

# 航空宇宙情報システム学第三 課題レポート

03-170360 森田涼介

2018 年 1 月 5 日

以下の 10 問に解答した。

- 2-4
- 2-7
- 3-2
- 3-8
- 4-5
- 5-1
- 6-1

## 練習問題 2-4

1.

$X$  の平均を  $\mu_X (= E[X])$ ,  $u = a \cdot X + b$  とおく。 $u$  の平均は, 期待値の性質を用いて,

$$\begin{aligned}\mu_u &= E[u] = E[a \cdot X + b] \\ &= a \cdot E[X] + b \\ &= a\mu_X + b\end{aligned}\tag{1}$$

よって,  $V(X) = E[(X - \mu_X)^2]$  であることに注意して,

$$\begin{aligned}V(a \cdot X + b) &= V(u) = E[(u - \mu_u)^2] \\ &= E[\{(aX + b) - (a\mu_X + b)\}^2] \\ &= E[(aX - a\mu_X)^2] \\ &= E[a^2(X - \mu_X)^2] \\ &= a^2 E[(X - \mu_X)^2] \\ &= a^2 V(X)\end{aligned}\tag{2}$$

2.

$Z = X + Y$  とおく。 $Z$  の平均は,

$$\mu_Z = E[Z] = E[X + Y] = E[X] + E[Y] = \mu_X + \mu_Y\tag{3}$$

よって, 分散は,

$$\begin{aligned}V(X + Y) &= V(Z) = E[(Z - \mu_Z)^2] \\ &= E[\{(X + Y) - (\mu_X + \mu_Y)\}^2] \\ &= E[\{(X - \mu_X) - (Y - \mu_Y)\}^2] \\ &= E[(X - \mu_X)^2 + 2(X - \mu_X)(Y - \mu_Y) + (Y - \mu_Y)^2] \\ &= E[(X - \mu_X)] + E[(Y - \mu_Y)^2] + 2E[(X - \mu_X)(Y - \mu_Y)] \\ &= V(X) + V(Y) + 2Cov(X, Y)\end{aligned}\tag{4}$$

3.

2. より,  $X$  と  $Y$  が無相関 ( $Cov(X, Y) = 0$ ) のとき,

$$V(X + Y) = V(X) + V(Y)\tag{5}$$

## 練習問題 2-7

- 物理量  $X$ : 平均  $\mu_X$ , 分散  $\sigma_X^2$
- 観測誤差  $W$ : 平均  $\mu_W = E[W] = 0$ , 分散  $\sigma_W^2 = V(W) = 1$

- 観測量  $Y = X + W$
- $E[XW] = 0$

このとき、 $X$  と  $W$  の共分散は、

$$\begin{aligned} \text{Cov}(X, W) &= E[(X - \mu_X)(W - \mu_W)] \\ &= E[XW] - \mu_X \mu_W \\ &= 0 \end{aligned} \tag{6}$$

$Y$  の平均・分散は、

$$\begin{aligned} \mu_Y &= E[Y] = E[X + W] \\ &= E[X] + E[W] \\ &= \mu_X + \mu_W \\ &= \mu_X \end{aligned} \tag{7}$$

$$\begin{aligned} \sigma_Y^2 &= V(Y) = V(X + W) \\ &= V(X) + V(W) + \text{Cov}(X, W) \\ &= \sigma_X^2 + \sigma_W^2 + 0 \\ &= \sigma_X^2 + 1 \end{aligned} \tag{8}$$

となる。ここで、 $V(X + Y)$  は、

$$V(X + Y) = V(2X + W) = V(2X) + V(W) + \text{Cov}(2X, W) \tag{9}$$

と表されるが、

$$\begin{aligned} V(2X) &= 2^2 V(X) = 4\sigma_X^2 \\ V(W) &= 1 \end{aligned} \tag{10}$$

であり、また、 $2X$  の平均は  $E[2X] = 2E[X] = 2\mu_X$  であるから、

$$\begin{aligned} \text{Cov}(2X, W) &= E[(2X - 2\mu_X)(W - \mu_W)] \\ &= 2E[(X - \mu_X)(W - \mu_W)] \\ &= 2\text{Cov}(X, W) \\ &= 0 \end{aligned} \tag{11}$$

となる。よって、

$$\begin{aligned} V(X + Y) &= V(2X) + V(W) + \text{Cov}(2X, W) \\ &= 4\sigma_X^2 + 1 \end{aligned} \tag{12}$$

一方、 $V(X + Y)$  は以下のようにも表される。

$$V(X + Y) = V(X) + V(Y) + \text{Cov}(X, Y) \tag{13}$$

よって、

$$\begin{aligned} \text{Cov}(X, Y) &= V(X + Y) - (V(X) + V(Y)) \\ &= (4\sigma_X^2 + 1) - (\sigma_X^2 + \sigma_X^2 + 1) \\ &= 2\sigma_X^2 \end{aligned} \tag{14}$$

## 練習問題 3-2

- 物理量 :  $X$
- 観測量  $Y_1 : Y_1 = X + W_1$
- 誤差量  $W_1 : E[W_1] = 0, V(W_1) = \sigma_1^2$
- 観測量  $Y_2 : Y_2 = X + W_2$
- 誤差量  $W_2 : E[W_2] = 0, V(W_2) = \sigma_2^2$
- $E[W_1 W_2] = 0$
- 推定量 :  $\hat{X} = aY_1 + bY_2$

(1)

$$\begin{aligned}
 E[\hat{X} - X] &= E[\hat{X}] - E[X] \\
 &= E[aY_1 + bY_2] - E[X] \\
 &= E[a(X + W_1) + b(X + W_2)] - E[X] \\
 &= E[(a + b)X] + E[aW_1] + E[bW_2] - E[X] \\
 &= E[(a + b - 1)X] \\
 &= (a + b - 1)E[X]
 \end{aligned} \tag{15}$$

これが 0 になるとき,  $E[X] = 0$  は一般に成立しないことから,

$$a + b = 1 \tag{16}$$

(2)

$a + b = 1$  のとき,

$$\begin{aligned}
 E[(\hat{X} - X)^2] &= E[(aW_1 + bW_2)^2] \\
 &= E[(a^2 W_1^2 + 2ab W_1 W_2 + b^2 W_2^2)] \\
 &= a^2 E[W_1^2] + b^2 E[W_2^2] + 2ab E[W_1 W_2]
 \end{aligned} \tag{17}$$

ここで,

$$\mu_1 = E[W_1] = 0 \tag{18}$$

$$\mu_2 = E[W_2] = 0 \tag{19}$$

とおくと,

$$V(W_1) = E[(W_1 - \mu_1)^2] = E[W_1^2] = \sigma_1^2 \tag{20}$$

$$V(W_2) = E[(W_2 - \mu_2)^2] = E[W_2^2] = \sigma_2^2 \tag{21}$$

$$E[W_1 W_2] = 0 \tag{22}$$

であるから,

$$E[(\hat{X} - X)^2] = a^2 \sigma_1^2 + b^2 \sigma_2^2 \tag{23}$$

となる。ここで、 $\partial b/\partial a = -1$  より、

$$\frac{\partial}{\partial a} E[(\hat{X} - X)^2] = 2a\sigma_1^2 + 2b \cdot (-1)\sigma_2^2 = 2(a\sigma_1^2 - b\sigma_2^2) \quad (24)$$

$$\frac{\partial^2}{\partial a^2} E[(\hat{X} - X)^2] = 2(\sigma_1^2 + \sigma_2^2) > 0 \quad (25)$$

よって、 $E[(\hat{X} - X)^2]$  は  $a$  について上に凸な関数であるから、これは  $\partial(E[(\hat{X} - X)^2])/\partial a = 0$  となる  $a$  で最小となる。

$$\therefore a\sigma_1^2 = b\sigma_2^2 \quad (26)$$

$$\Leftrightarrow a = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}, \quad b = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \quad (27)$$

また、このとき、 $E[(\hat{X} - X)^2]$  は以下のようにになる。

$$E[(\hat{X} - X)^2] = a^2\sigma_1^2 + b^2\sigma_2^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \quad (28)$$

## 1 練習問題 3-8

連続変数  $X$  が正規分布  $N(\mu, \sigma^2)$  に従うとき、 $X$  の確率分布は以下の式で表される。

$$p(X) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(X - \mu)^2}{2\sigma^2}\right) \quad (29)$$

よって、

$$\begin{aligned} H(X) &= E[-\log p(X)] \\ &= E\left[-\log\left\{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(X - \mu)^2}{2\sigma^2}\right)\right\}\right] \\ &= E\left[\frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(X - \mu)^2\right] \\ &= \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}E[(X - \mu)^2] \end{aligned} \quad (30)$$

ここで、 $E[(X - \mu)^2] = V(X) = \sigma^2$  より、

$$\begin{aligned} H(X) &= \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \cdot \sigma^2 \\ &= \frac{1}{2}(\log(2\pi\sigma^2) + 1) \\ &= \frac{1}{2}\log(2\pi e\sigma^2) \end{aligned} \quad (31)$$

## 練習問題 4-5

温度変化のモデルを、

$$T(t) = T_0 + a\sin(\omega t + \theta) + \varepsilon(t) \quad (32)$$

$$\varepsilon(t) \sim N(0, \sigma^2) \quad (33)$$

として、パラメータ  $T_0$ ,  $a$ ,  $\omega$ ,  $\theta$ ,  $\sigma^2$  を最尤推定する。ここで、誤差項  $\varepsilon(t)$  が正規分布に従うから、最尤法による解と最小二乗法による解は一致する。よって最小二乗法によってこれらのパラメータを求めることとする。

最小二乗法には Python の `scipy.optimize.leastsq` を用いた。また、それぞれのパラメータの初期値は、元データを概算で読み取り、それぞれ以下のようにした。

$$\begin{aligned} T_0 &= 13, & a &= 3.0 \\ \omega &= 2\pi/6000, & \theta &= 0.0 \end{aligned}$$

最尤推定の結果は以下ようになった。

表 1: 【練習問題 4-5】 最小二乗法によるパラメータ推定の結果

$T_0$ [°C]	:	12.67
$a$ [°C]	:	2.916
$\omega$ [/s]	:	0.001059
$\theta$	:	-0.1846
$\sigma^2$	:	0.3171

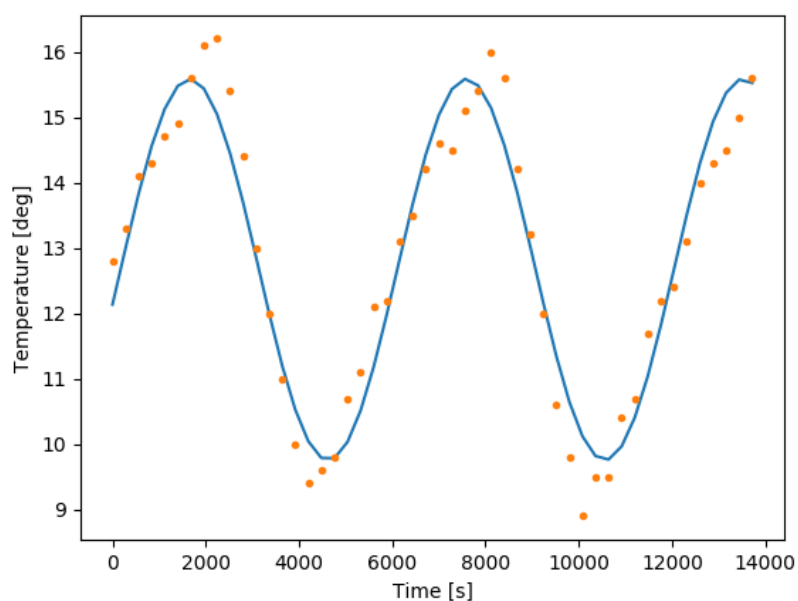


図 1: 【練習問題 4-5】 最尤推定によって求めたモデルのプロット

## 練習問題 5-1

中古車価格データの全てをプロットすると、以下の図 2 のようになる。

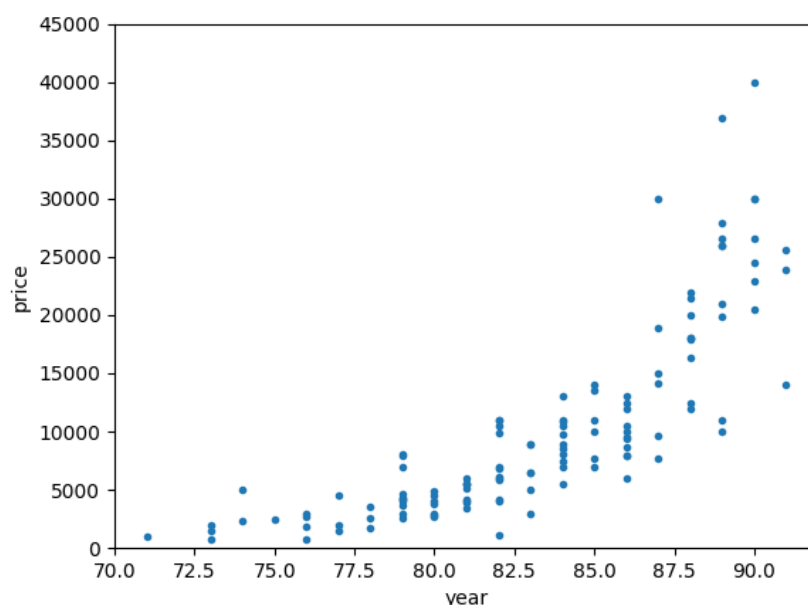


図 2: 【練習問題 5-1】 中古車価格データの完全版

これについて 1~4 次式モデルで多項式回帰を行う。ここでは最小二乗法を用いて解を求めることとする。

モデルの優劣の決め方について述べる。まず完全版データを教師データとテストデータに分け、教師データを用いてモデルを求め、テストデータでそのモデルを評価する。教師データは乱数によって選択することとし、その乱数の seed をいくつか変えて複数回試行を行う。モデル作成時の AICc とテストデータに対する  $Q$  (最小二乗和) を求め、それらを単純平均することによってモデルの評価を行う。

なお、赤池情報量基準については、 $n$  が小さいことから、

$$\text{AICc} = n \log \frac{\hat{Q}}{n} + \frac{2kn}{n-k-1} \quad (34)$$

を用いる。

ここでは試行回数を 100 回とする。また、配布資料に倣って、教師データの数 を 10 としてモデルを求めたところ、その結果は以下ようになった。なお、図は適当な seed のときのモデルをプロットしたものである。

表 2: 【練習問題 5-1】 テストデータ 10 個のときの各次元に対する AICc と  $Q$  の平均値

次元	AICc	$Q$
1	205.27	$3.80851 \times 10^9$
2	215.66	$3.80475 \times 10^9$
3	282.89	$4.98813 \times 10^{10}$
4	116.06	$3.60880 \times 10^{10}$

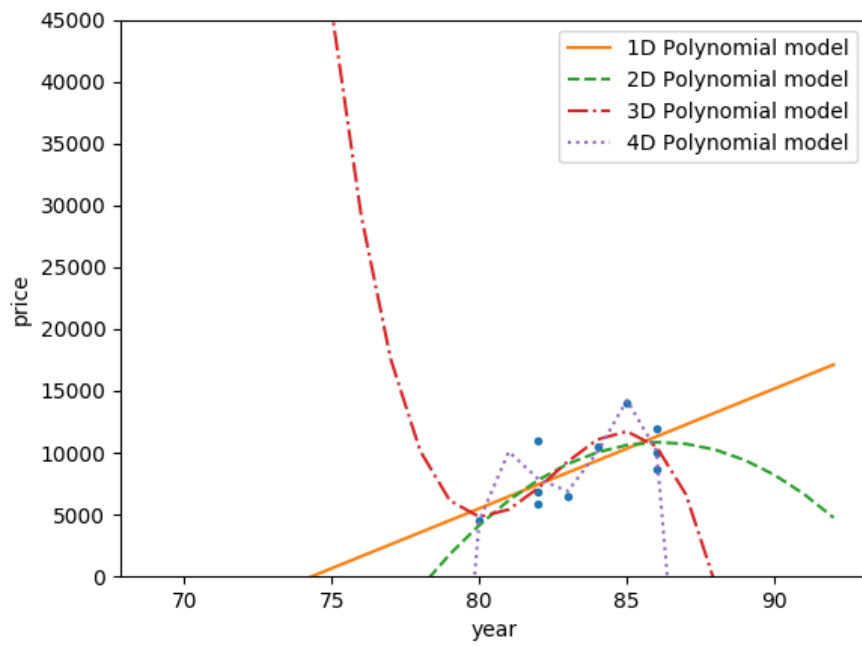


図 3: テストデータ 10 個のときの train 時の fitting の図

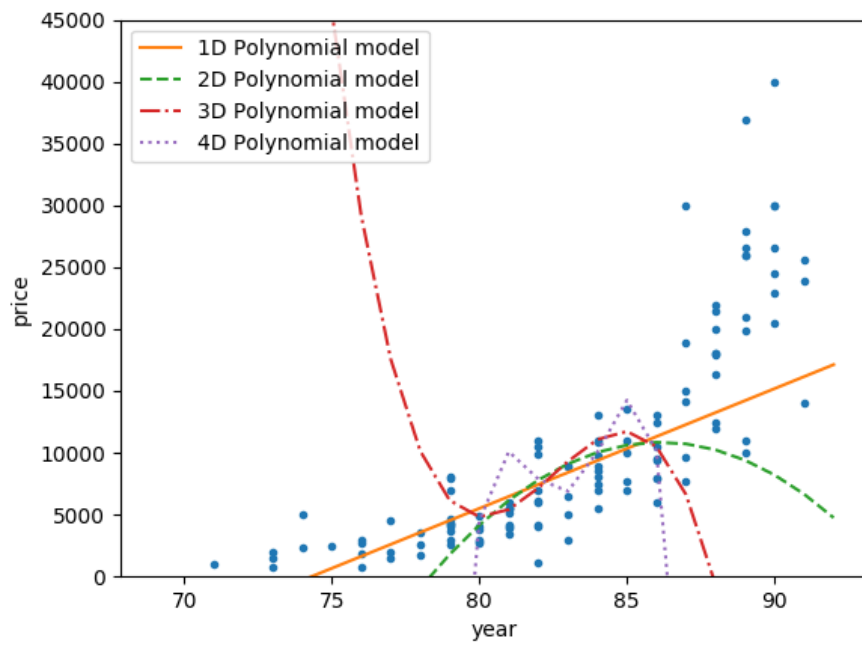


図 4: テストデータ 10 個のときの test 時の fitting の図



表 2 より,  $AICc$  の値からすれば 4 次が,  $Q$  の値からすれば 1, 2 次が優れているといえるが, 図 3, 4 を見れば 2, 4 次は妥当ではないと思われる。これは, テストデータが少なすぎるため, 2 次ではうまく fit できず, また 4 次では過学習してしまって  $AICc$  が小さくなっているのだと考えられる。従って 1 次式が優れたモデルである, と言えなくもないが, 元データは 1 次式というよりは曲線に見える。

以上の問題はテストデータが少なすぎるために起こっていると考えられる。そこで, 全てのデータの 30% をテストに利用した場合について調べてみたところ, その結果は以下のようになった。

表 3: 【練習問題 5-1】 テストデータ 30% のときの各次元に対する  $AICc$  と  $Q$  の平均値

次元	$AICc$	$Q$
1	671.43	$2.51412 \times 10^9$
2	662.66	$1.84339 \times 10^9$
3	667.49	$1.94358 \times 10^9$
4	676.40	$2.32340 \times 10^9$

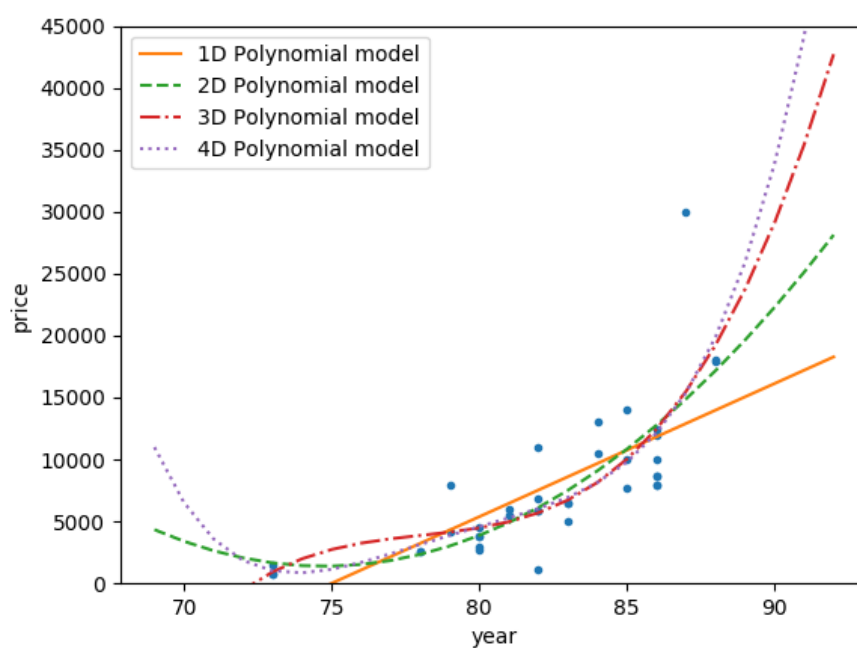


図 5: テストデータ 30% のときの train 時の fitting の図

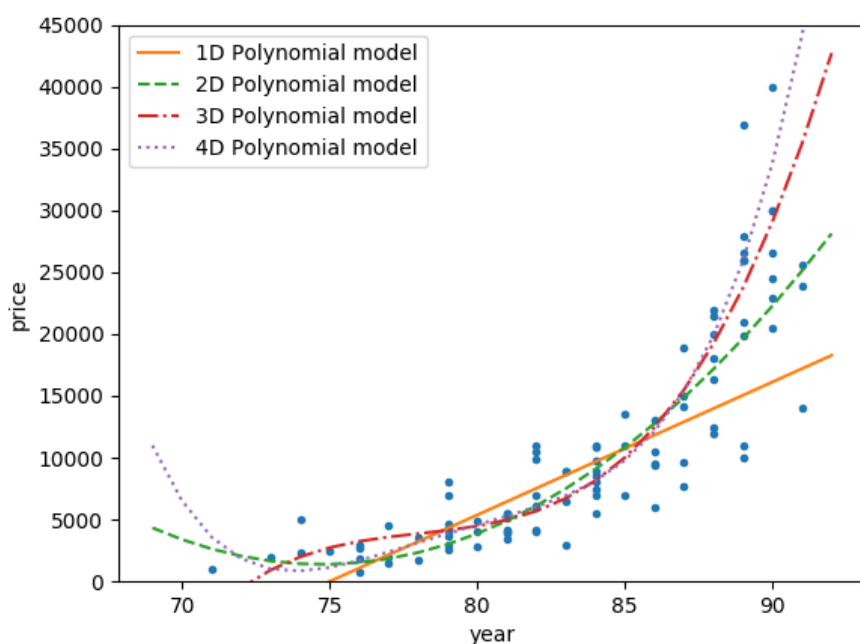


図 6: テストデータ 30% のときの test 時の fitting の図

表 3 より, テストデータを全体の 30% としたところ, AICc,  $Q$  ともに 2 次が最も優れている。また, 図を見ても大きな違和感はない。

以上の考察により, 2 次式モデルが最も優れているといえる。

## 練習問題 6-1

(1)

ノイズ (誤差項)  $\varepsilon$  が正規分布  $N(0, 1)$  に従うから, 最尤推定による解と最小二乗法による解は一致する。`scipy.optimize.leastsq` を用いて最小二乗法による解を求めると,

$$a = 1.059 \quad (35)$$

と求まる。

(2)

$a$  の事前分布が  $p(a) = N(1, 0.09)$  であるとする。ノイズの分散を  $\sigma_\varepsilon^2$ ,  $a$  の平均を  $\mu_a$ , 分散を  $\sigma_a^2$  とする。まず一般の線形回帰問題を考える。以下のようなモデルの MAP 推定を行う。

$$y = \mathbf{a}^T \mathbf{X} + \varepsilon \quad (36)$$

$$\varepsilon \sim N(0, \sigma_\varepsilon^2) \quad (37)$$

$$p(\mathbf{a}) = N(\mu_a, \sigma_a^2) \quad (38)$$

対数尤度関数は以下ようになる。

$$\begin{aligned}
LL(\mathbf{a}, \sigma_e^2) &= \log \prod_{i=1}^N p(y_i | \mathbf{x}_i, \mathbf{a}, \sigma_e^2) p(\mathbf{a}) \\
&= \sum_{i=1}^N \log \{ p(y_i | \mathbf{x}_i, \mathbf{a}, \sigma_e^2) p(\mathbf{a}) \} \\
&= \sum_{i=1}^N \log \left\{ \frac{1}{\sqrt{2\pi\sigma_e^2}} \exp\left(-\frac{(\varepsilon_i)^2}{2\sigma_e^2}\right) \cdot \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp\left(-\frac{(a_i - \mu_a)^2}{2\sigma_a^2}\right) \right\} \\
&= \sum_{i=1}^N \left\{ -\frac{1}{2\sigma_e^2} \varepsilon_i^2 - \frac{1}{2\sigma_a^2} (a_i - \mu_a)^2 \right\} - \frac{N}{2} \{ \log(2\pi\sigma_e^2) + \log(2\pi\sigma_a^2) \} \\
&= -\frac{1}{2\sigma_e^2} \left( \|\boldsymbol{\varepsilon}\|^2 + \frac{\sigma_e^2}{\sigma_a^2} \|(\mathbf{a} - \mu_a)\|^2 \right) - N \log(2\pi\sigma_e\sigma_a) \\
&= -\frac{1}{2\sigma_e^2} \left( \|\mathbf{y} - \mathbf{X}\mathbf{a}\|^2 + \frac{\sigma_e^2}{\sigma_a^2} \|(\mathbf{a} - \mu_a)\|^2 \right) - N \log(2\pi\sigma_e\sigma_a)
\end{aligned}$$

よって,

$$LL(\mathbf{a}, \sigma_e^2) \text{ が最大} \Leftrightarrow \|\mathbf{y} - \mathbf{X}\mathbf{a}\|^2 + (\sigma_e^2/\sigma_a^2) \|(\mathbf{a} - \mu_a)\|^2 \text{ が最小} \quad (39)$$

ここで,

$$\lambda = \frac{\sigma_e^2}{\sigma_a^2} \quad (40)$$

とおくと,  $\|\mathbf{y} - \mathbf{X}\mathbf{a}\|^2 + \lambda \|(\mathbf{a} - \mu_a)\|^2$  を最小にすればよいことがわかる。

ここで, 配布資料より,

$$\begin{aligned}
\log p(\mathbf{a} | \mathbf{X}, \mathbf{y}, \sigma^2) &= -\frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{X}\mathbf{a}\|^2 - \frac{\lambda}{2\sigma^2} \|\mathbf{a}\|^2 + \text{Const.} \\
&= -\frac{1}{2\sigma^2} (\|\mathbf{y} - \mathbf{X}\mathbf{a}\|^2 + \lambda \|\mathbf{a}\|^2) + \text{Const.}
\end{aligned} \quad (41)$$

を最大にする  $\mathbf{a}$  は,

$$\hat{\mathbf{a}}_{\text{MAP}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{y}) \quad (42)$$

となることがわかる。よって,

$$\mathbf{b} = \mathbf{a} - \mu_a, \quad \mathbf{z} = \mathbf{y} - \mu_a \mathbf{X} \quad (43)$$

と変換すると,

$$\|\mathbf{y} - \mathbf{X}\mathbf{a}\|^2 + \lambda \|(\mathbf{a} - \mu_a)\|^2 = \|\mathbf{z} - \mathbf{X}\mathbf{b}\|^2 + \lambda \|\mathbf{b}\|^2 \quad (44)$$

を最小にすればよいから, その解は,

$$\hat{\mathbf{b}}_{\text{MAP}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{z}) \quad (45)$$

結局，対数尤度関数  $LL(\mathbf{a}, \sigma_e^2)$  を最大にする  $\mathbf{a}$  は，

$$\begin{aligned}\hat{\mathbf{a}}_{\text{MAP}} &= \hat{\mathbf{b}}_{\text{MAP}} + \mu_a \\ &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{z}) + \mu_a \\ &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} (\mathbf{X}^T \mathbf{y} - \mu_a \mathbf{X}^T \mathbf{X}) + \mu_a\end{aligned}\tag{46}$$

と表せる。

式 (46) に，

$$\mu_a = 1\tag{47}$$

$$\lambda = \frac{\sigma_e^2}{\sigma_a^2} = \frac{1}{0.09} = \frac{100}{9}\tag{48}$$

$$(49)$$

と  $\mathbf{X}, \mathbf{y}$  を代入すると，

$$a = 1.028\tag{50}$$

を得る。

以下に MLE, MAP 推定それぞれの結果のグラフを示した。

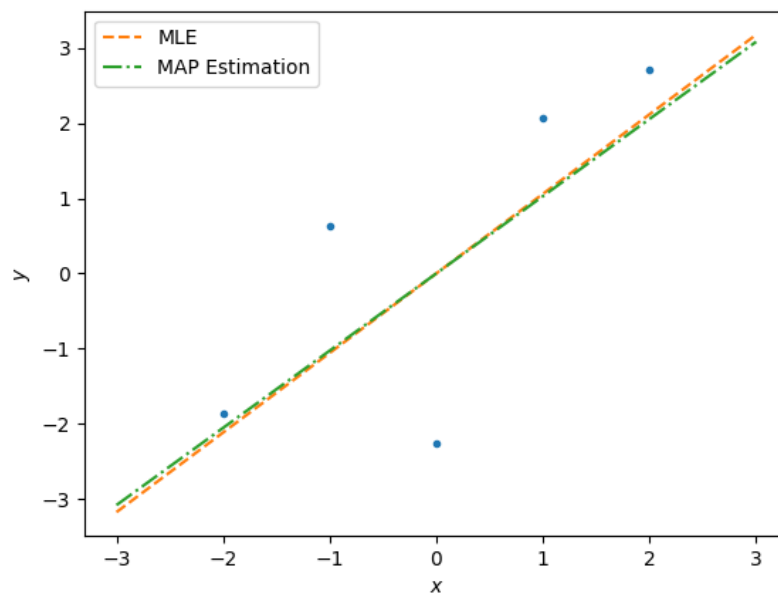


図 7: 【練習問題 6-1】 MLE と MAP 推定による線形回帰のグラフ

## 2 プログラム

プログラムには Python3 を用いた。

Listing 1: system\_report.py

```

1  #!/usr/bin/env Python3
2  # -*- coding: utf-8 -*-
3
4
5
6  ## modules -----
7  import numpy as np
8  from scipy import optimize
9  import csv
10 import matplotlib.pyplot as plt
11
12
13
14 ## functions -----
15 def main():
16     #q_1_2()
17     #q_2_2()
18     #q_4_5()
19     #q_4_5_2()
20     #q_5_1()
21     #q_5_3()
22     q_5_4()
23     #q_6_1()
24
25
26 def q_1_2():
27     q = Q_1_2()
28     q.calc()
29     q.plot()
30
31
32 def q_2_2():
33     q = Q_2_2()
34     q.mkElist()
35     q.plot_sum_eq()
36     q.plot_mlt()
37
38
39 def q_4_5():
40     q = Q_4_5()
41     q.calc()
42     q.print_result()
43     q.plot()
44
45
46 def q_4_5_2():

```

```

47     q = Q_4_5_2()
48     q.calc()
49     q.print_result()
50     q.plot()
51
52
53 def q_5_1():
54     np.random.seed(1) # 適当な seed を選ぶ seed 固定
55     seeds = np.random.randint(0, 1000000000, 100) # 適当な seed を選ぶ
56     AICc_sum = np.zeros(4, dtype=float)
57     Q_sum = np.zeros(4, dtype=float)
58     for seed in seeds: # 各 seed について計算
59         q = Q_5_1(seed)
60         q.calc()
61         #q.print_result()
62         #q.plot_train()
63         #q.plot_test()
64         for i in range(4):
65             AICc_sum[i] += q.result_list[i][2]
66             Q_sum[i] += q.result_list[i][3]
67     AICc_mean = AICc_sum / len(seeds)
68     Q_mean = Q_sum / len(seeds)
69     AICs_min_idx = np.argmin(AICc_mean)
70     Q_min_idx = np.argmin(Q_mean)
71     for i in range(4):
72         dim = i + 1
73         print("Dimension : {}".format(dim))
74         print("\t AICc ({} times mean) : {}".format(len(seeds),
75             AICc_mean[i]))
76         print("\t Q ({} times mean) : {}".format(len(seeds), Q_mean[i]))
77         print()
78     print("valid dimension is {} from AICc value".format(AICs_min_idx + 1))
79     print("valid dimension is {} from Q value".format(Q_min_idx + 1))
80
81 def q_5_3():
82     q = Q_5_3()
83     q.calc()
84     q.print_result()
85     q.plot()
86
87
88 def q_5_4():
89     q = Q_5_4()
90     q.calc()
91     q.plot()
92

```

```

93
94 def q_6_1():
95     q = Q_6_1()
96     q.calc1()
97     q.print_result(1)
98     q.calc2()
99     q.print_result(2)
100    q.plot()
101
102
103
104 ## classes -----
105 class Q_1_2:
106     def __init__(self):
107         self.t_f = 30.0 # s
108
109         self.T_analysis = np.arange(0, self.t_f, 0.001)
110         self.X_analysis = np.zeros_like(self.T_analysis)
111
112         self.dt = 1.0 # s
113         self.N = int(self.t_f/self.dt)
114         self.T_pts = np.arange(0, self.t_f, self.dt)
115         self.X_pts = np.zeros((self.N, 2), dtype=float)
116         self.X_pts[0, :] = np.array([1, 0], dtype=float)
117
118         self.A_dash = np.array([[0.582, 0.727], [-0.727, 0.364]],
119 dtype=float)
120         self.B_dash = np.array([0.418, 0.727], dtype=float)
121
122     def u(self, t):
123         return np.sin(2 * t)
124
125     def x_analysis(self, t):
126         alpha = -3/20
127         beta = np.sqrt(391)/20
128         first = (83/78) * np.exp(alpha * t) * (np.cos(beta * t) +
129 (1249/1560/beta)*np.sin(beta * t))
130         second = -(5/78) * (np.cos(2 * t) + 5 * np.sin(2 * t))
131         return first + second
132
133     def x_pts(self, k):
134         t = k * self.dt
135         x_k = self.X_pts[k, :]
136         x_kplus1 = np.dot(self.A_dash, x_k) + self.B_dash * self.u(t)
137         return x_kplus1
138
139     def calc(self):

```

```

138         self.X_analysis = self.x_analysis(self.T_analysis)
139         for k in range(0, self.N-1):
140             self.X_pts[k+1, :] = self.x_pts(k)
141             #print(self.X_analysis)
142             #print(self.X_pts)
143
144     def plot(self):
145         plt.plot(self.T_analysis, self.X_analysis, label="analyzed solution")
146         plt.plot(self.T_pts, self.X_pts[:, 0], label="parsed-time-system
solution ($\Delta t = 1.0$)")
147         plt.xlabel("$t[s]$")
148         plt.ylabel("$x[m]$")
149         plt.legend()
150         plt.savefig("../figures/q_1_2.png")
151         plt.show()
152
153
154 class Q_2_2:
155     def __init__(self):
156         self.n = 1000
157         self.N = np.arange(0, self.n, 1)
158         self.sum_sq_solution = (91/3) * np.ones(self.n, dtype=float)
159         self.mlt_solution = (49/4) * np.ones(self.n, dtype=float)
160         self.seeds = [10, 100, 1000] # seeds for np.random
161         self.E_sum_sq_list, self.E_mlt_list = [], []
162
163     def mkrndXY(self, seed):
164         np.random.seed(seed)
165         x = np.random.randint(1, 7, self.n)
166         y = np.random.randint(1, 7, self.n)
167         sum_sq = x ** 2 + y ** 2
168         mlt = x * y
169         # print(sum_sq)
170         # print(mlt)
171         return sum_sq, mlt
172
173     def calc_E(self, seed):
174         sum_sq, mlt = self.mkrndXY(seed)
175         E_sum_sq = np.zeros(self.n, dtype=float)
176         E_mlt = np.zeros(self.n, dtype=float)
177         for i in range(1, self.n):
178             E_sum_sq[i] = (E_sum_sq[i - 1] * (i - 1) + sum_sq[i]) / i
179             E_mlt[i] = (E_mlt[i - 1] * (i - 1) + mlt[i]) / i
180             # print(E_sum_sq)
181             # print(E_mlt)
182         return E_sum_sq, E_mlt
183

```



```

184     def mkElist(self):
185         for seed in self.seeds:
186             E_sum_sq, E_mlt = self.calc_E(seed)
187             self.E_sum_sq_list.append(E_sum_sq)
188             self.E_mlt_list.append(E_mlt)
189
190     def plot_sum_eq(self):
191         for E, seed in zip(self.E_sum_sq_list, self.seeds):
192             plt.plot(self.N, E, label="seed = {}".format(seed))
193         plt.plot(self.N, self.sum_sq_solution)
194         plt.xlim([1, self.n])
195         plt.ylim([20, 40])
196         plt.xlabel("$n$")
197         plt.ylabel("$E[X^2 + Y^2]$")
198         plt.legend()
199         plt.show()
200
201     def plot_mlt(self):
202         for E, seed in zip(self.E_mlt_list, self.seeds):
203             plt.plot(self.N, E, label="seed = {}".format(seed))
204         plt.plot(self.N, self.mlt_solution)
205         plt.xlim([1, self.n])
206         plt.ylim([5, 17.5])
207         plt.xlabel("$n$")
208         plt.ylabel("$E[XY]$")
209         plt.legend()
210         plt.savefig("../figures/q_2_2.png")
211         plt.show()
212
213
214     class Q_4_5:
215         def __init__(self):
216             self.smoking_rate = np.array([18.2, 25.82, 18.24, 28.6, 31.1, 33.6,
217             40.46, 28.27, 20.1, 27.91, 26.18, 22.12])
218             self.death_rate = np.array([17.05, 19.8, 15.98, 22.07, 22.83, 24.55,
219             27.27, 23.57, 13.58, 22.8, 20.3, 16.59])
220             self.a = 0
221             self.b = 0
222             self.y_curve = lambda x: self.a * x + self.b
223             self.e = np.zeros_like(self.smoking_rate)
224             self.mu = 0
225             self.sigma_sq = 0
226
227     def least_square(self, x, y):
228         c = np.polyfit(x, y, 1)
229         self.a = c[0]
230         self.b = c[1]

```

```

229         self.y_curve = np.poly1d(c)
230         return self.y_curve
231
232     def calc(self):
233         self.least_square(self.smoking_rate, self.death_rate)
234         y_pred = self.y_curve(self.smoking_rate)
235         self.e = self.death_rate - y_pred
236         self.mu = np.average(self.e)
237         self.sigma_sq = np.var(self.e)
238
239     def print_result(self):
240         print("a          : {}".format(self.a))
241         print("b          : {}".format(self.b))
242         print("mean       : {}".format(self.mu))
243         print("variance  : {}".format(self.sigma_sq))
244
245     def plot(self):
246         x_min = int(np.min(self.smoking_rate) - 3)
247         x_max = int(np.max(self.smoking_rate) + 3)
248         x = np.arange(x_min, x_max, 1)
249         plt.plot(x, self.y_curve(x), label="1D")
250         plt.plot(self.smoking_rate, self.death_rate, ".", label="real data")
251         plt.xlabel("smoking rate [%]")
252         plt.ylabel("death rate [%]")
253         plt.savefig("../figures/q_4_5.png")
254         plt.show()
255
256
257     class Q_4_5_2:
258         def __init__(self):
259             self.Time = np.arange(0, 13900, 280)
260             self.Temp = np.array([12.8, 13.3, 14.1, 14.3, 14.7, 14.9, 15.6,
261 16.1, 16.2, 15.4,
262                                     14.4, 13.0, 12.0, 11.0, 10.0, 9.4, 9.6, 9.8,
263 10.7, 11.1,
264                                     12.1, 12.2, 13.1, 13.5, 14.2, 14.6, 14.5,
265 15.1, 15.4, 16.0,
266                                     15.6, 14.2, 13.2, 12.0, 10.6, 9.8, 8.9, 9.5,
267 9.5, 10.4,
268                                     10.7, 11.7, 12.2, 12.4, 13.1, 14.0, 14.3,
269 14.5, 15.0, 15.6])
270             self.params = np.array([13.0, 3.0, 2*np.pi/6000.0, 0.1], dtype=float)
271             self.e = np.zeros_like(self.Time)
272             self.mu = 0
273             self.sigma_sq = 0
274
275     def T_model(self, param, t):

```

```

271         T_0 = param[0]
272         a = param[1]
273         omega = param[2]
274         theta = param[3]
275         T_pred = T_0 + a * np.sin(omega * t + theta)
276         return T_pred
277
278     def residual_func(self, param, t, T_observed):
279         T_pred = self.T_model(param, t)
280         return T_observed - T_pred
281
282     def least_square(self):
283         result = optimize.leastsq(self.residual_func, self.params,
284         args=(self.Time, self.Temp))
285         self.params = result[0]
286         return self.params
287
288     def calc(self):
289         self.least_square()
290         self.e = self.residual_func(self.params, self.Time, self.Temp)
291         self.mu = np.average(self.e)
292         self.sigma_sq = np.var(self.e)
293
294     def print_result(self):
295         print("T0          : {}".format(self.params[0]))
296         print("a          : {}".format(self.params[1]))
297         print("omega       : {}".format(self.params[2]))
298         print("theta      : {}".format(self.params[3]))
299         print("mean       : {}".format(self.mu))
300         print("variance  : {}".format(self.sigma_sq))
301
302     def plot(self):
303         t = np.arange(0, 13900, 280)
304         plt.plot(t, self.T_model(self.params, t), label="model")
305         plt.plot(self.Time, self.Temp, ".", label="real data")
306         plt.xlabel("Time [s]")
307         plt.ylabel("Temperature [deg]")
308         plt.savefig("../figures/q_4_5_2.png")
309         plt.show()
310
311 class Q_5_1:
312     def __init__(self, seed=10):
313         self.year, self.price = self.load_data()
314         self.sample = int(0.3 * len(self.year))      # use 30% of the data
315         for training

```

```

315         self.year_train, self.year_test, self.price_train, self.price_test =
\
316             self.parse_data(self.year, self.price, self.sample, seed)
317         self.n_dim = 4          # max dimension for fitting
318
319         self.params_list = []
320         self.result_list = []
321         for i in range(self.n_dim):
322             dim = i + 1
323             params = np.ones(dim+1, dtype=float)      # list for storing
parameters (like a_0, a_1, ...)
324             result = np.zeros(4, dtype=float)        # list for storing mean,
variance, AICc, and Q
325             self.params_list.append(params)
326             self.result_list.append(result)
327             self.polynomial_list = [self.poly_1D, self.poly_2D, self.poly_3D,
self.poly_4D]
328             self.residual_func_list = [self.residual_func_1D,
self.residual_func_2D, self.residual_func_3D, self.residual_func_4D]
329             self.linestyle_list = ["solid", "dashed", "dashdot", "dotted"]    #
line style for each dimensions
330
331     def load_data(self):
332         with open("../data/mazda_cars.txt", "r") as f:
333             f.readline()      # delete header
334             year_list = []
335             price_list = []
336             for line in f:
337                 year_str, price_str = line.split("\t")
338                 year = int(year_str)
339                 price = int(price_str)
340                 year_list.append(year)
341                 price_list.append(price)
342             year = np.array(year_list)
343             price = np.array(price_list)
344             return year, price
345
346     def parse_data(self, x, y, sample, seed=10):
347         n = len(x)
348         np.random.seed(seed)
349         idx_list = np.random.randint(0, n-1, sample)
350         x_train = np.zeros(sample, dtype=float)
351         y_train = np.zeros(sample, dtype=float)
352         for i, idx in enumerate(idx_list):
353             x_train[i] = x[idx]
354             y_train[i] = y[idx]
355         x_test = np.delete(x, idx_list)

```

```

356         y_test = np.delete(y, idx_list)
357         return x_train, x_test, y_train, y_test
358
359     def poly_1D(self, params, x):
360         return params[0] + params[1] * x
361
362     def poly_2D(self, params, x):
363         return params[0] + params[1] * x + params[2] * x**2
364
365     def poly_3D(self, params, x):
366         return params[0] + params[1] * x + params[2] * x**2 + params[3] *
x**3
367
368     def poly_4D(self, params, x):
369         return params[0] + params[1] * x + params[2] * x**2 + params[3] *
x**3 + params[4] * x**4
370
371     def residual_func_1D(self, params, x, price_known):
372         price_pred = self.poly_1D(params, x)
373         return price_known - price_pred
374
375     def residual_func_2D(self, params, x, price_known):
376         price_pred = self.poly_2D(params, x)
377         return price_known - price_pred
378
379     def residual_func_3D(self, params, x, price_known):
380         price_pred = self.poly_3D(params, x)
381         return price_known - price_pred
382
383     def residual_func_4D(self, params, x, price_known):
384         price_pred = self.poly_4D(params, x)
385         return price_known - price_pred
386
387     def least_square(self, residual_func, params):
388         result = optimize.leastsq(residual_func, params,
args=(self.year_train, self.price_train))
389         params = result[0]
390         return params
391
392     def AICc(self, n, k, Q):
393         aicc = n * np.log(Q/n) + (2*k*n)/(n-2*k-1)      # for not so large n
394         return aicc
395
396     def calc_params(self):
397         for i in range(self.n_dim):
398             func = self.residual_func_list[i]
399             params = self.params_list[i]

```

```

400         params_calced = self.least_square(func, params)
401         e = func(params_calced, self.year_train, self.price_train)
402         self.result_list[i][0] = np.average(e)    # mean
403         self.result_list[i][1] = np.var(e)        # variance
404         self.params_list[i] = params_calced
405
406     def calc_train_AICc(self):
407         n = len(self.year_train)
408         for i in range(self.n_dim):
409             func = self.residual_func_list[i]
410             params = self.params_list[i]
411             e = func(params, self.year_test, self.price_test)
412             Q = (np.linalg.norm(e))**2
413             k = len(params)
414             aicc = self.AICc(n, k, Q)
415             self.result_list[i][2] = aicc
416
417     def calc_test_Q(self):
418         for i in range(self.n_dim):
419             func = self.residual_func_list[i]
420             params = self.params_list[i]
421             e = func(params, self.year_test, self.price_test)
422             Q = (np.linalg.norm(e))**2
423             self.result_list[i][3] = Q
424
425     def calc(self):
426         self.calc_params()
427         self.calc_train_AICc()
428         self.calc_test_Q()
429
430     def print_result(self):
431         for i in range(self.n_dim):
432             dim = i + 1
433             params = self.params_list[i]
434             result = self.result_list[i]
435             print("Dimension : {}".format(dim))
436             for j in range(dim + 1):
437                 print("\t a{}          : {}".format(j, params[j]))
438             print("\t mean          : {}".format(result[0]))
439             print("\t variance    : {}".format(result[1]))
440             print("\t AICc        : {}".format(result[2]))
441             print("\t Q (test)   : {}".format(result[3]))
442             print()
443
444     def plot_train(self):
445         plt.plot(self.year_train, self.price_train, ".")
446         x = np.arange(self.year.min()-2, self.year.max()+2, 1)

```

```

447         for i in range(self.n_dim):
448             func = self.polynomial_list[i]
449             params = self.params_list[i]
450             plt.plot(x, func(params, x), linestyle=self.linestyle_list[i],
label="{}D Polynomial model".format(i+1))
451             plt.xlabel("year")
452             plt.ylabel("price")
453             plt.ylim([0, self.price.max()+5000])
454             plt.legend()
455             plt.savefig("../figures/q_5_1_train.png")
456             plt.show()
457
458     def plot_test(self):
459         plt.plot(self.year_test, self.price_test, ".")
460         x = np.arange(self.year.min()-2, self.year.max()+2, 1)
461         for i in range(self.n_dim):
462             func = self.polynomial_list[i]
463             params = self.params_list[i]
464             plt.plot(x, func(params, x), linestyle=self.linestyle_list[i],
label="{}D Polynomial model".format(i+1))
465             plt.xlabel("year")
466             plt.ylabel("price")
467             plt.ylim([0, self.price.max()+5000])
468             plt.legend()
469             plt.savefig("../figures/q_5_1_test.png")
470             plt.show()
471
472     def plot_all_point(self):
473         plt.plot(self.year, self.price, ".")
474         plt.xlabel("year")
475         plt.ylabel("price")
476         plt.ylim([0, self.price.max()+5000])
477         plt.legend()
478         plt.savefig("../figures/q_5_1_all.png")
479         plt.show()
480
481
482     class Q_5_3:
483         def __init__(self):
484             self.x = np.array([0.032, 0.034, 0.214, 0.263, 0.275, 0.275, 0.45,
0.5,
485                               0.5, 0.63, 0.8, 0.9, 0.9, 0.9, 0.9, 1.0,
486                               1.1, 1.1, 1.4, 1.7, 2.0, 2.0, 2.0, 2.0],
dtype=float)
487             self.y = np.array([170, 290, -130, -70, -185, -220, 200, 290,
270, 200, 300, -30, 650, 150, 500, 920,

```

```

489         450, 500, 500, 960, 500, 850, 800, 1090],
dtype=float)
490     self.polynomial_list = [self.poly_1D_noconst, self.poly_1D]
491     self.residual_func_list = [self.residual_func_1D_noconst,
self.residual_func_1D]
492     poly_1D_noconst_params = np.array([0], dtype=float)
493     poly_1D_params = np.array([0, 0], dtype=float)
494     self.params_list = [poly_1D_noconst_params, poly_1D_params]
495     poly_1D_noconst_result = np.array([0, 0, 0], dtype=float)
496     poly_1D_result = np.array([0, 0, 0], dtype=float)
497     self.result_list = [poly_1D_noconst_result, poly_1D_result]
498
499     def poly_1D_noconst(self, params, x):
500         return params[0] * x
501
502     def poly_1D(self, params, x):
503         return params[0] + params[1] * x
504
505     def residual_func_1D_noconst(self, params, x, y_known):
506         y_pred = self.poly_1D_noconst(params, x)
507         return y_known - y_pred
508
509     def residual_func_1D(self, params, x, y_known):
510         y_pred = self.poly_1D(params, x)
511         return y_known - y_pred
512
513     def least_square(self, residual_func, params):
514         result = optimize.leastsq(residual_func, params, args=(self.x,
self.y))
515         params = result[0]
516         return params
517
518     def AICc(self, n, k, Q):
519         aicc = n * np.log(Q/n) + (2*k*n)/(n-2*k-1)
520         return aicc
521
522     def calc_AICc(self):
523         n = len(self.x)
524         for i in range(2):
525             func = self.residual_func_list[i]
526             params = self.params_list[i]
527             e = func(params, self.x, self.y)
528             Q = (np.linalg.norm(e))**2
529             k = len(params)
530             aicc = self.AICc(n, k, Q)
531             self.result_list[i][2] = aicc
532

```



```

533     def calc(self):
534         for i in range(2):
535             func = self.residual_func_list[i]
536             params = self.params_list[i]
537             params_calced = self.least_square(func, params)
538             e = func(params_calced, self.x, self.y)
539             self.result_list[i][0] = np.average(e)
540             self.result_list[i][1] = np.var(e)
541             self.params_list[i] = params_calced
542         self.calc_AICc()
543
544     def print_result(self):
545         title_list = ["without const.", "with const."]
546         for i in range(2):
547             params = self.params_list[i]
548             result = self.result_list[i]
549             print(title_list[i])
550             for j, param in enumerate(params):
551                 print("    a{}          : {}".format(j, param))
552             print("    mean          : {}".format(result[0]))
553             print("    variance       : {}".format(result[1]))
554             print("    AICc          : {}".format(result[2]))
555             print()
556
557     def plot(self):
558         title_list = ["without const.", "with const."]
559         plt.plot(self.x, self.y, ".", label="real data")
560         x = np.arange(self.x.min()-1, self.x.max()+1, 1)
561         for i in range(2):
562             func = self.polynomial_list[i]
563             params = self.params_list[i]
564             plt.plot(x, func(params, x), label="1D Polynomial model
({})".format(title_list[i]))
565         plt.xlabel("distance")
566         plt.ylabel("recession velocity")
567         plt.legend()
568         plt.savefig("../figures/q_5_3.png")
569         plt.show()
570
571
572 class Q_5_4:
573     def __init__(self):
574         self.month = np.arange(0, 12*15+10, 1)
575         self.temp = self.load_data_from_npy()
576         self.n = 12*15
577         self.temp_train = self.temp[:self.n]
578         self.temp_test = self.temp[self.n:]

```

```

579         self.month_train = self.month[:self.n]
580         self.month_test = self.month[self.n:]
581         self.m_list = [1, 2, 12, 24]
582         self.temp_pred_list = []
583
584     def load_data_from_csv(self):
585         temp_list = []
586         with open("../data/TokyoTemperatureSince2001.csv", "r") as f:
587             reader = csv.reader(f)
588             for row in reader:
589                 temp_str = row[1]
590                 temp = float(temp_str)
591                 temp_list.append(temp)
592         temp = np.array(temp_list, dtype=float)
593         np.save("../data/q_5_4.npy", temp)
594         return temp
595
596     def load_data_from_npy(self):
597         temp = np.load("../data/q_5_4.npy")
598         return temp
599
600     def coeff(self, m, i, j):
601         x_i = self.temp_train[m+1-i: self.n-i].copy()
602         x_j = self.temp_train[m+1-j: self.n-j].copy()
603         c = np.dot(x_i, x_j)
604         return c
605
606     def calc_a(self, m):
607         C_ij = np.zeros((m, m), dtype=float)
608         C_i0 = np.zeros(m, dtype=float)
609         for idx_i in range(m):
610             i = idx_i + 1
611             C_i0[idx_i] = self.coeff(m, i, 0)
612             for idx_j in range(m):
613                 j = idx_j + 1
614                 C_ij[idx_i, idx_j] = self.coeff(m, i, j)
615         A = np.linalg.solve(C_ij, C_i0)
616         return A
617
618     def calc(self):
619         for m in self.m_list:
620             A_m = self.calc_a(m)
621             temp_future = np.zeros(self.n-m, dtype=float)
622             for t in range(m+1, self.n):
623                 temp_past = self.temp[t-1:t-m-1:-1]
624                 temp_future[t-(m+1)] = np.dot(A_m, temp_past)
625             print(A_m)

```

```

626         print(temp_past)
627         self.temp_pred_list.append(temp_future)
628         print(self.temp_pred_list)
629
630     def plot(self):
631         for i in range(len(self.m_list)):
632             plt.plot(self.month_train, self.temp_train, marker="o",
633                    label="real data")
634             m = self.m_list[i]
635             temp_pred = self.temp_pred_list[i]
636             months = self.month_train[m:]
637             plt.plot(months, temp_pred, label="m = {}".format(m))
638             plt.xlabel("Months from Jan. 2001")
639             plt.ylabel("Average Temperature")
640             plt.legend()
641             plt.savefig("../figures/q_5_4_{}.png".format(m))
642             plt.show()
643
644     class Q_6_1:
645         def __init__(self):
646             self.x = np.arange(-2, 3, 1)
647             self.y = np.array([-1.8623, 0.6339, -2.2588, 2.0622, 2.7188])
648             self.params1 = [0, 0, 1]
649             self.params2 = [0, 0, 1]
650
651         def poly_1D(self, params, x):
652             return params[0] * x
653
654         def residual_func_1D(self, params, x, y_known):
655             y_pred = self.poly_1D(params, x)
656             return y_known - y_pred
657
658         def least_square(self, residual_func, params):
659             result = optimize.leastsq(residual_func, params, args=(self.x,
660             self.y))
661             params = result[0]
662             return params
663
664         def calcl(self):
665             params_calced = self.least_square(self.residual_func_1D,
666             self.params1)
667             self.params1 = params_calced
668             e = self.residual_func_1D(self.params1, self.x, self.y)
669             self.params1[-2] = np.average(e)
670             self.params1[-1] = np.var(e)

```

```

670     def print_result(self, i):
671         if i == 1:
672             params = self.params1
673         else:
674             params = self.params2
675         print("a          : {}".format(params[0]))
676         print("mean       : {}".format(params[1]))
677         print("variance  : {}".format(params[2]))
678         print()
679
680     def estimate_a(self, x, y):
681         mu = 1
682         var = 0.09
683         A = (np.linalg.norm(x))**2 + 1/var
684         B = np.dot(x.T, y) - (np.linalg.norm(x))**2
685         a = mu + A**(-1) * B
686         return a
687
688     def calc2(self):
689         a = self.estimate_a(self.x, self.y)
690         self.params2[0] = a
691         y_pred = self.poly_1D([a], self.x)
692         e = self.y - y_pred
693         self.params2[-2] = np.average(e)
694         self.params2[-1] = np.var(e)
695
696     def plot(self):
697         plt.plot(self.x, self.y, ".")
698         x = np.arange(self.x.min()-1, self.x.max()+2, 1)
699         plt.plot(x, self.poly_1D(self.params1, x), linestyle="dashed",
700                label="MLE")
701         plt.plot(x, self.poly_1D(self.params2, x), linestyle="dashdot",
702                label="MAP Estimation")
703         plt.xlabel("$x$")
704         plt.ylabel("$y$")
705         plt.legend()
706         plt.savefig("../figures/q_6_1.png")
707         plt.show()
708
709     ## execution -----
710     if __name__ == "__main__":
711         main()

```