ＡＩ基礎セミナー

第９回
機械学習（３回目：機械学習の概要と教師あり学習（回帰））

改訂履歴

日付	担当者	内容
2021/05/08	M.Takeda	Git 公開

目次

(1) はじめに

・ｾﾐﾅｰ第９回では、「教師あり学習（回帰)」を取り上げます。

・連続値を取る説明変数と、連続値を取る目的変数の間に関数関係のﾓﾃﾞﾙを当てはめることを
「回帰」と呼びます。これは古くからある統計手法です。

・回帰の手法にも様々ありますが、①「線形回帰ﾓﾃﾞﾙ」を解析的に解く方法、
②「線形回帰ﾓﾃﾞﾙ」を数値的に解く方法、③「線形基底関数ﾓﾃﾞﾙ」を解析的に解く方法、
④回帰ﾓﾃﾞﾙの適合度の評価、⑤「基底関数」、⑥「ﾛｼﾞｽﾃｨｯｸ回帰」に話題を絞り込んで解説します。

①「線形回帰ﾓﾃﾞﾙ」を解析的に解く方法では、ｾﾐﾅｰ第４回「数学の基礎（１回目：代数学)」で
解説した「Ｄ次元線形回帰モデルの解法」を用います。

②「線形回帰ﾓﾃﾞﾙ」を数値的に解く方法では、ｾﾐﾅｰ第５回「数学の基礎（２回目：解析学)」で
解説した「勾配降下法」を用います。

⑥「ﾛｼﾞｽﾃｨｯｸ回帰」は回帰ではなく分類の為のﾓﾃﾞﾙですが、
本ｾﾐﾅｰ第８回の「機械学習（２回目：機械学習の概要と教師あり学習（分類）)」では、
扱って来なかったので、ここで扱うことにします。
「ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ」の作成では、ｾﾐﾅｰ第６回「数学の基礎（３回目：統計学)」で解説した
「ﾍﾞﾙﾇｰｲ分布」や、ｾﾐﾅｰ第８回「機械学習（２回目：機械学習の概要と教師あり学習（分類）)」で
解説した「対数尤度」を用います。

・上記のように、今回は以前に取り上げた数学手法の回帰ﾓﾃﾞﾙ作成への適用といった色彩が濃いものに
なっています。

・今回の目標を、代表的な回帰ﾓﾃﾞﾙの概要理解と、回帰ﾓﾃﾞﾙの作成・評価方法の理解という点に置きます。

・確認問題は、本文を要約したような問題となっております。
解いてみて、理解度を確認してみてください。

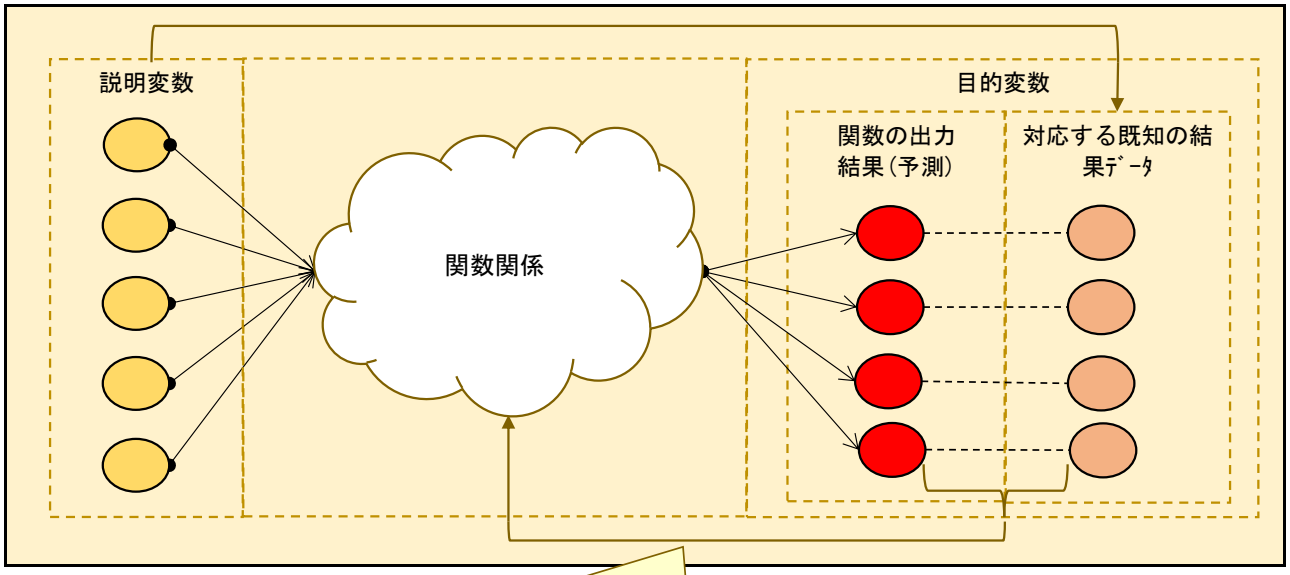
(2) 回帰ﾓﾃﾞﾙ

(※ この節は本ｾﾐﾅｰ(８回目)からの再掲を含みます)

・連続値を取る説明変数と、連続値を取る目的変数の間に関数関係のﾓﾃﾞﾙを当てはめることを
「回帰 (ｶｲｷ, Regression)」と呼び、当てはめるﾓﾃﾞﾙ式を
「回帰方程式または回帰式 (ｶｲｷｼｷ、Regression equation)」と呼びます。

・「回帰方程式」は、観測ﾃﾞｰﾀをもとに入力ﾃﾞｰﾀ（説明変数）と出力ﾃﾞｰﾀ（目的変数）の数値間の関係性から統計的手法によって推計します。
これを「回帰分析 (ｶｲｷﾌﾞﾝｾｷ, Regression analysis)」と呼びます。
説明変数が1次元ならば「単回帰分析 (ﾀﾝｶｲｷﾌﾞﾝｾｷ、Single regression analysis)」、
2次元以上ならば「重回帰分析 (ｼﾞｭｳｶｲｷﾌﾞﾝｾｷ、Multiple regression analysis)」と言います。

・「回帰式」は、方程式を解くことにより厳密な解を求めることができる場合があり、
このような解を「解析解 (ｶｲｾｷｶｲ、Analytical solution)」と呼びます。
「線形回帰ﾓﾃﾞﾙ」の解がこの例になります。
方程式を解くことができない場合、勾配法などの数値計算により近似的な解を求めます。
このような解を「数値解 (ｽｳﾁｶｲ、Numerical solution)」と呼びます。



関数の出力結果(目的変数の予測値)が、対応する既知の結果ﾃﾞｰﾀに近づけるように、関数関係を学習するのが「教師あり学習」。
目的変数が連続確率変数の場合、関数関係のﾓﾃﾞﾙを当てはめることを「回帰」と呼びます。

【出典・参考】
回帰⇒ https://ja.wikipedia.org/wiki/回帰分析
線形回帰⇒ https://ja.wikipedia.org/wiki/線形回帰

(3) 線形回帰

(※ この節は本ｾﾐﾅｰ(４回目)からの再掲を含みます)

・連続確率変数の目的変数 y に対し D個の説明変数の組

$$(x_1, x_2, \cdots, x_D)$$

があり、目的変数 y を、D個の説明変数の線形結合として表現する場合、
この回帰ﾓﾃﾞﾙを「D次元線形回帰ﾓﾃﾞﾙ (D-dimension linear regression model)」と言います。

D次元線形回帰ﾓﾃﾞﾙ

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D = \sum_{d=0}^{D} w_dx_d \quad \cdots(式3\text{-}1)$$

- w_i (i=0～D) : 偏回帰係数
- x_i (i=0～D) : 説明変数、但しx_0=1

・「D次元線形回帰ﾓﾃﾞﾙ」は、(D+1,1)型行列 $\boldsymbol{w}$, $\boldsymbol{x}$ を用いて以下の様に表現できます：

D次元線形回帰ﾓﾃﾞﾙの行列表現

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} \quad \cdots(式3\text{-}2)$$

$$\boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$$

- w_i (i=0～D) : 偏回帰係数
- x_i (i=0～D) : 説明変数、但しx_0=1

係数 w_d {d=1～D} は「偏回帰係数 (ﾍﾝｶｲｷｹｲｽｳ, Partial regression coefficient)」と言い、
説明変数 x_i {i≠d, i=1～D}を一定にして、説明変数 x_d {d=1～D} だけを１単位変化させたときの
目的変数 y の変化量を表します。

・D次元線形回帰ﾓﾃﾞﾙは、偏回帰係数 w_d {d=1～D} と定数項 w_0 を求めることにより決定します。

$$\boldsymbol{w} = (w_0, w_1, w_2, \cdots, w_D)$$

この係数 $\boldsymbol{w}$ は、N個の観測値の組に D次元線形回帰ﾓﾃﾞﾙを適用した時の、
目的変数の観測値 t_n (n=1～N) とﾓﾃﾞﾙ式の計算値 y_n (n=1～N) との誤差が最小になるように決定します。

No.	観測値		ﾓﾃﾞﾙ式の計算値
	説明変数 (D次元)	目的変数	
1	($x_{1,1}$、$x_{1,2}$、・・・、$x_{1,D}$)	t_1	y_1
2	($x_{2,1}$、$x_{2,2}$、・・・、$x_{2,D}$)	t_2	y_2
:	:	:	:
N	($x_{N,1}$、$x_{N,2}$、・・・、$x_{N,D}$)	t_N	y_N

- $x_{n,d}$: n番目(n=1～N)の観測値の組での、d番目(d=1～D)の説明変数の値
- t_n : n番目(n=1～N)の観測値の組での、目的変数の観測値
- y_n : n番目(n=1～N)の観測値の組での、ﾓﾃﾞﾙ式の計算値

・係数の決定の仕方には、解析的に解く方法(解析解)と、数値計算的に解く方法(数値解)があり、
以降で、各々について紹介します。

【出典・参考】

ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞ逆行列 ⇒ http://zellij.hatenablog.com/entry/20120811/p1

「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)

・これ以降では、次のデータを解析することを目標にして解説を行うこととします。

（資料09-(03)-1_秋田における日平均気温の月平均値）

（気象庁提供）
http://www.data.jma.go.jp/obd/stats/etrn/view/monthly_s3.php?prec_no=32&block_no=47582&year=2020&month=&day=&view=a1

国土交通省
気象庁
Japan Meteorological Agency

ホーム | 防災情報 | 各種データ・資料 | 知識・解説 | 気

ホーム > 各種データ・資料 > 過去の気象データ検索 > 観測開始からの毎月の値

・・・（途中、省略）・・・

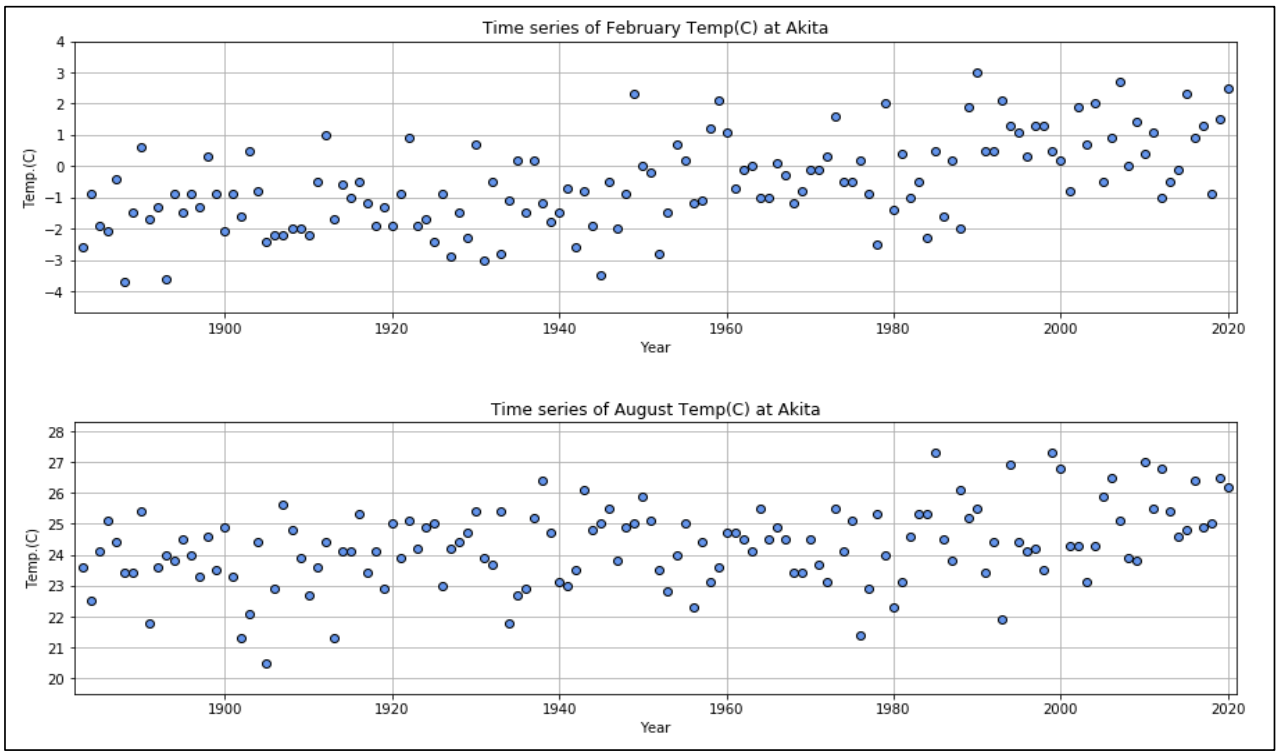
秋田　日平均気温の月平均値(℃)

年	1月	2月	3月	4月	5月	6月	7月	8月	9月	10月	11月	12月	年の値
1882										12.2	5.4	1.2	6.3]
1883	-1.1	-2.6	0.6	6.8	11.9	17.3	23.7	23.6	19.0	13.5	5.5	1.5	10.0
1884	-1.5	-0.9	1.0	7.5	12.7	17.6	22.2	22.5	19.3	11.0	4.3	-0.5	9.6
1885	-2.6	-1.9	1.2	7.4	12.3	17.4	20.9	24.1	20.4	13.3	7.5	3.1	10.3
1886	-1.2	-2.1	2.1	9.0	13.7	18.5	23.0	25.1	21.8	13.3	6.8	1.0	10.9
1887	-2.0	-0.4	2.9	8.0	12.1	18.2	22.1	24.4	17.7	13.5	8.1	1.7	10.5
1888	-1.5	-3.7	1.9	8.7	13.4	15.4	23.1	23.4	17.7	13.0	8.4	1.8	10.1
1889	-3.0	-1.5	1.7	8.6	12.4	17.0	21.7	23.4	17.4	10.3	4.8	0.8	9.5
1890	-1.6	0.6	5.1	9.9	13.8	17.9	21.5	25.4	22.1	12.7	8.0	5.5	11.7
1891	-2.1	-1.7	4.6	7.4	13.7	17.9	21.4	21.8	20.5	12.3	6.6	2.7	10.4

・・・（途中、省略）・・・

2001	-1.2	-0.8	3.7	10.7	16.1								
2002	1.0	1.9	5.5	11.8	14.4	18.8	23.8	24.3	20.4	14.4	4.8	1.0	11.8
2003	0.1	0.7	3.3	9.9	16.2	19.8	20.8	23.1	20.2	13.6	9.2	4.7	11.8
2004	0.6	2.0	4.2	9.2	15.9	19.4	24.1	24.3	21.1	14.1	10.9	4.2	12.5
2005	0.4	-0.5	3.0	9.3	13.8	21.1	22.3	25.9	21.0	15.2	8.4	0.5	11.7
2006	-0.7	0.9	3.7	8.1	14.9	19.5	22.2	26.5	20.5	14.4	8.9	3.6	11.9
2007	2.7	2.7	3.5	8.9	14.8	20.8	22.3	25.1	22.4	14.7	7.5	2.8	12.4
2008	-0.3	0.0	5.8	11.5	15.4	19.2	23.7	23.9	21.1	15.3	8.1	4.0	12.3
2009	1.3	1.4	3.9	9.6	15.8	19.9	22.4	23.8	19.5	14.5	8.8	3.1	12.0
2010	0.9	0.4	3.0	8.0	14.1	20.5	24.8	27.0	21.5	14.9	8.6	3.7	12.3
2011	-1.3	1.1	2.1	8.5	14.1)	19.2	25.0	25.5	21.6	14.8	9.6	1.8	11.8
2012	-1.1)	-1.0	3.3	9.7	15.2	19.7	23.6	26.8	24.2	15.4	8.1	1.2	12.1
2013	-0.9	-0.5	3.5	8.4	14.5	21.4	23.4	25.4	21.2	15.8	7.4	3.1	11.9
2014	0.0	-0.1	3.9	9.7	15.5	21.5	24.4	24.6	20.1	13.6	9.1	1.4	12.0
2015	1.2	2.3	5.7	10.9	16.7	19.9	23.8	24.8	20.1	13.4	9.4	4.3	12.7
2016	0.7	0.9	5.1	10.2	17.1	19.7	23.5	26.4	22.3	13.8	6.5	3.7	12.5
2017	0.6	1.3	3.9	10.0	16.1	17.6	24.6	24.9	20.1	14.0	7.9	1.6	11.9
2018	0.2	-0.9	5.2	10.3	15.6	19.6	25.3	25.0	20.5	15.3	9.2	2.7	12.3
2019	0.9	1.5	5.4	9.4	16.8	19.9	24.3	26.5	22.0	16.0	8.1	3.6	12.9
2020	2.3	2.5	6.4	8.6	15.5	20.9	23.0	26.2	22.9	14.5	8.9	2.3	12.8
2021	-0.4	2.1]											-0.4]

・「資料09-(03)-1_秋田における日平均気温の月平均値」のうち、
「1883年～2020年」の期間で 2月と8月に着目した時系列のグラフは以下のようになります。



(ﾘｽﾄ09-(03)-1_秋田における日平均気温の2月と8月の月平均値)

```
#*****************************************************************************/
# ﾘｽﾄ09-(03)-1_秋田における日平均気温の2月と8月の月平均値
#-----------------------------------------------------------------------------
# (気象庁提供)
#「秋田」における「観測開始からの毎月の値」のうち
#「日平均気温の月平均値（℃）」
#  ・・・「1883年～2020年」の期間で、2月と8月に着目
#
# http://www.data.jma.go.jp/obd/stats/etrn/view/monthly_s3.php?
#     prec_no=32&block_no=47582&year=2020&month=&day=&view=a1
#*****************************************************************************/

# ライブラリ
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# データ (「秋田」における2月と8月の平均気温の時系列) ---------
# データ (年)
yyyy= [1883,1884,1885,1886,1887,1888,1889,1890,1891,1892,1893,1894,1895,1896,1897,1898,1899,1900,
       1901,1902,1903,1904,1905,1906,1907,1908,1909,1910,1911,1912,1913,1914,1915,1916,1917,1918,1919,1920,
       1921,1922,1923,1924,1925,1926,1927,1928,1929,1930,1931,1932,1933,1934,1935,1936,1937,1938,1939,1940,
       1941,1942,1943,1944,1945,1946,1947,1948,1949,1950,1951,1952,1953,1954,1955,1956,1957,1958,1959,1960,
       1961,1962,1963,1964,1965,1966,1967,1968,1969,1970,1971,1972,1973,1974,1975,1976,1977,1978,1979,1980,
       1981,1982,1983,1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,
       2001,2002,2003,2004,2005,2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018,2019,2020]
# データ (2月の平均気温)
tFeb = [-2.6,-0.9,-1.9,-2.1,-0.4,-3.7,-1.5,0.6,-1.7,-1.3,-3.6,-0.9,-1.5,-0.9,-1.3,0.3,-0.9,-2.1,
        -0.9,-1.6,0.5,-0.8,-2.4,-2.2,-2.2,-2.0,-2.0,-2.2,-0.5,1.0,-1.7,-0.6,-1.0,-0.5,-1.2,-1.9,-1.3,-1.9,
        -0.9,0.9,-1.9,-1.7,-2.4,-0.9,-2.9,-1.5,-2.3,0.7,-3.0,-0.5,-2.8,-1.1,0.2,-1.5,0.2,-1.2,-1.8,-1.5,
        -0.7,-2.6,-0.8,-1.9,-3.5,-0.5,-2.0,-0.9,2.3,0.0,-0.2,-2.8,-1.5,0.7,0.2,-1.2,-1.1,1.2,2.1,1.1,
        -0.7,-0.1,0.0,-1.0,-1.0,0.1,-0.3,-1.2,-0.8,-0.1,-0.1,0.3,1.6,-0.5,-0.5,0.2,-0.9,-2.5,2.0,-1.4,
         0.4,-1.0,-0.5,-2.3,0.5,-1.6,0.2,-2.0,1.9,3.0,0.5,0.5,2.1,1.3,1.1,0.3,1.3,1.3,0.5,0.2,
        -0.8,1.9,0.7,2.0,-0.5,0.9,2.7,0.0,1.4,0.4,1.1,-1.0,-0.5,-0.1,2.3,0.9,1.3,-0.9,1.5,2.5 ]
```

```
# データ (8月の平均気温)
tAug = [23.6,22.5,24.1,25.1,24.4,23.4,23.4,25.4,21.8,23.6,24.0,23.8,24.5,24.0,23.3,24.6,23.5,24.9,
        23.3,21.3,22.1,24.4,20.5,22.9,25.6,24.8,23.9,22.7,23.6,24.4,21.3,24.1,24.1,25.3,23.4,24.1,22.9,25.0,
        23.9,25.1,24.2,24.9,25.0,23.0,24.2,24.4,24.7,25.4,23.9,23.7,25.4,21.8,22.7,22.9,25.2,26.4,24.7,23.1,
        23.0,23.5,26.1,24.8,25.0,25.5,23.8,24.9,25.0,25.9,25.1,23.5,22.8,24.0,25.0,22.3,24.4,23.1,23.6,24.7,
        24.7,24.5,24.1,25.5,24.5,24.9,24.5,23.4,23.4,24.5,23.7,23.1,25.5,24.1,25.1,21.4,22.9,25.3,24.0,22.3,
        23.1,24.6,25.3,25.3,27.3,24.5,23.8,26.1,25.2,25.5,23.4,24.4,21.9,26.9,24.4,24.1,24.2,23.5,27.3,26.8,
        24.3,24.3,23.1,24.3,25.9,26.5,25.1,23.9,23.8,27.0,25.5,26.8,25.4,24.6,24.8,26.4,24.9,25.0,26.5,26.2 ]

# データグラフ表示 ------------------------------------
plt.figure(figsize=(15, 9))
plt.subplots_adjust(wspace=0.1, hspace=0.4)

# 2 月
plt.subplot(2,1,1)
plt.title("Time series of February Temp(C) at Akita")
plt.plot(yyyy, tFeb, marker='o', linestyle='None', markeredgecolor='black', color='cornflowerblue')
plt.xlim(np.min(yyyy)-1, np.max(yyyy)+1)
plt.ylim(np.min(tFeb)-1, np.max(tFeb)+1)
plt.xlabel("Year")
plt.ylabel("Temp.(C)")
plt.grid(True)

# 8月
plt.subplot(2,1,2)
plt.title("Time series of August Temp(C) at Akita")
plt.plot(yyyy, tAug, marker='o', linestyle='None', markeredgecolor='black', color='cornflowerblue')
plt.xlim(np.min(yyyy)-1, np.max(yyyy)+1)
plt.ylim(np.min(tAug)-1, np.max(tAug)+1)
plt.xlabel("Year")
plt.ylabel("Temp.(C)")
plt.grid(True)

plt.show()
```

(3.1) 線形回帰（解析解）

(※ この節は本ｾﾐﾅｰ(第４回)からの再掲を含みます)

・線形回帰ﾓﾃﾞﾙの場合、「平均二乗誤差」Ｊ(***w***)を最小にする偏回帰係数 ***w*** を求めることが目標です。
偏回帰係数 ***w*** は方程式を解くことにより厳密な解を求めることができます。
このような解を「解析解」と呼びます。「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」を用いた導出方法を紹介します。

・Ｎ個の観測値の組で、n番目の組 (n=1～N) の説明変数群を $\boldsymbol{x}_n$ で表します。
$\boldsymbol{x}_n$ に対し「D次元線形回帰ﾓﾃﾞﾙ」を適用して得られる目的変数 $y(\boldsymbol{x}_n)$ の値は、
(式3-2)より偏回帰係数 ***w*** を用いて次式で与えられます：

$$y(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{x}_n \qquad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \qquad \boldsymbol{x}_n = \begin{pmatrix} 1 \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix} \qquad (n=1\sim N) \qquad \cdots(式3\text{-}3)$$

この値と、目的変数の観測値の値 t_n との差は次式のようになります。

$$y(\boldsymbol{x}_n) - t_n = \boldsymbol{w}^T \boldsymbol{x}_n - t_n \qquad \cdots(式3\text{-}4)$$

これが n番目の観測値の組についての、ﾓﾃﾞﾙ式の計算値の誤差になります。
これをＮ個の観測値全体で扱う為に、次の「平均二乗誤差 (ﾍｲｷﾝｼﾞｮｳｺﾞｻ, Mean squared error, MSE)」
Ｊ(***w***) を導入します。これは、全観測値の組についての、ﾓﾃﾞﾙ式の計算値の誤差の二乗の平均値です。

$$\begin{aligned} J(\boldsymbol{w}) &= (1/N) \times \Sigma_{n=1}^{N} \left(y(\boldsymbol{x}_n) - t_n\right)^2 \\ &= (1/N) \times \Sigma_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n - t_n\right)^2 \end{aligned} \qquad \cdots(式3\text{-}5)$$

Ｊが ***w*** の関数になっているのは、Ｊ の値が未定の ***w*** に依存するからです。
Ｊの最小値を与える ***w*** を求めるにあたり、両辺を w_i (i=0～D) で偏微分したものが、
0 となるという条件を用います。式を整理した結果は、次式のようになります：

$$\begin{aligned} \partial J(\boldsymbol{w})/\partial w_0 \Rightarrow \quad & 0 = \Sigma_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n - t_n\right) && (\because x_{n,0}=1) \qquad \cdots(式3\text{-}6') \\ \partial J(\boldsymbol{w})/\partial w_1 \Rightarrow \quad & 0 = \Sigma_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n - t_n\right) x_{n,1} \\ \partial J(\boldsymbol{w})/\partial w_2 \Rightarrow \quad & 0 = \Sigma_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n - t_n\right) x_{n,2} \\ \vdots \qquad\qquad & \vdots \qquad\qquad\quad \vdots \\ \partial J(\boldsymbol{w})/\partial w_D \Rightarrow \quad & 0 = \Sigma_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n - t_n\right) x_{n,D} \end{aligned}$$

これはベクトルを用いて、次のように表現できます：

$$\begin{aligned} (0,0,0,\cdots,0) &= \Sigma_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n - t_n\right) \boldsymbol{x}_n^T \\ &= \Sigma_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n \boldsymbol{x}_n^T - t_n \boldsymbol{x}_n^T\right) && (\because 行列の分配法則) \\ &= \boldsymbol{w}^T \Sigma_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^T - \Sigma_{n=1}^{N} t_n \boldsymbol{x}_n^T \end{aligned} \qquad \cdots(式3\text{-}6)$$

ここで、Ｎ個の観測値($\boldsymbol{x}_n$;t_n)の組：

$$\begin{cases} \text{No.1}: (x_{1,1},\ x_{1,2},\ \cdots,\ x_{1,D}\ ;\ t_1) \\ \text{No.2}: (x_{2,1},\ x_{2,2},\ \cdots,\ x_{2,D}\ ;\ t_2) \\ \quad\vdots \\ \text{No.N}: (x_{N,1},\ x_{N,2},\ \cdots,\ x_{N,D}\ ;\ t_N) \end{cases}$$

を、以下の様な(N,D+1)型行列 ***X***, (N,1)型行列 ***t*** を用いて表現します：

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{pmatrix} \qquad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

(***X*** の第一列が 1 なのは、$x_{n,0}=1$ として扱うことによります)

行列$\boldsymbol{X}$ により、(式3-6)の右辺第一項のΣ部分は、以下の様に表現できます：

$$\Sigma_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^{T} = \Sigma_{n=1}^{N} \begin{pmatrix} 1 \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix} \begin{pmatrix} 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{pmatrix}$$

$$= \Sigma_{n=1}^{N} \begin{pmatrix} 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,D} \\ x_{n,1} & x_{n,1}*x_{n,1} & x_{n,1}*x_{n,2} & \cdots & x_{n,1}*x_{n,D} \\ x_{n,2} & x_{n,2}*x_{n,1} & x_{n,2}*x_{n,2} & \cdots & x_{n,2}*x_{n,D} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ x_{n,D} & x_{n,D}*x_{n,1} & x_{n,D}*x_{n,2} & \cdots & x_{n,D}*x_{n,D} \end{pmatrix}$$

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

$$\therefore \quad \Sigma_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^{T} = \boldsymbol{X}^{T}\boldsymbol{X} \qquad \cdots(式3\text{-}7)$$

行列$\boldsymbol{X}$、$\boldsymbol{t}$ により、(式3-6)の右辺第二項は、以下の様に表現できます：

$$\Sigma_{n=1}^{N} t_n \boldsymbol{x}_n^{T} = \Sigma_{n=1}^{N} t_n \times \begin{pmatrix} 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{pmatrix}$$

$$= \Sigma_{n=1}^{N} \begin{pmatrix} t_n & t_n*x_{n,1} & t_n*x_{n,2} & \cdots & t_n*x_{n,D} \end{pmatrix}$$

$$= \begin{pmatrix} \Sigma_{n=1}^{N} t_n & \Sigma_{n=1}^{N} t_n*x_{n,1} & \Sigma_{n=1}^{N} t_n*x_{n,2} & \cdots & \Sigma_{n=1}^{N} t_n*x_{n,D} \end{pmatrix}$$

$$= \begin{pmatrix} t_1 & t_2 & \cdots & t_N \end{pmatrix} \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{pmatrix}$$

$$= \boldsymbol{t}^{T}\boldsymbol{X}$$

$$\therefore \quad \Sigma_{n=1}^{N} t_n \boldsymbol{x}_n^{T} = \boldsymbol{t}^{T}\boldsymbol{X} \qquad \cdots(式3\text{-}8)$$

上記関係式(式3-7)、(式3-8)により、(式3-6)は以下の様に整理されます：

$$(0, 0, 0, \cdots, 0) = \boldsymbol{w}^{T}\boldsymbol{X}^{T}\boldsymbol{X} - \boldsymbol{t}^{T}\boldsymbol{X} \qquad \cdots(式3\text{-}4')$$

更にこれに転置行列の関係式を用いることにより、次のように変形できます：

$$(0, 0, 0, \cdots, 0)^{T} = (\boldsymbol{w}^{T}\boldsymbol{X}^{T}\boldsymbol{X} - \boldsymbol{t}^{T}\boldsymbol{X})^{T}$$
$$= (\boldsymbol{w}^{T}\boldsymbol{X}^{T}\boldsymbol{X})^{T} - (\boldsymbol{t}^{T}\boldsymbol{X})^{T}$$
$$= (\boldsymbol{X}^{T}\boldsymbol{X})^{T}(\boldsymbol{w}^{T})^{T} - \boldsymbol{X}^{T}\boldsymbol{t}$$
$$= (\boldsymbol{X}^{T}(\boldsymbol{X}^{T})^{T})\boldsymbol{w} - \boldsymbol{X}^{T}\boldsymbol{t}$$
$$= (\boldsymbol{X}^{T}\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^{T}\boldsymbol{t}$$

$$\therefore \quad (\boldsymbol{X}^{T}\boldsymbol{X})\boldsymbol{w} = \boldsymbol{X}^{T}\boldsymbol{t} \qquad \cdots(式3\text{-}9)$$

(式3-9)の両辺に左側から $(\boldsymbol{X}^{T}\boldsymbol{X})$ の逆行列 $(\boldsymbol{X}^{T}\boldsymbol{X})^{-1}$ を掛けることにより、
(式3-1)の w の解析解を得ることが出来ます：

D次元線形回帰ﾓﾃﾞﾙの解

$$\boldsymbol{w} = (\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{t} \qquad \cdots(式3\text{-}10)$$

これが、「D次元線形回帰ﾓﾃﾞﾙ」の解となります。

(式3-10)の右辺の $(X^TX)^{-1}X^T$ には、
「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列（Moore-Penrose pseudo-inverse matrix)」という名前がついており、
線形代数学における逆行列の一般化になっています。

ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列

$$(X^TX)^{-1}X^T$$

・以上をまとめると「D次元線形回帰ﾓﾃﾞﾙ」は次式で与えられます：

D次元線形回帰ﾓﾃﾞﾙ（解析解）

$$y(\boldsymbol{x}) = \left((X^TX)^{-1}X^T\boldsymbol{t} \right)^T \boldsymbol{x} \qquad \cdots(式3\text{-}11)$$

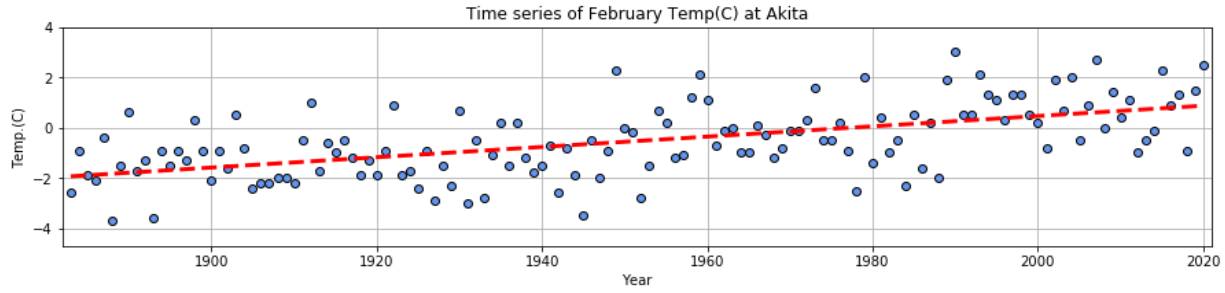
$$\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{pmatrix} \quad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

- $y(\boldsymbol{x})$ は説明変数 $\boldsymbol{x}$ に対するモデルの計算値
- x_d (d=1～D) は、d番目の説明変数
- $x_{n,d}$ (d=1～D) は、観測値の n番目の組(n=1～N) でのd番目の説明変数 x_d の値
- t_n は、観測値の n番目の組(n=1～N) での目的変数 y の観測値（教師ﾃﾞｰﾀ）

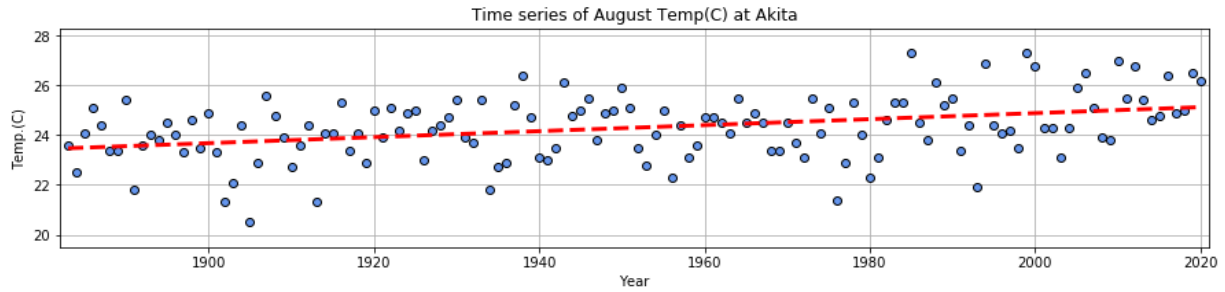
・以下では実ﾃﾞｰﾀに対して「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」を用いた解析解による線形回帰ﾓﾃﾞﾙを実装し、
その結果を見てみましょう。(ﾓﾃﾞﾙがどの程度実ﾃﾞｰﾀを説明できているかの評価は、後節で扱います)

・データは「資料09-(03)-1_秋田における日平均気温の月平均値」で、
「年」を説明変数、「指定月(2月、8月)の平均気温」を目的変数とした、単回帰ﾓﾃﾞﾙを作成します。

```
次元数   D=1
データ数 N=138
(wList)=([-2.77555756e-17  5.72290719e-01])
```




```
次元数   D=1
データ数 N=138
(wList)=([-5.67688258e-16  3.78217748e-01])
```



(ﾘｽﾄ09-(03)-2_秋田における月平均気温の時系列を解析(解析解))

```
#**************************************************************************/
# ﾘｽﾄ09-(03)-2_秋田における月平均気温の時系列を解析(解析解)
#--------------------------------------------------------------------------
# D次元線形回帰ﾓﾃﾞﾙをムーア・ペンローズの擬似逆行列を用いて解析 (D=1)
#--------------------------------------------------------------------------
# (気象庁提供)
#「秋田」における「観測開始からの毎月の値」のうち
#「日平均気温の月平均値 (℃) 」
#  ・・・「1883年～2020年」の期間で、2月と8月に着目
#
# http://www.data.jma.go.jp/obd/stats/etrn/view/monthly_s3.php?
#     prec_no=32&block_no=47582&year=2020&month=&day=&view=a1
#**************************************************************************/
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#**************************************************************************/
# データ (「秋田」における2月と8月の平均気温の時系列)
#**************************************************************************/
# データ (年)
yyyy= [1883,1884,1885,1886,1887,1888,1889,1890,1891,1892,1893,1894,1895,1896,1897,1898,1899,1900,
       1901,1902,1903,1904,1905,1906,1907,1908,1909,1910,1911,1912,1913,1914,1915,1916,1917,1918,1919,1920,
       1921,1922,1923,1924,1925,1926,1927,1928,1929,1930,1931,1932,1933,1934,1935,1936,1937,1938,1939,1940,
       1941,1942,1943,1944,1945,1946,1947,1948,1949,1950,1951,1952,1953,1954,1955,1956,1957,1958,1959,1960,
       1961,1962,1963,1964,1965,1966,1967,1968,1969,1970,1971,1972,1973,1974,1975,1976,1977,1978,1979,1980,
       1981,1982,1983,1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,
       2001,2002,2003,2004,2005,2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018,2019,2020]
# データ (2月の平均気温)
tFeb = [-2.6,-0.9,-1.9,-2.1,-0.4,-3.7,-1.5,0.6,-1.7,-1.3,-3.6,-0.9,-1.5,-0.9,-1.3,0.3,-0.9,-2.1,
        -0.9,-1.6,0.5,-0.8,-2.4,-2.2,-2.2,-2.0,-2.0,-2.2,-0.5,1.0,-1.7,-0.6,-1.0,-0.5,-1.2,-1.9,-1.3,-1.9,
        -0.9,0.9,-1.9,-1.7,-2.4,-0.9,-2.9,-1.5,-2.3,0.7,-3.0,-0.5,-2.8,-1.1,0.2,-1.5,0.2,-1.2,-1.8,-1.5,
        -0.7,-2.6,-0.8,-1.9,-3.5,-0.5,-2.0,-0.9,2.3,0.0,-0.2,-2.8,-1.5,0.7,0.2,-1.2,-1.1,1.2,2.1,1.1,
        -0.7,-0.1,0.0,-1.0,-1.0,0.1,-0.3,-1.2,-0.8,-0.1,-0.1,0.3,1.6,-0.5,-0.5,0.2,-0.9,-2.5,2.0,-1.4,
         0.4,-1.0,-0.5,-2.3,0.5,-1.6,0.2,-2.0,1.9,3.0,0.5,0.5,2.1,1.3,1.1,0.3,1.3,1.3,0.5,0.2,
        -0.8,1.9,0.7,2.0,-0.5,0.9,2.7,0.0,1.4,0.4,1.1,-1.0,-0.5,-0.1,2.3,0.9,1.3,-0.9,1.5,2.5 ]
# データ (8月の平均気温)
tAug = [23.6,22.5,24.1,25.1,24.4,23.4,23.4,25.4,21.8,23.6,24.0,23.8,24.5,24.0,23.3,24.6,23.5,24.9,
        23.3,21.3,22.1,24.4,20.5,22.9,25.6,24.8,23.9,22.7,23.6,24.4,21.3,24.1,24.1,25.3,23.4,24.1,22.9,25.0,
        23.9,25.1,24.2,24.9,25.0,23.0,24.2,24.4,24.7,25.4,23.9,23.7,25.4,21.8,22.7,22.9,25.2,26.4,24.7,23.1,
        23.0,23.5,26.1,24.8,25.0,25.5,23.8,24.9,25.0,25.9,25.1,23.5,22.8,24.0,25.0,22.3,24.4,23.1,23.6,24.7,
        24.7,24.5,24.1,25.5,24.5,24.9,24.5,23.4,23.4,24.5,23.7,23.1,25.5,24.1,25.1,21.4,22.9,25.3,24.0,22.3,
        23.1,24.6,25.3,25.3,27.3,24.5,23.8,26.1,25.2,25.5,23.4,24.4,21.9,26.9,24.4,24.1,24.2,23.5,27.3,26.8,
        24.3,24.3,23.1,24.3,25.9,26.5,25.1,23.9,23.8,27.0,25.5,26.8,25.4,24.6,24.8,26.4,24.9,25.0,26.5,26.2 ]

#**************************************************************************/
# fStandardize                                                            */
# 指定したデータを標準化し、その結果と平均・標準偏差を返す。              */
#-------------------------------------------------------------------------*/
# 引数:                                                                   */
#   x :対象のデータリスト (ﾃﾞｰﾀの組分)                                    */
#-------------------------------------------------------------------------*/
# 戻り値:                                                                 */
# (第１返値) z    :指定したデータの標準化結果                             */
# (第２返値) xAve :指定したデータの平均                                   */
# (第３返値) xStd :指定したデータの・標準偏差                             */
#**************************************************************************/
def fStandardize( x ):
    xAve = np.average(x)
    xStd = np.std(x)

    if xStd == 0.0 :
        return x, xAve, xStd
    else:
```

```
        z = np.zeros([len(x)])
        for n in range(0, len(x)):
            z[n] = (x[n] - xAve)/ xStd
        return z, xAve, xStd

#*****************************************************************************/
# fCalcAndShow                                                               */
# 解析解による線形回帰ﾓﾃﾞﾙの作成と、回帰曲線と学習曲線の表示                    */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   title  ：ｸﾞﾗﾌ表示のﾀｲﾄﾙ                                                  */
#   yyyy   ：説明変数 [n]  (n=1～N：観測値の組、1次元)                       */
#   t      ：目的変数 [n]  (n=1～N：観測値の組、1次元)                       */
#-----------------------------------------------------------------------------*/
# 戻り値：  なし                                                             */
#*****************************************************************************/
def fCalcAndShow(title, yyyy, t):

    # 入力ﾃﾞｰﾀの次元数
    D = 1
    print("次元数   D={0}".format(D))

    # データ数
    N = len(t)
    print("データ数 N={0}".format(N))

    # 説明変数の標準化（計算の収束と桁あふれ防止用）
    zList, xAve, xStd = fStandardize(yyyy)

    # 目的変数の標準化（計算の収束と桁あふれ防止用）
    tList, tAve, tStd = fStandardize(t)

    # 標準化した説明変数から(N,D+1)型行列 X を作成
    xList = np.zeros([N, D+1])
    for n in range(0, N):
        xList[n,0] = 1
        for d in range(1, D+1):
            xList[n,d] = zList[n]

    # ムーア・ペンローズの擬似逆行列を用いて偏回帰係数を算出
    inv_XtX = np.linalg.inv(np.dot(xList.T, xList))
    mtrxM = np.dot(inv_XtX, xList.T)
    wList = np.dot(mtrxM, tList)
    print("(wList)=({0})".format(wList))

    # 回帰直線の座標を計算する
    y = np.zeros(N)
    for n in range(0, N):
        y[n] = (np.dot(wList, xList[n]) * tStd) + tAve

    # 観測データ、および、モデルによる計算結果を表示
    plt.figure(figsize=(15, 3))
    plt.title(title)
    plt.plot(yyyy, t, marker='o', linestyle='None',
             markeredgecolor='black', color='cornflowerblue')
    plt.plot(yyyy, y, color='red', linewidth=3, linestyle='--', )
    plt.xlim(np.min(yyyy)-1, np.max(yyyy)+1)
    plt.ylim(np.min(t)-1, np.max(t)+1)
    plt.xlabel("Year")
    plt.ylabel("Temp.(C)")
    plt.grid(True)
    plt.show()

#*****************************************************************************/
# メイン処理                                                                 */
```

```
#*****************************************************************************/

# (2月)
fCalcAndShow("Time series of February Temp(C) at Akita", yyyy, tFeb)

# (8月)
fCalcAndShow("Time series of August Temp(C) at Akita", yyyy, tAug)
```

(3.2) 線形回帰（数値解）

・線形回帰ﾓﾃﾞﾙの場合、「平均二乗誤差」Ｊ(**w**)を最小にする偏回帰係数 **w** を求めることが目標です。
偏回帰係数 **w** は数値計算により近似的な解を求めることもできます。
このような解を「数値解」と呼びます。以下に「勾配法」を用いた導出方法を紹介します。

・初めに関連する数式のおさらいをしておきましょう。

・D個の説明変数 x_d (d=1～D)からなる線形回帰ﾓﾃﾞﾙ

$$y(\boldsymbol{x}) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D \quad \cdots(式3\text{-}1)再掲$$

$$= \boldsymbol{w}^T \boldsymbol{x} \quad (但し\ x_0=1) \quad \cdots(式3\text{-}2)再掲$$

・Ｎ個の観測値の組で、n番目の組（n=1～N：観測値の組数）の説明変数群 $\boldsymbol{x}_n$ に対して
「D次元線形回帰ﾓﾃﾞﾙ」を適用して得られる目的変数の値 $y(\boldsymbol{x}_n)$ ：

$$y(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{x}_n \quad \cdots(式3\text{-}3)再掲$$

・Ｎ個の観測値の組についての、「D次元線形回帰ﾓﾃﾞﾙ」の平均二乗誤差Ｊ(**w**)：

$$J(\boldsymbol{w}) = (1/N) \sum_{n=1}^{N} (y(\boldsymbol{x}_n) - t_n)^2$$

$$= (1/N) \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n)^2 \quad \cdots(式3\text{-}5)再掲$$

$$\boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \quad \boldsymbol{x}_n = \begin{pmatrix} 1 \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix} \quad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

- $y(\boldsymbol{x})$ は説明変数 $\boldsymbol{x}$ に対する線形回帰ﾓﾃﾞﾙの計算値
- x_d (d=1～D) は、d番目の説明変数
- $x_{n,d}$ (d=1～D) は、観測値の n番目の組(n=1～N) でのd番目の説明変数 x_d の値
- t_n は、観測値の n番目の組(n=1～N) での目的変数 y の観測値（教師ﾃﾞｰﾀ）

・「平均二乗誤差」Ｊ(**w**)を最小にする **w** を求めるｱﾙｺﾞﾘｽﾞﾑ
「勾配法（最急降下法, steepest descent method)」は、本ｾﾐﾅｰの
「第５回　数学の基礎（２回目：解析学)」の「(5.3) 勾配降下法」、あるいは、
「第８回　機械学習（２回目：機械学習の概要と教師あり学習（分類）)」の「(8) ﾓﾃﾞﾙの学習」
で既に詳述しているので、ここでは線形回帰ﾓﾃﾞﾙ作成への応用として解説します。

・勾配降下法を線形回帰ﾓﾃﾞﾙ作成へ応用する際の基本的アルゴリズムは、以下のようになります：

勾配降下法の基本的ｱﾙｺﾞﾘｽﾞﾑ（線形回帰ﾓﾃﾞﾙ作成への応用）

(1) tを繰返しのｽﾃｯﾌﾟ数として、t=0 から開始する。

(2) 学習係数 η (>0) を与える(※1)。

(3) 偏回帰係数 $\boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$ の初期値 $\boldsymbol{w}(0) = \begin{pmatrix} w_{0,0} \\ w_{0,1} \\ w_{0,2} \\ \vdots \\ w_{0,D} \end{pmatrix}$ を任意に与える。

(4) 以下の様なモデルの最適判定を行い、計算の収束判定を行う：

(4.1) 偏回帰係数 $\boldsymbol{w}(t)$ での平均二乗誤差 $J(\boldsymbol{w}(t))$ を計算し、その値が十分小さければ、
本処理を終了する。終了時の偏回帰係数 $\boldsymbol{w}(t)$ がﾓﾃﾞﾙの最適値となる。

$$J(\boldsymbol{w}(t)) = (1/N)\sum_{n=1}^{N} (\boldsymbol{w}(t)^T \boldsymbol{x}_n - t_n)^2 \qquad \cdots(式3-⑤ 再掲)$$

(4.2) 上記の平均二乗誤差 $J(\boldsymbol{w}(t))$ の減少が殆どなく、計算が収束したとみなせる場合、
本処理を終了する。終了時の偏回帰係数 $\boldsymbol{w}(t)$ がﾓﾃﾞﾙの最適値となる。

(4.3) 繰り返し回数が制限を超過した場合、本処理を終了する。
この時は 最小値を与える点は見つかっておらず、モデル等の見直しを行う必要あり。

(4.4) 上記(4.1)～(4.3)以外の場合、処理(5)の偏回帰係数の更新処理へ。

(5) 偏回帰係数 $\boldsymbol{w}$ について $\boldsymbol{w}(t)$ からの移動量ベクトル $\Delta\boldsymbol{w}$ を次式により計算し $\boldsymbol{w}(t+1)$ へ更新する。

$$\Delta\boldsymbol{w} = -\eta \nabla J$$

ここで、∇J は平均二乗誤差 J の $\boldsymbol{w}(t)$ における勾配ベクトル：

$$\nabla J = (\partial J/\partial w_0,\ \partial J/\partial w_1,\ \cdots,\ \partial J/\partial w_D) \mid \boldsymbol{w}(t)$$

$\therefore$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta \nabla J$$

$$\begin{pmatrix} w_{t+1,0} \\ w_{t+1,1} \\ w_{t+1,2} \\ \vdots \\ w_{t+1,D} \end{pmatrix} = \begin{pmatrix} w_{t,0} \\ w_{t,1} \\ w_{t,2} \\ \vdots \\ w_{t,D} \end{pmatrix} - \eta \begin{pmatrix} \partial J/\partial w_0 \\ \partial J/\partial w_1 \\ \partial J/\partial w_2 \\ \vdots \\ \partial J/\partial w_D \end{pmatrix} \Bigg| \boldsymbol{w}(t)$$

ここで、平均二乗誤差 J の偏回帰係数 w_d (d=0～D:偏回帰係数の組) による偏微分は

$$\partial J/\partial w_d = \partial (1/N)\sum_{n=1}^{N} (\boldsymbol{w}(t)^T \boldsymbol{x}_n - t_n)^2 / \partial w_d \qquad (x_{n,0}=1)$$

$$= (2/N)\sum_{n=1}^{N} (\boldsymbol{w}(t)^T \boldsymbol{x}_n - t_n)\, x_{n,d} \qquad (n=1\sim N：観測値の組)$$

$$= (2/N)\sum_{n=1}^{N} (y(\boldsymbol{x}_n) - t_n)\, x_{n,d}$$

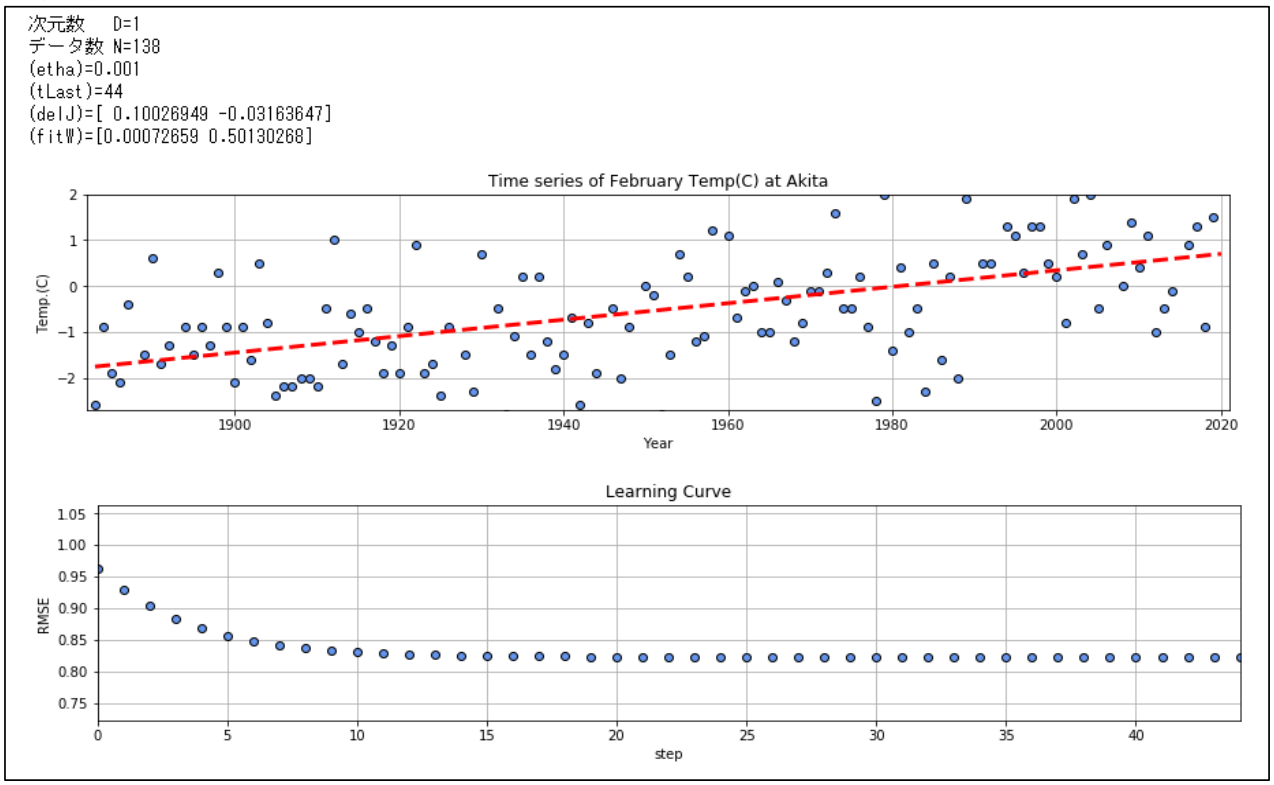
(6) 繰返しのｽﾃｯﾌﾟ数 t ← t+1 として手順(4)へ

(※1) 学習係数 η を、手順(4)～(6)の過程で動的に決定する方法もあります。

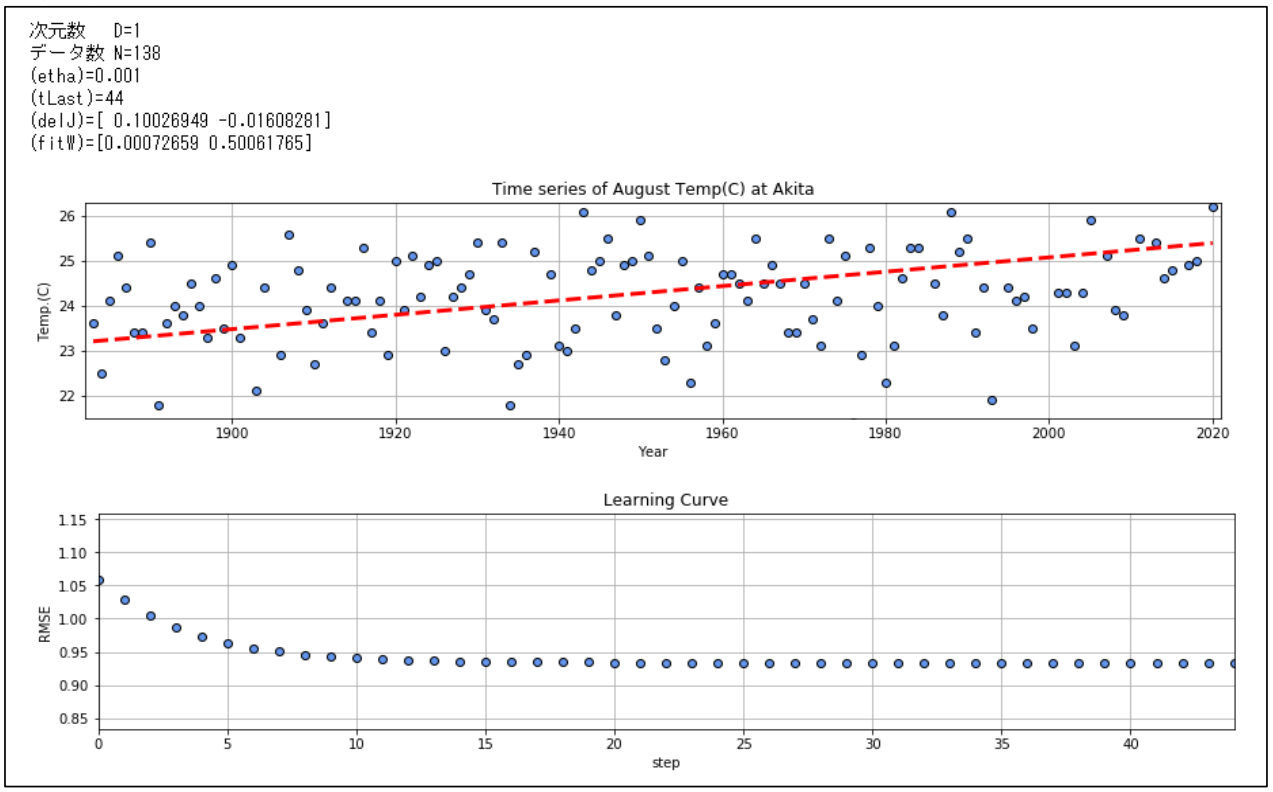
・上記の勾配降下法による線形回帰ﾓﾃﾞﾙ作成の実装例と、その回帰直線と学習曲線を、以下に示します。
データは「資料09-(03)-1_秋田における日平均気温の月平均値」です。

・解析解の場合は、数式で導出されるため計算ｺｽﾄは少なくて済みますが、
数値解の場合は、学習係数の設定や収束判定などの処理が入り、
計算ｺｽﾄと精度で検証が必要になります。
(ﾓﾃﾞﾙがどの程度実ﾃﾞｰﾀを説明できているかの評価は、後節で扱います)

(1) ２月のﾃﾞｰﾀ



(2) ８月のﾃﾞｰﾀ



(ﾘｽﾄ09-(03)-3_秋田における月平均気温の時系列を解析(数値解))

```
#*************************************************************************/
# ﾘｽﾄ09-(03)-3_秋田における月平均気温の時系列を解析(数値解)
#-------------------------------------------------------------------------
# D次元線形回帰ﾓﾃﾞﾙを勾配法を用いて解析 (D=1)
#-------------------------------------------------------------------------
# (気象庁提供)
#「秋田」における「観測開始からの毎月の値」のうち
#「日平均気温の月平均値 (℃) 」
#  ・・・「1883年～2020年」の期間で、2月と8月に着目
#
# http://www.data.jma.go.jp/obd/stats/etrn/view/monthly_s3.php?
#     prec_no=32&block_no=47582&year=2020&month=&day=&view=a1
#*************************************************************************/
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#*************************************************************************/
# データ (「秋田」における2月と8月の平均気温の年系列)
#*****************************|*****************************************/
# データ (年)
yyyy= [1883,1884,1885,1886,1887,1888,1889,1890,1891,1892,1893,1894,1895,1896,1897,1898,1899,1900,
       1901,1902,1903,1904,1905,1906,1907,1908,1909,1910,1911,1912,1913,1914,1915,1916,1917,1918,1919,1920,
       1921,1922,1923,1924,1925,1926,1927,1928,1929,1930,1931,1932,1933,1934,1935,1936,1937,1938,1939,1940,
       1941,1942,1943,1944,1945,1946,1947,1948,1949,1950,1951,1952,1953,1954,1955,1956,1957,1958,1959,1960,
       1961,1962,1963,1964,1965,1966,1967,1968,1969,1970,1971,1972,1973,1974,1975,1976,1977,1978,1979,1980,
       1981,1982,1983,1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,
       2001,2002,2003,2004,2005,2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018,2019,2020]
# データ (2月の平均気温)
tFeb = [-2.6,-0.9,-1.9,-2.1,-0.4,-3.7,-1.5,0.6,-1.7,-1.3,-3.6,-0.9,-1.5,-0.9,-1.3,0.3,-0.9,-2.1,
        -0.9,-1.6,0.5,-0.8,-2.4,-2.2,-2.2,-2.0,-2.0,-2.2,-0.5,1.0,-1.7,-0.6,-1.0,-0.5,-1.2,-1.9,-1.3,-1.9,
        -0.9,0.9,-1.9,-1.7,-2.4,-0.9,-2.9,-1.5,-2.3,0.7,-3.0,-0.5,-2.8,-1.1,0.2,-1.5,0.2,-1.2,-1.8,-1.5,
        -0.7,-2.6,-0.8,-1.9,-3.5,-0.5,-2.0,-0.9,2.3,0.0,-0.2,-2.8,-1.5,0.7,0.2,-1.2,-1.1,1.2,2.1,1.1,
        -0.7,-0.1,0.0,-1.0,-1.0,0.1,-0.3,-1.2,-0.8,-0.1,-0.1,0.3,1.6,-0.5,-0.5,0.2,-0.9,-2.5,2.0,-1.4,
         0.4,-1.0,-0.5,-2.3,0.5,-1.6,0.2,-2.0,1.9,3.0,0.5,0.5,2.1,1.3,1.1,0.3,1.3,1.3,0.5,0.2,
        -0.8,1.9,0.7,2.0,-0.5,0.9,2.7,0.0,1.4,0.4,1.1,-1.0,-0.5,-0.1,2.3,0.9,1.3,-0.9,1.5,2.5 ]
# データ (8月の平均気温)
tAug = [23.6,22.5,24.1,25.1,24.4,23.4,23.4,25.4,21.8,23.6,24.0,23.8,24.5,24.0,23.3,24.6,23.5,24.9,
        23.3,21.3,22.1,24.4,20.5,22.9,25.6,24.8,23.9,22.7,23.6,24.4,21.3,24.1,24.1,25.3,23.4,24.1,22.9,25.0,
        23.9,25.1,24.2,24.9,25.0,23.0,24.2,24.4,24.7,25.4,23.9,23.7,25.4,21.8,22.7,22.9,25.2,26.4,24.7,23.1,
        23.0,23.5,26.1,24.8,25.0,25.5,23.8,24.9,25.0,25.9,25.1,23.5,22.8,24.0,25.0,22.3,24.4,23.1,23.6,24.7,
        24.7,24.5,24.1,25.5,24.5,24.9,24.5,23.4,23.4,24.5,23.7,23.1,25.5,24.1,25.1,21.4,22.9,25.3,24.0,22.3,
        23.1,24.6,25.3,25.3,27.3,24.5,23.8,26.1,25.2,25.5,23.4,24.4,21.9,26.9,24.4,24.1,24.2,23.5,27.3,26.8,
        24.3,24.3,23.1,24.3,25.9,26.5,25.1,23.9,23.8,27.0,25.5,26.8,25.4,24.6,24.8,26.4,24.9,25.0,26.5,26.2 ]

#*************************************************************************/
# fStandardize                                                           */
# 指定したデータを標準化し、その結果と平均・標準偏差を返す。             */
#-------------------------------------------------------------------------*/
# 引数:                                                                  */
#   x :対象のデータリスト(ﾃﾞｰﾀの組分)                                    */
#-------------------------------------------------------------------------*/
# 戻り値:                                                                */
# (第１返値) z    :指定したデータの標準化結果                            */
# (第２返値) xAve :指定したデータの平均                                  */
# (第３返値) xStd :指定したデータの・標準偏差                            */
#*************************************************************************/
def fStandardize( x ):
    xAve = np.average(x)
    xStd = np.std(x)

    if xStd == 0.0 :
        return x, xAve, xStd
    else:
```

```
        z = np.zeros([len(x)])
        for n in range(0, len(x)):
            z[n] = (x[n] - xAve)/ xStd
        return z, xAve, xStd

#*****************************************************************************/
# fGetEtha                                                                   */
# 学習係数η を取得する関数                                                   */
#-----------------------------------------------------------------------------*/
# 引数： 無し                                                                */
#-----------------------------------------------------------------------------*/
# 戻り値：学習係数η                                                          */
#*****************************************************************************/
def fGetEtha():
    return 0.001

#*****************************************************************************/
# fGetDelJ                                                                   */
# 平均二乗誤差Ｊの勾配∂J/∂wd(d=0～D:偏回帰係数の組)を計算する関数           */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   dimNo  : 線形回帰ﾓﾃﾞﾙの次元数D (≧1)                                      */
#   xList  : 説明変数 [n,d]  (n=1～N：観測値の組、d=0～D:説明変数の組)        */
#   tList  : 目的変数 [n]    (n=1～N：観測値の組)                            */
#   wList  : 偏回帰係数 [d]  (d=0～D:偏回帰係数の組)                         */
#-----------------------------------------------------------------------------*/
# 戻り値：                                                                   */
#   第1戻り値：delJ[d]：平均二乗誤差Ｊの勾配∂J/∂w[d] (d=0～D)                */
#   第2戻り値：J       ：平均二乗誤差Ｊ(RMSE)                                */
#*****************************************************************************/
def fGetDelJ(dimNo, xList, tList, wList):
    delJ = np.zeros(dimNo+1)
    dtN = len(tList)
    se = 0.0

    for n in range(0, dtN):
        # ﾓﾃﾞﾙの計算値
        yn = np.dot(wList, xList[n])
        se = se + (yn - tList[n])**2

        #∂J/∂wd の計算
        for d in range(0, dimNo+1):
            delJ[d] = delJ[d] + (yn - tList[n]) * xList[n,d]
        delJ[d] = 2* delJ[d] / dtN

    rmse = np.sqrt(se/dtN)
    return delJ, rmse

#*****************************************************************************/
# fFitting                                                                   */
# 勾配法による線形回帰ﾓﾃﾞﾙの作成                                             */
#-----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   dimNo  : 線形回帰ﾓﾃﾞﾙの次元数D (≧1)                                      */
#   xList  : 説明変数 [n,d]  (n=1～N：観測値の組、d=0～D:説明変数の組)        */
#   tList  : 目的変数 [n]    (n=1～N：観測値の組)                            */
#-----------------------------------------------------------------------------*/
# 戻り値：                                                                   */
#   第1戻り値：delJ[d]：平均二乗誤差Ｊの勾配 (d=0～D:偏回帰係数の組)          */
#   第2戻り値：fitW[d]：偏回帰係数 (d=0～D:偏回帰係数の組)                    */
#   第3戻り値：jHist[t]：平均二乗誤差Ｊの (t:計算ｽﾃｯﾌﾟ)                       */
#   第4戻り値：tLast：   最終的な計算ｽﾃｯﾌﾟ数                                 */
#*****************************************************************************/
def fFitting(dimNo, xList, tList):
    if dimNo <= 0 :
```

```
        return None, None

    # 計算用の定数
    GRAD_THRESHOLD= 0.001 # 勾配の変化率のしきい値
    GRAD_REPEAT = 30      # 勾配の変化率がしきい値未満となる回数の最大値
    REPEAT_MAX = 300      # 最大繰り返し回数

    # 学習係数η の初期値
    etha = fGetEtha()

    # 偏回帰係数の初期化
    wHist = np.zeros([REPEAT_MAX, dimNo+1])
    wHist[0,:] = np.full(dimNo+1, 1/(dimNo+1))

    # 勾配降下法による逐次近似
    jHist = np.zeros([REPEAT_MAX])
    tLast = 0
    jLast = 0
    gradRptNo = 0
    diffJ = 0

    for t in range(0, REPEAT_MAX-1):
        tLast = t
        delJ, jHist[t] = fGetDelJ(dimNo, xList, tList, wHist[t,:])

        # 収束判定用の勾配の変化率が、
        # 勾配の大きさの1000分の１以下の状態が、30回続いたら
        # 逐次近似処理を打ち切る
        diffJ = np.abs(jHist[t] - jLast)
        if (diffJ / jHist[t]) < GRAD_THRESHOLD :
            gradRptNo = gradRptNo + 1;
        else:
            gradRptNo = 0
        if gradRptNo > GRAD_REPEAT :
            break

        # 偏回帰係数の更新
        for d in range(0, dimNo+1):
            wHist[t+1,d] = wHist[t,d] - etha * delJ[d]

        # 直前のJ
        jLast = jHist[t]

    # 偏回帰係数の最終値
    fitW = wHist[tLast,:]

    print("(etha)={0}".format(etha))
    print("(tLast)={0}".format(tLast))
    print("(delJ)={0}".format(delJ))
    print("(fitW)={0}".format(fitW))

    return delJ, fitW, jHist, tLast

#*****************************************************************************/
# fCalcAndShow                                                               */
# 線形回帰ﾓﾃﾞﾙの作成と、回帰曲線と学習曲線の表示                               */
#----------------------------------------------------------------------------*/
# 引数：                                                                     */
#   title  ：ｸﾞﾗﾌ表示のﾀｲﾄﾙ                                                  */
#   yyyy   ：説明変数 [n]  (n=1～N：観測値の組、1次元)                       */
#   t      ：目的変数 [n]  (n=1～N：観測値の組、1次元)                       */
#----------------------------------------------------------------------------*/
# 戻り値：  なし                                                             */
#*****************************************************************************/
def fCalcAndShow(title, yyyy, t):
```

```
    # 入力ﾃﾞｰﾀの次元数
    dimNo = 1
    print("次元数　D={0}".format(dimNo))

    # データ数
    N = len(t)
    print("データ数 N={0}".format(N))

    # 説明変数（計算の収束と桁あふれ防止用にﾃﾞｰﾀの標準化を施す）
    zList, xAve, xStd = fStandardize(yyyy)
    xList = np.zeros([N, dimNo+1])
    for n in range(0, N):
        xList[n,0] = 1
        for d in range(1, dimNo+1):
            xList[n,d] = zList[n]

    # 目的変数の標準化（計算の収束と桁あふれ防止用）
    tList, tAve, tStd = fStandardize(t)

    # 偏回帰係数を算出し、回帰直線の座標を計算
    delJ, fitW, jHist, tLast = fFitting(dimNo, xList, tList)
    y = np.zeros(N)
    for n in range(0, N):
        y[n] = (np.dot(fitW, xList[n]) * tStd) + tAve

    # データとモデルによる計算結果を表示
    plt.figure(figsize=(15, 3))
    plt.title(title)
    plt.plot(yyyy, t, marker='o', linestyle='None',
             markeredgecolor='black', color='cornflowerblue')
    plt.plot(yyyy, y, color='red', linewidth=3, linestyle='--', )
    plt.xlim(np.min(yyyy)-1, np.max(yyyy)+1)
    plt.ylim(np.min(t)+1, np.max(t)-1)
    plt.xlabel("Year")
    plt.ylabel("Temp.(C)")
    plt.grid(True)

    # 学習の履歴を表示
    jstep = np.zeros(len(jHist))
    for t in range(0, len(jHist)):
        jstep[t] = t

    plt.figure(figsize=(15, 3))
    plt.title("Learning Curve")
    plt.plot(jstep, jHist, marker='o', linestyle='None',
             markeredgecolor='black', color='cornflowerblue')
    plt.xlim(0, tLast)
    plt.ylim(np.min(jHist[:tLast])-0.1, np.max(jHist[:tLast])+0.1)
    plt.xlabel("step")
    plt.ylabel("RMSE")
    plt.grid(True)
    plt.show()

#*****************************************************************************/
# メイン処理                                                                 */
#*****************************************************************************/

# (2月)
fCalcAndShow("Time series of February Temp(C) at Akita", yyyy, tFeb)

# (8月)
fCalcAndShow("Time series of August Temp(C) at Akita", yyyy, tAug)
```

(4) 基底関数

(4.1) 線形基底関数ﾓﾃﾞﾙ

・「D次元線形回帰ﾓﾃﾞﾙ」は、説明変数 $\boldsymbol{x} = (x_d)$ {d=1～D} の線形結合です。

D次元線形回帰ﾓﾃﾞﾙ（定式化）

$$y(\boldsymbol{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D \quad \cdots(式3\text{-}1)\ 再掲$$

$$= \sum_{d=0}^{D} w_d x_d$$

$$= \boldsymbol{w}^T \boldsymbol{x} \quad \cdots(式3\text{-}2)\ 再掲$$

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \quad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$$

但し $x_0 = 1$

D：説明変数の次元数

・しかしながら「D次元線形回帰ﾓﾃﾞﾙ」では表現力が乏しいため、
説明変数 $\boldsymbol{x} = (x_d)$ {d=1～D：説明変数の次元数} の線形結合の代わりに、
「基底関数（ｷﾃｲｶﾝｽｳ、basis function)」と呼ぶ説明変数 $\boldsymbol{x}$ の関数 $\boldsymbol{\Phi}_m(\boldsymbol{x})$ {m=1～M：関数の数} の組
を用いて、その線形結合として表現するモデルがあります。
それを「線形基底関数ﾓﾃﾞﾙ（ｾﾝｹｲｷﾃｲｶﾝｽｳﾓﾃﾞﾙ、linear basis function model)」と呼びます。

線形基底関数ﾓﾃﾞﾙ（定式化）

$$y(\boldsymbol{x}) = w_0 + w_1 \boldsymbol{\Phi}_1(\boldsymbol{x}) + w_2 \boldsymbol{\Phi}_2(\boldsymbol{x}) + \cdots + w_M \boldsymbol{\Phi}_M(\boldsymbol{x}) \quad \cdots(式4\text{-}1)$$

$$= \sum_{m=0}^{M} w_m \boldsymbol{\Phi}_m(\boldsymbol{x})$$

$$= \boldsymbol{w}^T \boldsymbol{\Phi}(\boldsymbol{x}) \quad \cdots(式4\text{-}2)$$

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \quad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_M \end{pmatrix} \quad \boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\Phi}_0 \\ \boldsymbol{\Phi}_1 \\ \boldsymbol{\Phi}_2 \\ \vdots \\ \boldsymbol{\Phi}_M \end{pmatrix}$$

但し $\boldsymbol{\Phi}_0(\boldsymbol{x}) = 1$

D：説明変数の次元数

M：基底関数の数

・「線形基底関数ﾓﾃﾞﾙ」は、「D次元線形回帰モデル」の解析解(式3-11)と同様に
「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」を用いた解析解を得ることができます。

線形基底関数ﾓﾃﾞﾙ（解析解）

$$y(\boldsymbol{x}) = \left((\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \right)^T \boldsymbol{x} \quad \cdots(式4\text{-}3)$$

$$\boldsymbol{\Phi} = \begin{pmatrix} 1 & \boldsymbol{\Phi}_1(\boldsymbol{x}_1) & \boldsymbol{\Phi}_2(\boldsymbol{x}_1) & \cdots & \boldsymbol{\Phi}_M(\boldsymbol{x}_1) \\ 1 & \boldsymbol{\Phi}_1(\boldsymbol{x}_2) & \boldsymbol{\Phi}_2(\boldsymbol{x}_2) & \cdots & \boldsymbol{\Phi}_M(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \boldsymbol{\Phi}_1(\boldsymbol{x}_N) & \boldsymbol{\Phi}_2(\boldsymbol{x}_N) & \cdots & \boldsymbol{\Phi}_M(\boldsymbol{x}_N) \end{pmatrix} \quad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix} \quad \boldsymbol{x}_n = \begin{pmatrix} x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix}$$

- $y(\boldsymbol{x})$ は説明変数 $\boldsymbol{x}$ に対するモデルの計算値
- $\boldsymbol{x}_n$ (n=1～N) は、N個の観測値の組での n番目の説明変数の組
- $x_{n,d}$ (d=1～D) は、$\boldsymbol{x}_n$ での d番目の説明変数の値
- t_n は、N個の観測値の組での n番目の目的変数 y の値（教師ﾃﾞｰﾀ）
- $\boldsymbol{\Phi}_m$ (m=0～M) は M個の基底関数のうちの m番目の基底関数、但し $\boldsymbol{\Phi}_0=1$

(4.2) 線形基底関数ﾓﾃﾞﾙの例

・基底関数として、任意の関数を指定できますが、代表的なのは
「多項式基底関数 (ﾀｺｳｼｷｷﾃｲｶﾝｽｳ、Polynomial basis functions)」と、
「ガウス基底関数 (ｶﾞｳｽｷﾃｲｶﾝｽｳ、Gaussian basis functions)」です。

・以下に、基底関数線形ﾓﾃﾞﾙを幾つか示します。

・「D次元線形回帰ﾓﾃﾞﾙ」は、基底関数 $\boldsymbol{\Phi}_m(\boldsymbol{x})$ が $\boldsymbol{x}$ の第m成分を取得する関数から構成される
「線形基底関数ﾓﾃﾞﾙ」とみることもできます：

D次元線形回帰ﾓﾃﾞﾙ

$$y(\boldsymbol{x}) = w_0 + w_1*x_1 + w_2*x_2 + \cdots + w_D*x_D$$

$$\text{基底関数 } \boldsymbol{\Phi}_m(\boldsymbol{x}) = x_m \qquad \cdots(式4\text{-}4)$$

{ m=1～M：基底関数の数(この場合は説明変数の次元の数D)。$\boldsymbol{\Phi}_0(\boldsymbol{x})=1$ }

・「多項式基底関数」は、基底関数 $\boldsymbol{\Phi}_m(\boldsymbol{x})$ が $\boldsymbol{x}$ の m乗から構成されるものです。
説明変数が一次元(x)の場合は、多項式近似のことになります。

多項式基底関数

$$y(x) = w_0 + w_1*x^1 + w_2*x^2 + \cdots + w_D*x^D$$

$$\text{基底関数 } \boldsymbol{\Phi}_m(\boldsymbol{x}) = x^m \qquad \cdots(式4\text{-}5)$$

{ m=1～M：多項式基底関数ﾓﾃﾞﾙの基底関数の数。$\boldsymbol{\Phi}_0(\boldsymbol{x})=1$ }

・「ガウス基底関数」は、基底関数 $\boldsymbol{\Phi}_m(\boldsymbol{x})$ が ガウス分布曲線から構成されるものです。
説明変数が一次元(x)の場合は、以下のような式になります。

ガウス基底関数

$$y(x) = w_0 + w_1*\exp\{-(x-\mu_1)/(2\sigma_1^2)\} + w2*\exp\{-(x-\mu_2)/(2\sigma_2^2)\} \cdots + w_M*\exp\{-(x-\mu_M)/(2\sigma_M^2)\}$$

$$\text{基底関数 } \boldsymbol{\Phi}_m(\boldsymbol{x}) = \exp\{-(x-\mu_m)/(2\sigma_m^2)\} \qquad \cdots(式4\text{-}6)$$

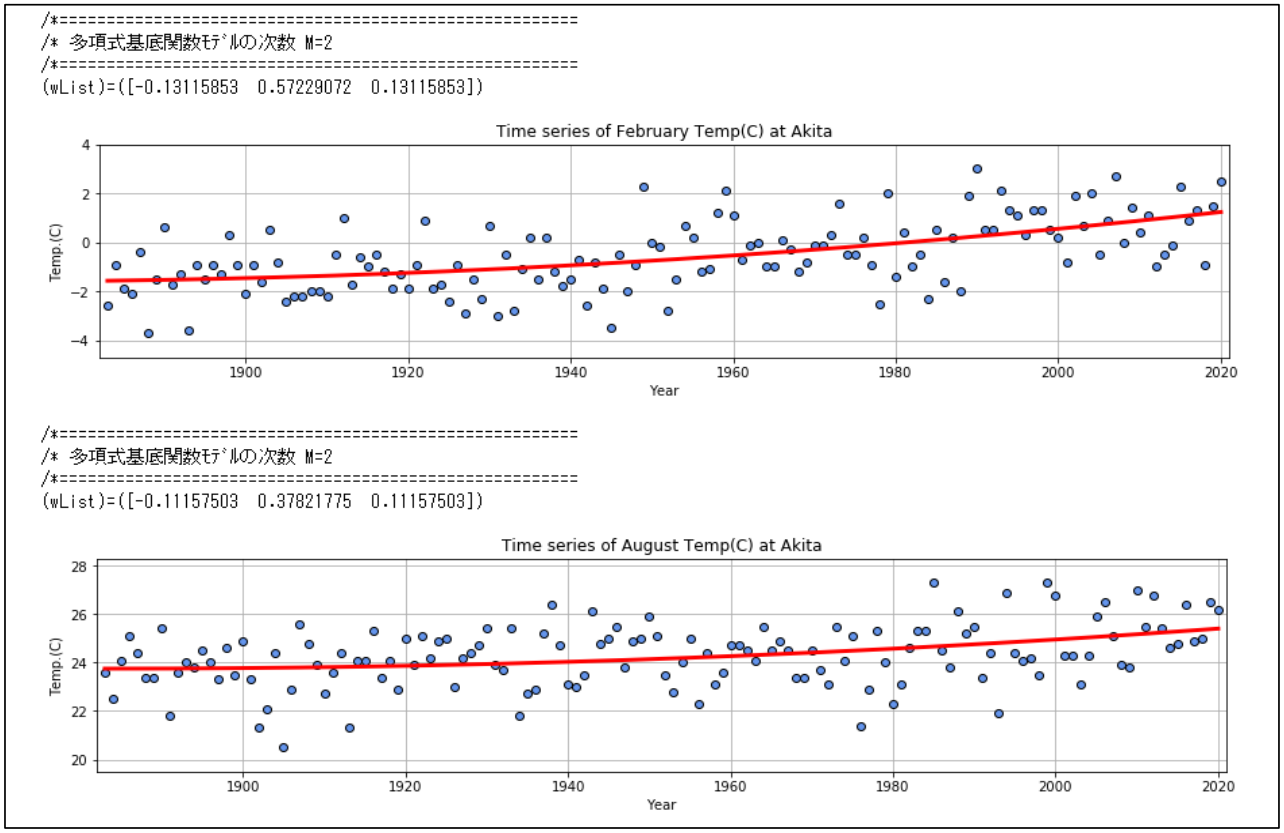
{ m=1～M：ガウス基底関数ﾓﾃﾞﾙの基底関数の数。$\boldsymbol{\Phi}_0(\boldsymbol{x})=1$
μ_m：m番目のｶﾞｳｽ分布関数での平均、
σ_m：m番目のｶﾞｳｽ分布関数での標準偏差 }

※「ガウス基底関数」は「ラジアル基底関数」の一種であり、
本ｾﾐﾅｰ第2回目の「(2.4.3) ラジアル基底関数ネットワーク (Radial Basis Network、 RBN)」
でも少し言及しています。
(ｶﾞｳｽ基底関数は、複数のｶﾞｳｽ分布の線形和になります。
ｶﾞｳｽ分布は平均値の周りに釣り鐘型に対称的な分布をしており、
「ﾗｼﾞｱﾙ (放射状)」な分布関数となっています)

【出典・参考】
「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)

(4.3) 線形基底関数ﾓﾃﾞﾙの実装

・「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」を用いた解析解による
M次の多項式基底関数ﾓﾃﾞﾙ作成の実装例を、以下に示します。
データは「資料09-(03)-1_秋田における日平均気温の月平均値」で、次数M＝２で作成したものです。
(実装は「ﾘｽﾄ09-(04)-1_多項式基底関数ﾓﾃﾞﾙのﾑｰｱ･ﾍﾟﾝﾛｰｽﾞの擬似逆行列を用いた解析解」を参照。



(ﾘｽﾄ09-(04)-1_多項式基底関数ﾓﾃﾞﾙのﾑｰｱ･ﾍﾟﾝﾛｰｽﾞの擬似逆行列を用いた解析解)

```
#*****************************************************************************/
# ﾘｽﾄ09-(04)-1_多項式基底関数ﾓﾃﾞﾙのﾑｰｱ･ﾍﾟﾝﾛｰｽﾞの擬似逆行列を用いた解析解
#*****************************************************************************/
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#*****************************************************************************/
# データ （「秋田」における2月と8月の平均気温の年系列）
#*****************************|***********************************************/
# データ （年）
yyyy= [1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900,
       1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920,
       1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940,
       1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960,
       1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980,
       1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
       2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
# データ （2月の平均気温）
tFeb = [-2.6, -0.9, -1.9, -2.1, -0.4, -3.7, -1.5, 0.6, -1.7, -1.3, -3.6, -0.9, -1.5, -0.9, -1.3, 0.3, -0.9, -2.1,
        -0.9, -1.6, 0.5, -0.8, -2.4, -2.2, -2.2, -2.2, -2.0, -2.0, -2.2, -0.5, 1.0, -1.7, -0.6, -1.0, -0.5, -1.2, -1.9, -1.3, -1.9,
        -0.9, 0.9, -1.9, -1.7, -2.4, -0.9, -2.9, -1.5, -2.3, 0.7, -3.0, -0.5, -2.8, -1.1, 0.2, -1.5, 0.2, -1.2, -1.8, -1.5,
        -0.7, -2.6, -0.8, -1.9, -3.5, -0.5, -2.0, -0.9, 2.3, 0.0, -0.2, -2.8, -1.5, 0.7, 0.2, -1.2, -1.1, 1.2, 2.1, 1.1,
        -0.7, -0.1, 0.0, -1.0, -1.0, 0.1, -0.3, -1.2, -0.8, -0.1, -0.1, 0.3, 1.6, -0.5, -0.5, 0.2, -0.9, -2.5, 2.0, -1.4,
         0.4, -1.0, -0.5, -2.3, 0.5, -1.6, 0.2, -2.0, 1.9, 3.0, 0.5, 0.5, 2.1, 1.3, 1.1, 0.3, 1.3, 1.3, 0.5, 0.2,
         -0.8, 1.9, 0.7, 2.0, -0.5, 0.9, 2.7, 0.0, 1.4, 0.4, 1.1, -1.0, -0.5, -0.1, 2.3, 0.9, 1.3, -0.9, 1.5, 2.5 ]
# データ （8月の平均気温）
```

```
tAug = [23.6,22.5,24.1,25.1,24.4,23.4,23.4,25.4,21.8,23.6,24.0,23.8,24.5,24.0,23.3,24.6,23.5,24.9,
       23.3,21.3,22.1,24.4,20.5,22.9,25.6,24.8,23.9,22.7,23.6,24.4,21.3,24.1,24.1,25.3,23.4,24.1,22.9,25.0,
       23.9,25.1,24.2,24.9,25.0,23.0,24.2,24.4,24.7,25.4,23.9,23.7,25.4,21.8,22.7,22.9,25.2,26.4,24.7,23.1,
       23.0,23.5,26.1,24.8,25.0,25.5,23.8,24.9,25.0,25.9,25.1,23.5,22.8,24.0,25.0,22.3,24.4,23.1,23.6,24.7,
       24.7,24.5,24.1,25.5,24.5,24.9,24.5,23.4,23.4,24.5,23.7,23.1,25.5,24.1,25.1,21.4,22.9,25.3,24.0,22.3,
       23.1,24.6,25.3,25.3,27.3,24.5,23.8,26.1,25.2,25.5,23.4,24.4,21.9,26.9,24.4,24.1,24.2,23.5,27.3,26.8,
       24.3,24.3,23.1,24.3,25.9,26.5,25.1,23.9,23.8,27.0,25.5,26.8,25.4,24.6,24.8,26.4,24.9,25.0,26.5,26.2 ]

#*********************************************************************/
# f_Standardize                                                      */
# 指定したデータを標準化し、その結果と平均・標準偏差を返す。         */
#---------------------------------------------------------------------*/
# 引数：                                                             */
#   d ：対象のデータリスト（ﾃﾞｰﾀの組分）                             */
#---------------------------------------------------------------------*/
# 戻り値：                                                           */
# （第１返値）z     ：指定したデータの標準化結果                     */
# （第２返値）dAve  ：指定したデータの平均                           */
# （第３返値）dStd  ：指定したデータの・標準偏差                     */
#*********************************************************************/
def f_Standardize( d ):
    dAve = np.average(d)
    dStd = np.std(d)

    if dStd == 0.0 :
        return d, dAve, dStd
    else:
        z = np.zeros([len(d)])
        for n in range(0, len(d)):
            z[n] = (d[n] - dAve)/ dStd
        return z, dAve, dStd

#*********************************************************************/
# f_PseudoInvMatrix                                                  */
# Ｍ次の多項式基底関数ﾓﾃﾞﾙの回帰曲線を、                             */
# ムーア・ペンローズの擬似逆行列を解くことにより、算出する。         */
#---------------------------------------------------------------------*/
# 引数：                                                             */
#   x ：説明変数のリスト（ﾃﾞｰﾀの組分）                               */
#   t ：目的変数のリスト（ﾃﾞｰﾀの組分）                               */
#   orderM：多項式基底関数ﾓﾃﾞﾙの多項式の最大次数                     */
#---------------------------------------------------------------------*/
# 戻り値：                                                           */
#   wList ：作成したＭ次の多項式基底関数ﾓﾃﾞﾙの偏回帰係数（Ｍ+1個分）*/
#*********************************************************************/
def f_PseudoInvMatrix( x, t, orderM ):
    # 基底関数の行列Φ(mtrxF)
    dataN = len(t)
    mtrxF = np.zeros([dataN, orderM+1])
    for n in range(0, dataN):
        mtrxF[n,0] = 1
        for m in range(1, orderM+1):
            mtrxF[n,m] = x[n]**m

    # 偏回帰係数を算出して、回帰ﾓﾃﾞﾙを作成
    inv_FtF = np.linalg.inv(np.dot(mtrxF.T, mtrxF))
    mtrxM = np.dot(inv_FtF, mtrxF.T) # ムーア・ペンローズの擬似逆行列
    wList = np.dot(mtrxM, t)     # Ｍ次元線形回帰ﾓﾃﾞﾙの解析解

    print("(wList)=({0})".format(wList))
    return wList
```

```
#*************************************************************************/
# f_PredAndPlot                                                          */
# 指定データからM次の多項式基底関数ﾓﾃﾞﾙを作成し、                          */
# その観測値のﾌﾟﾛｯﾄとﾓﾃﾞﾙ予測値の時系列グラフを表示する。                  */
#-------------------------------------------------------------------------*/
# 引数：                                                                 */
#   xList ：観測値のうち説明変数のリスト（ﾃﾞｰﾀの組分）                    */
#   tList ：観測値のうち目的変数のリスト（ﾃﾞｰﾀの組分）                    */
#   orderM：多項式基底関数ﾓﾃﾞﾙの多項式の最大次数                          */
#-------------------------------------------------------------------------*/
# 戻り値： なし                                                          */
#*************************************************************************/
def f_PredAndPlot( title, x, t, orderM ):
    print("//======================================================")
    print("// 多項式基底関数ﾓﾃﾞﾙの次数 M={0}".format(orderM))
    print("//======================================================")

    # データ数
    dataN = len(t)

    # 説明変数について、計算の収束と桁あふれ防止用にﾃﾞｰﾀの標準化を施す
    xList, xAve, xStd = f_Standardize( x )

    # 目的変数について、計算の収束と桁あふれ防止用にﾃﾞｰﾀの標準化を施す
    tList, tAve, tStd = f_Standardize( t )

    # 全データから偏回帰係数を算出して、回帰ﾓﾃﾞﾙを作成
    wList = f_PseudoInvMatrix( xList, tList, orderM )

    # 作成した回帰ﾓﾃﾞﾙで、座標を計算
    y = np.zeros(dataN)
    for n in range(0, dataN):
        xmList = np.zeros([orderM+1])
        for m in range(0, orderM+1):
            xmList[m] = xList[n]**m
        y[n] = (np.dot(wList, xmList) * tStd) + tAve

    # グラフ表示
    plt.figure(figsize=(15, 3))
    plt.title(title)
    plt.plot(x, t, marker='o', linestyle='None',
             markeredgecolor='black', color='cornflowerblue')
    plt.plot(x, y, color='red', linewidth=3, linestyle='-', )
    plt.xlim(np.min(x)-1, np.max(x)+1)
    plt.ylim(np.min(t)-1, np.max(t)+1)
    plt.xlabel("Year")
    plt.ylabel("Temp.(C)")
    plt.grid(True)
    plt.show()

    return

#*************************************************************************/
# メイン処理                                                             */
#*************************************************************************/
M = 2  # 多項式基底関数ﾓﾃﾞﾙの次数（=2:二次曲線）

# 2 月
f_PredAndPlot( "Time series of February Temp(C) at Akita", yyyy, tFeb, M )

# 8 月
f_PredAndPlot( "Time series of August Temp(C) at Akita", yyyy, tAug, M )
```

(5) 適合度

・ここまで「資料09-(03)-1_秋田における日平均気温の月平均値」を取り上げて、
回帰ﾓﾃﾞﾙを幾つか作成してきましたが、どのﾓﾃﾞﾙが最適なものと言えるのでしょうか？
以下のような点が気になります：

(1) 学習ﾃﾞｰﾀで作成した回帰ﾓﾃﾞﾙが、元の学習ﾃﾞｰﾀをどの程度説明できているか？
(2) 作成した回帰ﾓﾃﾞﾙが、学習ﾃﾞｰﾀ以外のﾃﾞｰﾀをどの程度の精度で予測できるのか？

・回帰ﾓﾃﾞﾙが実ﾃﾞｰﾀをどのくらいの精度で説明できるのかの程度を
「適合度（ﾃｷｺﾞｳﾄﾞ, goodness of fit)」と言います。
ここでは、その指標について幾つか紹介します。

・この章では、観測値と回帰値について、以下のような記法を用いることにします。

D個の説明変数 x_d (d=1～D)で説明される目的変数y についてのN個の観測値と回帰値の組
（回帰値＝回帰ﾓﾃﾞﾙによる目的変数の計算値）：

No.	観測値		回帰値 (ﾓﾃﾞﾙ計算値)	残差 (観測値-回帰値)
	説明変数 (D次元)	目的変数		
1	$(x_{1,1}、x_{1,2}、・・・、x_{1,D})$	t_1	y_1	$e_1=t_1-y_1$
2	$(x_{2,1}、x_{2,2}、・・・、x_{2,D})$	t_2	y_2	$e_2=t_2-y_2$
:	:	:	:	:
N	$(x_{N,1}、x_{N,2}、・・・、x_{N,D})$	t_N	y_N	$e_N=t_N-y_N$

- $x_{n,d}$ ：n番目(n=1～N)の観測値の組での、d番目(d=1～D)の説明変数の値
- t_n ：n番目(n=1～N)の観測値の組での、目的変数の観測値
- y_n ：n番目(n=1～N)の観測値の組での、目的変数の回帰値
- e_n ：n番目(n=1～N)の観測値の組での、残差(観測値t_n と回帰値y_n との差t_n-y_n)
- t_{mean} ：N個の観測値の組での、目的変数の平均値 ($t_{mean}=1/N\ \Sigma_{n=1}^{N}\ t_n$)

(5.1) 平均二乗誤差(MSE)、二乗平均平方根誤差(RMSE)

・回帰ﾓﾃﾞﾙの適合度の尺度として、
「平均二乗誤差 (ﾍｲｷﾝﾆｼﾞｮｳｺﾞｻ、Mean Squared Error、MSE)」と、その平方根を取った
「二乗平均平方根誤差 (ﾆｼﾞｮｳﾍｲｷﾝﾍｲﾎｳｺﾝｺﾞｻ、Root Mean Squared Error、RMSE)」があります。

・n番目の観測値t_n と回帰値y_n との差e_n (= $t_n - y_n$)を「残差 (ｻﾞﾝｻ、residual)」といいます。
「平均二乗誤差 (MSE)」は、「残差」の二乗の平均値のことです。
これは、元のﾃﾞｰﾀを二乗した次元の値となっています。

平均二乗誤差 (MSE)

$$MSE = (1/N) \sum_{n=1}^{N} (t_n - y_n)^2 = (1/N) \sum_{n=1}^{N} e_n^2 \quad \cdots(式5\text{-}1)$$

・これまで扱ってきた、
「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」を用いた線形回帰ﾓﾃﾞﾙや基底関数ﾓﾃﾞﾙの解析解の導出や、
「勾配法」を用いた数値解の導出は、「平均二乗誤差(MSE)を最小にする」という目標を達成する
為の方法です。

・「平均二乗誤差 (MSE)」の平方根をとったものが「二乗平均平方根誤差 (RMSE)」で、
元のﾃﾞｰﾀと同じ次元の値です。

二乗平均平方根誤差 (RMSE)

$$RMSE = \sqrt{MSE} = \sqrt{(1/N) \sum_{n=1}^{N} (t_n - y_n)^2} \quad \cdots(式5\text{-}2)$$

【出典・参考】
MSE⇒ 「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)
RMSE⇒ https://funatsu-lab.github.io/open-course-ware/basic-theory/accuracy-index/#rmse

(5.2) 決定係数(R^2)

・回帰ﾓﾃﾞﾙの適合度の別の尺度として、
「決定係数 (ｹｯﾃｲｹｲｽｳ、coefficient of determination)」 R^2 (ｱｰﾙｽｰ)があります。

・「決定係数 (R^2)」を以下のように定義します。
これは、回帰値の変動が全変動に占める割合を示しています。

決定係数 (R^2)

$$R^2 = ESS／TSS \quad \cdots(式5\text{-}3)$$
$$= \sum_{n=1}^{N} (y_n - t_{mean})^2 ／ \sum_{n=1}^{N} (t_n - t_{mean})^2 \quad \cdots(式5\text{-}4)$$
$$= 1 - \{ \sum_{n=1}^{N} e_n^2 ／ \sum_{n=1}^{N} (t_n - t_{mean})^2 \} \quad \cdots(式5\text{-}5)$$
$$= 1 - RSS／TSS \quad \cdots(式5\text{-}6)$$

ESS (回帰値 y_n の平均周りの変動、Explained Sum of squares)

$$ESS = \sum_{n=1}^{N} (y_n - t_{mean})^2 \quad \cdots(式5\text{-}7)$$

これは、回帰値y_n の、観測値の平均t_{mean} 周りの変動です。

TSS (全変動、ｾﾞﾝﾍﾝﾄﾞｳ、Total Sum of squares)

$$TSS = \sum_{n=1}^{N} (t_n - t_{mean})^2 \quad \cdots(式5\text{-}8)$$

これは、観測値t_n の、観測値の平均t_{mean} 周りの変動です。

RSS (残差変動(ｻﾞﾝｻﾍﾝﾄﾞｳ)、残差平方和(ｻﾞﾝｻﾍｲﾎｳﾜ)、Residual Sum of squares)

$$RSS = \sum_{n=1}^{N} (t_n - y_n)^2 = \sum_{n=1}^{N} e_n^2 \quad \cdots(式5\text{-}9)$$

これは、回帰値y_n と観測値t_n の残差($e_n = t_n - y_n$)の平方和です。

・「決定係数 (R^2)」は、(式5-5)を見てもわかるように、
回帰値y_n と観測値t_n との差(残差e_n)が小さい程、１に近い値になります。

【出典・参考】

決定係数⇒「統計学」森棟,照井,中川,西埜,黒住 共著 有斐閣 2017年12月 改訂版第3刷

決定係数⇒ https://aizine.ai/r2-score0411/

(5.3) 決定係数(R^2) と重相関係数

・観測値t_n と、回帰値y_n (n=1～N)の相関係数を
「重相関係数 (ｼﾞｭｳｿｳｶﾝｹｲｽｳ、Multiple correlation coefficient)」と言います。
重相関係数が１に近いほど、回帰ﾓﾃﾞﾙの精度が高いといえます。

・最小二乗法による回帰ﾓﾃﾞﾙの場合、「決定係数 (R^2)」の平方根は、
実は「重相関係数」に等しいことが分かっています。

相関係数については、既にｾﾐﾅｰ第６回の「(5.3) 共分散と相関係数」で取り上げていますが、
それを、観測値t_n と、回帰値y_n (n=1～N)の相関関係に適用することで、この関係が分かります。

重相関係数 R_{ty}

$$= \frac{\sum_{n=1}^{N}(t_n-t_{mean})(y_n-y_{mean})}{\sqrt{\sum_{n=1}^{N}(t_n-t_{mean})^2}\ \sqrt{\sum_{n=1}^{N}(y_n-y_{mean})^2}}$$ ←共分散の定義

$$= \frac{\sum_{n=1}^{N}(y_n-t_{mean})^2}{\sqrt{\sum_{n=1}^{N}(t_n-t_{mean})^2}\ \sqrt{\sum_{n=1}^{N}(y_n-t_{mean})^2}}$$ ←$y_{mean}=t_{mean}$ (※1)

$$= \frac{\sqrt{\sum_{n=1}^{N}(y_n-t_{mean})^2}}{\sqrt{\sum_{n=1}^{N}(t_n-t_{mean})^2}}$$

$$= \sqrt{R^2}$$ ←(式5-4)

＝「決定係数 (R2)」の平方根

(※1) 回帰値平均 y_{mean} ＝ 観測値平均 t_{mean}

【出典・参考】

決定係数(R^2) と重相関係数
⇒「統計学」森棟,照井,中川,西埜,黒住 共著 有斐閣 2017年12月 改訂版第3刷

回帰ﾓﾃﾞﾙの評価⇒https://pythondatascience.plavox.info/scikit-learn/回帰モデルの評価

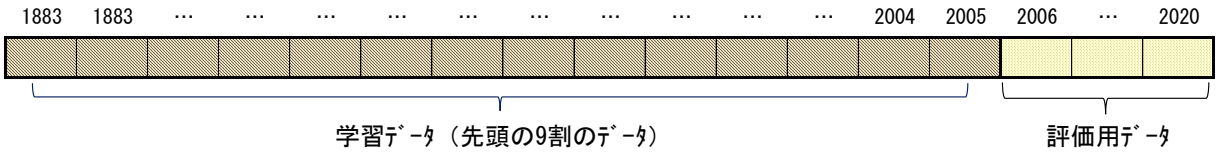
(5.4) 回帰ﾓﾃﾞﾙとその適合度の例

・以下に、M次(M=1～10)の多項式基底関数ﾓﾃﾞﾙを
「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」による解析解で作成し、その適合度の評価を行ってみます。

・データはこれまで用いてきた
「資料09-(03)-1_秋田における日平均気温の月平均値」のうち、「1883年～2020年」の期間で、
2月のものを用います。

・最初の９割ほどの期間のデータ（1883年～2005年）を学習ﾃﾞｰﾀとして用いて、
M次の多項式基底関数ﾓﾃﾞﾙを解析解から作成します。

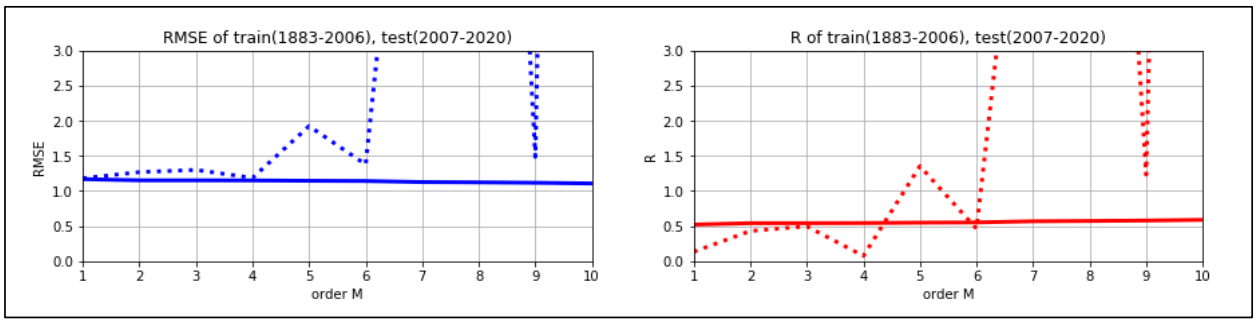
・最後の１割ほどの期間のデータ（2006年～2020年）を評価用ﾃﾞｰﾀとし、
学習ﾃﾞｰﾀで作成したﾓﾃﾞﾙを用いて、評価用ﾃﾞｰﾀの計算値を求めます。



・これにより、全期間（1883年～2020年）に渡る二乗平均平方根誤差(RMSE)と決定係数(R2)を算出して、
ﾓﾃﾞﾙの適合度を評価してみます。
以下に実装例とその出力結果および結果概要を述べます。

(1) 学習ﾃﾞｰﾀで作成した回帰ﾓﾃﾞﾙが、元の学習ﾃﾞｰﾀをどの程度説明できているか？
⇒多項式基底関数ﾓﾃﾞﾙは次数を上げるほど、学習ﾃﾞｰﾀ自体の適合度は上がっています。
下のｸﾞﾗﾌで、実線が学習ﾃﾞｰﾀ自体に対する予測値のRMSEとR^2ですが、
RMSEは次数が上がるほど、RMSEが僅かずつ減少し、R(=$\sqrt{R^2}$)が増加する様子が分かります。

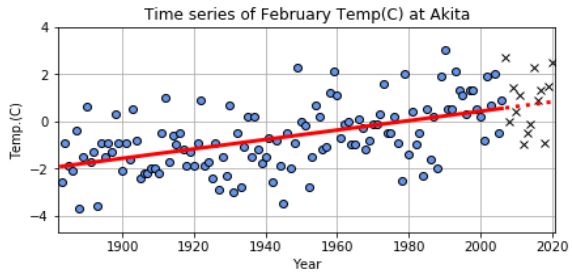
(2) 作成した回帰ﾓﾃﾞﾙが、学習ﾃﾞｰﾀ以外のﾃﾞｰﾀをどの程度の精度で予測できるのか？
⇒「学習ﾃﾞｰﾀ以外のﾃﾞｰﾀ」は「評価用ﾃﾞｰﾀ」のことですが、
多項式基底関数ﾓﾃﾞﾙで次数を上げても、評価用ﾃﾞｰﾀの適合度がよくなるわけではありません。
下のｸﾞﾗﾌで、点線が評価用ﾃﾞｰﾀに対する予測値のRMSEとR^2ですが、
RMSEについては、1～4 の次数では小さ目ですが、5以上になると大きく増加しています。
R(=$\sqrt{R^2}$)については、4以上の次数になると不安定になっています。

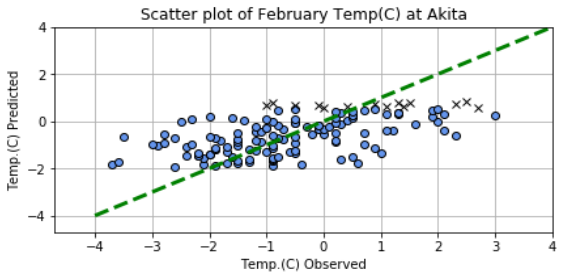


(3) 実際の観測値とﾓﾃﾞﾙの計算値の時系列と、その相関図を見ても上記の傾向はわかります。
次数が５以上のあたりから、過学習が目立っていて、
評価用ﾃﾞｰﾀに対する予測には使えそうにないことが分かります。
本時系列ﾃﾞｰﾀへの多項式基底関数ﾓﾃﾞﾙの当てはめという観点では、
次数１～４が適合度が高いといえるでしょう。

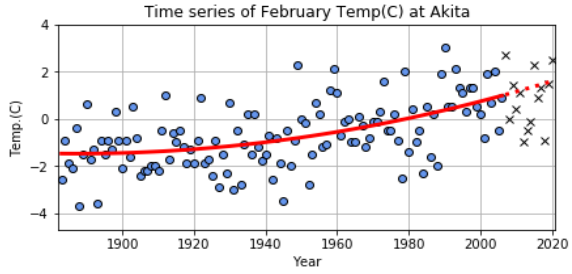
実装(ﾘｽﾄ09-(05)-1_多項式基底関数ﾓﾃﾞﾙの適合度による評価)の実行結果

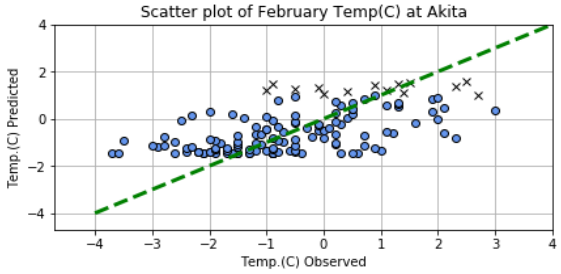
```
//=======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=1
//=======================================================
(wList)=([-0.00961914  0.55733766])
Train: RMSE=1.1688755142303715, R=0.5215104848328811
Test : RMSE=1.1813948662913125, R=0.13380722983540289
```



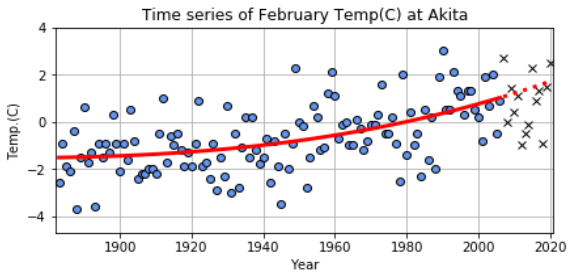


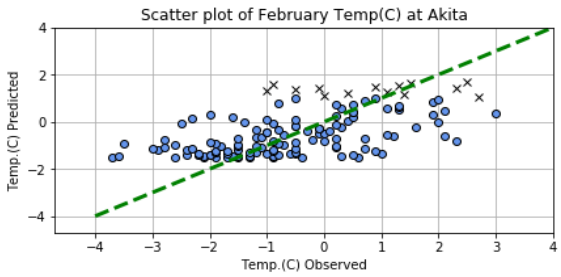

```
//=======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=2
//=======================================================
(wList)=([-0.16010325  0.62544547  0.1937965 ])
Train: RMSE=1.1517019967864262, R=0.541487642100144
Test : RMSE=1.2683484695446088, R=0.43012525301202115
```



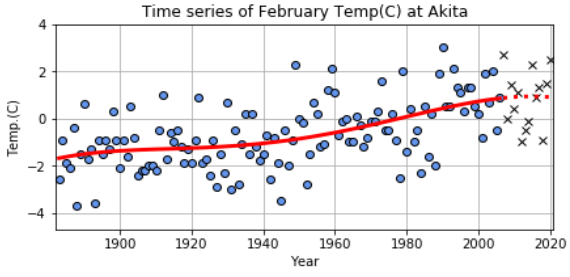


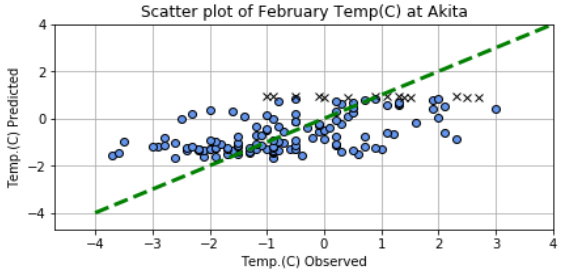

```
//=======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=3
//=======================================================
(wList)=([-0.16500573  0.59875746  0.2041372   0.01961587])
Train: RMSE=1.1515916452082242, R=0.5416126881629991
Test : RMSE=1.3000564379292538, R=0.4974390855057646
```



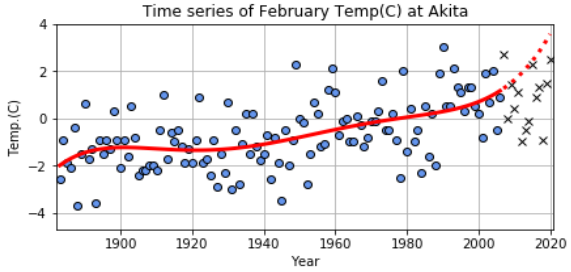


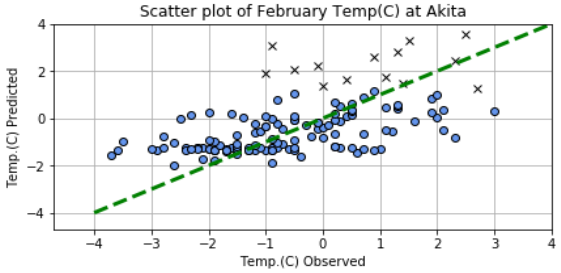

```
//=======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=4
//=======================================================
(wList)=([-0.20659713  0.66574112  0.383046   -0.04690469 -0.09484004])
Train: RMSE=1.1500118937707704, R=0.5433983366819742
Test : RMSE=1.1867653143853083, R=0.07487137316748849
```



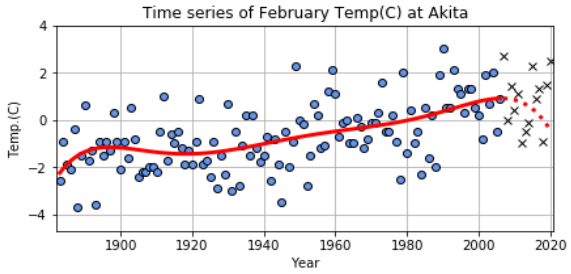


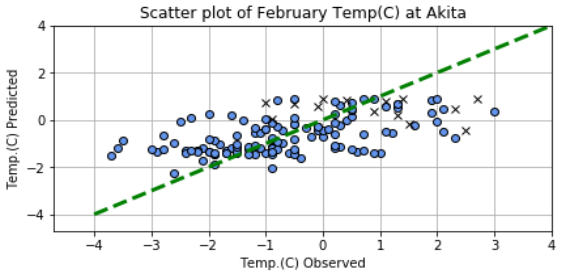
```
//======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=5
//======================================================
(wList)=([-0.15759346  0.9101528   0.09347457 -0.55250603  0.09189365  0.21230797])
Train: RMSE=1.1451431450446998, R=0.5488498728870218
Test : RMSE=1.923978747895218, R=1.3538306831741553
```



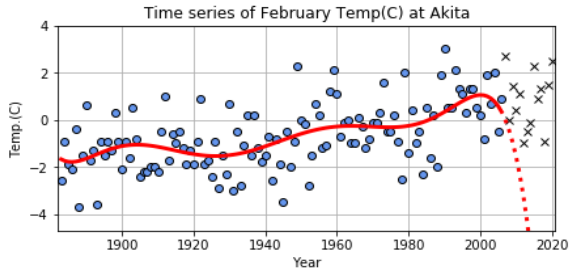


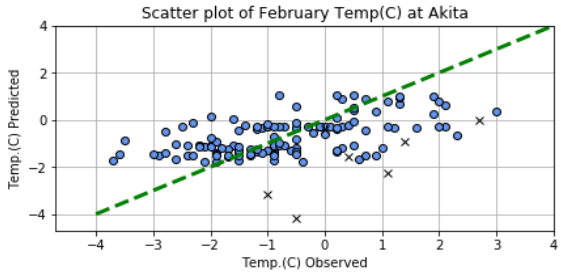

```
//======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=6
//======================================================
(wList)=([-0.10870135  0.72444723 -0.34990089 -0.07788006  0.70088862 -0.01395299
 -0.21460379])
Train: RMSE=1.1421062080517743, R=0.552211492926259
Test : RMSE=1.3775601846644876, R=0.4624716193966872
```



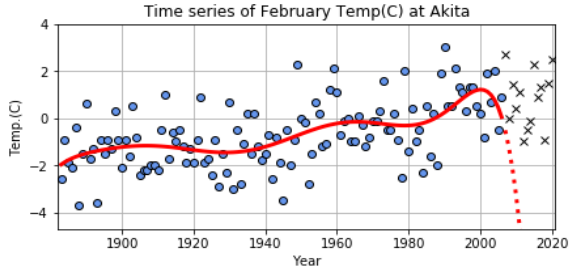


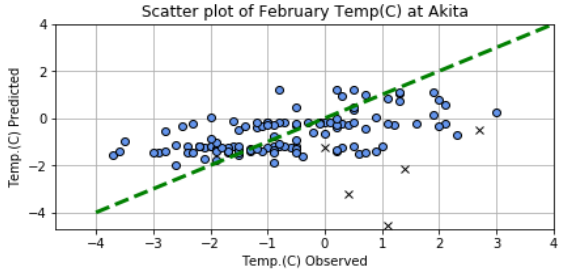

```
//======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=7
//======================================================
(wList)=([ 2.07365480e-03  1.20089125e+00 -1.61332551e+00 -1.99694029e+00
  2.69274548e+00  1.98776696e+00 -9.69673565e-01 -6.13858207e-01])
Train: RMSE=1.1269085085724877, R=0.5686057304237229
Test : RMSE=9.110186552106256, R=7.567149431992996
```



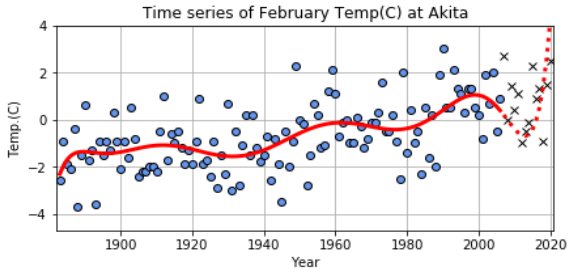


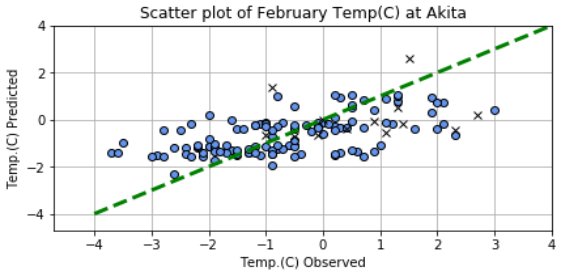

```
//======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=8
//======================================================
(wList)=([-0.04077012  1.53950304 -0.94243073 -3.5854253   0.97216097  3.78061932
  0.49898454 -1.1792667  -0.40220861])
Train: RMSE=1.1229247865860812, R=0.5727899454267945
Test : RMSE=16.81485442728724, R=14.138994553650333
```



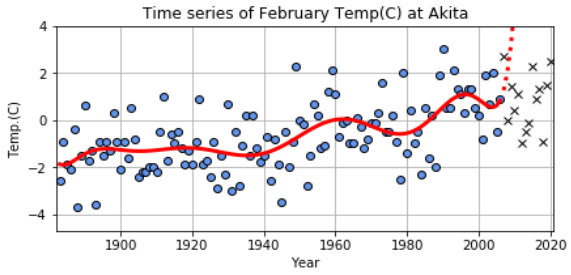


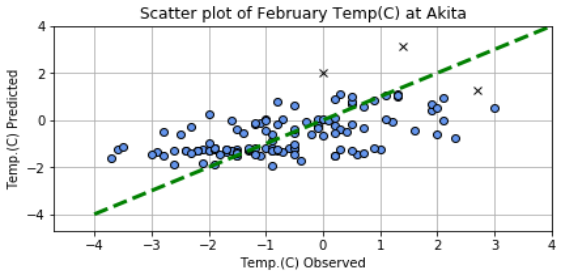
```
//=======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=9
//=======================================================
(wList)=([ 0.03901272  1.80663894 -2.42688853 -5.37267637  5.12780628  7.19011382
 -3.14434948 -3.67922006  0.58376094  0.62344768])
Train: RMSE=1.117115823196911, R=0.5788108732236767
Test : RMSE=1.4859558534538961, R=1.232738326621701
```






```
//=======================================================
// 多項式基底関数ﾓﾃﾞﾙの次数 M=10
//=======================================================
(wList)=([-4.75445502e-03  2.55071551e+00 -1.36299100e+00 -1.08815968e+01
  6.54452049e-01  1.79755054e+01  3.80375587e+00 -1.14107166e+01
 -3.94801521e+00  2.45988545e+00  1.04509371e+00])
Train: RMSE=1.1072008626460719, R=0.5888749158653319
Test : RMSE=41.26418764294936, R=35.24857980239155
```





(ﾘｽﾄ09-(05)-1_多項式基底関数ﾓﾃﾞﾙの適合度による評価)

```
#*****************************************************************************
# ﾘｽﾄ09-(05)-1_多項式基底関数ﾓﾃﾞﾙの適合度による評価
#-----------------------------------------------------------------------------
#「秋田」の2月の平均気温の年系列のデータについて、
# 先頭の9割のデータからM次の多項式基底関数ﾓﾃﾞﾙを
# 「ムーア・ペンローズの擬似逆行列」を用いた解析解を導出して作成します。
#
# 作成したﾓﾃﾞﾙを用いて、全期間の計算値を求めて実測値と比較することにより、
# ﾓﾃﾞﾙを作成した期間と、それ以外の期間について、
# その二乗平均平方根誤差(RMSE)と決定係数(R2)を算出して評価します。
#*****************************************************************************
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#*****************************************************************************/
# データ （「秋田」における2月の平均気温の年系列）
#*****************************************************************************/
# データ （年）
yyyy= [1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900,
       1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920,
       1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940,
       1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960,
       1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980,
       1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
       2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
```

```
# データ (2月の平均気温)
tFeb = [-2.6,-0.9,-1.9,-2.1,-0.4,-3.7,-1.5,0.6,-1.7,-1.3,-3.6,-0.9,-1.5,-0.9,-1.3,0.3,-0.9,-2.1,
        -0.9,-1.6,0.5,-0.8,-2.4,-2.2,-2.2,-2.2,-2.0,-2.0,-2.2,-0.5,1.0,-1.7,-0.6,-1.0,-0.5,-1.2,-1.9,-1.3,-1.9,
        -0.9,0.9,-1.9,-1.7,-2.4,-0.9,-2.9,-1.5,-2.3,0.7,-3.0,-0.5,-2.8,-1.1,0.2,-1.5,0.2,-1.2,-1.8,-1.5,
        -0.7,-2.6,-0.8,-1.9,-3.5,-0.5,-2.0,-0.9,2.3,0.0,-0.2,-2.8,-1.5,0.7,0.2,-1.2,-1.1,1.2,2.1,1.1,
        -0.7,-0.1,0.0,-1.0,-1.0,0.1,-0.3,-1.2,-0.8,-0.1,-0.1,0.3,1.6,-0.5,-0.5,0.2,-0.9,-2.5,2.0,-1.4,
         0.4,-1.0,-0.5,-2.3,0.5,-1.6,0.2,-2.0,1.9,3.0,0.5,0.5,2.1,1.3,1.1,0.3,1.3,1.3,0.5,0.2,
        -0.8,1.9,0.7,2.0,-0.5,0.9,2.7,0.0,1.4,0.4,1.1,-1.0,-0.5,-0.1,2.3,0.9,1.3,-0.9,1.5,2.5 ]

#*************************************************************************/
# f_RMSE                                                                 */
# 計算値と実測値の二乗平均平方根誤差 (RMSE)を算出する。                  */
#-------------------------------------------------------------------------*/
# 引数:                                                                  */
#   t   : 実測値 (教師ﾃﾞｰﾀ)                                              */
#   y   : 予測値                                                         */
#-------------------------------------------------------------------------*/
# 戻り値:二乗平均平方根誤差 (RMSE)                                       */
#*************************************************************************/
def  f_RMSE(t, y):
    if (len(t) <= 0):
        return 0.0

    err2Sum = 0.0
    for n in range(0, len(t)):
        err2Sum = err2Sum + (t[n] - y[n])**2
    return np.sqrt(err2Sum / len(t))

#*************************************************************************/
# f_R2                                                                   */
# 計算値と実測値の決定係数(R2)を算出する。                               */
#-------------------------------------------------------------------------*/
# 引数:                                                                  */
#   t   : 実測値 (教師ﾃﾞｰﾀ)                                              */
#   y   : 予測値                                                         */
#-------------------------------------------------------------------------*/
# 戻り値:決定係数(R2)                                                    */
#*************************************************************************/
def  f_R2(t, y):
    if (len(t) <= 0):
        return 0.0

    tmean = np.mean(t)
    ess = 0.0
    tss = 0.0
    for n in range(0, len(t)):
        ess = ess + (y[n] - tmean)**2
        tss = tss + (t[n] - tmean)**2
    return ess / tss

#*************************************************************************/
# f_PseudoInvMatrix                                                      */
# M次の多項式基底関数ﾓﾃﾞﾙの回帰曲線を、                                  */
# ムーア・ペンローズの擬似逆行列を解くことにより、算出する。             */
#-------------------------------------------------------------------------*/
# 引数:                                                                  */
#   x :説明変数のリスト (ﾃﾞｰﾀの組分)                                     */
#   t :目的変数のリスト (ﾃﾞｰﾀの組分)                                     */
#   orderM:多項式基底関数ﾓﾃﾞﾙの多項式の最大次数                          */
#-------------------------------------------------------------------------*/
# 戻り値:                                                                */
#   wList :作成したM次の多項式基底関数ﾓﾃﾞﾙの偏回帰係数 (M+1個分)          */
#*************************************************************************/
def f_PseudoInvMatrix( x, t, orderM ):
    # 基底関数の行列Φ(mtrxF)
```

```
    dataN = len(t)
    mtrxF = np.zeros([dataN, orderM+1])
    for n in range(0, dataN):
        mtrxF[n,0] = 1
        for m in range(1, orderM+1):
            mtrxF[n,m] = x[n]**m

    # 偏回帰係数を算出して、回帰ﾓﾃﾞﾙを作成
    inv_FtF = np.linalg.inv(np.dot(mtrxF.T, mtrxF))
    mtrxM = np.dot(inv_FtF, mtrxF.T) # ムーア・ペンローズの擬似逆行列
    wList = np.dot(mtrxM, t)    # M次元線形回帰ﾓﾃﾞﾙの解析解

    print("(wList)=({0})".format(wList))
    return wList

#*********************************************************************/
# f_Standardize                                                      */
# 指定したデータを標準化し、その結果と平均・標準偏差を返す。          */
#---------------------------------------------------------------------*/
# 引数：                                                             */
#   d ：対象のデータリスト（ﾃﾞｰﾀの組分）                             */
#---------------------------------------------------------------------*/
# 戻り値：                                                           */
# （第１返値）z     ：指定したデータの標準化結果                     */
# （第２返値）dAve  ：指定したデータの平均                           */
# （第３返値）dStd  ：指定したデータの・標準偏差                     */
#*********************************************************************/
def f_Standardize( d ):
    dAve = np.average(d)
    dStd = np.std(d)

    if dStd == 0.0 :
        return d, dAve, dStd
    else:
        z = np.zeros([len(d)])
        for n in range(0, len(d)):
            z[n] = (d[n] - dAve)/ dStd
        return z, dAve, dStd

#*********************************************************************/
# f_PredAndPlot                                                      */
# 先頭の9割のデータからM次の多項式基底関数ﾓﾃﾞﾙを作成し、              */
# モデルの計算値と、実測値を比較することにより、                     */
# その精度指標とグラフを表示する。                                   */
#---------------------------------------------------------------------*/
# 引数：                                                             */
#   xList ：観測値のうち説明変数のリスト（ﾃﾞｰﾀの組分）               */
#   tList ：観測値のうち目的変数のリスト（ﾃﾞｰﾀの組分）               */
#   orderM：多項式基底関数ﾓﾃﾞﾙの多項式の最大次数                     */
#---------------------------------------------------------------------*/
# 戻り値：                                                           */
# （第1返値）学習ﾃﾞｰﾀ（先頭の9割のﾃﾞｰﾀ）の最後の説明変数の配列添え字 */
# （第2返値）学習ﾃﾞｰﾀ（先頭の9割のﾃﾞｰﾀ）の二乗平均平方根誤差（RMSE） */
# （第3返値）学習ﾃﾞｰﾀ（先頭の9割のﾃﾞｰﾀ）決定係数(R2)の平方根（相関係数）*/
# （第4返値）評価ﾃﾞｰﾀ（最後の1割のﾃﾞｰﾀ）二乗平均平方根誤差（RMSE）   */
# （第5返値）評価ﾃﾞｰﾀ（最後の1割のﾃﾞｰﾀ）決定係数(R2)の平方根（相関係数）*/
#*********************************************************************/
def f_PredAndPlot( x, t, orderM ):
    print("//=====================================================")
    print("// 多項式基底関数ﾓﾃﾞﾙの次数 M={0}".format(orderM))
    print("//=====================================================")

    # データ数
    dataN = len(t)
```

```
# 説明変数について、計算の収束と桁あふれ防止用にﾃﾞｰﾀの標準化を施す
xList, xAve, xStd = f_Standardize( x )

# 目的変数について、計算の収束と桁あふれ防止用にﾃﾞｰﾀの標準化を施す
tList, tAve, tStd = f_Standardize( t )

# 先頭の９割のデータから偏回帰係数を算出して、回帰ﾓﾃﾞﾙを作成
dataNc = int(np.trunc(dataN * 0.9))
wList = f_PseudoInvMatrix( xList[:dataNc], tList[:dataNc], orderM )

# 作成した回帰ﾓﾃﾞﾙで、座標を計算
y = np.zeros(dataN)
for n in range(0, dataN):
    xmList = np.zeros([orderM+1])
    for m in range(0, orderM+1):
        xmList[m] = xList[n]**m
    y[n] = (np.dot(wList, xmList) * tStd) + tAve

# RMSE、R2を計算（ﾓﾃﾞﾙ作成用の学習ﾃﾞｰﾀ（先頭の9割のデータ））
o_TrainRmse = f_RMSE(t[:dataNc], y[:dataNc])
o_TrainR2 = f_R2(t[:dataNc], y[:dataNc])
o_TrainR = np.sqrt(o_TrainR2)
print("Train: RMSE={0}, R={1}".format(o_TrainRmse, o_TrainR))

# RMSE、R2を計算（評価用のﾃﾞｰﾀ（最後の1割のデータ））
o_TestRmse = f_RMSE(t[dataNc:], y[dataNc:])
o_TestR2 = f_R2(t[dataNc:], y[dataNc:])
o_TestR = np.sqrt(o_TestR2)
print("Test : RMSE={0}, R={1}".format(o_TestRmse, o_TestR))

# グラフ表示範囲
diagList = np.linspace(int(np.min(t)-1),int(np.max(t)+1), 100)
plt.figure(figsize=(15, 3))

# 時系列
plt.subplot(1, 2, 1)
plt.title("Time series of February Temp(C) at Akita")
plt.plot(x[:dataNc], t[:dataNc], marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue') # 学習ﾃﾞｰﾀ
plt.plot(x[dataNc:], t[dataNc:], marker='x', linestyle='None',
         markeredgecolor='black', color='blue') # 試験ﾃﾞｰﾀ
plt.plot(x[:dataNc], y[:dataNc], color='red', linewidth=3, linestyle='-', ) # 学習ﾃﾞｰﾀ
plt.plot(x[dataNc:], y[dataNc:], color='red', linewidth=3, linestyle=':', ) # 試験ﾃﾞｰﾀ
plt.xlim(np.min(x)-1, np.max(x)+1)
plt.ylim(np.min(t)-1, np.max(t)+1)
plt.xlabel("Year")
plt.ylabel("Temp.(C)")
plt.grid(True)

# 観測値と予測値の相関分布
plt.subplot(1, 2, 2)
plt.title("Scatter plot of February Temp(C) at Akita")
plt.plot(t[:dataNc], y[:dataNc], marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue') # 学習ﾃﾞｰﾀ
plt.plot(t[dataNc:], y[dataNc:], marker='x', linestyle='None',
         markeredgecolor='black', color='blue') # 試験ﾃﾞｰﾀ
plt.plot(diagList, diagList, color='green', linewidth=3, linestyle='--', )
plt.xlim(np.min(t)-1, np.max(t)+1)
plt.ylim(np.min(t)-1, np.max(t)+1)
plt.xlabel("Temp.(C) Observed")
plt.ylabel("Temp.(C) Predicted")
plt.grid(True)

plt.show()
```

```
    return  dataNc-1, o_TrainRmse, o_TrainR, o_TestRmse, o_TestR

#*****************************************************************************/
# メイン処理                                                                  */
#*****************************************************************************/
maxM = 10  # 多項式基底関数ﾓﾃﾞﾙの最大次数
mList = np.zeros(maxM)
trainLastI = 0
trainRmse = np.zeros(maxM)
trainR = np.zeros(maxM)
testRmse = np.zeros(maxM)
testR = np.zeros(maxM)

for m in range(0, maxM):
    mList[m] = m+1
    trainLastI, trainRmse[m], trainR[m],testRmse[m], testR[m] = f_PredAndPlot( yyyy, tFeb, m+1 )

# 多項式基底関数ﾓﾃﾞﾙの次数毎の適合度の変化
plt.figure(figsize=(15, 3))

plt.subplot(1, 2, 1)
plt.title("RMSE of train({0}-{1}), test({2}-{3})".format(
    yyyy[0], yyyy[trainLastI], yyyy[trainLastI+1], yyyy[len(yyyy)-1]))
plt.plot(mList, trainRmse, color='blue', linewidth=3, linestyle='-', label='$RMSE (train)$')
plt.plot(mList, testRmse, color='blue', linewidth=3, linestyle=':', label='$RMSE (test)$')
plt.xlim(1, maxM)
plt.ylim(0, 3)
plt.xlabel("order M")
plt.ylabel("RMSE")
plt.grid(True)

plt.subplot(1, 2, 2)
plt.title("R of train({0}-{1}), test({2}-{3})".format(
    yyyy[0], yyyy[trainLastI], yyyy[trainLastI+1], yyyy[len(yyyy)-1]))
plt.plot(mList, trainR, color='red', linewidth=3, linestyle='-', label='$R (train)$')
plt.plot(mList, testR, color='red', linewidth=3, linestyle=':', label='$R (test)$')
plt.xlim(1, maxM)
plt.ylim(0, 3)
plt.xlabel("order M")
plt.ylabel("R")
plt.grid(True)
plt.show()
```

【出典・参考】

「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)

(6) ロジスティック回帰

(6.1) ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ

・これまでは、連続値をとる目的変数 y と D個の説明変数 x_d (d=1～D)との関係を扱ってきました。
これに対し、二値（成功/失敗、True/False、Yes/No、といった2種類の値を取る）の目的変数 y と D個の説明変数 x_d (d=1～D)との関数関係を解析する手法が
「ﾛｼﾞｽﾃｨｯｸ回帰分析（ﾛｼﾞｽﾃｨｯｸｶｲｷﾌﾞﾝｾｷ, Logistic Regression Analysys)」で、その数式ﾓﾃﾞﾙを
「ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ（ﾛｼﾞｽﾃｨｯｸｶｲｷﾓﾃﾞﾙ, Logistic Regression Model)」と言います。
「ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ」で目的変数 y は、着目する事象の発生確率 p (0≦p≦1) を表します。

・二値を目的変数とすることから判るように、「ﾛｼﾞｽﾃｨｯｸ回帰」は回帰ではなく分類の為のﾓﾃﾞﾙですが、
本ｾﾐﾅｰ第８回目の「機械学習（２回目：機械学習の概要と教師あり学習（分類））」では、
扱って来なかったので、ここで扱うことにします。

・ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙは次式で表現され、値域は 0≦y≦1 となっています。

ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ

$$y = 1 / \{ 1 + \exp(-(w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D)) \} \quad \cdots(式6\text{-}1)$$

$$= 1 / \{ 1 + \exp(-\sum_{d=0}^{D} w_dx_d) \} \quad \cdots(式6\text{-}2)$$

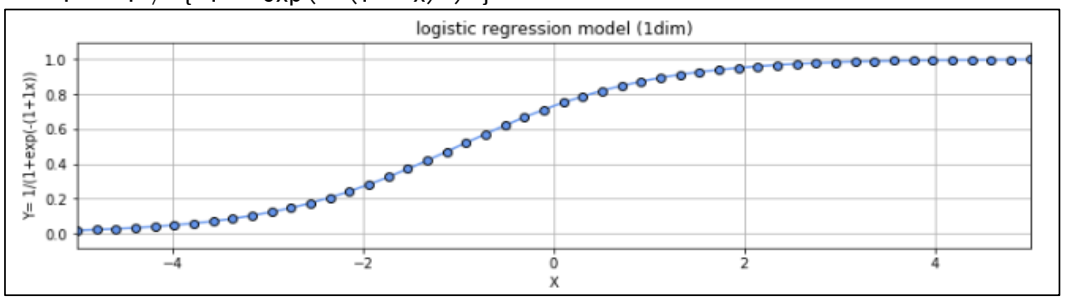
$$= 1 / \{ 1 + \exp(-\boldsymbol{w}^T \boldsymbol{x}) \} \quad \cdots(式6\text{-}3)$$

$$\boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$$

x_d ：d番目の説明変数 {d=1～D} (x_0=1)
w_d ：d番目の説明変数にかかる偏回帰係数 {d=1～D} (w_0 は定数項)

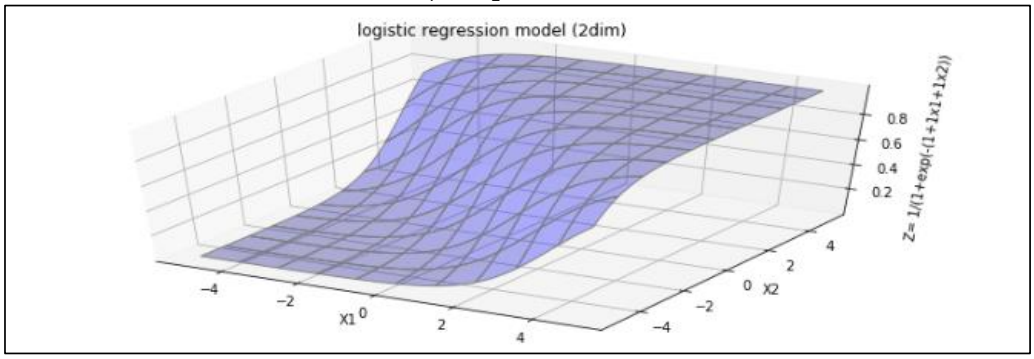
・説明変数が1個のﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ例 (ﾘｽﾄ09-(06)-1_ﾛｼﾞｽﾃｨｯｸ回帰のｸﾞﾗﾌ)：

$$Y = 1 / \{ 1 + \exp(-(1 + x)) \}$$



・説明変数が2個のﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ例 (ﾘｽﾄ09-(06)-1_ﾛｼﾞｽﾃｨｯｸ回帰のｸﾞﾗﾌ)：

$$Y = 1 / \{ 1 + \exp(-(1 + x_1 + x_2)) \}$$



・着目する事象の発生確率 p と発生しない確率 (1-p) との比を「オッズ (odds)」と言います。
更に、オッズの自然対数を取った値を「対数オッズ (ﾀｲｽｳｵｯｽﾞ、logarithmic odds)」と言います。

オッズ、対数オッズ

$$\text{オッズ} = p/(1-p) \qquad \cdots(式6\text{-}4)$$

$$\text{対数オッズ} = \ln\left(p/(1-p)\right) \qquad \cdots(式6\text{-}5)$$

- p : 着目する事象の発生する確率
- $1-p$: 着目する事象の発生しない確率

・「対数オッズ」は発生確率 p の関数であり、
これを「ﾛｼﾞｯﾄ関数 (ﾛｼﾞｯﾄｶﾝｽｳ、logit function)」と言いますが、
発生確率 p の確率分布がロジスティック回帰モデルに従うとした場合、
以下のような関係式があります：

ﾛｼﾞｯﾄ関数とﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ

$$\text{ﾛｼﾞｯﾄ関数} = \ln\left(p/(1-p)\right) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D \qquad \cdots(式6\text{-}6)$$

- p : 着目する事象の発生する確率
- $1-p$: 着目する事象の発生しない確率
- x_d : 説明変数 (d=0～D、但し x_0=1)
- w_d : 説明変数 x_d の偏回帰係数 (d=0～D)

・(式6-6)の導出は以下のようになります：

L を

$$L = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D$$

で定義すると、ロジスティック回帰モデルは

$$p = 1 / \{1 + \exp(-L)\} \qquad \cdots(式6\text{-}1')$$
$$= \{1 + \exp(-L) - \exp(-L)\} / \{1 + \exp(-L)\}$$
$$= 1 - \exp(-L) / \{1 + \exp(-L)\}$$
$$\therefore \quad 1 - p = \exp(-L) / \{1 + \exp(-L)\}$$
$$= \exp(-L) * p$$
$$\therefore \quad (1 - p)/p = \exp(-L)$$
$$\therefore \quad p/(1 - p) = \exp(L)$$

両辺の自然対数を取ることにより、以下の関係式が得られます：

$$\ln\left(p/(1 - p)\right) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D \qquad \cdots(式6\text{-}6)$$

【出典・参考】

ﾛｼﾞｽﾃｨｯｸ回帰⇒https://udemy.benesse.co.jp/ai/logistic-regression-analysis.html
ﾛｼﾞｽﾃｨｯｸ回帰⇒https://to-kei.net/regression/logistic_regression/
ﾛｼﾞｽﾃｨｯｸ回帰⇒「統計学が最強の学問である[実践編]」西内啓著 ﾀﾞｲﾔﾓﾝﾄﾞ社 2017年3月 第7刷
ﾛｼﾞｽﾃｨｯｸ回帰⇒「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)
オッズ⇒https://ja.wikipedia.org/wiki/オッズ
ロジット⇒https://ja.wikipedia.org/wiki/ロジット

(ﾘｽﾄ09-(06)-1_ﾛｼﾞｽﾃｨｯｸ回帰のｸﾞﾗﾌ)

```
#*****************************************************************************
# ﾘｽﾄ09-(06)-1_ﾛｼﾞｽﾃｨｯｸ回帰のｸﾞﾗﾌ
#*****************************************************************************
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from mpl_toolkits.mplot3d import axes3d

#*****************************************************************/
# f_LogisticReg1                                                 */
# ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ (1次元)                                          */
#-----------------------------------------------------------------*/
# 引数：                                                         */
#   w   :偏回帰係数 (D+1次元)                                    */
#   x   :説明変数 (1次元)                                        */
#-----------------------------------------------------------------*/
# 戻り値：                                                       */
# (第１返値) logis：ﾛｼﾞｽﾃｨｯｸ回帰の計算値                          */
#*****************************************************************/
def f_LogisticReg1( w, x ):
    logit = w[0] + w[1]*x
    logis = 1 / (1 + np.exp(-logit))
    return logis

#*****************************************************************/
# f_LogisticReg2                                                 */
# ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ (2次元)                                          */
#-----------------------------------------------------------------*/
# 引数：                                                         */
#   w    :偏回帰係数 (D+1次元)                                   */
#   x1   :説明変数1 (1次元)                                      */
#   x2   :説明変数2 (1次元)                                      */
#-----------------------------------------------------------------*/
# 戻り値：                                                       */
# (第１返値) logis：ﾛｼﾞｽﾃｨｯｸ回帰の計算値                          */
#*****************************************************************/
def f_LogisticReg2( w, x1, x2 ):
    logit = w[0] + w[1]*x1 + w[2]*x2
    logis = 1 / (1 + np.exp(-logit))
    return logis

#*****************************************************************/
# f_ShowLogistic1                                                */
# 指定した偏回帰係数で、1次元ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの計算値を計算し、     */
# グラフを表示する。                                             */
#-----------------------------------------------------------------*/
# 引数：                                                         */
#   plt     : 表示ｵﾌﾞｼﾞｪｸﾄ                                       */
#   xMeshNo : 表示時のメッシュ数                                 */
#   xRange  : 説明変数 の表示範囲                                */
#   wList   : 偏回帰係数                                        */
#-----------------------------------------------------------------*/
# 戻り値：                                                       */
#*****************************************************************/
def f_ShowLogistic1( plt, xMeshNo, xRange, wList ):
    xList = np.linspace(xRange[0], xRange[1], xMeshNo)
    yylogis = f_LogisticReg1(wList, xList)

    plt.title("logistic regression model (1dim)")
    plt.plot(xList, yylogis, marker='o', linestyle='-',
             markeredgecolor='black', color='cornflowerblue')

    plt.xlim(xRange[0], xRange[1])
    plt.ylim(np.min(yylogis)-0.1, np.max(yylogis)+0.1)
```

```
    plt.xlabel("X")
    plt.ylabel("Y= 1/(1+exp(-({0}+{1}x))".format(wList[0],wList[1]))
    plt.grid(True)
    return

#**************************************************************************/
# f_ShowLogistic2                                                         */
# 指定した偏回帰係数で、2次元ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの計算値を計算し、      */
# 3次元グラフを表示する。                                                 */
#-------------------------------------------------------------------------*/
# 引数：                                                                  */
#   ax       : 3次元表示ｵﾌﾞｼﾞｪｸﾄ                                         */
#   xMeshNo : 表示時のメッシュ数                                          */
#   x1Range : 説明変数1 の表示範囲                                        */
#   x2Range : 説明変数2 の表示範囲                                        */
#   wList    : 偏回帰係数                                                 */
#-------------------------------------------------------------------------*/
# 戻り値：                                                                */
#**************************************************************************/
def f_ShowLogistic2( ax, xMeshNo, x1Range, x2Range, wList ):

    x1 = np.linspace(x1Range[0], x1Range[1], xMeshNo)
    x2 = np.linspace(x2Range[0], x2Range[1], xMeshNo)
    xx1, xx2 = np.meshgrid(x1, x2)
    yylogis = f_LogisticReg2(wList, xx1, xx2)

    ax.set_title("logistic regression model (2dim)")
    ax.set_xlabel("X1")
    ax.set_ylabel("X2")
    ax.set_zlabel("Z= 1/(1+exp(-({0}+{1}x1+{2}x2))".format(
                    wList[0],wList[1],wList[2]))
    ax.plot_surface(xx1,xx2, yylogis, color='blue', edgecolor='gray',
                    rstride=5, cstride=5, alpha=0.3)
    ax.view_init(elev=45, azim=-60)
    return

#**************************************************************************/
# メイン処理                                                              */
#**************************************************************************/
# ｸﾞﾗﾌ表示ﾊﾟﾗﾒｰﾀ
xMeshNo = 50
plt. 琦ure( 琦size=(12, 7))
plt.subplots_adjust(wspace=0.5, hspace=0.5)

# 1次元ﾛｼﾞｽﾃｨｯｸ回帰曲線
plt.subplot(2, 1, 1)
wList1 = [1,1]
xRange = [-5, 5]
f_ShowLogistic1( plt, xMeshNo, xRange, wList1 )

# 2次元ﾛｼﾞｽﾃｨｯｸ回帰曲面
plt. 琦ure( 琦size=(12, 10))
ax = plt.subplot(2, 1, 2, projection='3d')
wList2 = [1,1,1]
x1Range = [-5, 5]
x2Range = [-5, 5]
f_ShowLogistic2( ax, xMeshNo, x1Range, x2Range, wList2 )
```

(6.2) 最尤推定と交差ｴﾝﾄﾛﾋﾟｰ誤差

(※ この節は本ｾﾐﾅｰ(６、８回目)からの再掲を含みます)

・ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙを作成する場合、「平均交差ｴﾝﾄﾛﾋﾟｰ誤差」を最小化するように偏回帰係数を求めます。
この節では、このことについて解説します。

・統計学において、未知または既知のある確率分布の下での確率変数の出現結果から、
元々の確率分布を推測する時の「尤もらしさ（ﾓｯﾄﾓﾗｼｻ）」を表す数値を、
「尤度（ﾕｳﾄﾞ、likelihood)」または「尤度関数（ﾕｳﾄﾞｶﾝｽｳ、likelihood function)」と言います。

・この「尤度」を最大化するように元々の確率分布を推測することを
「最尤推定（ｻｲﾕｳｽｲﾃｲ、maximum likelihood estimation、MLE)」と言います。
「離散型確率分布」の場合は「確率質量関数」を、
「連続型確率分布」の場合は「確率密度関数」を推定することになります。

・「コインを投げたときに表が出るか裏が出るか」のように、何かを行ったときに起こる結果が
事象 A, B の２つ(A：成功する、B：失敗する)しかない試行のことを
「ﾍﾞﾙﾇｰｲ試行（Bernoulli trial)」といいます(本ｾﾐﾅｰ６回目でも取り上げています)。

・１回のﾍﾞﾙﾇｰｲ試行で成功する回数（0, 1 の何れか）を確率変数 Y としたとき、
これが従う確率分布が「ﾍﾞﾙﾇｰｲ分布（ﾍﾞﾙﾇｰｲﾌﾞﾝﾌﾟ、Bernoulli distribution)」です。
１回のﾍﾞﾙﾇｰｲ試行で成功する確率を p とすると、確率変数 Y=k の時の
ﾍﾞﾙﾇｰｲ分布の確率質量関数 P(Y=k) は以下の式で表されます。

ﾍﾞﾙﾇｰｲ分布の確率質量関数

$$P(Y=k) = p^k (1-p)^{1-k} \qquad \text{for } k \in \{0,1\} \qquad \cdots(式6\text{-}7)$$

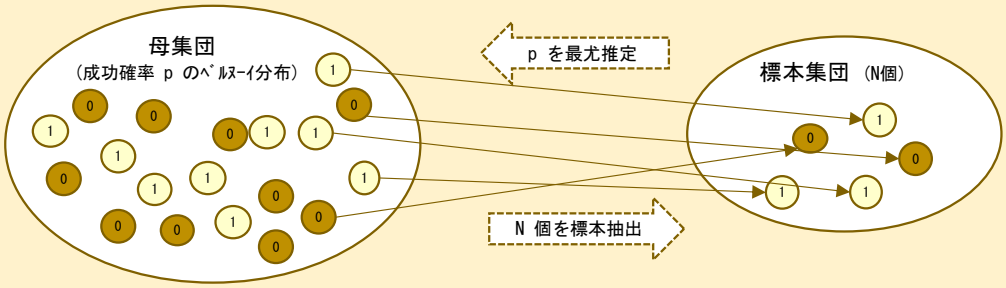
※ (式6-7)は以下の場合分けの式を
1つにまとめたもの：

$$\begin{cases} P(Y=1) = p \\ P(Y=0) = 1-p \end{cases}$$

・以下に、ﾍﾞﾙﾇｰｲ分布の最尤推定方法を示します。

ﾍﾞﾙﾇｰｲ分布の最尤推定方法

標本 $\{ Y_1, Y_2, \cdots, Y_N \}$ はある母集団からの N 個の無作為標本であるとします。
その母集団が従う確率分布がﾍﾞﾙﾇｰｲ分布で、
１回のﾍﾞﾙﾇｰｲ試行で成功する(1となる)確率を p（未知数)とします。



この時、標本 $\{ Y_1, Y_2, \cdots, Y_N \}$ の観測値 $\{ t_1, t_2, \cdots, t_N \}$ の各 t_n は、
{1, 0} の何れかを各々 {p, 1-p} の確率で取ります。
各 Y_n が値 t_n を取るというﾍﾞﾙﾇｰｲ分布の確率質量関数 $P(Y_n=t_n)$ は次式となります：

$$P(Y_n=t_n) = p^{t_n} (1-p)^{1-t_n} \qquad \text{for } t_n \in \{0,1\},\ n=1 \sim N \qquad \cdots(式6\text{-}8)$$

この式より、標本 $\{Y_1, Y_2, \cdots, Y_N\}$ の観測値が $\{t_1, t_2, \cdots, t_N\}$ となる確率
(同時確率)は、各観測結果 Y_n についての確率(式6-8)の直積で以下の式になります：

$$\begin{aligned} & P(Y_1=t_1, Y_2=t_2, \cdots, Y_N=t_N) \\ &= \prod_{n=1}^{N} P(Y_n=t_n) \\ &= \prod_{n=1}^{N} p^{t_n} (1-p)^{1-t_n} \\ &= p^z (1-p)^{N-z} \end{aligned} \quad \cdots(式6\text{-}9)$$

(… zは$t_n=1$となるﾃﾞｰﾀの個数、
$N-z$は$t_n=0$となるﾃﾞｰﾀの個数。以下同様)

これが観測値が $\{t_1, t_2, \cdots, t_N\}$ の下での
成功確率 p を推定する為の尤度関数 L となります。

$$\begin{aligned} & L(p|t_1, t_2, \cdots, t_N) \\ &= \prod_{n=1}^{N} p^{t_n} (1-p)^{1-t_n} \\ &= p^z (1-p)^{N-z} \end{aligned} \quad \cdots(式6\text{-}10)$$

尤度関数 L の対数をとったものを
「対数尤度関数 (ﾀｲｽｳﾕｳﾄﾞｶﾝｽｳ、log likelihood function)」と言い、
以下の式になります（対数関数の性質を用いて式を変形します）：

$$\begin{aligned} & \log\{ L(p|t_1, t_2, \cdots, t_N) \} \\ &= \log\{ \prod_{n=1}^{N} p^{t_n} (1-p)^{1-t_n} \} \\ &= \sum_{n=1}^{N} \{ \log p^{t_n} + \log(1-p)^{1-t_n} \} \\ &= \sum_{n=1}^{N} \{ t_n * \log p + (1-t_n) * \log(1-p) \} \\ &= z * \log p + (N-z) * \log(1-p) \\ &= \log\{p^z (1-p)^{N-z}\} \end{aligned} \quad \cdots(式6\text{-}11)$$

p の最尤推定値は、尤度関数 L を最大にする p の値です。

ここで自然対数関数 $\log(x)$ が x について単調増加であり、
関数値の大小関係が x の大小関係を保つという性質から、
尤度関数 L を最大にする p の値は、対数尤度関数の値も最大にします。

最尤推定値であることの条件として、
「p が、対数尤度関数 $\log(L)$ の p についての偏微分が 0 を与える」があります：

$$\begin{aligned} \partial \log(L)/\partial p &= \partial \log\{p^z (1-p)^{N-z}\}/\partial p \\ &= \partial \{z * \log p + (N-z) * \log(1-p)\}/\partial p \\ &= z/p - (N-z)/(1-p) \\ &= 0 \end{aligned}$$

$$\therefore \quad p = z/N$$

$$\therefore \quad p = 1/N \sum_{n=1}^{N} t_n \quad \cdots(式6\text{-}12)$$

これは、標本平均に一致しています。
例えば、標本$\{t_1, t_2, \cdots, t_N\}$ のうち、50% が 1（成功） ならば
$p=0.5$ が最尤推定値となります。

・対数尤度関数にﾏｲﾅｽ1(-1) をかけたものを
「交差ｴﾝﾄﾛﾋﾟｰ誤差 (ｺｳｻｴﾝﾄﾛﾋﾟｰｺﾞｻ、Cross Entropy Error)」と言い、この平均を取ったものを、
「平均交差ｴﾝﾄﾛﾋﾟｰ誤差 (ﾍｲｷﾝｺｳｻｴﾝﾄﾛﾋﾟｰｺﾞｻ、Mean Cross Entropy Error)」と言います。

交差ｴﾝﾄﾛﾋﾟｰ誤差、平均交差ｴﾝﾄﾛﾋﾟｰ誤差

$$\text{交差ｴﾝﾄﾛﾋﾟｰ誤差} = -\sum_{n=1}^{N} \{t_n \log(y_n) + (1-t_n)\log(1-y_n)\} \quad \cdots(式6\text{-}13)$$

$$\text{平均交差ｴﾝﾄﾛﾋﾟｰ誤差} = -1/N \sum_{n=1}^{N} \{t_n \log(y_n) + (1-t_n)\log(1-y_n)\} \quad \cdots(式6\text{-}14)$$

t_n ：N個の標本 $\{ Y_1, Y_2, \cdots, Y_N \}$ のうちの n番目の観測値
y_n ：N個の標本 $\{ Y_1, Y_2, \cdots, Y_N \}$ のうちの n番目の予測値

・ﾍﾞﾙﾇｰｲ分布の最尤推定にあたり、尤度を最大化することは対数尤度を最大化することでもあります。
更に対数尤度を最大化することは、(平均)交差ｴﾝﾄﾛﾋﾟｰ誤差を最小化することでもあります。

・ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙでは、説明変数が指定した値の時に、着目する事象が発生するかどうかに関心があり、
回帰式では、着目する事象の発生確率 y_n ($0 \leqq y_n \leqq 1$) を予測します。
実際の観測値 t_n (0 or 1)と回帰式の計算値 y_n を比較評価することによりﾓﾃﾞﾙを作成しますが、
上記のように、平均交差ｴﾝﾄﾛﾋﾟｰ誤差Ｅ(***w***) を損失関数として扱い、
これを最小にする偏回帰係数 ***w*** を求めて行います。

【出典・参考】
交差ｴﾝﾄﾛﾋﾟｰ誤差
⇒「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)

(6.3) 平均交差ｴﾝﾄﾛﾋﾟｰ誤差の偏微分

・上記のようにﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙでは、平均交差ｴﾝﾄﾛﾋﾟｰ誤差Ｅ(***w***) を損失関数として扱い、
これを最小化するように、勾配法などを用いて偏回帰係数 ***w*** を求めます。

・勾配法を適用するにあたり、平均交差ｴﾝﾄﾛﾋﾟｰ誤差Ｅ(***w***) の偏回帰係数による偏微分が必要です。
この節では、この偏微分を求める方法を述べます。

説明では、以下の記法を用います：

ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの
偏回帰係数ﾍﾞｸﾄﾙ

$$\boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$$

ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの
説明変数ﾍﾞｸﾄﾙ

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$$

観測値の組(n=1～N)の
n番目の観測値の説明変数ﾍﾞｸﾄﾙ

$$\boldsymbol{x}_n = \begin{pmatrix} 1 \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix}$$

観測値の組(n=1～N)の
目的変数の観測値ﾍﾞｸﾄﾙ

$$\boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

観測値の組(n=1～N)の
目的変数の回帰計算結果ﾍﾞｸﾄﾙ

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

- y ：はﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの目的変数
- x_d ：(d=0～D) はﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙのd番目の説明変数、但しx_0=1
- w_d ：(d=0～D) はﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの説明変数x_d にかかる偏回帰係数、w_0 は定数項
- t_n ：(n=1～N) はn番目の観測値の組での目的変数y の観測値 ($t_n \in \{0,1\}$, 教師ﾃﾞｰﾀ)
- y_n ：(n=1～N) はn番目の観測値の組についてのﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの計算値
- $x_{n,d}$ ：(n=1～N) はn番目の観測値の組でのd番目(d=0～D)の説明変数x_d の値、但し$x_{n,0}$=1

まず、平均交差ｴﾝﾄﾛﾋﾟｰ誤差Ｅ(***w***)を、
n番目(n=1～N)の観測値の組の交差ｴﾝﾄﾛﾋﾟｰ誤差E_n(***w***)の総和として表します：

$$E(\boldsymbol{w}) = -1/N \sum_{n=1}^{N} \{t_n \log(y_n) + (1-t_n)\log(1-y_n)\} \quad \cdots(式6\text{-}14)$$

$$= 1/N \sum_{n=1}^{N} E_n(\boldsymbol{w}) \quad \cdots(式6\text{-}15)$$

ここで $E_n(\boldsymbol{w}) = -t_n \log(y_n) - (1-t_n)\log(1-y_n)$ …(式6-16)

ﾛｼﾞｯﾄ関数(式6-6)の右辺を入力総和 a(***w***) と呼ぶことにします。

入力総和 $a(\boldsymbol{w}) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D$ …(式6-17)

この入力総和 a(***w***) を用いると、ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ(式6-1)は、 a(***w***)のｼｸﾞﾓｲﾄﾞ関数σになります：

$$y = 1 / \{ 1 + \exp(-(w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D)) \} \quad \cdots(式6\text{-}1)$$

$$\therefore \quad y = 1 / \{ 1 + \exp(-a(\boldsymbol{w})) \} \quad \cdots(式6\text{-}18)$$

$$\therefore \quad y = \sigma(a(\boldsymbol{w})) \quad \cdots(式6\text{-}19)$$

n番目(n=1～N)の観測値の組の入力総和 a(**w**) を $a_n(\boldsymbol{w})$ で、
ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの計算値 y を y_n で表すと、

$$a_n(\boldsymbol{w}) = w_0 + w_1x_{n,1} + w_2x_{n,2} + \cdots + w_Dx_{n,D} \quad \cdots(式6\text{-}17')$$

$$y_n = \sigma(a_n(\boldsymbol{w})) \quad \cdots(式6\text{-}19')$$

n番目(n=1～N)の観測値の組の交差ｴﾝﾄﾛﾋﾟｰ誤差 $E_n(\boldsymbol{w})$ は、y_n の関数(式6-16)で、
更に y_n は $a_n(\boldsymbol{w})$ の関数(式6-19')なので、以下のような合成関数になります：

$$E_n(\boldsymbol{w}) = E_n(y_n(a_n(\boldsymbol{w}))) \quad \cdots(式6\text{-}20)$$

従って、合成関数の偏微分の連鎖律により、

$$\partial E_n(\boldsymbol{w})/\partial w_d = \partial E_n/\partial y_n \cdot \partial y_n/\partial a_n \,.\, \partial a_n/\partial w_d \quad (d=0\sim D) \quad \cdots(式6\text{-}21)$$

(式6-21)の右辺の第１番目の部分の偏微分は、(式6-16)より

$$\partial E_n/\partial y_n = \partial\{-t_n\log(y_n) - (1-t_n)\log(1-y_n)\}/\partial y_n$$

$$\therefore \quad = -t_n/y_n + (1-t_n)/(1-y_n) \quad \cdots(式6\text{-}22)$$

(式6-21)の右辺の第２番目の部分の偏微分は、(式6-19')とｼｸﾞﾓｲﾄﾞ関数の微分の公式より

$$\partial y_n/\partial a_n = \partial\sigma(a_n)/\partial a_n$$

$$= \sigma(a_n)\{1 - \sigma(a_n)\}$$

$$\therefore \quad = y_n(1 - y_n) \quad \cdots(式6\text{-}23)$$

(式6-21)の右辺の第３番目の部分の偏微分は、(式6-17')より

$$\partial a_n/\partial w_d = x_{n,d} \quad (d=0\sim D,\ x_{n,0}=1) \quad \cdots(式6\text{-}24)$$

以上により、n番目(n=1～N)の観測値の組の交差ｴﾝﾄﾛﾋﾟｰ誤差 $E_n(\boldsymbol{w})$ の、偏回帰係数 w_d による偏微分は

$$\partial E_n(\boldsymbol{w})/\partial w_d = \{-t_n/y_n + (1-t_n)/(1-y_n)\} \cdot \{y_n(1 - y_n)\} \cdot x_{n,d}$$

$$\therefore \quad = (y_n - t_n) \cdot x_{n,d} \quad \cdots(式6\text{-}25)$$

・以上をまとめると、平均交差ｴﾝﾄﾛﾋﾟｰ誤差 $E(\boldsymbol{w})$ の偏回帰係数 w_d による偏微分は次式により与えられます：

平均交差ｴﾝﾄﾛﾋﾟｰ誤差 $E(\boldsymbol{w})$ の偏回帰係数 w_d による偏微分

ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ：

$$y = 1 / \{1 + \exp(-(w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D))\} \quad \cdots(式6\text{-}1)$$

について、
その平均交差ｴﾝﾄﾛﾋﾟｰ誤差 $E(\boldsymbol{w})$ の偏回帰係数 w_d による偏微分は次式で与えられる：

$$\partial E(\boldsymbol{w})/\partial w_d = 1/N \sum_{n=1}^{N} (y_n - t_n) \cdot x_{n,d} \quad \cdots(式6\text{-}26)$$

- w_d : (d=0～D) はﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの説明変数x_d にかかる偏回帰係数、w_0 は定数項
- t_n : (n=1～N) はn番目の観測値の組での目的変数y の観測値 ($t_n \in \{0,1\}$, 教師ﾃﾞｰﾀ)
- y_n : (n=1～N) はn番目の観測値の組についてのﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの計算値
- $x_{n,d}$: (n=1～N) はn番目の観測値の組でのd番目(d=0～D)の説明変数x_d の値、但し$x_{n,0}=1$

【出典・参考】

「Pythonで動かして学ぶ！あたらしい機械学習の教科書」(2018年01月 翔泳社 伊藤真著)

(6.4) ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙの作成

・上記のようにﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙでは、平均交差ｴﾝﾄﾛﾋﾟｰ誤差Ｅ(***w***) を損失関数として扱います。
これを最小にする偏回帰係数 ***w*** の数値解は、勾配法を用いて求めます。

・勾配法のｱﾙｺﾞﾘｽﾞﾑと実装については、「(3.2) 線形回帰 (数値解)」などで既述の為、省略します。
「(3.2) 線形回帰 (数値解)」との違いは、
線形回帰ﾓﾃﾞﾙでは、平均二乗誤差Ｊ(***w***)を損失関数として扱っていたのに対し、
ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙでは、平均交差ｴﾝﾄﾛﾋﾟｰ誤差Ｅ(***w***) を損失関数として扱う点です。
また、使用する偏微分の計算は (6.3) で述べたとおりです。

(7) まとめ

・今回は回帰ﾓﾃﾞﾙについて解説してきましたが、「(5.4) 回帰ﾓﾃﾞﾙとその適合度の例」でも見たように、
ﾓﾃﾞﾙとしてどれを採用するかについては、単に学習ﾃﾞｰﾀの適合度が高いというだけでは不十分で、
未知のﾃﾞｰﾀに対しても適用に耐えられるﾓﾃﾞﾙにするために、
経験やｱｲﾃﾞｱや科学的論拠に基づいた知見を取入れての判断が必要になります。
実際の問題に適用するにあたり、このような注意が必要です。

・線形回帰の実例として「秋田」における
1883年～2020年 の約 140 年間にわたる 2月と8月の「日平均気温の月平均値」の変化を取り上げて
みましたが、この間に回帰直線(1次元線形回帰ﾓﾃﾞﾙ)でみて(P.8)、2月で約＋3.0度、8月で約＋1.2度程度
気温上昇が起こっていることが読み取れます。温暖化の速さに驚きと不安を感じます。

以上。

(8) 確認問題

(1) 回帰ﾓﾃﾞﾙとは？

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・(1.1)　　を取る説明変数と、(1.2)　　を取る目的変数の間に関数関係のﾓﾃﾞﾙを当てはめることを
「回帰（ｶｲｷ, Regression)」と呼び、当てはめるﾓﾃﾞﾙ式を
「回帰方程式または回帰式（ｶｲｷｼｷ、Regression equation)」と呼びます。

・「回帰方程式」は、観測ﾃﾞｰﾀをもとに入力ﾃﾞｰﾀ（説明変数）と出力ﾃﾞｰﾀ（目的変数）の数値間の関係性
から統計的手法によって推計します。
これを「回帰分析（ｶｲｷﾌﾞﾝｾｷ, Regression analysis)」と呼びます。
説明変数が1次元ならば (1.3)　　、
2次元以上ならば (1.4)　　と言います。

・「回帰式」は、方程式を解くことにより厳密な解を求めることができる場合があり、
このような解を(1.5)　　と呼びます。
方程式を解くことができない場合、勾配法などの数値計算により近似的な解を求めます。
このような解を(1.6)　　と呼びます。

・解析解の場合は、数式で導出されるため計算ｺｽﾄは少なくて済みますが、
数値解の場合は、学習係数の設定や収束判定などの処理が入り、計算ｺｽﾄと精度で検討が必要になります。

（選択肢）

(a)　「離散値（ﾘｻﾝﾁ、Discrete value)」
(b)　「解析解（ｶｲｾｷｶｲ、Analytical solution)」
(c)　「重回帰分析（ｼﾞｭｳｶｲｷﾌﾞﾝｾｷ、Multiple regression analysis)」
(d)　「数値解（ｽｳﾁｶｲ、Numerical solution)」
(e)　「単回帰分析（ﾀﾝｶｲｷﾌﾞﾝｾｷ、Single regression analysis)」
(f)　「連続値（ﾚﾝｿﾞｸﾁ、Continuous value)」

（回答）

(1.1)	
(1.2)	
(1.3)	
(1.4)	
(1.5)	
(1.6)	

(2) 線形回帰ﾓﾃﾞﾙ

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・連続確率変数の目的変数 y に対し、これを説明するD個の説明変数の組 $\boldsymbol{x}$

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_D)$$

があり、目的変数 y が、D個の説明変数の線形結合として表現できる場合、
この回帰ﾓﾃﾞﾙを (2.1) と言います。

$$y(\boldsymbol{x}) = w_0+w_1x_1+w_2x_2+ \cdots +w_Dx_D = \sum_{d=0}^{D} w_dx_d \qquad (但し、x_0 = 1)$$

係数 w_d {d=1～D} を (2.2) と言い、
説明変数 x_i {i≠d, i=1～D}を一定にして、説明変数 x_d {d=1～D} だけを１単位変化させたときの
目的変数 y の変化量を表します。

・D次元線形回帰ﾓﾃﾞﾙは、偏回帰係数 w_d {d=1～D} と定数項 w_0 を求めることにより決定します。

$$\boldsymbol{w} = (w_0, w_1, w_2, \cdots, w_D)$$

（選択肢）

(a) 「D次元線形回帰ﾓﾃﾞﾙ (D-dimension linear regression model)」
(b) 「偏回帰係数 (ﾍﾝｶｲｷｹｲｽｳ, Partial regression coefficient)」

（回答）

(2.1)	
(2.2)	

(3) 線形回帰ﾓﾃﾞﾙの解析解と数値解

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・線形回帰ﾓﾃﾞﾙでは、偏回帰係数 w_d {d=1～次元数D} と定数項 w_0 を決定することが、
ﾓﾃﾞﾙを解くことに相当します。

$$y(\boldsymbol{x}) = w_0+w_1x_1+w_2x_2+ \cdots +w_Dx_D = \sum_{d=0}^{D} w_dx_d$$

(x_d : 説明変数 d=1～次元数D、x_0=1)

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_D)$$

$$\boldsymbol{w} = (w_0, w_1, w_2, \cdots, w_D)$$

・線形回帰ﾓﾃﾞﾙを解くにあたり、
「数値解」と「解析解」の何れも (3.1) J($\boldsymbol{w}$)を最小にする $\boldsymbol{w}$ を求めるのが目標です。

$$J(\boldsymbol{w}) = (1/N) \times \sum_{n=1}^{N} (y(\boldsymbol{x}_n) - t_n)^2$$

- N : 観測値の個数
- $\boldsymbol{x}_n$: n番目の観測値の組(n=1～N) での説明変数 $\boldsymbol{x}$
- t_n : n番目の観測値の組(n=1～N) での目的変数 y の観測値

・線形回帰ﾓﾃﾞﾙの場合、方程式を解くことにより厳密な「解析解」を求めることができます。
その一つとして (3.2) を用いる方法があります。

線形回帰ﾓﾃﾞﾙの解析解の導出

$$y(\boldsymbol{x}) = ((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t})^T \boldsymbol{x}$$

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \quad \boldsymbol{X} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{pmatrix} \quad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

- $y(\boldsymbol{x})$ は説明変数 $\boldsymbol{x}$ に対するモデルの計算値
- x_d (d=1～D) は、d番目の説明変数
- $x_{n,d}$ (d=1～D) は、観測値の組 No.n (n=1～N) でのd番目の説明変数 x_d の値
- t_n は、観測値の組 No.n での目的変数 y の観測値（教師ﾃﾞｰﾀ）
- $(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$ は「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」

・線形回帰ﾓﾃﾞﾙの数値解の導出に (3.3) を用いる方法があります。

線形回帰ﾓﾃﾞﾙの数値解の導出（勾配降下法）

(1) tを繰返しのｽﾃｯﾌﾟ数として、t=0 から開始する。

(2) 学習係数η (>0) を与える(※1)。

(3) 偏回帰係数 $\boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix}$ の初期値 $\boldsymbol{w}(0) = \begin{pmatrix} w_{0,0} \\ w_{0,1} \\ w_{0,2} \\ \vdots \\ w_{0,D} \end{pmatrix}$ を任意に与える。

(4) 以下の様なモデルの最適判定を行い、計算の収束判定を行う：

(4.1) 偏回帰係数 $\boldsymbol{w}(t)$ での平均二乗誤差 $J(\boldsymbol{w}(t))$ を計算し、その値が十分小さければ、
本処理を終了する。偏回帰係数 $\boldsymbol{w}(t)$ がﾓﾃﾞﾙの最適値となる。

$$J(\boldsymbol{w}(t)) = (1/N)\sum_{n=1}^{N} (\boldsymbol{w}(t)^T \boldsymbol{x}_n - t_n)^2$$

(4.2) 上記の平均二乗誤差 $J(\boldsymbol{w}(t))$ の減少が殆どなく、計算が収束したとみなせる場合、
本処理を終了する。偏回帰係数 $\boldsymbol{w}(t)$ がﾓﾃﾞﾙの最適値となる。

(4.3) 繰り返し回数が制限を超過した場合、本処理を終了する。
この時は 最小値を与える点は見つかっておらず、モデル等の見直しを行う必要あり。

(4.4) 上記(4.1)～(4.3)以外の場合、処理(5)の偏回帰係数の更新処理へ。

(5) 偏回帰係数 $\boldsymbol{w}$ について $\boldsymbol{w}(t)$ からの移動量ベクトル $\Delta\boldsymbol{w}$ を次式により計算して $\boldsymbol{w}(t+1)$ へ更新する。

$$\Delta\boldsymbol{w} = -\eta \nabla J$$

ここで、∇J は平均二乗誤差 J の $\boldsymbol{w}(t)$ における勾配ベクトル：

$$\nabla J = \left(\partial J/\partial w_0,\ \partial J/\partial w_1,\ \cdots,\ \partial J/\partial w_D\right) \Big|_{\boldsymbol{w}(t)}$$

∴

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \eta \nabla J$$

$$\begin{pmatrix} w_{t+1,0} \\ w_{t+1,1} \\ w_{t+1,2} \\ \vdots \\ w_{t+1,D} \end{pmatrix} = \begin{pmatrix} w_{t,0} \\ w_{t,1} \\ w_{t,2} \\ \vdots \\ w_{t,D} \end{pmatrix} - \eta \left.\begin{pmatrix} \partial J/\partial w_0 \\ \partial J/\partial w_1 \\ \partial J/\partial w_2 \\ \vdots \\ \partial J/\partial w_D \end{pmatrix}\right|_{\boldsymbol{w}(t)}$$

ここで、平均二乗誤差 J の偏回帰係数 w_d (d=0～D:偏回帰係数の組) による偏微分は

$$\partial J/\partial w_d = \partial (1/N)\sum_{n=1}^{N} (\boldsymbol{w}(t)^T \boldsymbol{x}_n - t_n)^2 / \partial w_d \qquad (x_{n,0}=1)$$

$$= (2/N)\sum_{n=1}^{N} (\boldsymbol{w}(t)^T \boldsymbol{x}_n - t_n)\, x_{n,d} \qquad (n=1\sim N：観測値の組)$$

$$= (2/N)\sum_{n=1}^{N} (y(\boldsymbol{x}_n) - t_n)\, x_{n,d}$$

(6) 繰返しのｽﾃｯﾌﾟ数 t ← t+1 として手順(4)へ

(※1) 学習係数 η を、手順(4)～(6)の過程で動的に決定する方法もあります。

（選択肢）

(a) 「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」
(b) 「勾配降下法」
(c) 「平均二乗誤差」
(d) 「交差ｴﾝﾄﾛﾋﾟｰ誤差」

（回答）

(3.1)	
(3.2)	
(3.3)	

(4) 線形基底関数ﾓﾃﾞﾙ

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・回帰ﾓﾃﾞﾙのうち「D次元線形回帰ﾓﾃﾞﾙ」では表現力が乏しいため、
説明変数 $\boldsymbol{x} = (x_d)$ {d=1～D：説明変数の次元数} の線形結合の代わりに、
(4.1) と呼ぶ説明変数 $\boldsymbol{x}$ の関数 $\boldsymbol{\Phi}_m(\boldsymbol{x})$ {m=1～M：関数の数} の組
を用いて、その線形結合として表現するモデルがあります。
それを (4.2) と呼びます。

線形基底関数ﾓﾃﾞﾙ（定式化）

$$y(\boldsymbol{x}) = w_0 + w_1 * \boldsymbol{\Phi}_1(\boldsymbol{x}) + w_2 * \boldsymbol{\Phi}_2(\boldsymbol{x}) + \cdots + w_M * \boldsymbol{\Phi}_M(\boldsymbol{x})$$
$$= \sum_{m=0}^{M} w_m * \boldsymbol{\Phi}_m(\boldsymbol{x})$$
$$= \boldsymbol{w}^T \boldsymbol{\Phi}(\boldsymbol{x})$$

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \quad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_M \end{pmatrix} \quad \boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\Phi}_0 \\ \boldsymbol{\Phi}_1 \\ \boldsymbol{\Phi}_2 \\ \vdots \\ \boldsymbol{\Phi}_M \end{pmatrix}$$

但し $\boldsymbol{\Phi}_0(\boldsymbol{x}) = 1$
D：説明変数の次元数
M：基底関数の数

・「線形基底関数ﾓﾃﾞﾙ」は、「ﾑｰｱ・ﾍﾟﾝﾛｰｽﾞの擬似逆行列」を用いて解析解を得ることができます。

線形基底関数ﾓﾃﾞﾙ（解析解）

$$y(\boldsymbol{x}) = \left((\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \right)^T \boldsymbol{x}$$

$$\boldsymbol{\Phi} = \begin{pmatrix} 1 & \boldsymbol{\Phi}_1(\boldsymbol{x}_1) & \boldsymbol{\Phi}_2(\boldsymbol{x}_1) & \cdots & \boldsymbol{\Phi}_M(\boldsymbol{x}_1) \\ 1 & \boldsymbol{\Phi}_1(\boldsymbol{x}_2) & \boldsymbol{\Phi}_2(\boldsymbol{x}_2) & \cdots & \boldsymbol{\Phi}_M(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \boldsymbol{\Phi}_1(\boldsymbol{x}_N) & \boldsymbol{\Phi}_2(\boldsymbol{x}_N) & \cdots & \boldsymbol{\Phi}_M(\boldsymbol{x}_N) \end{pmatrix} \quad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix} \quad \boldsymbol{x}_n = \begin{pmatrix} x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix}$$

- $y(\boldsymbol{x})$ は説明変数 $\boldsymbol{x}$ に対するモデルの計算値
- $\boldsymbol{x}_n$ (n=1～N) は、N個の観測値の組での n番目の説明変数の組
- $x_{n,d}$ (d=1～D) は、$\boldsymbol{x}_n$ での d番目の説明変数の値
- t_n は、N個の観測値の組での n番目の目的変数 y の値（教師ﾃﾞｰﾀ）
- $\boldsymbol{\Phi}_m$ (m=0～M) は M個の基底関数のうちの m番目の基底関数、但し $\boldsymbol{\Phi}_0=1$

・以下に基底関数ﾓﾃﾞﾙの例を示します：

ﾓﾃﾞﾙ名	基底関数 $\boldsymbol{\Phi}_m(\boldsymbol{x})$	m=1～M：基底関数の数
(4.3)	x_m	M：説明変数の次元の数D
(4.4)	x^m	
(4.5)	$\exp\{ -(x - \mu_m) / (2\sigma_m^2) \}$	μ_m：m番目のｶﾞｳｽ分布関数での平均 σ_m：m番目のｶﾞｳｽ分布関数での標準偏差

（選択肢）

(a) 「線形基底関数ﾓﾃﾞﾙ（ｾﾝｹｲｷﾃｲｶﾝｽｳﾓﾃﾞﾙ、linear basis function model)」
(b) 「D次元線形回帰ﾓﾃﾞﾙ（D-dimension linear regression model)」
(c) 「ガウス基底関数（ｶﾞｳｽｷﾃｲｶﾝｽｳ、Gaussian basis functions)」
(d) 「多項式基底関数（ﾀｺｳｼｷｷﾃｲｶﾝｽｳ、Polynomial basis functions)」
(e) 「基底関数（ｷﾃｲｶﾝｽｳ、basis function)」

（回答）

(4.1)	
(4.2)	
(4.3)	
(4.4)	
(4.5)	

(5) 適合度

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・回帰ﾓﾃﾞﾙの適合度の尺度として、

(5.1) _____ と、その平方根を取った

「二乗平均平方根誤差（ﾆｼﾞｮｳﾍｲｷﾝﾍｲﾎｳｺﾝｺﾞｻ、Root Mean Squared Error、RMSE)」があります。

・観測値t_nと回帰値y_nとの差e_n (= $t_n - y_n$)を「残差（ｻﾞﾝｻ、residual)」といいます。

「平均二乗誤差（MSE)」は、「残差」の二乗の平均値のことです。

これは、元のﾃﾞｰﾀを二乗した次元の値となっています。

平均二乗誤差（MSE)

$$MSE = (1/N) \sum_{n=1}^{N} (t_n - y_n)^2 = (1/N) \sum_{n=1}^{N} e_n^2$$

・「平均二乗誤差（MSE)」の平方根をとったものが「二乗平均平方根誤差（RMSE)」で、

元のﾃﾞｰﾀと同じ次元の値です。

二乗平均平方根誤差（RMSE)

$$RMSE = \sqrt{MSE} = \sqrt{(1/N) \sum_{n=1}^{N} (t_n - y_n)^2}$$

・回帰ﾓﾃﾞﾙの適合度の別の尺度として、

(5.2) _____ R^2（ｱｰﾙﾂｰ）があります。

これは、回帰値の変動が全変動に占める割合を示しています。

決定係数（R^2)

R^2 ＝ ESS／TSS

＝ 1 － RSS／TSS

ESS（回帰値 y_n の平均周りの変動、Explained Sum of squares)

$ESS = \sum_{n=1}^{N} (y_n - t_{mean})^2$

これは、回帰値y_nの、観測値の平均t_{mean}周りの変動です。

TSS（全変動、ｾﾞﾝﾍﾝﾄﾞｳ、Total Sum of squares)

$TSS = \sum_{n=1}^{N} (t_n - t_{mean})^2$

これは、観測値t_nの、観測値の平均t_{mean}周りの変動です。

RSS（残差変動(ｻﾞﾝｻﾍﾝﾄﾞｳ)、残差平方和(ｻﾞﾝｻﾍｲﾎｳﾜ)、Residual Sum of squares)

$RSS = \sum_{n=1}^{N} (t_n - y_n)^2 = \sum_{n=1}^{N} e_n^2$

これは、回帰値y_nと観測値t_nの残差($e_n = t_n - y_n$)平方和です。

（選択肢）

(a) 「二乗平均平方根誤差（ﾆｼﾞｮｳﾍｲｷﾝﾍｲﾎｳｺﾝｺﾞｻ、Root Mean Squared Error、RMSE)」

(b) 「平均交差ｴﾝﾄﾛﾋﾟｰ誤差（ﾍｲｷﾝｺｳｻｴﾝﾄﾛﾋﾟｰｺﾞｻ、Mean Cross Entropy Error)」

(c) 「平均二乗誤差（ﾍｲｷﾝﾆｼﾞｮｳｺﾞｻ、Mean Squared Error、MSE)」

(d) 「決定係数（ｹｯﾃｲｹｲｽｳ、coefficient of determination)」

(e) 「交差ｴﾝﾄﾛﾋﾟｰ誤差（ｺｳｻｴﾝﾄﾛﾋﾟｰｺﾞｻ、Cross Entropy Error)」

（回答）

(5.1)	
(5.2)	

(6) ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ

以下の空欄に最もあてはまる語句を選択肢から選び、その記号を回答欄に記入してください。

・二値（成功/失敗、True/False、Yes/No、といった2種類の値を取る）の目的変数 y と
D個の説明変数 x_d (d=1～D)との関数関係を解析する手法が
「ﾛｼﾞｽﾃｨｯｸ回帰分析（ﾛｼﾞｽﾃｨｯｸｶｲｷﾌﾞﾝｾｷ, Logistic Regression Analysys)」で、その数式ﾓﾃﾞﾙを
(6.1)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿と言います。
「ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ」で目的変数 y は、着目する事象の発生確率 p（0≦p≦1）を表します。

・二値を目的変数とすることから判るように、「ﾛｼﾞｽﾃｨｯｸ回帰」は回帰ではなく分類の為のﾓﾃﾞﾙです。

ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ

$$y = 1 / \{ 1 + \exp(-(w_0 + w_1x_1 + w_2x_2 + \cdots + w_Dx_D)) \}$$

- x_d ：d番目の説明変数 {d=1～D} （x_0=1）
- w_d ：d番目の説明変数にかかる偏回帰係数 {d=1～D} （w_0 は定数項）

・ﾓﾃﾞﾙの最適度を示す対数尤度関数にﾏｲﾅｽ1(-1) をかけたものを、
(6.2)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿と言い、この平均を取ったものを、
(6.3)＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿と言います。

交差ｴﾝﾄﾛﾋﾟｰ誤差、平均交差ｴﾝﾄﾛﾋﾟｰ誤差

$$\text{交差ｴﾝﾄﾛﾋﾟｰ誤差} = -\sum_{n=1}^{N} \{t_n \log(y_n) + (1-t_n)\log(1-y_n)\} \quad \cdots(式6\text{-}2\text{-}⑦)$$

$$\text{平均交差ｴﾝﾄﾛﾋﾟｰ誤差} = -1/N \sum_{n=1}^{N} \{t_n \log(y_n) + (1-t_n)\log(1-y_n)\} \quad \cdots(式6\text{-}2\text{-}⑧)$$

- t_n ：N個の標本 { $Y_1, Y_2, \cdots, Y_N$ } のうちの n番目の観測値
- y_n ：N個の標本 { $Y_1, Y_2, \cdots, Y_N$ } のうちの n番目の予測値

・ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙを作成する場合、平均交差ｴﾝﾄﾛﾋﾟｰ誤差Ｅ(***w***) を(6.4)＿＿＿＿＿＿＿＿として扱い、
これを最小化するように、勾配法などで偏回帰係数 ***w*** を求めます。

（選択肢）

(a) 「ﾛｼﾞｽﾃｨｯｸ回帰ﾓﾃﾞﾙ（ﾛｼﾞｽﾃｨｯｸｶｲｷﾓﾃﾞﾙ, Logistic Regression Model)」
(b) 「交差ｴﾝﾄﾛﾋﾟｰ誤差（ｺｳｻｴﾝﾄﾛﾋﾟｰｺﾞｻ、Cross Entropy Error)」
(c) 「二乗平均平方根誤差（ﾆｼﾞｮｳﾍｲｷﾝﾍｲﾎｳｺﾝｺﾞｻ、Root Mean Squared Error、RMSE)」
(d) 「平均交差ｴﾝﾄﾛﾋﾟｰ誤差（ﾍｲｷﾝｺｳｻｴﾝﾄﾛﾋﾟｰｺﾞｻ、Mean Cross Entropy Error)」
(e) 「平均二乗誤差（ﾍｲｷﾝﾆｼﾞｮｳｺﾞｻ、Mean Squared Error、MSE)」
(f) 「損失関数（ｿﾝｼﾂｶﾝｽｳ、loss function)」
(g) 「活性化関数（ｶｯｾｲｶｶﾝｽｳ、activation function)」

（回答）

(6.1)	
(6.2)	
(6.3)	
(6.4)	

(9) 確認問題回答用紙

提出者　：

提出日　：　　年　　月　　日

回答　（全 22 問）

No.	回答
(1.1)	
(1.2)	
(1.3)	
(1.4)	
(1.5)	
(1.6)	
(2.1)	
(2.2)	
(3.1)	
(3.2)	
(3.3)	
(4.1)	
(4.2)	
(4.3)	
(4.4)	
(4.5)	

No.	回答
(5.1)	
(5.2)	
(6.1)	
(6.2)	
(6.3)	
(6.4)	

（※黄色の枠　のみ記入をお願いします）

※ ご意見・ご要望など有りましたら、下欄に記してください。