

データ分析（1）

《学修項目》

- ◎回帰分析
- ロジスティック分析
- 時系列分析

《キーワード》

説明変数との関係、予測モデル、最小二乗法、単回帰分析、重回帰分析、次数選択、観測値の非線形変換、ロジット変換、離散変数、尤度、最尤推定量、周期的変動、パワースペクトル、移動平均、季節調整法、自己相関関数、ARモデル

《参考文献、参考書籍》

- [1] 東京大学MIセンター公開教材 「1-4 データ分析」 《利用条件CC BY-NC-SA》
- [2] 応用基礎としてのデータサイエンス（講談社 データサイエンス入門シリーズ）
- [3] データサイエンスの考え方 社会に役立つAI×データ活用のために（オーム社）
- [4] Pythonによるあたらしいデータ分析の教科書 第2版（翔泳社）
- [5] 数理・データサイエンス・AI公開講座（放送大学）

1. 線形回帰分析（量的目的変数）

1.1 説明変数との関係 [1]

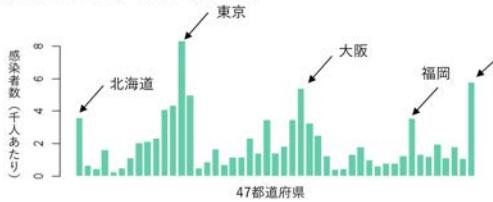
日本の47都道府県の人口千人あたりの新型コロナウイルスの累積感染者数（2021年4月当時）のデータから、棒グラフによる可視化を行ってみると、都道府県によって感染者の割合が大きく異なることが分かる。

都道府県による感染者の違いを各県の県庁所在地の人口密度と関連づけて説明することを考える（仮説）。

各県の県庁所在地の人口密度を横軸に、千人当たりの感染者数を縦軸にとって散布図を描くと、人口密度と感染者数には明確な関係があることが分かる[1]。

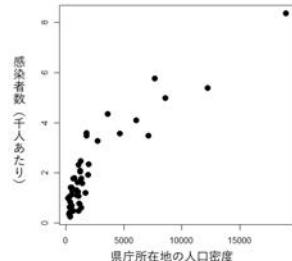
説明変数との関係を見る

以下の棒グラフは左から北海道、青森、……、沖縄の順に47都道府県の人口千人あたりの新型コロナウイルスの累積感染者数を示しています。都道府県によって感染した人の割合が大きく異なることがわかります。



このような都道府県（以下では単に県と書きます）による感染者の違いを各県の県庁所在地の人口密度と関連づけて説明することを考えてみましょう。

実際に、各县の県庁所在地の人口密度を横軸に、千人当たりの感染者数を縦軸にとって散布図を描いてみると、右のように人口密度と感染者数には明確な関係がみえます。



東京大学 数理・情報教育研究センター 北川源四郎 2021 CC BY-NC-SA

5

Pythonによる散布図プロットの実装例

日本の47都道府県の人口千人あたりの新型コロナウイルスの累積感染者数（2021年4月当時）のデータ

- オリジナルデータとプログラム例（公開教材[1]ただし実装はR）

```
In [ ]: # CSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする。
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DS/main/4/
# wgetしなくとも,Google colab.の左メニュー【ファイル】アイコンをクリックして,ブラウザへファイルを
# ファイル (udon.csv) がダウンロード・配置できたことを確認する
!ls -al ./
```

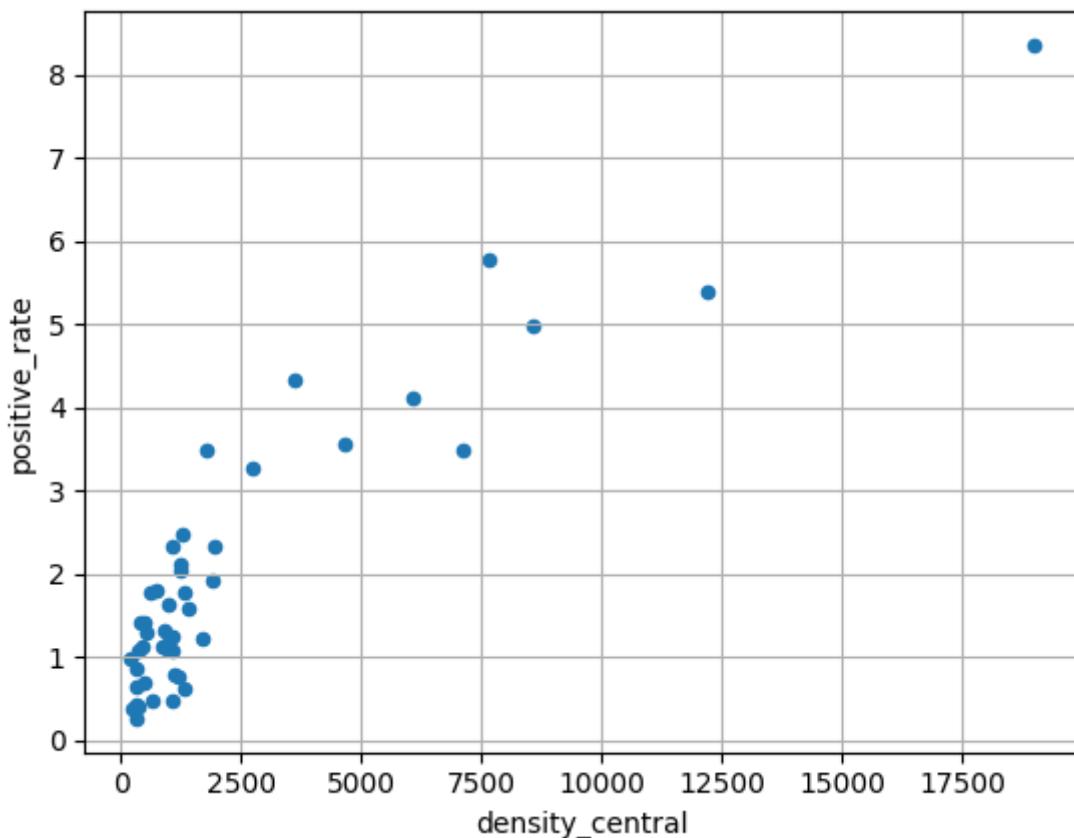
```
In [3]: # 全ての警告メッセージを非表示(オプション)
import warnings
warnings.simplefilter('ignore')
```

```
In [4]: # オリジナルのCSVファイルをpandasで読み込んでデータフレームdfに格納
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Tokyo_Covid-19_regression.csv', header=1)
df.columns = ['prefecture', 'positive_rate', 'population', 'density', 'aging_rate']
df.head()
```

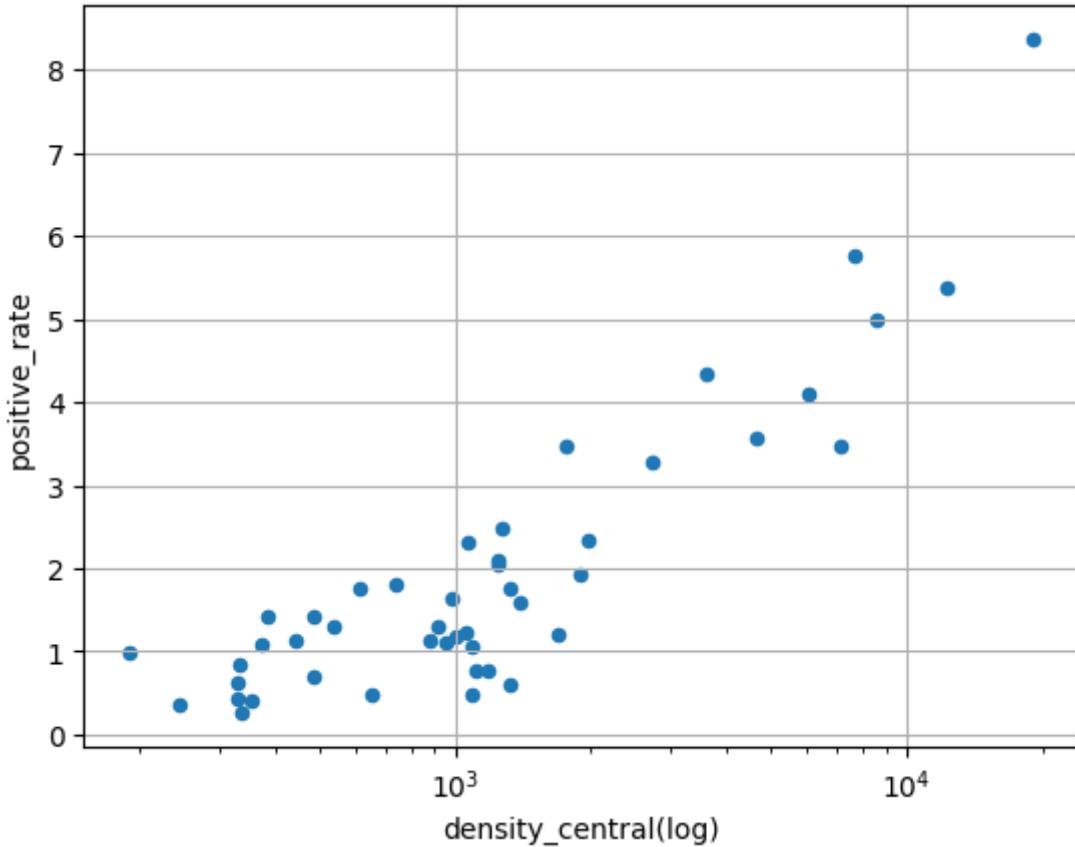
Out[4]:

	prefecture	positive_rate	population	density	aging_rate	density_central	2hours	4hour
0	青森県	0.63	1308	135.6	30.15	331.16	0.59	6.5
1	岩手県	0.43	1280	83.8	30.41	328.63	1.53	13.2
2	宮城県	1.59	2334	320.5	25.74	1388.68	1.79	15.3
3	秋田県	0.26	1023	87.9	33.92	335.55	0.58	5.7
4	山形県	0.48	1124	120.5	30.84	650.37	0.77	7.1

In [5]: # 散布図 X軸:各県の県庁所在地の人口密度(線形) Y軸:千人当たりの感染者数
df.plot(kind='scatter', grid=True, x=df.columns[5], y=df.columns[1])
plt.show()
右上で突出している1点は、東京都である。



In [6]: # 散布図 X軸:各県の県庁所在地の人口密度(対数) Y軸:千人当たりの感染者数
df.plot(kind='scatter', grid=True, logx=True, x=df.columns[5], y=df.columns[1])
plt.xlabel("density_central(log)")
plt.show()
対数人口密度と、コロナ感染率は、なんなく比例している感じがする(東京都を除く)



1.2 単回帰分析 [2]

コロナの感染率のように、値の変動の様子に興味がある変数 y と、人口密度のように y の変動の要因と考えられる変数 x があると仮定する。このとき、 y を**被説明変数（従属変数）**、 x を**説明変数**という。

ここで、 a, b を定数として、 x_i が与えられたときに y_i の値が

$$y_i = a + bx_i + \varepsilon_i$$

という法則（モデル）に従って決まると思定してみる。ここで、 ε_i は**誤差項**と呼ばれる。このような表現を**単回帰モデル**(simple regression model)といい、単回帰モデルを用いて y と x の関係を考察する分析を**単回帰分析** (simple regression analysis) という。

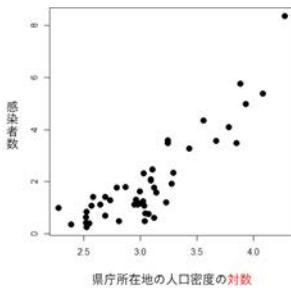
単回帰モデルでは、 y の変動の主要な決定要因は x であり、 y と x との関係は誤差項を除けば直線 $y = a + bx$ で記述（予測）できると想定する。この直線のことを**回帰直線（回帰式）** という。回帰直線の切片 a と傾き b は**回帰係数**と呼ばれる[2]。

単回帰分析

そこで、人口密度の情報を利用することによって、感染者数の値を説明したり、予測したりすることが考えられます。

右の散布図において、注目する変数を縦軸に表し、**目的変数**あるいは出力と呼びます。一方、横軸には**説明変数**あるいは入力を表します。

【注意】前頁の散布図と違って、右図では人口密度の対数を取って表示しています。このような変換によって二つの変数の関係が直線に近くなり、関係をとらえやすくなることがあることに注意すべきです。



単回帰分析では、目的変数（出力） y の値を

$$y = a + bx$$

のようにひとつの説明変数（入力） x を用いて**回帰直線**で表します。

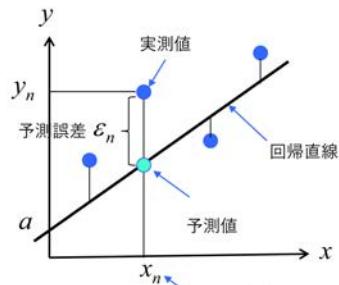
回帰モデル

ただし、実際のデータではすべての点がこの直線上にのるわけではありません。そこで回帰直線と説明変数から定まる**予測値**と実際の値との差 ε_n を**予測誤差**とみなし、**回帰モデル**

$$y_n = a + bx_n + \varepsilon_n$$

を考えることにします。 a と b は**回帰係数**と呼ばれます。特に a を切片、 b を直線の傾きと呼ぶこともあります。

以下では回帰係数をデータから推定する方法を考えます。



1.3 最小二乗法による回帰係数の決定[2]

データの情報を用いて、回帰直線を求める手続きを**推定**という。

単回帰モデルにおいてある（確率）分布を仮定すると、手元にあるデータは回帰直線の周りに分布しているはずである。そこで、データに対して「当てはまりのよい直線を回帰直線とする」ことが単純なアイディアとして出てくる。

データに対して当てはまりのよい直線とはどのような直線だろうか。回帰直線の候補 $y = a + bx$ が与えられたとき、この直線と各データ点 (x_i, y_i) とは

$$\varepsilon_i = y_i - (a + bx_i)$$

だけ離れていることになる。この ε_i を **残差(各点誤差)** という。一般に残差の値が全体的に 0 に近い直線は、直線と各データ点との差が小さく、データへの当てはまりが良いと言える。全体的な差の総量を評価するため、残差を2乗して和をとった **残差平方和(RSS)** を定義し、このRSS値が最小となるような係数 a, b を求めることは可能である（なぜなら、2次関数は凸であるから）。このような RSS の最小化を用いて回帰直線（回帰係数）を求める方法を、**最小二乗法**（ordinary least squares, OLS法）という。RSSが最適化された回帰係数を \hat{a}, \hat{b} と表す[2]。

最小二乗法による回帰直線の推定

回帰モデルの係数 a と b の推定にあたっては、なるべく予測誤差 ε_n を小さくすることを目指します。そのために N 組のデータ $(x_1, y_1), \dots, (x_N, y_N)$ があるとき予測誤差 ε_n の二乗の総和

$$S(a, b) = \sum_{n=1}^N \varepsilon_n^2 = \sum_{n=1}^N (y_n - a - bx_n)^2$$

を最小とするように係数 a と b を求めます。このように誤差の二乗和を最小化することによって未知数を推定する方法を**最小二乗法**と呼びます。

最小二乗法を適用すると**係数の最小二乗推定値**が次のように得られます。データから推定された係数などには「 $\hat{\cdot}$ 」をつけて表現します。

$$\hat{a} = \bar{y} - b\bar{x}, \quad \hat{b} = \frac{\sum_{n=1}^N (x_n - \bar{x})(y_n - \bar{y})}{\sum_{n=1}^N (x_n - \bar{x})^2}$$

なお、以下のように $S(\hat{a}, \hat{b})$ をデータ数 N で割ると**残差分散**が得られます。

$$\hat{\sigma}^2 = S(\hat{a}, \hat{b}) / N$$

東京大学 数理・情報教育研究センター 北川源四郎 2021 CC BY-NC-SA

8

RSSが最適化され、予測モデル $y = \hat{a} + \hat{b}x$ が求められたとする。このとき、 $x = x_i$ での予測値を \hat{y}_i とする。つまり、各点予測は $\hat{y}_i = \hat{a} + \hat{b}x_i$ である。 \bar{y} は全ての点での y の平均値とする。

総平方和 TSS の定義 :

$$TSS = \sum_{i=1}^n (y_i - \bar{y})^2$$

回帰平方和 ESS の定義 :

$$ESS = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$$

残差平方和 RSS も含めた3つの平方和の間には、 $TSS = ESS + RSS$ という関係がある。これらの関係から、 y の変動全体(ESS)のうち、回帰直線(TSS)で説明できた割合を計算することで、回帰直線にとデータ点との当てはまりの良さを評価できる。この値を**決定係数 R**として以下で定義する[2]。

$$R^2 = ESS/TSS = 1 - RSS/TSS$$

Pythonによる単回帰分析例：対数コロナ感染率と人口密度の関係

日本の47都道府県の人口千人あたりの新型コロナウイルスの累積感染者数（2021年4月当時）のデータ

- オリジナルデータとプログラム例（公開教材[1]ただし実装はR）
- [参考: Seaborn で散布図・回帰モデルを可視化する]([

<https://pythondatascience.plavox.info/seaborn/%E6%95%A3%E5%B8%83%E5%9B%B3%E>

```
In [7]: ## seaborn regplotで線形回帰直線と信頼区間を描画してみる.
import warnings
warnings.simplefilter('ignore')

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv('Tokyo_Covid-19_regression.csv', header=1)
df.columns = ['prefecture', 'positive_rate', 'population', 'density', 'aging_rate']

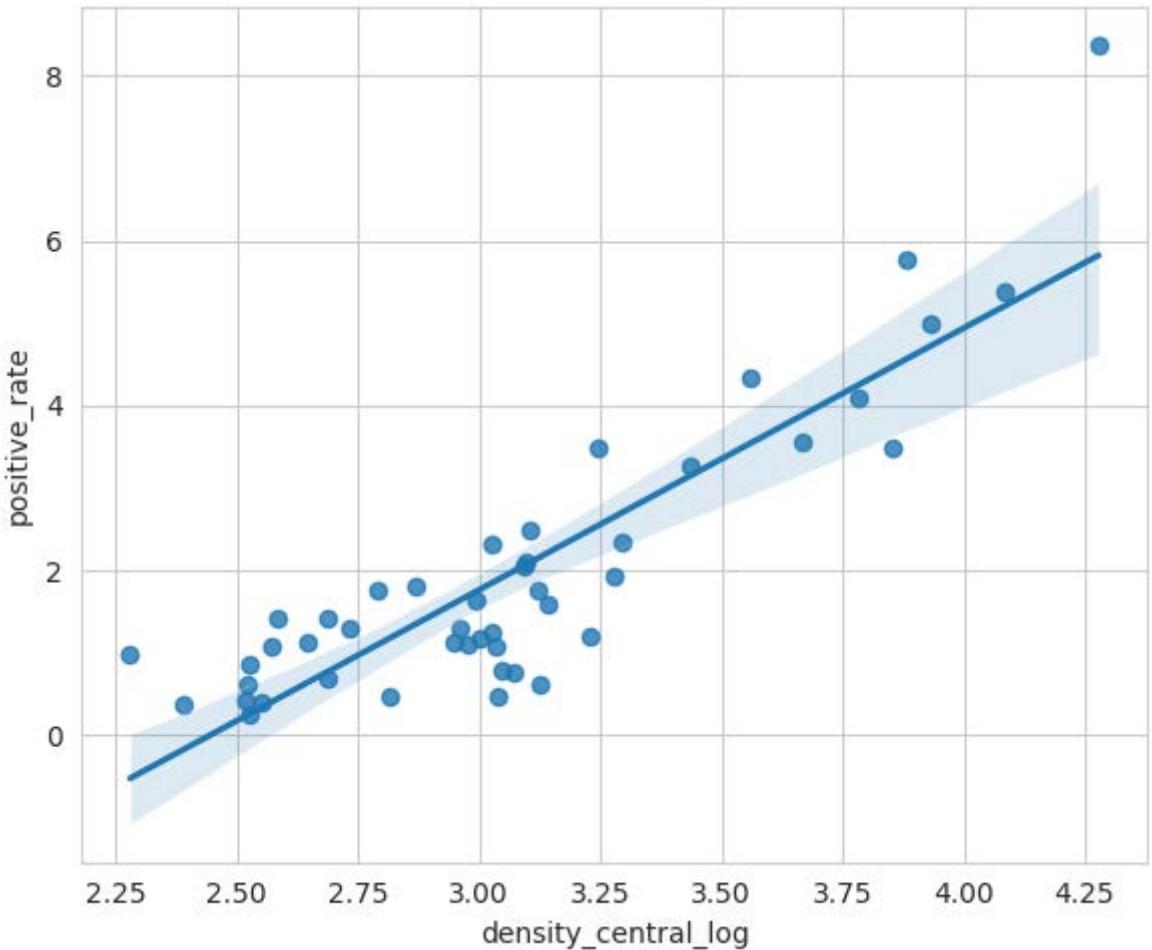
# 対数化したxデータ列を追加
df['density_central_log'] = np.log10(df['density_central'] + 1)

x = df.loc[:, ['density_central_log']]
y = df.loc[:, ['positive_rate']]

sns.set_style("whitegrid")
plt.figure(figsize=(6, 5))

sns.regplot(x='density_central_log', y='positive_rate', data=df.loc[:, ['density_central_log', 'positive_rate']])
# plt.xscale('log')
plt.tight_layout()
plt.show()

# 予測モデルの信頼区間(95%)から、(上振れ、下振れ両方で)かなり外れている都道府県があるようだ(調査)
```



```
In [8]: ## scikit-learn を用いた線形回帰の実行例
# sklearn.linear_model.LinearRegression クラスを読み込み
from sklearn import linear_model
clf = linear_model.LinearRegression() # 線形回帰モデル(教師無し学習)を作る

X = df.loc[:, ['density_central_log']]
Y = df.loc[:, ['positive_rate']]
clf.fit(X, Y) # 線形予測モデルを作成

coef = float(clf.coef_); print(coef) # 回帰係数の表示
intercept = float(clf.intercept_); print(intercept) # 切片(誤差)の表示
print(clf.score(X, Y)) # 決定係数の表示

# 線形回帰直線による予測関数の定義
def positive_rate_regression(den):
    return float(coef * den + intercept)
```

3.173743832267236
-7.756488900631577
0.7697379449105892

1.4 多項式回帰モデル [1]

単回帰モデルはデータに直線をあてはめるが、入力・出力の関係が直線では十分に表現できない場合、**多項式回帰モデル**を使うとよいことがある。

ここで、 $a_j (j = 0, 1, \dots, m)$ を定数として、 x_i が与えられたときに y_i の値が

$$y_i = a_0 + a_1 x_i + \dots + a_m x_i^m + \varepsilon_i$$

という法則（モデル）に従って決まると思定する。ここで m は多項式の次数、 a_j は回帰係数である[1]。

Pythonによる多項式回帰分析例：対数コロナ感染率と人口密度の関係

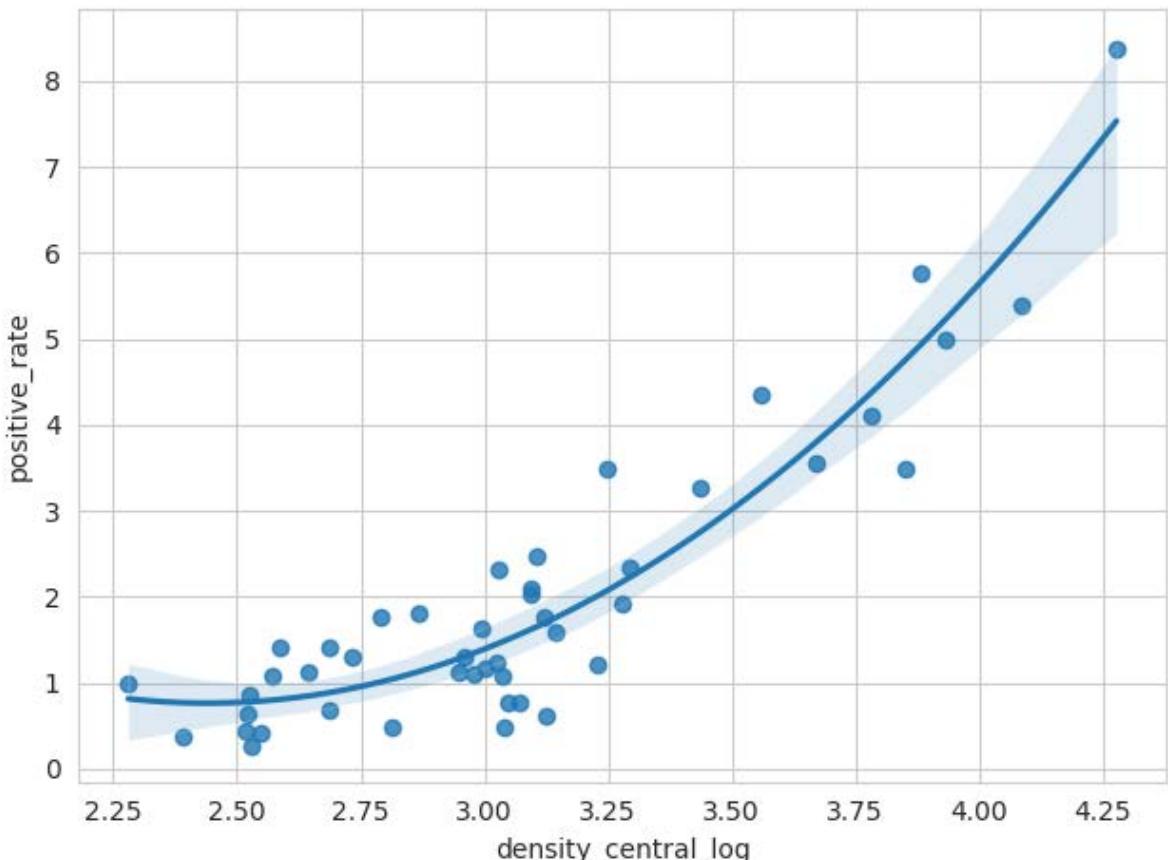
日本の47都道府県の人口千人あたりの新型コロナウイルスの累積感染者数（2021年4月当時）のデータ

- オリジナルデータとプログラム例（公開教材[1]ただし実装はR）
- [参考: Seaborn で散布図・回帰モデルを可視化する]（

<https://pythonfordatascience.plavox.info/seaborn/%E6%95%A3%E5%B8%83%E5%9B%B3%E>

```
In [9]: ## seaborn regplotで次数order=2の多項式回帰曲線と信頼区間を描画してみる.
```

```
sns.regplot(x='density_central_log', y='positive_rate', data=df.loc[:, ['density_central_log', 'positive_rate']])
plt.tight_layout()
plt.show()
# 2次の予測モデルの場合、東京度も信頼区間(95%)にギリギリ入る感じになる
```



```
In [10]: ## scikit-learn を用いた多項式回帰の実行例
```

```
# scikit-learnのPolynomialFeaturesを読み込み
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures

X = df.loc[:, ['density_central_log']]
Y = df.loc[:, ['positive_rate']]

# 2次元の特徴量に変換
polynomial_features = PolynomialFeatures(degree=2)
```

```

X_poly = polynomial_features.fit_transform(X)

clf = linear_model.LinearRegression() # 線形回帰モデル(教師無し学習)を作る
clf.fit(X_poly, Y) # 線形予測モデルを作成
Y_pred = clf.predict(X_poly)

# 評価
rmse = np.sqrt(mean_squared_error(Y, Y_pred)) #RMSE 二乗平均平方根誤差を計算
r2 = r2_score(Y, Y_pred) #R 決定係数を計算
print(f'RMSE : {rmse}')
print(f'R2 : {r2}')
print(clf.coef_) # 回帰係数の表示

```

RMSE : 0.6023220609772968
 R2 : 0.8695361540597994
 [[0. -9.77240526 2.002669]]

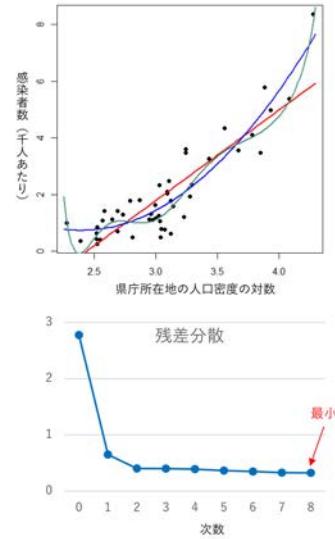
1.5 過学習 (Overfitting) 、次数選択

表現力が高いモデルを使う時の注意：

直線を用いた単回帰の代わりに2次式を用いると良い当てはまりが得られたので、もっと高い次数を使ってみたくなるのは自然です。

実際、8次の多項式を使ってみると右上の図の緑色の曲線のように、データの両端ではちょうどデータの上を通っていてデータを忠実に表現しています。

実際、残差分散をプロットしてみると、次数が高くなると残差分散は単調に減少しており、高次のモデルほどデータへの適合がよいことが分かります。しかし、この8次の回帰曲線は本当に良いモデルでしょうか？

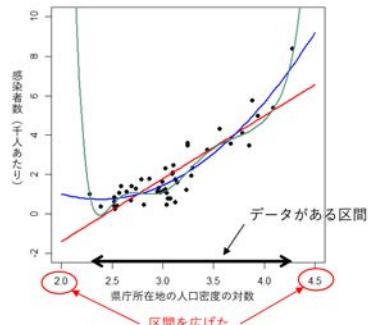


表現力が高いモデル利用時の注意： 過学習

表現力が高すぎるモデルを使うと、データにはよく適合するが観測されたデータ以外の予測、特に外挿を行ったときの予測精度が著しく悪くなる恐れがあります。

実際、説明変数の範囲を2.0から4.5に広げて回帰曲線を描いてみると図のようになります。外挿区間を広げても直線や2次式は安定していますが、緑色の8次式は両端で跳ね上がっていて、外挿能力が低いことがわかります。

データをよく表現しているが、予測能力が低くなる現象は過学習あるいは過剰適合(オーバーフィッティング)と呼ばれます。学習に使ったデータだけはよく表現しているが、それ以外のデータにも使える汎用性がないのです。

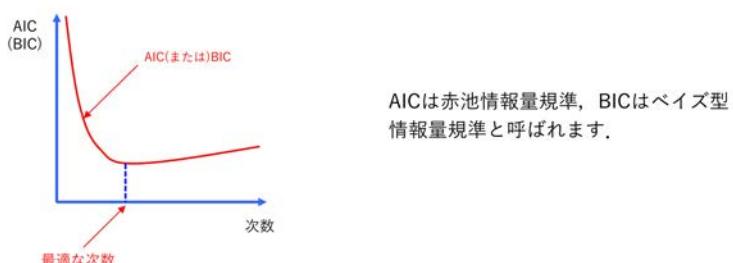


次数選択、モデル選択

多項式の次数は低すぎると現象を十分に表現できませんが、高すぎると過剰適合を起こします。したがって次数を適切に選ぶことが重要で、**次数選択**の問題と呼ばれます。

予測の観点からモデルの良さを評価し、次数を自動的に選択するための評価基準として情報量規準 AIC や BIC が用いられます。

候補となるすべての次数 m に対して AIC_m (あるいは BIC_m) を計算してその中で最小にする m を探すと、それが**最適な次数**となります。



情報量規準AICとBIC

m 次の多項式回帰モデルの場合、 N をデータ数、 σ_m^2 は m 次の多項式回帰モデルの残差分散とするとき、AICとBICは（共通の定数を無視すると）次のように定義されます。

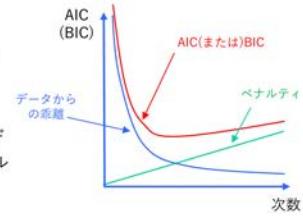
$$\begin{aligned} \text{AIC}_m &= N \log 2\pi\sigma_m^2 + 2(m+2) \\ \text{BIC}_m &= N \log 2\pi\sigma_m^2 + (m+2)\log N \end{aligned}$$

↑
データからの乖離 ↑
モデルの複雑さへのペナルティ

ここで、右辺の第1項はデータからの乖離を、第2項はパラメータ数を増やすことによるモデルの複雑さへのペナルティ項を考えることができます。

データからの乖離は次数を増やすと減少する一方、ペナルティ項は増大するので、通常適当な次数で最小となります。

【注意】上記の定義は回帰モデルの場合で、情報量規準はもっと一般的なモデルに適用できます。また、AICやBICはモデルの次数選択だけでなく、モデルの選択や変数選択にも利用できます。



1.6 重回帰モデル [1]

単回帰モデルは説明変数1つ：従属変数1つの関係性の解析であるが、説明変数を複数にして（選択した）場合の入力・出力の関係を表す予測モデルは、**重回帰モデル**を使うといふことがある（説明変数の従属変数に対する「寄与度」を、データの様相に応じて自動計算する手法は、次章の次元削減の項で説明する）。

ここで、 $a_j (j = 0, 1, \dots, m)$ を定数として、 j 番目の説明変数 x_j の i 番目のデータ $x_{i,j}$ が与えられたときに y_i の値が

$$y_i = a_0 + a_1 x_{i,1} + \dots + a_m x_{i,m} + \varepsilon_i$$

という法則（モデル）に従って決まると思定する。ここで m は説明変数の個数、 a_j は回帰係数である[2]。

重回帰分析

単回帰分析の例では、各県の千人あたりの感染者数を県庁所在地の人口密度で単回帰しましたが、感染者数に関連すると思われる変数は

県全体の人口密度
高齢者率
交通便り度

など、その他にもいろいろ考えられます。したがって、これらの変数を説明変数と考えて、目的変数を

$$y_n = a_0 + a_1 x_{n1} + \cdots + a_m x_{nm} + \varepsilon_n$$

m 個の説明変数

のように、定数項 a_0 と m 個の説明変数 x_1, \dots, x_m の影響として表現するモデルが**重回帰モデル**です。単回帰の時と同様に a_1, \dots, a_m は回帰係数で、残差 ε_n は正規分布 $N(0, \sigma^2)$ に従うものと仮定します。

重回帰分析：都道府県別感染者数の例

以下ではコロナウィルスの**都道府県別感染者数**を例に考えてみます。説明変数の候補として、表のように x_{n1} から x_{n8} までの 8変数を考えることにします。 x_{n1} は定数項に対応します。 x_{n8} は県庁所在地の人口密度 x_{n4} の2乗です。この変数は多項式回帰で2次のモデルが最もよかったです。採用したものです。

これらの説明変数は、コロナウィルスの感染が人口密度や年齢あるいは人の移動に関連して発生しやすいと言われていることから、候補として採用したものですが、実際にはこのほかにもいろいろな変数を考えることができます。

表：目的変数と説明変数

y_n	コロナウィルスの累積患者数
x_{n1}	定数項
x_{n2}	県全体の人口密度
x_{n3}	高齢化率
x_{n4}	都道府県庁所在地の人口密度
x_{n5}	交通便り度(2時間)
x_{n6}	交通便り度(4時間)
x_{n7}	交通便り度(6時間)
x_{n8}	x_{n4}^2 (人口密度の2乗)

Pythonによる重回帰分析例：対数コロナ感染率と、<高齢化率, 人口密度, 人口密度^2>の関係

日本の47都道府県の人口千人あたりの新型コロナウィルスの累積感染者数（2021年4月当時）のデータ

- オリジナルデータとプログラム例（公開教材[1]ただし実装はR）
- [参考: Seaborn で散布図・回帰モデルを可視化する](

<https://pythondatascience.plavox.info/seaborn/%E6%95%A3%E5%B8%83%E5%9B%B3%E>

```
In [11]: # density_central_logを2乗したデータ列を生成し、説明変数3個としてデータXを構成
df['density_central_log2'] = np.square(df['density_central_log'])
X = df.loc[:, ['aging_rate', 'density_central_log', 'density_central_log2']]

clf = linear_model.LinearRegression() # 線形回帰モデル(教師無し学習)を作る
clf.fit(X, Y) # 重回帰予測モデルを作成(Xが複数の説明変数である場合、自動的に重回帰モデル)
Y_pred = clf.predict(X) # 予測値計算

# 評価
rmse = np.sqrt(mean_squared_error(Y, Y_pred)) #RMSE 二乗平均平方根誤差を計算
r2 = r2_score(Y, Y_pred) #R 決定係数を計算
print(f'RMSE : {rmse}')
print(f'R2 : {r2}')
print(clf.coef_) # 回帰係数の表示

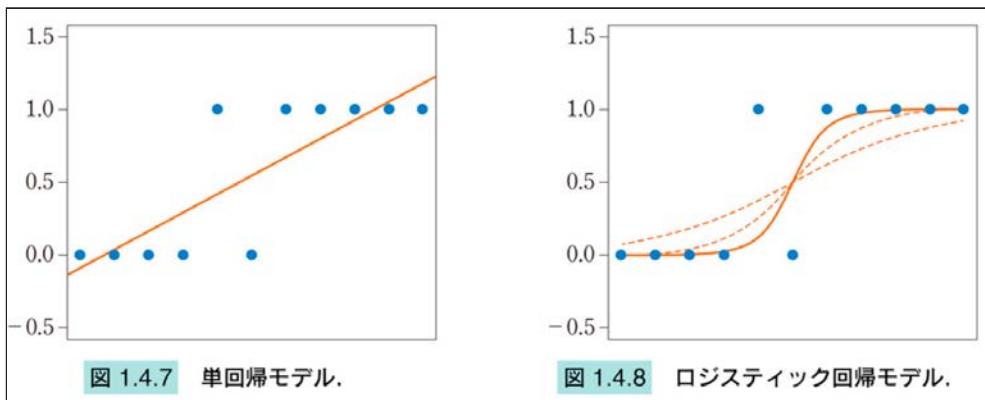
# 決定係数 R の評価結果
# 単回帰(人口密度) 0.769
# 多項式回帰(人口密度, order=2) 0.869
# 重回帰(高齢化率, 人口密度, 人口密度^2) 0.908
```

2. ロジスティック回帰分析（離散目的変数） [1][2]

被説明変数が2値の質的変数の場合は、ロジスティック関数と呼ばれるS字型の曲線の関数を考え、その曲線パラメータ（例えば2つ）を推定する。予測モデルとデータ当てはめの制度は、**尤度（確からしさ）**を用いて評価する。

2.1 ロジスティック回帰の概要

病気の発症の有無、商品購入の有無のような質的変数を目的変数（従属変数） y とする場合、例えば $y = \{0, 1\}$ の2値に変換した後、説明変数 x に対して回帰分析を行う手法が考えられる。しかし、2値しかとらない目的変数であったとしても、通常の線形回帰は「直線」でモデルを予測するので、例えば下の図1.4.7のような单回帰モデルの場合、予測値が0以下や1以上の値を取ってしまう領域がある。



単回帰モデル、ロジスティック回帰モデル ([2]より引用)

この問題を解決するため、目的変数を2値ではなく、開区間 $(-\infty, +\infty)$ を定義域、閉区間 $[0, 1]$ を値域とするような単調増加な連続関数へ変換するアイディアが考えられる。このよ

うな性質を持つ関数は数多く存在するが、例えば指数関数の逆関数を用いたロジスティック関数

$$f(x) = \frac{1}{1 + \exp\{-(\alpha + \beta x)\}}$$

を変換に用いることを考えてみよう（図1.4.8）[2]。

（つづき）このようなロジスティック関数によって変換すれば、図1.4.8に見られるように、任意の説明変数 x の値によって、0以上1以下の値を取るようになる。言い換えれば、変換後の関数 $f(x)$ は、説明変数の値が x であるときに、 $y = 1$ になる確率を示すと考えてよい（後述の「ロジット変換」の項を参照）。

予測モデルとしての目標は「データに最もよく当てはまるようなロジスティック関数のパラメータ α, β を求める」とあるが、これは、与えられたデータ点とロジスティックス関数によって示された「確からさ」をより良く満足するような曲線の形を決定したいということになる。

このような予測モデルをロジスティック回帰モデル(logistic regression model)といい、このモデルを用いた分析をロジスティック回帰分析 (logistic regression analysis) という。

ロジスティック回帰モデルの回帰係数の推定には、最小二乗法ではなく、最尤法(maximum likelihood method)という方法が用いられる。線形回帰のときのようにデータ点と平均値との2乗誤差といった単純な最適化は用いることができず、一言でいえば手元にあるデータの出現の尤もらしさを説明できるモデルを求める方法である[2]。

2.2 ロジット変換とロジスティック回帰

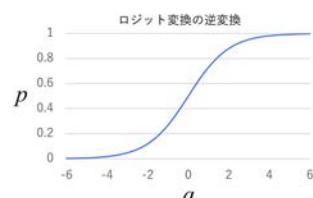
ロジット変換

まず、携帯電話普及率のように、説明変数（年）に対する割合 p_n が大量な観測数に基づいて得られている場合には、 p_n をそのまま回帰分析するのではなく、 p のロジット変換

$$q = \log\left(\frac{p}{1-p}\right)$$

を定義しそのモデル化を考えます。逆に q から

$$p = \frac{e^q}{1+e^q}$$



で確率 p を求めると、 q がどんな値をとっても p は常に 0 と 1 の間の値をとるので q の空間では自由なモデリングが可能となります。

【参考】 $p/(1-p)$ はオッズと呼ばれ、競馬などのギャンブルでも利用されます。

ロジスティック回帰モデル

N 組のデータ $(x_1, y_1), \dots, (x_N, y_N)$ があり, x_n は説明変数, y_n は病気に罹患している場合は0, いない場合は1となる離散変数とします.

0か1の2値をとる変数に対して予測誤差は考えにくいので, 1という事象が起きる確率 p を考え, そのロジット変換 q を説明変数 x_n で表現するモデル

$$q_n = a + bx_n$$

を考えることにします. このとき

$$\log\left(\frac{p_n}{1-p_n}\right) = a + bx_n$$

なので, 事象1および0が起きる確率はそれぞれ次のように表されます.

$$p_n = \frac{\exp(a + bx_n)}{1 + \exp(a + bx_n)}, \quad 1 - p_n = \frac{1}{1 + \exp(a + bx_n)}$$

このように事象1と0が起きる確率を表現するモデルを**ロジスティック回帰モデル**と呼びます.

2.3 ロジスティック回帰モデルの尤度、最尤推定

ロジスティック回帰モデルの尤度

ロジスティック回帰モデルを想定すると, 説明変数 x_n が与えられているとき, y_n の確率は

$$P(y_n | x_n, a, b) = \begin{cases} \frac{1}{1 + \exp(a + bx_n)} & y_n = 0 \text{ のとき} \\ \frac{\exp(a + bx_n)}{1 + \exp(a + bx_n)} & y_n = 1 \text{ のとき} \end{cases}$$
$$= \frac{\exp\{y_n(a + bx_n)\}}{1 + \exp(a + bx_n)} \quad \text{【注意】 } y_n \text{ は } 0 \text{ か } 1 \text{ なので上の式は} \\ \text{このように一つの式で書ける.}$$

で与えられます. したがって, N 組の独立なデータ $(x_1, y_1), \dots, (x_N, y_N)$ が与えられているとき**尤度**は以下のように定義されます.

$$L(a, b) = \prod_{n=1}^N p(y_n | x_n, a, b) = \prod_{n=1}^N \frac{\exp\{y_n(a + bx_n)\}}{1 + \exp(a + bx_n)}$$

次頁ではこの尤度を用いてパラメータ a, b を推定する方法を考えます.

最尤法

尤度の最大化によって、未知数 $\theta=(a,b)$ を推定する方法は**最尤法**と呼ばれます。ただし、通常は尤度の対数をとった以下の**対数尤度**を最大化します。対数関数は単調増加関数なので、どちらでも同じ推定値が得られます。

ロジスティック回帰モデルの場合、対数尤度は次のようになります。

$$\begin{aligned}\ell(a,b) &= \log L(a,b) \\ &= \sum_{n=1}^N \left\{ y_n(a + bx_n) - \log(1 + \exp\{(a + bx_n)\}) \right\}\end{aligned}$$

したがって、この対数尤度を最大化、すなわち

$$\max_{a,b} \ell(a,b) = \ell(\hat{a}, \hat{b})$$

となる \hat{a} と \hat{b} を求めるこによって、係数の推定値が得られます。このように、最尤法で推定された係数を**最尤推定値**と呼びます。

30

東京大学 数理・情報教育研究センター 北川源四郎 2021 CC BY-NC-SA

Pythonによるロジスティック回帰分析の実装例

東京の1年間（2020年）の気象データから、1日の最高湿度、最低湿度、降水現象（雨が降った/降らなかった）のデータを抽出したもの。湿度を説明変数、降水事象を目的変数として、湿度からどの程度降雨を予測できるかを見てみる[1]。

- オリジナルデータとプログラム例（公開教材[1]ただし実装はR）

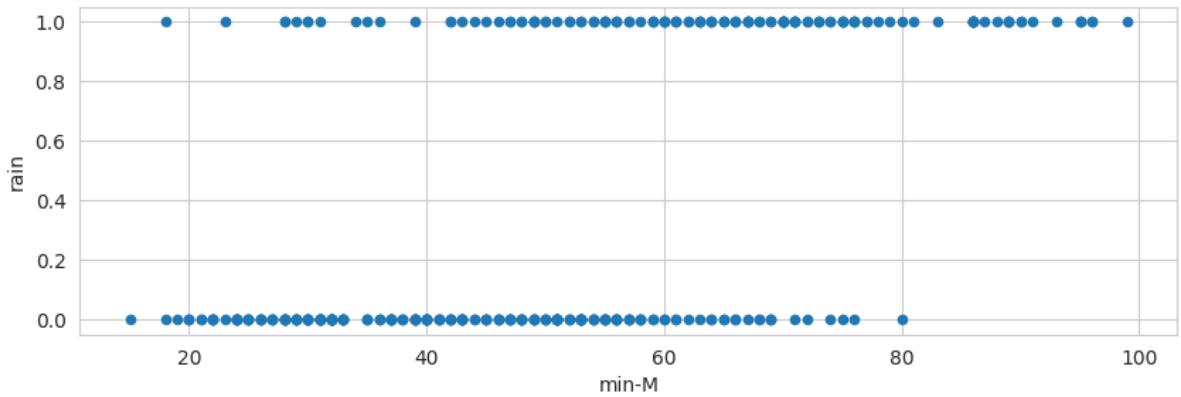
```
In [ ]: # CSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする。
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DS/main/4/
# wgetしなくても,Google colab.の左メニュー [ファイル] アイコンをクリックして,ブラウザへファイルを
# ファイル (udon.csv) がダウンロード・配置できることを確認する
!ls -al ./
```

```
In [13]: # オリジナルのCSVファイルをpandasで読み込んでデータフレームdfに格納
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Tokyo_rain.csv', header=1)
df.columns = ['Moisture', 'min-M', 'rain']
df.head()
```

```
Out[13]:   Moisture  min-M  rain
0          60       40     0
1          63       42     0
2          53       33     0
3          61       34     1
4          44       31     0
```

```
In [14]: # 散布図 X軸:最低湿度(%:線形) Y軸:降水現象の有無(0:無し, 1:有り)
df.plot(figsize=(10, 3), kind='scatter', grid=True, x=df.columns[1], y=df.co
```

```
plt.show()
```



```
In [15]: # scikit-learnのLogisticRegressionモデルを適用した結果 (コード引用:TauStation http://
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# 説明変数: 最低湿度、目的変数: 降水現象
X = np.array(df['min-M']).reshape(-1, 1)
y = df['rain']

# C: 正則化強度=1e5, L2正則化を指定してロジスティック回帰モデルを構築して最適化実行
logreg = LogisticRegression(C=1e5, solver='lbfgs')
logreg.fit(X, y)

# 回帰係数、50%しきい値を表示
b = logreg.intercept_[0]
w = logreg.coef_[0][0]
x_border = 0.0 - b / w
print("intercept : {}".format(b))
print("coefficient: {}".format(w))
print("x_border : {}".format(x_border))

# 元データと予測モデルのグラフ描画
xl = 0
xr = 100

x_graph = np.linspace(xl, xr)
y_graph = 1 / (1 + np.exp(-b - w * x_graph))

plt.rcParams["figure.figsize"] = [10,3]
fig, ax = plt.subplots()

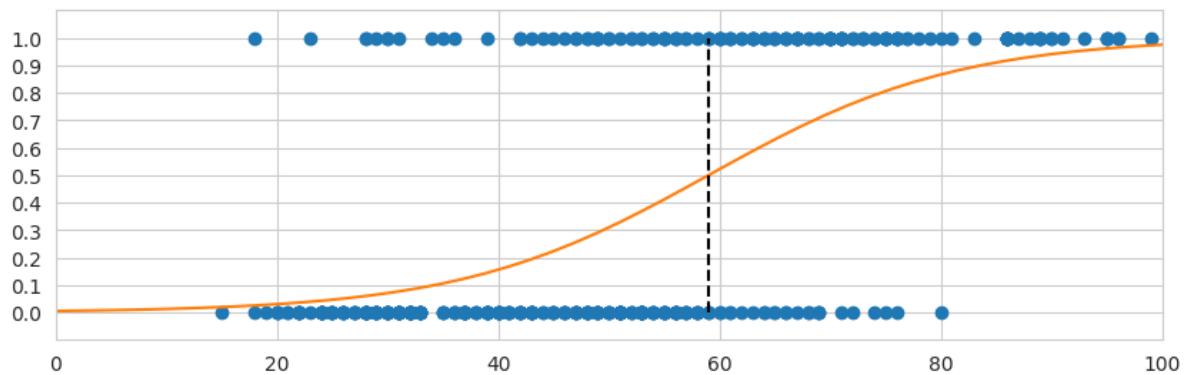
ax.scatter(X, y, c='tab:blue')
ax.plot(x_graph, y_graph, c='tab:orange')
ax.plot([x_border, x_border], [0, 1], c='k', linestyle='dashed')

ax.set_xlim(xl, xr)
ax.set_ylim(-0.1, 1.1)
ax.set_yticks(np.linspace(0, 1, 11))
ax.grid(True)
plt.show()

# α切片 intercept、β傾き coefficient の推定結果、予測モデルとしての曲線(降水確率)が表示され
# 最大尤度推定した結果の確率50%しきい値(湿度)は、x_border として表示されている。
```

```
# 湿度が40%,60%,80%の時、降水確率は各々 およそ15%,50%,85%となることがわかる(あくまで確率)
# 曲線の傾きはかなり「なだらか」であるから、湿度を使って雨降りを言い当てるのは(よほど)高湿度の状況!
```

```
intercept : -5.25429137264403
coefficient: 0.0890115222159052
x_border   : 59.029339593803236
```



ロジスティック回帰分析結果から、 $a = -5.254, b = 0.0890$ という推定値が得られたから、最低湿度 x による降水確率 $p(x)$ は、

$$p(x) = \frac{exp(a + bx)}{1 + exp(a + bx)} = \frac{exp(-5.254 + 0.0890x)}{1 + exp(-5.254 + 0.0890x)}$$

という予測モデルが得られた。

3. 時系列データ分析 [1][2]

回帰分析で扱ったコロナ感染率のデータや、降水現象のデータなどは、いずれも「時点」を固定した上で対象から得られたデータである。このようなデータを横断面データ（cross sectional data）という。

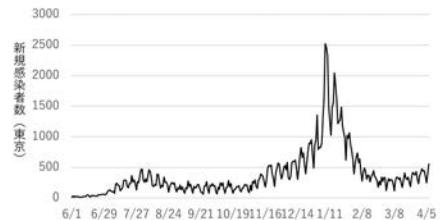
これに対して、時間軸上で（時間の流れに沿って連続的に）得られたデータを、**時系列データ**（time series data）という[2]。何かを目的変数にすることが無いから、いわゆるノンパラメトリック分析に相当する。

3.1 時系列データの概要

時系列データ y が1期から N 期まであるときに、これらを y_1, y_2, \dots, y_N のように表す。表現自体はこれまでのデータと同じだが、時系列データの場合は**時間軸上で添え字の順番にデータが出現している**という点が違う。分析をする上でも、**時間軸上での変化率や周期が興味の対象であることが多い**。時系列データを折れ線グラフで可視化したものを持て**時系列グラフ**（time series chart）という[2]。

時系列とは

図は横軸に月日、縦軸に東京の新型コロナウィルス新規感染者数を示します。このように時間と共に変化する現象を記録したものが**時系列データ**で、それを図示したものは**時系列グラフ**と呼ばれます。



データが何番目に観測されたかを n で表し、そのときの観測値を y_n とします。 N 個の観測値 y_1, \dots, y_N が得られたとき、 N は**時系列の長さ**あるいはデータ数と呼ばれます。

時系列分析の目的は、時系列の特徴を捉えたり、モデルを推定することによって、今後の変化を予測したり、意思決定に必要な情報を得ることです。そのための基本的なツールとして、以下では時系列の周期を捉える方法と時系列の傾向を捉える方法を考えます。

3.2 ピリオドグラムとパワースペクトル

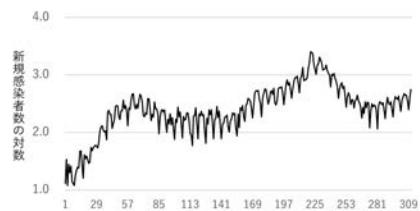
一般的に、変動過程のパワースペクトル密度 (Power Spectrum Density; PSD) を推定する1つの方法は、過程のサンプルの離散フーリエ変換を求め (通常はFFTを使用)、結果の振幅の2乗を適宜スケーリングするというものである。この推定は、"ピリオドグラム"と呼ばれる。

- [MathWorks - ノンパラメトリック推定法](#)

ピリオドグラムに似たものとしてパワースペクトルがある。パワースペクトルはピリオドグラムと違って連続関数になるが、時系列モデルを推定することによって求められる。パワースペクトルでも同様な周波数にピークが見られる[2]。

時系列を変換してみる

人数や金額などを数えたデータの場合は対数をとった方が特徴を捉えやすいこともあります。右図は新規感染者数の**常用対数**をとって表示したものです。



前ページの原データでは、時刻によって変動の幅が大きく変化していましたが、対数変換の結果ほぼ同じような変動幅となり、同じようなパターンで周期的に上下を繰り返していることが顕著になります。この周期性は、検査体制の影響で日曜日や月曜日の新規感染者数が少なくなるからです。

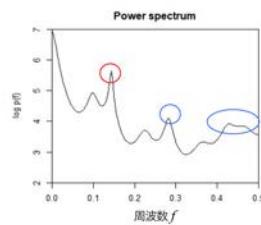
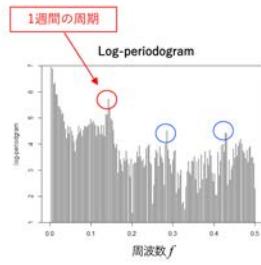
ただし、時系列の場合は完全な**周期的変動**を示すことは少なく、多くの場合周期的な変動パターン自体が徐々に変化します。このような時系列の周期的変動を可視化するツールとして、ピリオドグラムとパワースペクトルがあります。

例： Covid-19データの周期性

Covid-19 時系列のピリオドグラムの例です。横軸 f は周期の逆数で**周波数**と呼ばれます。また、縦軸は各周波数におけるピリオドグラムの対数を表します。 $0 \leq f \leq 0.5$ の範囲が図示されていますが、 $f = 0.5$ は2日に1回の周期成分、 $f = 0$ は周期無限大の成分です。ピリオドグラムが極大値をとる $f = 1/7 = 0.1429$ は1週間の周期成分に対応します。

ピリオドグラムからこのデータには顕著な1週間周期が存在することがわかります。また、 $f = 2/7$ と $3/7$ にもピークが見られますが、これは**高調波**と呼ばれるもので1週間の周期成分が単純なサイン・コサインではなく、複雑な波形であることを示しています。

ピリオドグラムに似たものとして**パワースペクトル**があります。パワースペクトルはピリオドグラムと違って連続関数になりますが、時系列モデルを推定することによって求められます。パワースペクトルでも同様な周波数にピークが見られます。



3.3 時系列データの変動分解、周期性の除去（移動平均）

周期性の除去

● 周期性を除去する理由

Covid-19の場合について考えてみると、検査体制の事情から毎週日曜日や月曜日は小さな値になりますが、この事情を考慮しないと、患者数が減少したと誤った判断をしかねません。このような場合には、時系列の周期成分を除去してしまう方が簡単です。

● 周期性を除去する方法

周期の長さが分っている周期成分を除去する簡単な方法として、移動平均があります。

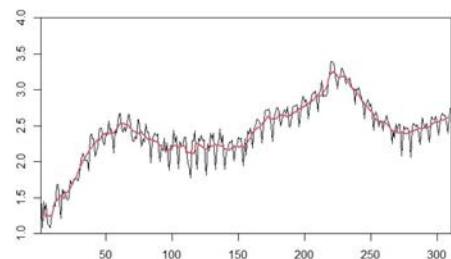
一般に、時系列 y_1, \dots, y_n が与えられるとき **($2k+1$)項の移動平均** は

$$t_n = \frac{1}{2k+1} (y_{n-k} + \dots + y_n + \dots + y_{n+k})$$

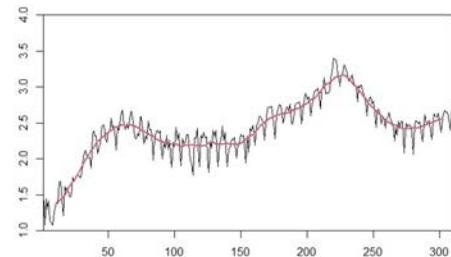
で定義されます。ここで、 $k=3$ とすると項数は $2k+1=7$ となり、カッコ内には日曜から土曜までの各曜日が1回ずつ含まれるので、曜日の影響を除去できます。この方法では直近の t_{n-2}, t_{n-1}, t_n は計算できないので、上記の式で求めたものを現時点の推定値とみなして \tilde{t}_{n+k} と定義することもあります。ただし、この片側移動平均の場合は、 k 時点だけ波形が遅れることに注意する必要があります。

例：Covid-19データの場合

上図の赤線は $k=3$ として求めた **7項移動平均** です。1週間の変動パターンがほとんど除去されて滑らかになり、感染者の増減の傾向がよく捉えられているようにみえます。



【参考】移動平均において $k=10$ とすると、前後の1週間を加え、合計3週間の平均をとることになり、さらに滑らかな移動平均が得られます(下図)。ただし、その反面急激な構造変化を捉えにくくなるという副作用があり、 $n=230$ 付近の急激な変化も滑らかになっている事には注意する必要があります。



Pythonによる時系列データの変動分解の実装例

東京都のコロナ感染者数（2021年4月8日までの過去312日間）のデータを抽出したもの。

- オリジナルデータとプログラム例（公開教材[1]ただし実装はR）

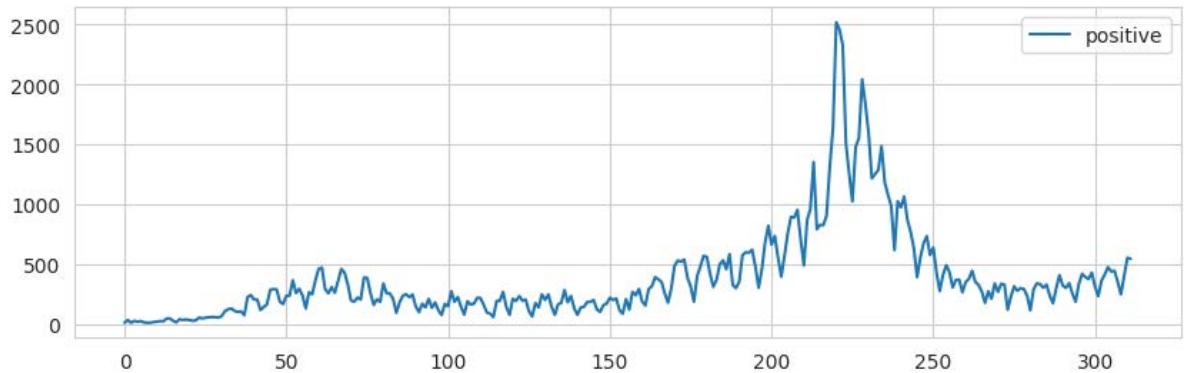
```
In [ ]: # CSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする.  
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DS/main/4/  
  
# wgetしなくとも,Google colab.の左メニュー [ファイル] アイコンをクリックして,ブラウザへファイルを  
  
# ファイル (udon.csv) がダウンロード・配置できたことを確認する  
!ls -al ./
```

```
In [17]: # オリジナルのCSVファイルをpandasで読み込んでデータフレームdfに格納
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Covid-19_tokyo_20210408.csv', header=None, dtype='float')
df.columns = ['positive']
df.head()
```

Out[17]:

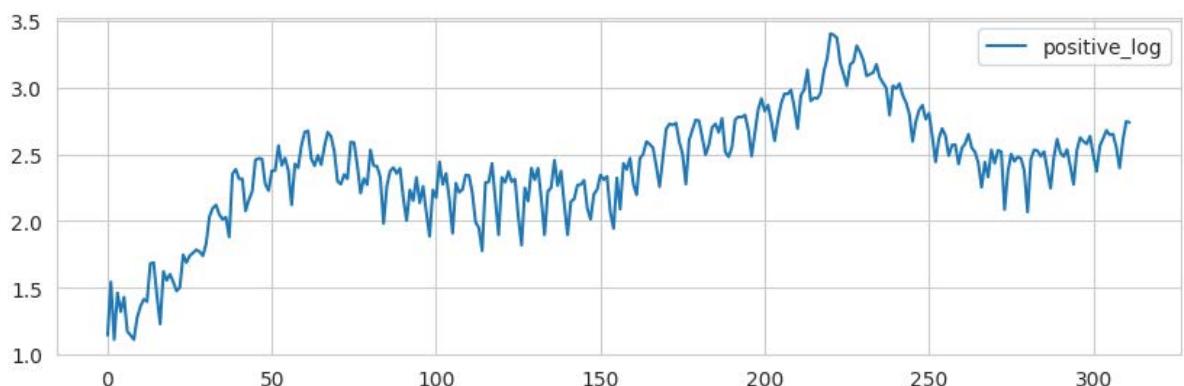
	positive
0	13.0
1	34.0
2	12.0
3	28.0
4	20.0

```
In [18]: # 折れ線グラフの描画(Y軸:線形)
df.plot(grid=True, y=df.columns[0])
plt.show()
```



```
In [19]: # 対数化したxデータ列を追加
df['positive_log'] = np.log10(df['positive'] + 1)

# 折れ線グラフの描画(Y軸:対数)
df.plot(grid=True, y=df.columns[1])
plt.show()
```



statsmodels API - Lomb-Scargle ピリオドグラムで周期性解析を行う

- 東京大学 - Lomb-Scargle ピリオドグラム 周期性解析

```
In [20]: # Scipy の Lomb-Scargle ピリオドグラムを使う - コード引用 https://zenn.dev/chimuichim
```

```

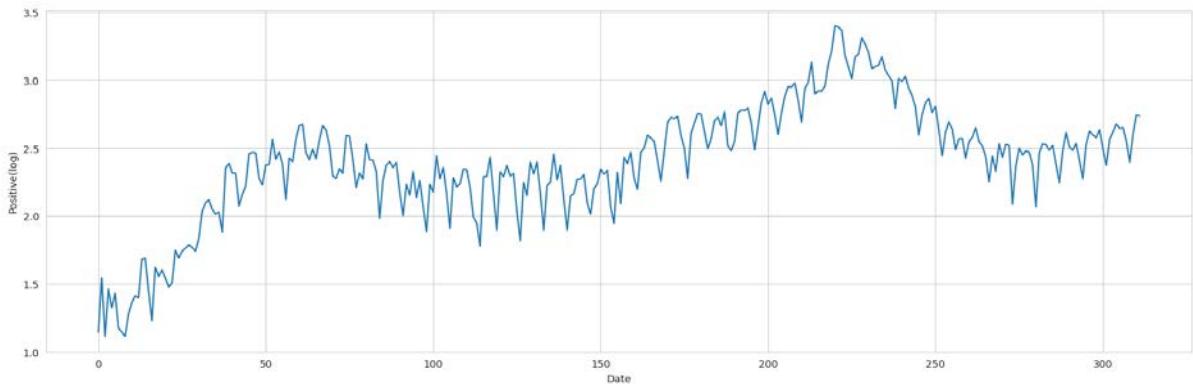
# データのロードと時系列プロット
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import scipy.signal as signal

x = list(range(len(df)))
y = df['positive_log']

fig = plt.figure(figsize=(20, 6))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel("Date")
ax.set_ylabel("Positive(log)")
ax.plot(x, y)

fig.show()

```



```

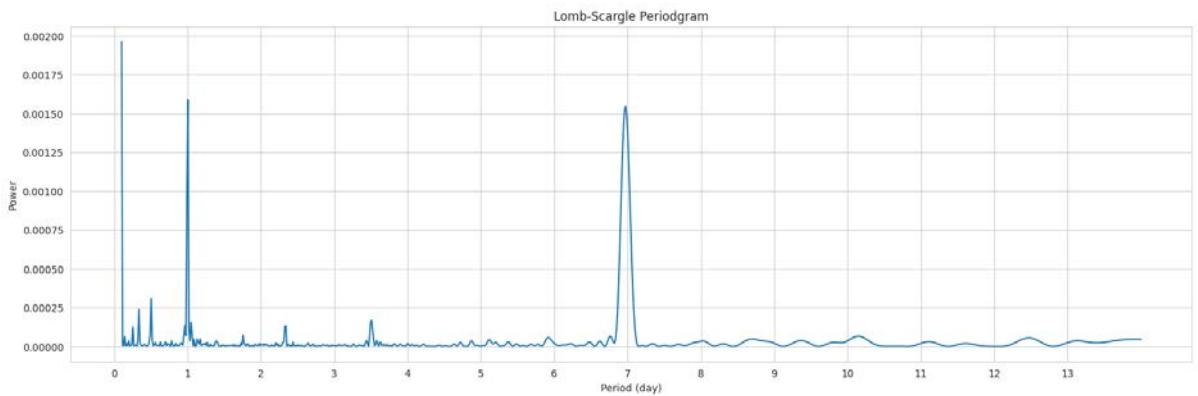
In [21]: # Lomb-Scargle ピリオドグラムの計算と出力
periods = np.linspace(0.1, 14, 1000)
ang_freq = 2 * np.pi / periods

pgram = signal.lombscargle(
    x,
    y,
    ang_freq,
    precenter = True,
    normalize = True
)

fig = plt.figure(figsize=(20, 6))
ax = fig.add_subplot(1,1,1)
ax.set_title("Lomb-Scargle Periodogram")
ax.set_xlabel("Period (day)")
ax.set_ylabel("Power")
ax.set_xticks(np.arange(0,14))
ax.plot(periods, pgram)

fig.show()
# 周期性解析の結果、7日でのパワースペクトルが存在することがわかる(検査状況の関係)

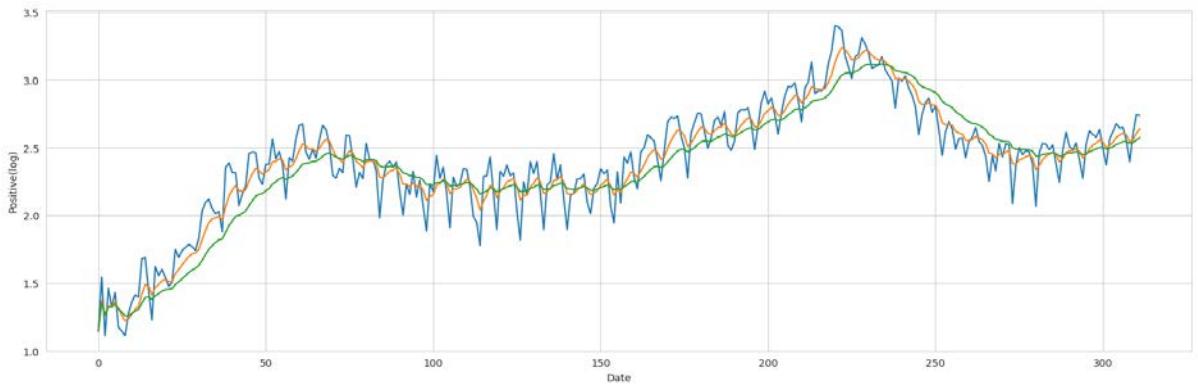
```



```
In [22]: # 指数平滑移動平均の計算(1週間、3週間)
df_ewm7 = df.ewm(span=7).mean()
df_ewm21 = df.ewm(span=21).mean()
y_ewm7 = df_ewm7['positive_log']
y_ewm21 = df_ewm21['positive_log']

fig = plt.figure(figsize=(20, 6))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel("Date")
ax.set_ylabel("Positive(log)")
ax.plot(x, y)
ax.plot(x, y_ewm7)
ax.plot(x, y_ewm21)

fig.show()
```



3.3 自己相関関数、自己回帰予測モデル(AR)

自己回帰モデルは時点 t におけるモデル出力が時点 t 以前のモデル出力に依存する確率過程である。AutoRegressiveと書かれることから ARモデルと呼ばれることが多い。

- [Wikipedia 自己回帰モデル](#)

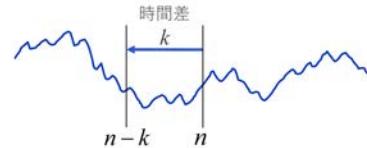
自己相関関数

周期的変動に限らず時系列の変動の特徴は自己相関関数で表現することができます。ある時刻と k だけ離れた時刻との相関係数を計算します。

$$R(k) = \text{Cor}(y_n, y_{n-k})$$

と表します。 $R(k)$ は y_n と y_{n-k} の相関係数で k はラグ、 $R(k)$ は **自己相関関数** と呼ばれます。

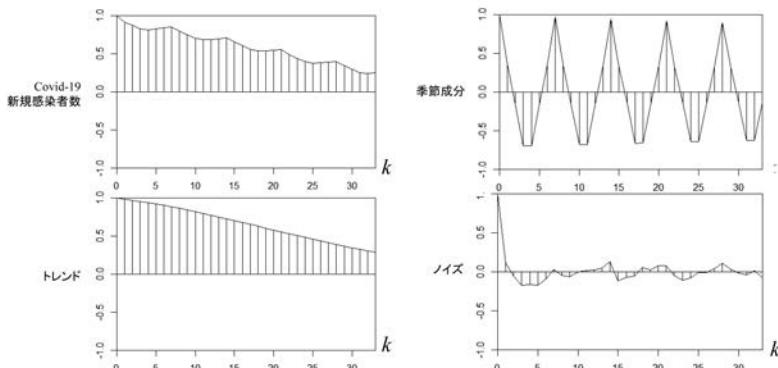
- $R(k)$ は k だけ離れた地点との相関の大きさを表します。
- y_n と y_{n-k} に正の相関があるとき $R(k) > 0$ となります。
- y_n と y_{n-k} に負の相関があるとき $R(k) < 0$ となります。



例：Covid-19 データの自己相関関数

Covid-19データのラグ32までの自己相関関数を示します。

- 左上は原データの自己相関関数で、7日周期で変動しながらゆっくり減衰しています。
- 左下は季節調整で得られたトレンドの場合で、非常にゆっくり減衰しています。また原データの変動の主要部分がトレンドであることもわかります。
- 右上は季節成分で7日周期でほぼ規則的に変動しています。
- 右下はノイズ成分で、ほとんど自己相関がない系列であることがわかります。



時系列の予測

時系列分析の重要な応用が**予測**です。時刻 n までの時系列 y_1, \dots, y_n が与えられたとき、将来の値 y_{n+1}, \dots, y_{n+k} を推定する問題が**予測**です。とくに1時点先の y_{n+1} を推定する問題を**1期先予測**、 y_{n+j} (j は2以上)を推定するときは**長期予測**と呼びます。

● ARモデルによる予測

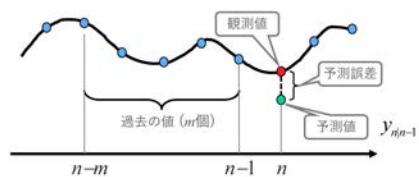
代表的な予測方法はARモデルを用いる方法で、時刻 n までのデータを使って**ARモデル**

$$y_n = a_1 y_{n-1} + \dots + a_m y_{n-m} + \varepsilon_n$$

が推定できると、 y_{n+1} の予測値は

$$y_{n+1|n} = a_1 y_n + \dots + a_m y_{n-m+1}$$

となります。以下、逐次代入により2期先以上の予測値も求められます。



Python : statsmodels API - SARIMAモデルの推定

ARモデル (AutoRegressive model)、MAモデル (Moving Average model) を組み合わせた ARMAモデルは、定常な時系列のモデルであり、ARMA(p,q) の2パラメータを持つ。

ARIMAモデル (AR Integrated MA model)は、d階差分をとった系列に対してARMA(p,q)を考えるモデルであり、ARIMA(p,d,q) の3パラメータを持つ。

ARIMAモデルに周期成分を取り入れたモデルをSARIMAモデル (Seasonal ARIMA model) という。ARIMAに加えて、季節性変動周期性を考慮した seasonal(p,d,q) ならびに注目している周期 period の4パラメータを指定する。

- 時系列分析～統計的手法編～
- Pythonによる時系列分析の基礎
- 自動SARIMAモデル推定

```
In [23]: import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
import matplotlib.pyplot as plt
import pandas as pd

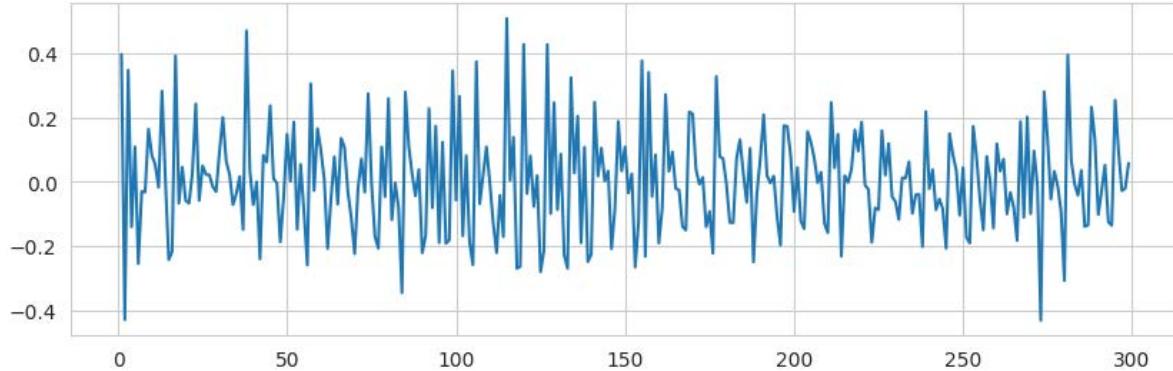
x = list(range(len(df)))
ts_all = df['positive_log'] # データ全部
ts = df['positive_log'].head(300) # 先頭から300日分を訓練データとして使用

# 差分系列を生成
diff = ts - ts.shift()
diff.head()
```

```
Out[23]: 0      NaN
         1      0.397940
         2     -0.430125
         3      0.348455
         4     -0.140179
Name: positive_log, dtype: float64
```

```
In [24]: # 差分系列のグラフ
plt.plot(diff)
```

```
Out[24]: [<matplotlib.lines.Line2D at 0x7f9d7391f970>]
```



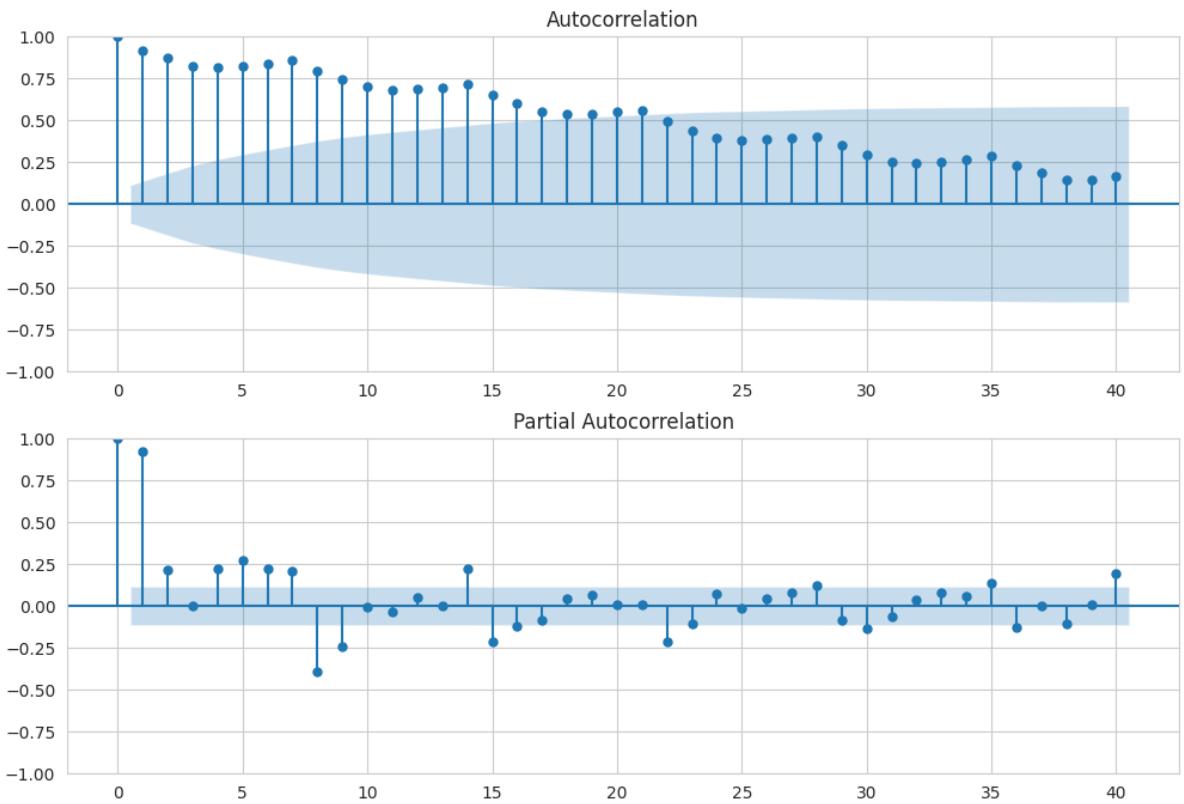
```
In [25]: # 自己相関を求める
ts_acf = sm.tsa.stattools.acf(ts, nlags=40)
ts_acf
```

```
Out[25]: array([1.          , 0.91679062, 0.87380393, 0.82507231, 0.81323599,
       0.82255791, 0.84024703, 0.85727613, 0.79838253, 0.74534535,
       0.7023909 , 0.68227085, 0.68796147, 0.69442285, 0.71289434,
       0.65212822, 0.60254167, 0.55191134, 0.53676006, 0.54058754,
       0.54960428, 0.55600462, 0.49322869, 0.43729582, 0.39566474,
       0.37712283, 0.38443775, 0.39346572, 0.40525867, 0.34903186,
       0.29764382, 0.25375037, 0.24169804, 0.25158383, 0.26796041,
       0.2853877 , 0.23188283, 0.18985427, 0.14806347, 0.14343072,
       0.16537673])
```

```
In [26]: # 偏自己相関
ts_pacf = sm.tsa.stattools.pacf(ts, nlags=40, method='ols')
ts_pacf
```

```
Out[26]: array([ 1.          ,  0.91776651,  0.2014535 ,  0.05929372,  0.24125532,
       0.31893368,  0.28900929,  0.38746279, -0.30592676, -0.15760208,
      -0.06615554, -0.06820625, -0.00953559, -0.01387226,  0.12568882,
      -0.33979603, -0.11677725, -0.10331131, -0.01282727, -0.0313394 ,
       0.06450924,  0.08501514, -0.09225216, -0.11140619, -0.0232582 ,
      -0.01994493,  0.05664235,  0.06987429,  0.17241436, -0.01926213,
      -0.04519232, -0.05060917, -0.01464231, -0.02561164,  0.07212645,
       0.16719951, -0.0897206 ,  0.01730886, -0.03914269, -0.00821575,
       0.08819341])
```

```
In [27]: # 自己相関のグラフ
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(ts, lags=40, ax=ax2)
# 7日間の周期性が見られることがわかる(ts.diffは一日遅れであることに注意)。
```



ARIMAモデルの推定（差分実行）

```
In [28]: # 差分系列への自動ARMA推定関数の実行(AIC基準,BIC基準)
resDiff = sm.tsa.arma_order_select_ic(diff, ic=['aic', 'bic'], trend='c')
resDiff
# ARMA(4,1)が最も良いらしい。
```

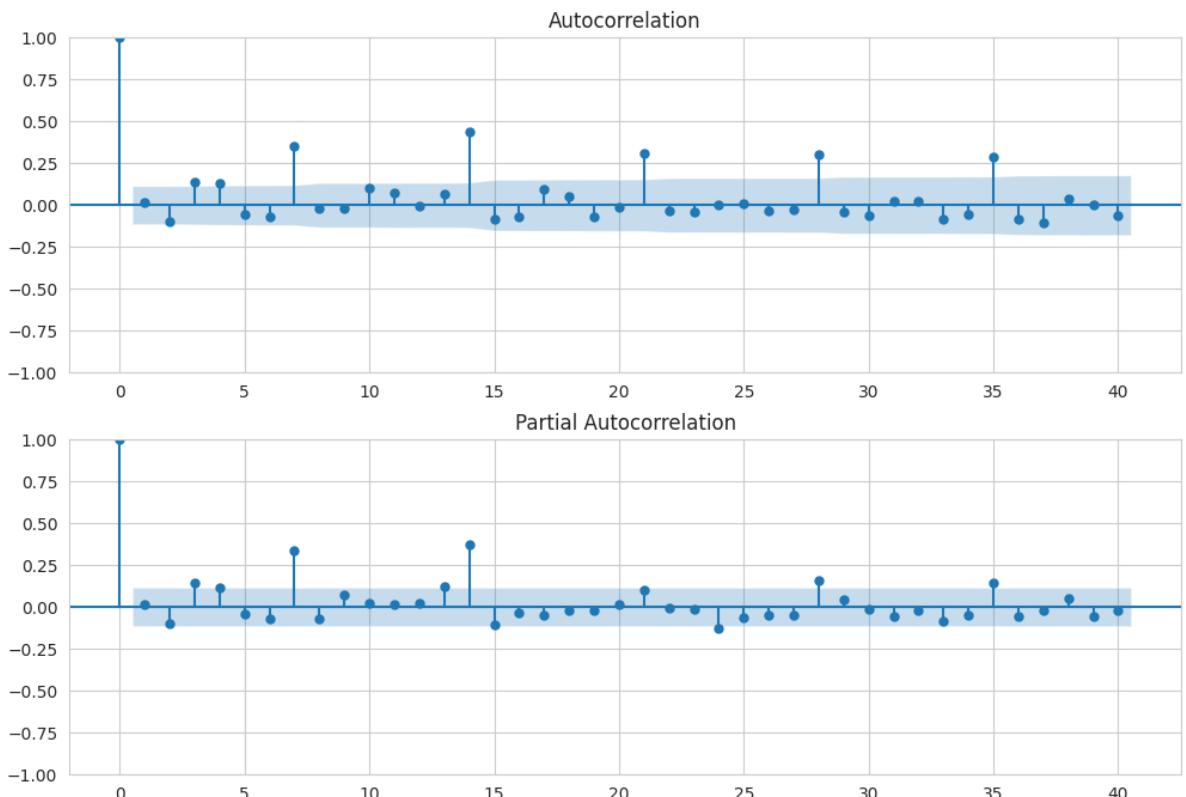
```
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check model_rets
  warnings.warn("Maximum Likelihood optimization failed to "
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:604: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check model_rets
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
Out[28]: {'aic': 0 1 2
 0 -235.260100 -260.612552 -283.337957
 1 -249.391277 -285.740192 -284.710345
 2 -249.364002 -286.231335 -281.740291
 3 -268.356589 -311.350088 -304.220565
 4 -301.248802 -327.405054 -308.688321,
 'bic': 0 1 2
 0 -227.852535 -249.501204 -268.522827
 1 -238.279930 -270.925063 -266.191432
 2 -234.548873 -267.712423 -259.517596
 3 -249.837676 -289.127393 -278.294088
 4 -279.026107 -301.478576 -279.058062,
 'aic_min_order': (4, 1),
 'bic_min_order': (4, 1)}
```

```
In [29]: # p=4, q=1 が最善だったので、それをモデル化
from statsmodels.tsa.arima_model import ARIMA
ARIMA_4_1_1 = sm.tsa.ARIMA(ts, order=(4, 1, 1)).fit()
ARIMA_4_1_1.params
```

```
Out[29]: ar.L1      0.033606
ar.L2     -0.087843
ar.L3     -0.299241
ar.L4     -0.284038
ma.L1    -0.497597
sigma2    0.018816
dtype: float64
```

```
In [30]: # 残差のチェック
resid = ARIMA_4_1_1.resid
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(resid.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(resid, lags=40, ax=ax2)
# 7日間の周期性に関する残差がかなり抑えられているが、完全には除去できていないことがわかる。
```



SARIMAモデルの推定(季節性・周期性トレンドを入れた自己回帰予測モデル)

- 定常時系列の解析に使われるARMAモデル・SARIMAモデル

```
In [31]: import itertools

# 各パラメータの範囲を決める(自動ARMA推定の結果&グリッドサーチ)
p = range(4, 5)
q = range(1, 2)
d = range(0, 2)

sp = sd = sq = range(0, 2)

# p, d, q の組み合わせを列挙するリストを作成
pdq = list(itertools.product(p, d, q))
```

```
# P, D, Q の組み合わせを列挙するリストを作成すると同時に、後ろに s = 7 を決め打ちでつけている。
seasonal_pdq = [(x[0], x[1], x[2], 7) for x in list(itertools.product(sp, sc))]
```

In []: # pdq, seasonal_pdq を範囲内でグリッドサーチ(時間がかかる)

```
import warnings
warnings.simplefilter('ignore')

best_result = [0, 0, 10000000]
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.SARIMAX(ts,
                                  order=param,
                                  seasonal_order=param_seasonal,
                                  enforce_stationarity=True,
                                  enforce_invertibility=True)

            results = mod.fit(method='bfgs', maxiter=300)

            print('order{}, s_order{} - AIC: {}'.format(param, param_seasonal, results.aic))

            if results.aic < best_result[2]:
                best_result = [param, param_seasonal, results.aic]
        except:
            continue

print('\nAICが最も良いモデル:', best_result)
```

In [34]: # SARIMAモデル:自動ARMA推定の結果&seasonal_pdq グリッドサーチの結果、ならびに既知情報「SARIMA_4_1_1_101」

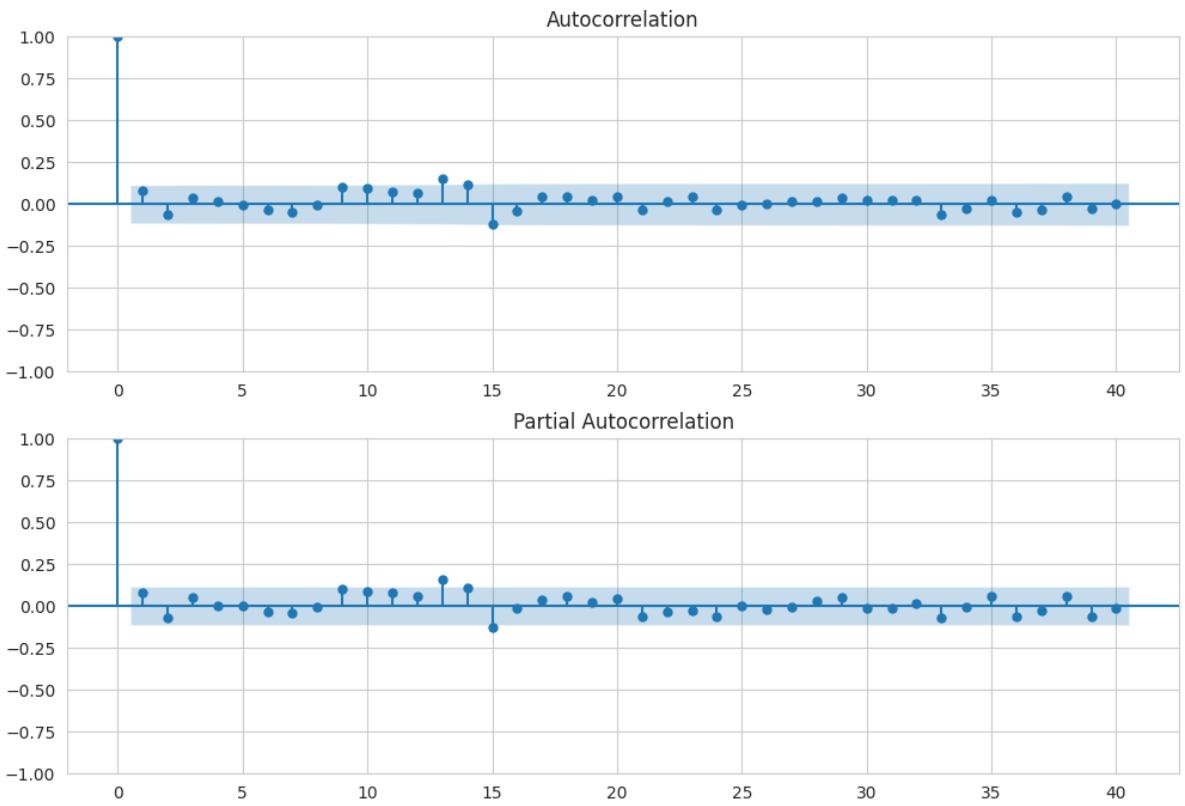
```
SARIMA_4_1_1_101 = sm.tsa.SARIMAX(ts, order=(4,1,1), seasonal_order=(1,0,1,7))

# AIC基準での評価結果
print(SARIMA_4_1_1_101.aic);
#print(SARIMA_4_1_1_101.summary())
```

```
Optimization terminated successfully.
    Current function value: -0.823860
    Iterations: 29
    Function evaluations: 35
    Gradient evaluations: 35
-478.3162922803077
```

In [35]: # 残差のチェック

```
residSARIMA = SARIMA_4_1_1_101.resid
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(residSARIMA.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(residSARIMA, lags=40, ax=ax2)
# 7日間の周期性に関する残差は、ほぼ完全には除去できていることがわかる。
```



```
In [36]: # 実験:301日から312日までを SARIMAモデルによって予測する
pred = SARIMA_4_1_1_101.predict(301, 312)
print(pred)
```

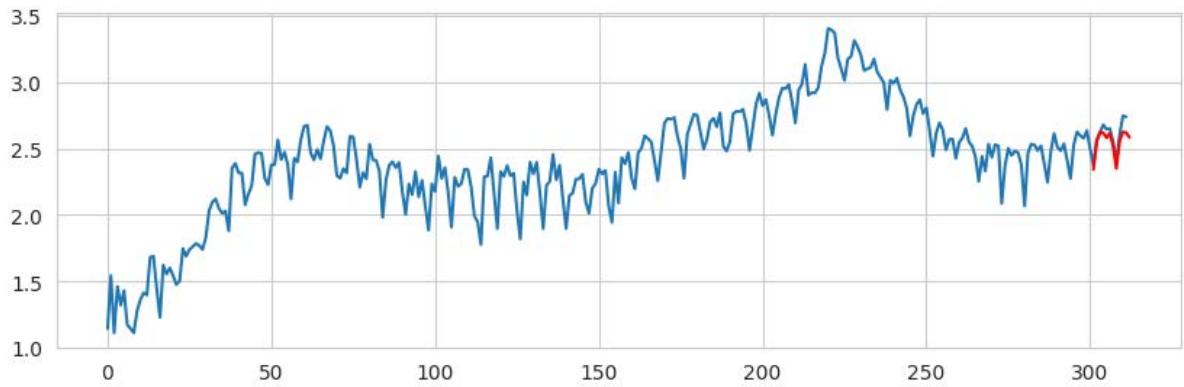
```
301    2.341660
302    2.550025
303    2.621476
304    2.617091
305    2.580526
306    2.613302
307    2.522645
308    2.349795
309    2.555436
310    2.623917
311    2.618820
312    2.582882
Name: predicted_mean, dtype: float64
```

```
In [37]: # 実データと予測結果の図示
plt.plot(ts_all) # 全てのデータ(301日目以降は実データ)
plt.plot(pred, "r") # 301日目以降の予測値

# 7日間周期の部分は、うまく予測できている。中期的なアップトレンドの部分は要調整。
```

```
Out[37]: [

```



memo