本書では、あまり使わないので、用語等については覚えなくても大丈夫ですが、これら以外にも、さまざまなアプローチ方法があるので、時間がある方は調べてみてください。

入力

```
# 計算実行
print(minimize_scalar(my_function, method = 'Brent'))
```

出力

```
     fun: 0.0
    nfev: 9
     nit: 4
 success: True
       x: -1.0000000000000002
```

Scipyは、積分や微分方程式などにも使えますが、この章では、いったんこれで終わりにします。Scipyを使った、さまざまな科学計算については、後の章で改めて説明します。

Let's Try

my_function関数の計算式を$f(x) = 0$から、さまざまな関数に変更して、最小値などの計算を実行してみましょう。

Practice

【練習問題 2-4】

以下の行列について、行列式を求めてください。

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 3 & 1 & 2 \end{pmatrix} \quad (式2\text{-}3\text{-}2)$$

【練習問題 2-5】

練習問題 2-4と同じ行列について、逆行列、固有値と固有ベクトルを求めてください。

【練習問題 2-6】

以下の関数が0となる解を、ニュートン法を用いて求めてみましょう。

$$f(x) = x^3 + 2x + 1 \quad (式2\text{-}3\text{-}3)$$

答えはAppendix 2

Chapter 2-4 Pandasの基礎

Keyword インデックス、Series、DataFrame、データの操作、データの結合、ソート

PandasはPythonでモデリングする（機械学習等を使う）前のいわゆる前処理をするときに便利なライブラリです。さまざまなデータのさまざまな加工処理をスムーズに柔軟に実施することができ、表計算やデータの抽出、検索などの操作ができるようになります。具体例を挙げると、データの中からある条件（男性だけ）を満たす行を抽出したり、ある軸（男女別など）を設定してそれぞれの平均値（身長、体重など）を算出したり、データを結合するなどの操作ができます。DB（データベース）のSQLに慣れている方には扱いやすいと思います。

2-4-1 Pandasのライブラリのインポート

ここでは、Pandasのライブラリをインポートします。

前述の「2.1.3 この章で使うライブラリのインポート」において、「import pandas as pd」としてPandasをインポートしているので、「pd.機能名」と表記することでPandasライブラリを使えるようになっています。

以下ではさらに、一次元の配列を扱うときのSeriesライブラリと、二次元の配列を扱うときのDataFrameライブラリをインポートします。

入力

```
from pandas import Series, DataFrame
```

2-4-2 Seriesの使い方

Seriesは1次元の配列のようなオブジェクトです。PandasのベースはNumpyのarrayです。以下に、Seriesオブジェクトに10個の要素を設定する、簡単な例を示します。

実行結果を見ると分かるように、Seriesオブジェクトをprintすると、2つの組の値が表示されます。先頭の10行文は要素のインデックスと値です。dtypeはデータの型です。

入力

```
# Series
sample_pandas_data = pd.Series([0,10,20,30,40,50,60,70,80,90])
print(sample_pandas_data)
```

出力

```
0     0
1    10
2    20
```

```
3    30
4    40
5    50
6    60
7    70
8    80
9    90
dtype: int64
```

インデックスは要素を特定するキーのことです。この例のように、[0, 10, 20, 30, 40,…]のようにSeriesオブジェクトに対して、値だけを指定した場合、インデックスは先頭から0、1、2…のように連番が付きます。
データの値とインデックスの値は、それぞれ次のように、valuesプロパティとindexプロパティを指定することで、別々に取り出すこともできます。

入力

```
# indexをアルファベットでつける
sample_pandas_index_data = pd.Series(
    [0, 10,20,30,40,50,60,70,80,90],
    index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'])
print(sample_pandas_index_data)
```

出力

```
a     0
b    10
c    20
d    30
e    40
f    50
g    60
h    70
i    80
j    90
dtype: int64
```

入力

```
print('データの値:', sample_pandas_index_data.values)
print('インデックスの値:', sample_pandas_index_data.index)
```

出力

```
データの値: [ 0 10 20 30 40 50 60 70 80 90]
インデックスの値: Index(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'], dtype='object')
```


2-4-3 DataFrameの使い方

DataFrameオブジェクトは2次元の配列です。それぞれの列で、異なるdtype(データ型)を持たせることもできます。
下記は、ID、City、Birth_year、Nameの4つの列を持つデータ構造を示した例です。print関数で表示すると、そのデータは表形式で表示されます。

入力

```
attri_data1 = {'ID':['100','101','102','103','104'],
               'City':['Tokyo','Osaka','Kyoto','Hokkaido','Tokyo'],
               'Birth_year':[1990,1989,1992,1997,1982],
               'Name':['Hiroshi','Akiko','Yuki','Satoru','Steve']}

attri_data_frame1 = DataFrame(attri_data1)

print(attri_data_frame1)
```

出力

```
    ID      City  Birth_year     Name
0  100     Tokyo        1990  Hiroshi
1  101     Osaka        1989    Akiko
2  102     Kyoto        1992     Yuki
3  103  Hokkaido        1997   Satoru
4  104     Tokyo        1982    Steve
```

一番左列に表示されている0, 1, 2, 3, 4の値は、インデックスの値です。DataFrameオブジェクトもSeriesオブジェクトと同様にインデックスを変更したり、インデックスとして文字を指定したりすることもできます。
次のようにインデックスを指定すると、attri_data_1の値に対して新しいインデックスを指定したattri_data_frame_index1というDataFrameオブジェクトを作ることができます(ここではDataFrameオブジェクトに対して操作しましたが、Seriesも同様の操作で、何か他のSeriesオブジェクトからインデックスを変更したSeriesオブジェクトを作ることができます)。

入力

```
attri_data_frame_index1 = DataFrame(attri_data1,index=['a','b','c','d','e'])
print(attri_data_frame_index1)
```

出力

```
   ID      City  Birth_year     Name
a  100     Tokyo        1990  Hiroshi
b  101     Osaka        1989    Akiko
c  102     Kyoto        1992     Yuki
d  103  Hokkaido        1997   Satoru
e  104     Tokyo        1982    Steve
```

2-4-3-1 Jupyter環境におけるデータ表示

ここまではSeriesオブジェクトやDataFrameオブジェクトを表示する際に、print(attri_data_frame_index1)のようにprint関数を使ってきました。しかしデータの変数を、そのまま次のように記述することで、表示することもできます。この場合、Jupyter環境によって、これがSeriesオブジェクトやDataFrameオブジェクトであることが認識され、罫線などが付いた見やすい表示になります。

以下では、この方法で表示していきます。

入力

```
attri_data_frame_index1
```

出力

	ID	City	Birth_year	Name
a	100	Tokyo	1990	Hiroshi
b	101	Osaka	1989	Akiko
c	102	Kyoto	1992	Yuki
d	103	Hokkaido	1997	Satoru
e	104	Tokyo	1982	Steve

2-4-4 行列操作

DataFrameは、さまざまな行列操作ができます。

2-4-4-1 転置

行列の転置のように、行と列を入れ替える場合には、.Tメソッドを使います。

入力

```
# 転置
attri_data_frame1.T
```

出力

	0	1	2	3	4
ID	100	101	102	103	104
City	Tokyo	Osaka	Kyoto	Hokkaido	Tokyo
Birth_year	1990	1989	1992	1997	1982
Name	Hiroshi	Akiko	Yuki	Satoru	Steve

2-4-4-2 特定列のみを取り出す

特定の列だけを指定したいときは、データの後にその列名を指定します。複数の列を指定したいときは、それらをPythonのリストの形式で指定します。

入力

```
# 列名の指定（1つの場合）
attri_data_frame1.Birth_year
```

出力

```
0    1990
1    1989
2    1992
3    1997
4    1982
Name: Birth_year, dtype: int64"
```


入力

```
# 列名の指定（複数の場合）
attri_data_frame1[['ID', 'Birth_year']]
```

出力

	ID	Birth_year
0	100	1990
1	101	1989
2	102	1992
3	103	1997
4	104	1982

2-4-5 データの抽出

DataFrameオブジェクトでは、特定の条件を満たすデータだけを取り出したり、複数のデータを結合したりすることもできます。

次の例は、データのうち、CityがTokyoのみのデータを抽出する例です。ここで指定している条件であるattri_data_frame1['City'] == 'Tokyo'は、dtypeがboolであるSeriesオブジェクトです。この処理は、attri_data_frame1['City'] == 'Tokyo'がTrueであるデータをすべてattri_data_frame1から抽出するもので、フィルターの役割を果たしています。

入力

```
#  条件（フィルター）
attri_data_frame1[attri_data_frame1['City'] == 'Tokyo']
```

出力

	ID	City	Birth_year	Name
0	100	Tokyo	1990	Hiroshi
4	104	Tokyo	1982	Steve

なお、条件部分である式はCity列の要素1つ1つとTokyoを比較しており、以下のようにそこだけ取り出して表示すると、TrueかFalseになっていることがわかります。

入力

```
attri_data_frame1['City'] == 'Tokyo'
```

出力

```
0     True
1    False
2    False
3    False
4     True
Name: City, dtype: bool"
```

条件を複数指定したいときは、次のようにisin(リスト)を使います。以下は、CityがTokyoかOsakaであるデータを抽出しています。この使い方は、あとの章でも使います。

入力

```
#  条件（フィルター、複数の値）
attri_data_frame1[attri_data_frame1['City'].isin(['Tokyo','Osaka'])]
```

出力

	ID	City	Birth_year	Name
0	100	Tokyo	1990	Hiroshi
1	101	Osaka	1989	Akiko
4	104	Tokyo	1982	Steve

Let's Try

他にも条件を変更(Birth_yearが1990未満など)して、フィルターを実行してみましょう。

2-4-6 データの削除と結合

DataFrameオブジェクトでは、必要のない列や行を削除したり、他のDataFrameオブジェクトと結合したりすることもできます。

2-4-6-1 列や行の削除

ある特定の列や行を削除するにはdropメソッドを実行します。axisパラメータに軸を指定します。「axis=0が行」「axis=1が列」です。なお、このaxisパラメータは他の場面でも使うので、覚えておいてください。

- **行削除の場合**：1つ目の引数に削除したい行のインデックスをリストとして指定します。axisパラメータには「0」を指定します。
- **列の削除の場合**：1つ目の引数に削除したい列名をリストとして指定します。axisパラメータには「1」を指定します。

次の例は、Birth_year列を削除する例です。

入力

```
attri_data_frame1.drop(['Birth_year'], axis = 1)
```

出力

	ID	City	Name
0	100	Tokyo	Hiroshi
1	101	Osaka	Akiko
2	102	Kyoto	Yuki
3	103	Hokkaido	Satoru
4	104	Tokyo	Steve

なお、上記で列を削除しても元のデータの列が削除されたわけではないので、注意しましょう。置き換えたい場合は、あらためてattri_data_frame1 = attri_data_frame1.drop(['Birth_year'],axis=1)のように設定します。もしくは、オプションのinplace=Trueをパラメータとして指定すると、元のデータを置き換えることもできます。

2-4-6-2 データの結合

DataFrameオブジェクト同士は結合できます。データ分析ではさまざまなデータがある場合に、それらを結合して分析することは多々ありますから、実行できるようになりましょう。まずは例として、結合先のDataFrameオブジェクトを、次のようにattri_data_frame2という変数で用意します。

入力

```
# 別のデータの準備
attri_data2 = {'ID':['100','101','102','105','107'],
               'Math':[50,43,33,76,98],
               'English':[90,30,20,50,30],
               'Sex':['M','F','F','M','M']}
attri_data_frame2 = DataFrame(attri_data2)
attri_data_frame2
```

出力

	ID	Math	English	Sex
0	100	50	90	M
1	101	43	30	F
2	102	33	20	F
3	105	76	50	M
4	107	98	30	M

そして、これまで使ってきたattri_data_frame1と、このattri_data_fame2を結合してみます。
結合するにはmergeメソッドを使います。キーを明示しないときは、自動で同じキーの値であるものを見つけて結合します。この例の場合、キーはIDです。100、101、102が共通であるため、それが合致するデータが結合されます。

入力

```
# データのマージ（内部結合、詳しくは6章で）
pd.merge(attri_data_frame1,attri_data_frame2)
```

出力

	ID	City	Birth_year	Name	Math	English	Sex
0	100	Tokyo	1990	Hiroshi	50	90	M
1	101	Osaka	1989	Akiko	43	30	F
2	102	Kyoto	1992	Yuki	33	20	F

2-4-7 集計

DataFrameオブジェクトでは、データを集計することもきます。
さらにgroupbyメソッドを使うと、ある特定の列を軸とした集計ができます。以下は「Sexの列」を軸として、数学のスコア平均を算出する例です。スコア平均を計算するにはmeanメソッドを使います。ほかにも、最大値を計算するmaxメソッドや最小値を計算するminメソッドなどもあります。

入力

```
# データのグループ集計(詳しくは次の章で)
attri_data_frame2.groupby('Sex')['Math'].mean()
```

出力

```
Sex
F    38.000000
M    74.666667
Name: Math, dtype: float64"
```

Let's Try

他にも変数を変えて、実行してみましょう。集計対象をEnglishにした場合はどうなりますか。また、最大値や最小値を求めてみましょう。

2-4-8 値のソート

SeriesオブジェクトやDataFrameオブジェクトのデータは、ソートすることもできます。値だけではなく、インデックスをベースにソートできます。まずはソート対象のサンプルデータを次のように定義します。ソートの効果がわかりやすくなるよう、わざとデータを適当な順で並べてあります。

入力

```
# データの準備
attri_data2 = {'ID':['100','101','102','103','104'],
               'City':['Tokyo','Osaka','Kyoto','Hokkaido','Tokyo'],
               'Birth_year':[1990,1989,1992,1997,1982],
               'Name':['Hiroshi','Akiko','Yuki','Satoru','Steve']}
attri_data_frame2 = DataFrame(attri_data2)
attri_data_frame_index2 = DataFrame(attri_data2,index=['e','b','a','d','c'])
attri_data_frame_index2
```

出力

	ID	City	Birth_year	Name
e	100	Tokyo	1990	Hiroshi
b	101	Osaka	1989	Akiko
a	102	Kyoto	1992	Yuki
d	103	Hokkaido	1997	Satoru
c	104	Tokyo	1982	Steve

インデックスでソートするには、次のようにsort_indexメソッドを実行します。

入力

```
# indexによるソート
attri_data_frame_index2.sort_index()
```

出力

	ID	City	Birth_year	Name
a	102	Kyoto	1992	Yuki
b	101	Osaka	1989	Akiko
c	104	Tokyo	1982	Steve
d	103	Hokkaido	1997	Satoru
e	100	Tokyo	1990	Hiroshi

値でソートする場合には、次のようにsort_valuesメソッドを使います。

入力

```
# 値によるソート、デフォルトは昇順
attri_data_frame_index2.Birth_year.sort_values()
```

出力

```
c    1982
b    1989
e    1990
a    1992
d    1997
Name: Birth_year, dtype: int64"
```


2-4-9 nan (null) の判定

データ分析ではデータが欠損しており、該当のデータが存在しないことがあります。それらをそのまま計算すると、平均などを求めたときに正しい値が得られないので、除外するなどの操作が必要です。欠損値などのデータはnanという特別な値で格納されるので、その扱いについて補足します。

2-4-9-1 条件に合致したデータの比較

まずはnanの話ではなく、ふつうに条件検索する例から説明します。

次の例は、attri_data_frame_index2の全要素を対象に、Tokyoという文字列があるかどうかをisinで調べる例です。その結果は、それぞれのセルにTrueかFalseが返されます。入っていれば(条件を満たしていれば)True、入っていなければ(条件を満たしていなければ)Falseが設定されます。この操作が、条件に合致するデータを探すときの基本です。

入力

```
# 値があるかどうかの確認
attri_data_frame_index2.isin(['Tokyo'])
```

出力

	ID	City	Birth_year	Name
e	False	True	False	False
b	False	False	False	False
a	False	False	False	False
d	False	False	False	False
c	False	True	False	False

2-4-9-2 nanとnullの例

次の例は、Name列の値をわざとnanに設定した例です。nanかどうかを判定するにはisnullメソッドを使います。

入力

```
# 欠損値の取り扱い
# name をすべてnanにする
attri_data_frame_index2['Name'] = np.nan
attri_data_frame_index2.isnull()
```

出力

	ID	City	Birth_year	Name
e	False	False	False	True
b	False	False	False	True
a	False	False	False	True
d	False	False	False	True
c	False	False	False	True

そしてnanであるものの総数を求めるには、次のようにします。Nameが5になっているのは、上記の結果でわかるように、Trueが5つあるため、それをカウントしているからです。

入力

```
# nullを判定し、合計する
attri_data_frame_index2.isnull().sum()
```

出力

```
ID          0
City        0
Birth_year  0
Name        5
dtype: int64"
```

以上で、Pandasの簡単な説明は終わりです。3章では実際のデータの加工処理をしていきますので、ここで学んだことはしっかりと身につけてください。

Practice

【練習問題 2-7】

以下のデータに対して、Moneyが500以上の人を絞り込んで、レコードを表示してください。

入力

```
from pandas import Series,DataFrame
import pandas as pd

attri_data1 = {'ID':['1','2','3','4','5'],
               'Sex':['F','F','M','M','F'],
               'Money':[1000,2000,500,300,700],
               'Name':['Saito','Horie','Kondo','Kawada','Matsubara']}
attri_data_frame1 = DataFrame(attri_data1)
```

【練習問題 2-8】

練習問題 2-7のデータに対して、男女別（MF別）の平均Moneyを求めてください。

【練習問題 2-9】

練習問題2-7のデータに対して、以下のデータの同じIDの人をキーとして、データをマージしてください。そして、MoneyとMathとEnglishの平均を求めてください。

入力

```
attri_data2 = {'ID':['3','4','7'],
               'Math':[60,30,40],
               'English':[80,20,30]}

attri_data_frame2 = DataFrame(attri_data2)
```

答えはAppendix 2

Chapter 2-5 Matplotlibの基礎

Keyword データビジュアライゼーション、散布図、ヒストグラム

データ分析をする上で、対象となるデータを可視化することはとても重要です。単に数字を眺めているだけでは、データに潜む傾向がなかなか見えなかったりしますが、データをビジュアル化することで、データ間の関係性なども見えてきます。特に、近年はインフォグラフィックスなどといって、可視化が注目されています。

ここでは、主にMatplotlibとSeabornを使って、データを可視化する基本的な方法を身につけましょう。巻末の参考URL「B-5」が参考になります。

2-5-1 Matplotlibを使うための準備

前述の「2.1.3 この章で使うライブラリのインポート」において、MatplotlibとSeabornをすでにインポートしています。Matplotlibでは、描画に関するほとんどの機能が「pyplot.機能名」で提供されています。そこで「2-1-3 この章で使うライブラリのインポート」では「import matplotlib.pyplot as plt」とインポートし、「plt.機能名」と略記できるようにしています。

SeabornはMatplotlibのグラフを、さらにきれいにするライブラリです。インポートするだけでグラフがきれいになり、また、いくつかの追加のスタイルを指定できるようになります。

以下の「%matplotlib inline」は、Jupyter Notebook上にグラフを表示するためのマジックコマンドです。Jupyter環境の初学者の方はグラフを書くときに忘れやすいので、注意しましょう。

入力

```
# Matplotlib と Seabornの読み込み
# Seabornはきれいに図示できる
import matplotlib as mpl
import seaborn as sns

# pyplotにはpltの別名で実行できるようにする
import matplotlib.pyplot as plt

# Jupyter Notebook上でグラフを表示させるために必要なマジックコマンド
%matplotlib inline
```

2-5-2 散布図

Matplotlibでは、さまざまなグラフを描けますが、まずは、散布図から始めましょう。散布図は、2つの組み合わせデータに対して、x-y座標上に点をプロットしたグラフです。plt.plot(x, y, 'o')で描写でき、最後の引数はグラフの形状を指定するもので'o'は点で描くという意味です。その他の動作については、コード中のコメントを参考にしてください。