グラフ描画ライブラリMatplotlib

ダウンロード・データあります

佐藤 聖

Matplotlibは，Pythonで頻繁に利用されるグラフ作成ライブラリです．数値データを元に折れ線グラフ，棒グラフ，ヒストグラムや散布図など，多種多様なグラフを作れます．データをきれいなグラフで可視化できるので自動レポート作成プログラムを作るときに重宝します．

以下の画像フォーマットでグラフを保存できます．

```
eps, jpeg, jpg, pdf, pgf, png, ps, raw, rgba,
svg, svgz, tif, tiff
```

● インストール

Raspberry Pi OSには標準インストールされていないので，下記のコマンドを実行してインストールします．

```
sudo apt-get update↵
sudo apt-get -y upgrade↵
sudo␣apt-get␣install␣-y␣
                python3-matplotlib↵
```

apt-getコマンドでは，Raspberry Pi OSのリポジトリに登録されている最新版がインストールされます．

10-1 折れ線グラフ…時系列データの変動を可視化しやすい

● 折れ線グラフの特徴

折れ線グラフは，Line2Dクラスのplot関数を使って描画します．あるデータが時系列に沿ってどのように変化しているか，その傾向をつかむのに適しています．

● 折れ線グラフの書式

折れ線グラフの書式は次になります．

```
plot(*args, scalex=True, scaley=True,
data=None, **kwargs)
```

● 引数

plot関数は，可変長の引数を渡すのに*argsを利用します．

▶必須の引数

通常は，設定が必須な引数はx軸とy軸の値になります．

▶オプションの引数

・scalexとscaley

通常はデフォルトのTrueで利用します．このパラメータはビュー制限がデータ制限に適合しているかどうかを決定します．

・data

デフォルトでNoneが設定されています．dist型(辞書型)オブジェクトをデータとして扱う場合，x軸とy軸でデータを指定する代わりに，dataパラメータでオブジェクトを指定し，x軸とy軸のラベルを指定できます．

・kwargs**

グラフの見た目を細かく調整するためにLine2Dクラスのプロパティやオプションも利用できます．詳細は省略しますが，Line2Dクラスには41種類の引数が用意されており，線，マーカ，枠線，グリッド線，凡例などを自由に変更できます．クラスの主要な引数を**表1**に示します．

表1 線の見た目を設定するLine2Dクラスの主要な引数

引　数	意　味
xdata(必須)	x軸方向の数値
ydata(必須)	y軸方向の数値
linewidth	線の太さ
linestyle	線の種類．solid(実線)，dashed(破線)，dashdot(破線＆点線)，dotted(点線)から選択できる．デフォルトはsolid
color	線の色
marker	マーカの種類(デフォルト：None)
markersize	マーカの大きさ

● 実行例

▶折れ線グラフを描画

図1の1行目でnumpyライブラリを読み込み，npの別名を付けています．2行目で同じようにmatplotlibライブラリのpyplotクラスを読み込んでpltの別名を付けています．

3行目でx軸の値をnumpyの配列に変換して変数xに代入しています．

4行目でy軸の値をNumPyの配列に変換して変数yに代入しています．これらのデータを使ってグラフを描画します．

5行目で画像ファイルに書き出すため，インスタンスを作成して変数figに代入します．

6行目でplot関数にプロットするデータとして変数xと変数yを指定し，プロット・オブジェクトが作成されると7行目のようにメッセージが表示されます．プロット・オブジェクトを画面に出力するときには，show関数を使いplt.show()を実行します．

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1, 2, 3, 4, 5])
y = np.array([10, 50, 30, 20, 40])
fig = plt.figure()
plt.plot(x, y)
```

5行目

7行目

```
[<matplotlib.lines.Line2D at 0x7f78c5f9e090>]
```

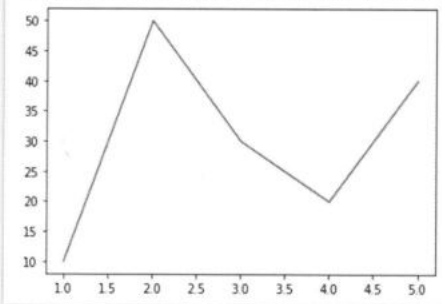


図1　時系列での変化が分かりやすい折れ線グラフ1

・グラフの保存とインスタンスの解放

pngファイルに出力するため，**図2**の1行目でsavefig関数にファイルの出力先のパス(省略するとカレント・ディレクトリ)とファイル名を指定します．

2行目のclose関数でインスタンスを閉じます．インスタンスを閉じないとメモリにデータが残り続けます．Pythonを終了するまで，そこで使っていたメモリが解放されなくなります．

```
fig.savefig("plot1.png")
plt.close()
```

生成したグラフは，画像ファイルとして保存できます．他のアプリケーションに取り込み，レポートに使用することがよくあります

図2　時系列での変化が分かりやすい折れ線グラフ2…グラフの保存

▶折れ線グラフの見た目を調整

線を太く赤色にし，描画領域にグリッド線も追加した折れ線グラフを作ります．

図3の1～4行目は上記の例と同じです．

6行目でgrid関数の引数にTrueを指定してグリッド線を有効に指定します．このようにするとグラフが読みやすくなります．

7行目のplot関数の引数linewidthを4にして線を太くし，引数colorをredにして線を赤色にしています(**図4**)．実はこれらはplot関数の引数ではなく，Line2Dクラスの機能を呼び出して利用しています．

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1, 2, 3, 4, 5])
y = np.array([10, 50, 30, 20, 40])
fig = plt.figure()
plt.grid(True)
plt.plot(x, y, linewidth=4, color="red")
```

7行目

図3　時系列での変化が分かりやすい折れ線グラフ3…グラフの見た目を調整

```
[<matplotlib.lines.Line2D at 0x7ff8e3651c10>]
```

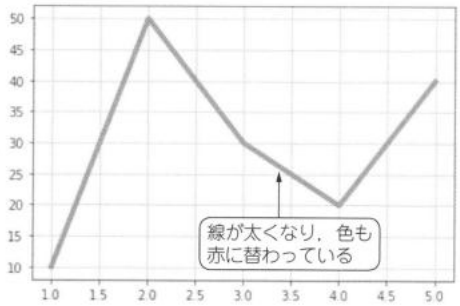


図4　時系列での変化が分かりやすい折れ線グラフ4…線の太さと色を変更

ここで紹介した以外にも，matplotlibの各種関数を使うことで，タイトル，軸名，凡例，軸の間隔なども追加したり，調整したりできます．

図5の1行目で画像ファイルを保存し，2行目でインスタンスを閉じています．

```
fig.savefig("plot2.png")
plt.close()
```

図5　時系列での変化が分かりやすい折れ線グラフ5…インスタンスを閉じる

▶複数の折れ線グラフを描画

先ほどの折れ線グラフのソースコードをベースにして，種類別にplot関数を記述することで，複数の線を描画した折れ線グラフを作ることができます．

図6の1～5行目は上記の例と同じです．

7行目以降のplot関数の引数linestyleに指定するものによって，線の種類を変更しています．変更箇所は次の通りです．

- 7行目：solid
- 8行目：dashed
- 9行目：dashdot

1つのグラフにたくさんの要素をプロットするには，要素数分plot関数を記述します．この部分は，for文でループ処理にするなど応用ができます．

図7の1行目で画像ファイルを保存し，インスタンスを閉じています．

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1, 2, 3, 4, 5])
y = np.array([10, 50, 30, 20, 40])
fig = plt.figure()
plt.grid(True)
plt.plot(x, y, linestyle="solid")
plt.plot(x, y/2, linestyle="dashed")
plt.plot(x, y/3, linestyle="dashdot")
plt.plot(x, y, marker="o")
```

7行目

```
[<matplotlib.lines.Line2D at 0x7ff8e3612b90>]
```

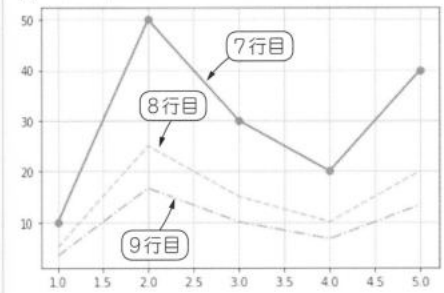


図6　時系列での変化が分かりやすい折れ線グラフ6…複数のグラフ

複数データをグラフに描いて比較するときに使用します．
データを入力すると定型デザインのグラフを生成できるので，グラフ生成を繰り返すときにプログラムを作成すると，定型作業として処理できます

```
fig.savefig("plot3.png")
plt.close()
```

図7　時系列での変化が分かりやすい折れ線グラフ7…インスタンスを閉じる

10-2　ヒストグラム…データの分布を可視化しやすい

●ヒストグラムの特徴

ヒストグラムはhist関数で描画できます．ヒストグラムは全体の分布を，段階ごとに切って見るのに適しています．

●hist関数の書式

hist関数の書式は次になります．

```
hist()←引数は表2に示します
```

matplotlib.pyplotクラスに含まれるhist関数には17種類の引数があります．主要な引数を**表2**に示します．

●実行例

初めに基本的なヒストグラムを作成します．

▶ヒストグラム表示

図8の1～2行目でNumPyライブラリを読み込み，npの別名を付けています．同じようにmatplotlibライブラリのpyplotクラスを読み込み，pltの別名を付けました．

3行目でNumPyライブラリのrandom関数を使って，平均50，標準偏差10の正規乱数を1,000件生成して変数xに格納しています．このデータを使ってヒストグラムを作ります．

4行目で画像ファイルに書き出すため，インスタンスを作成して変数figに代入しています．

5行目でhist関数に変数xを渡しています．この関数は階級数を指定しないとデフォルトの10本のバーでヒストグラムを作成します(**図9**)．

プロット・オブジェクトが作成されると6行目のように変数xのデータが表示されます．

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.normal(50, 10, 1000)
fig = plt.figure()
plt.hist(x)
```

3行目

図8　全体の分布が分かりやすいヒストグラム1…ソースコード

表2　hist関数の主要な引数

引　数	意　味
x	ヒストグラムを作成するための生データの配列(必須)
bins	表示する棒(ビン)の数(階級数)．デフォルトは10
range	ビンの最小値と最大値を指定．デフォルトはx.min()，x.max()の値
bottom	各棒の下側の余白を数値または配列で指定
histtype	bar(デフォルト)，barstacked(積み上げヒストグラム)，step(線)，stepfilled(塗りつぶしありの線)から選択

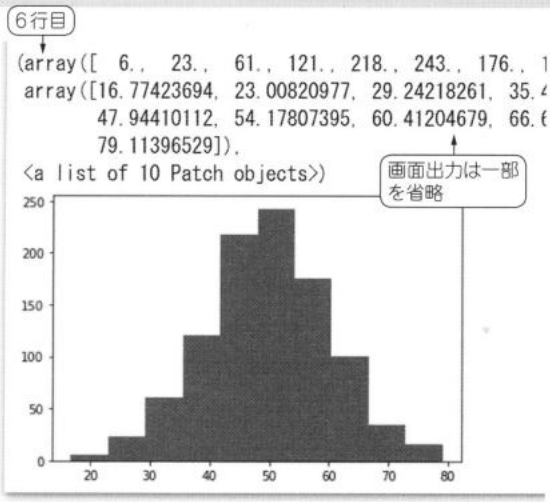


図9　全体の分布が分かりやすいヒストグラム1…グラフ

・グラフの保存とインスタンスの解放

図10の1行目でsavefig関数にファイルの出力先のパス（省略するとカレント・ディレクトリ）とファイル名を指定します．

2行目のclose関数でインスタンスを閉じます．

```
fig.savefig("hist1.png")
plt.close()
```

図10　全体の分布が分かりやすいヒストグラム1…グラフを画像として保存

▶ヒストグラムの範囲を調整

横軸の上限と下限を指定したヒストグラムを作成します．

図11の1～4行目は上記のプログラムと同じです．

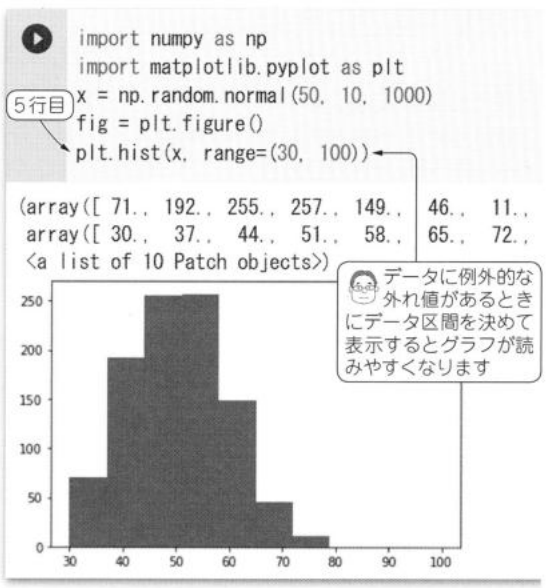


図11　全体の分布が分かりやすいヒストグラム2…データ区間を設定

・グラフの保存とインスタンスの解放

図12の1～2行目で画像ファイルを保存し，インスタンスを閉じます．

```
fig.savefig("hist2.png")
plt.close()
```

図12　全体の分布が分かりやすいヒストグラム2…グラフを画像として保存

5行目のhist関数の引数rangeに下限値と上限値を代入します．

▶ヒストグラムの見た目を調整

図13の4行目までは上記と同じです．

5行目のhist関数の引数rwidthで棒の幅として0.7，引数colorでorangeを指定しています．

ここでは紹介していませんが，階級数，正規化の有無，累積ヒストグラム出力，対数目盛りの表示，凡例などを引数で指定することもできます．

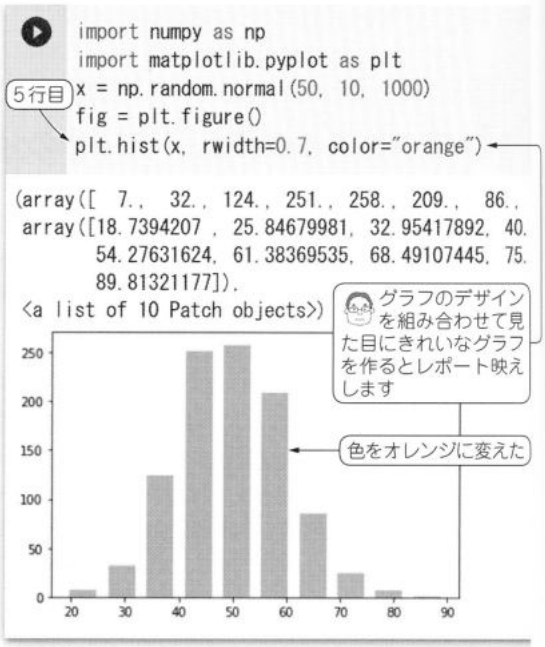


図13　全体の分布が分かりやすいヒストグラム3…グラフの見た目を調整

・グラフの保存とインスタンスの解放

図14の1～2行目で画像ファイルを保存し，インスタンスを閉じます

```
fig.savefig("hist3.png")
plt.close()
```

図14　全体の分布が分かりやすいヒストグラム3…グラフを画像として保存

10-3　散布図…データの相関を可視化しやすい

● 散布図の特徴

2つのデータの相関を見るのに適したグラフである散布図は，scatter関数を使って描画します．2つの値に正の相関があるときには，散布図は右肩上がりの線形に近い形を描きます．

● scatter関数の書式

scatter関数の書式は次になります．

```
scatter()←引数は表3に示します
```

matplotlib.pyplotクラスに含まれるscatter関数には17種類の引数があります．関数の主要な引数を**表3**に示します．

● 実行例

▶簡単な散布図

図15の1行目でmatplotlibライブラリのpyplotクラスを読み込み，pltの別名を付けています．

2行目でリストを変数xに，3行目でリストを変数yに代入しています．これらのデータを使って散布図を描画します．

4行目で画像ファイルに書き出すため，インスタンスを作成して変数figに代入しています．

5行目でsubplotインスタンスを生成しています．代入している1，1，1は，それぞれ行数，列数，1番目のプロットを表しています．このサブ・インスタンス作成を工夫すると，複数のグラフを行列に並べて表示できます．

6行目にmatplotlibライブラリのpyplotクラスに含まれるscatter関数に変数x，変数yを渡しています．引数cにbを代入し，色を青に指定しています．

プロット・オブジェクトが作成されると，7行目のようにメッセージが表示されます．

```
import matplotlib.pyplot as plt
x = [1, 2, 3]                                    ← 4行目
y = [2, 4, 6]
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)                    ← 複数のグラフを行列に並べて表示できます
ax.scatter(x, y, c='b')                          ← メッセージは省略している
<matplotlib.collections.PathCollection at 0x
```

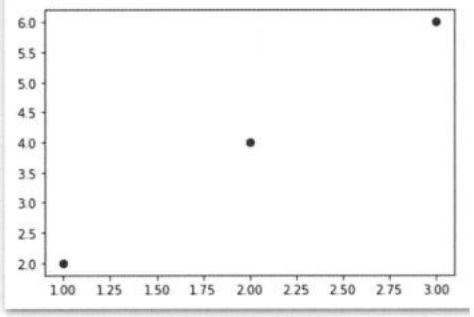


図15　データの相関が分かりやすい散布図1…簡単な散布図

• グラフの保存とインスタンスの解放

図16の1行目でsavefig関数にファイルの出力先のパス（省略するとカレント・ディレクトリ）とファイル名を指定し，作った画像を保存します．

2行目のclose関数でインスタンスを解放します．

```
fig.savefig("scatter1.png")
plt.close()
```

図16　データの相関が分かりやすい散布図1…グラフを画像として保存

▶散布図の見た目を調整

マーカのサイズを大きくしたり，グリッド線を加えたりすれば，グラフの読みやすさが向上します．

次にプロット・マーカの色や透明度，線の太さや色を指定し，グリッド線を追加した散布図を作成します．

図17の1～2行目はNumPyライブラリとmatplotlibライブラリの読み込みです．

3～4行目でNumPyライブラリのrandom関数のサブ関数であるrandに30を渡して，0.0以上1.0未満の範囲からランダムに30個の値を返します．値は変数x1と変数y1にそれぞれ代入します．この2つの変数に格納されたx軸とy軸の座標を使って散布図をプロットします．

5～6行目でpyplotインスタンスとsubplotインスタンスを生成しています．

ここからが先ほどの散布図と異なる引数の設定になります．

表3　scatter関数の主要な引数

引　数	意　味
x, y	x軸，y軸のデータ配列
s	マーカ・サイズ
c	色
marker	マーカの形
alpha	マーカの透明度 （0～1を指定，0：完全透明，1：不透明）
linewidths	マーカのエッジ線の太さ
edgecolors	マーカのエッジ線の色

7行目でmatplotlibライブラリのpyplotクラスに含まれるscatter関数の引数(**表3**)を設定しています.

```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.random.rand(30)
y1 = np.random.rand(30)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(x1, y1, s=300, alpha=0.5,
           linewidths='2', c='#aaaaFF', edgecolor
plt.grid(True)
```

3行目

6行目

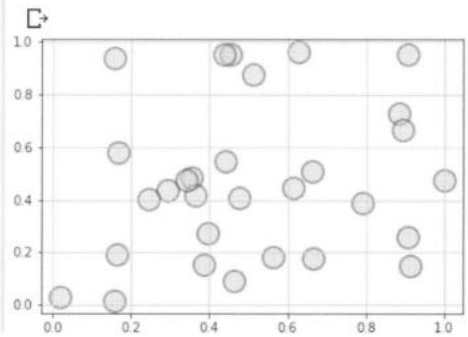


図17　データの相関が分かりやすい散布図2…見た目を調整

・グラフの保存とインスタンスの解放

図18の1～2行目で画像ファイルを保存し，インスタンスを閉じています.

```
fig.savefig("scatter2.png")
plt.close()
```

図18　データの相関が分かりやすい散布図2…グラフを画像として保存

▶2つのデータを1つの散布図に描画

上の散布図をベースに，2つのデータを1つの散布図に描画することもできます.

図19の3～4行目で1つ目のデータを，5～6行目で2つ目のデータを用意しています.

7～8行目は上記のソースコードと同様です.

9行目で1つ目のデータをscatter関数でプロットしています.

```
import numpy as np
import matplotlib.pyplot as plt
x1 = np.random.rand(30)
y1 = np.random.rand(30)
x2 = np.random.rand(30)
y2 = np.random.rand(30)
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(x1, y1, s=300, alpha=0.5,
           linewidths='2', c='#aaaaFF', edgecolors=
plt.grid(True)
```

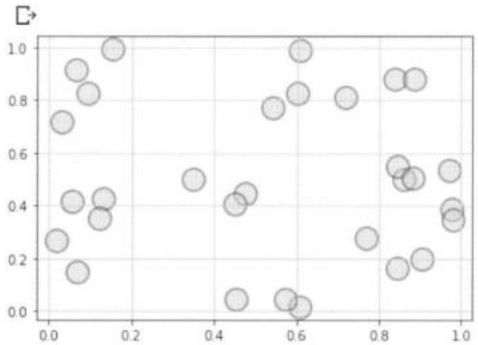


図19　データの相関が分かりやすい散布図3…2つのデータを描画

・グラフの保存とインスタンスの解放

図20の1行目で2つ目のデータをscatter関数でプロットし，オブジェクトを作成しています.

これらは引数cの色指定だけ異なります．このようにデータごとにマーカを変えるときには，scatter関数を別々に記述する必要があります.

```
ax.scatter(x2, y2, s=300, alpha=0.5,
           linewidths='2', c='#FFaaaa', edgecolors='r

<matplotlib.collections.PathCollection at 0

fig.savefig("scatter3.png")
plt.close()
```

図20　データの相関が分かりやすい散布図3…グラフを画像として保存

10-4　棒グラフ…同じ尺度の複数データを並べて比べやすい

● bar関数の棒グラフの特徴

棒グラフは，bar関数を使って作成できます．同じ尺度の複数のデータを並べて比較するのに適しています.

どのデータがどれだけ頭1つ抜き出ているか視覚的に把握できます.

● bar関数の書式

bar関数の書式は次になります.

```
matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)
```

matplotlib.pyplotクラスに含まれるbar関数の主要な引数を次に示します.

- **x**
  棒グラフを描画するためのデータとして，floatまたは配列を指定する.
- **height**
  バーの高さを指定する.
- **width**
  バーの幅を指定する.
- **bottom**
  バーベースのy座標を指定する.
- **align**
  バーのx座標への配置方法を指定する.

● 実行例

初めに簡単な棒グラフを作成します.

図21の1行目でmatplotlibライブラリのpyplotクラスを読み出し，pltの別名を付けています.

2行目でリストを変数xに，3行目でもリストを変数yに代入しています．このデータを使って棒グラフを描画します.

4行目で画像ファイルに書き込むため，インスタンスを作成して変数figに代入しています.

6行目では変数x，変数yを引数にして，matplotlibライブラリのpyplotクラスに含まれるbar関数を呼び出しています.

プロット・オブジェクトが作成されると7行目のようにメッセージが表示されます.

8行目でsavefig関数にファイルの出力先のパス(省略するとカレント・ディレクトリ)とファイル名を指定します.

9行目のclose関数でインスタンスを閉じています.

また，x軸とy軸のラベルは使用したデータから自動的に割り当てられたものが使用され，画像内のバーの高さや幅も自動的に調整されています．これらは，matplotlibの他の関数を利用することで調整できます.

▶棒グラフの装飾その1

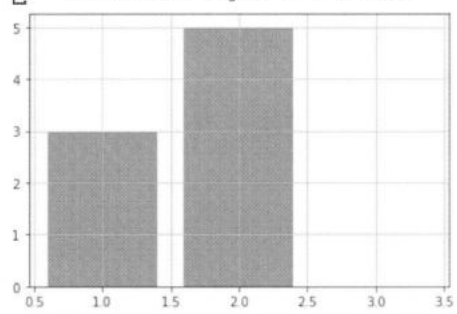
グラフを読みやすくするため，バーの色を赤に変え，グリッド線を加えてみます.

図22の1～4行目のソースコードにこれまでと変更はありません.

```
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [3, 5, 0]
fig = plt.figure()
plt.grid(True)
plt.bar(x, y, color='r')
```

7行目

```
<BarContainer object of 3 artists>
```




```
fig.savefig("bar2.png")
plt.close()
```

8行目

図21　同じ尺度のデータを比べやすい棒グラフ1…簡単な棒グラフ

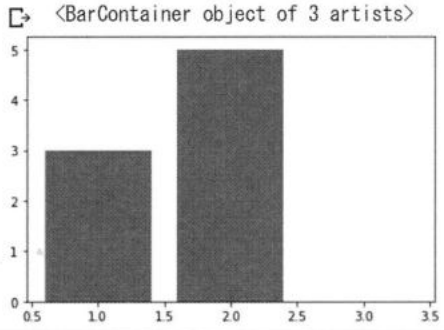
5行目でbar関数の引数colorにrを設定して，バーの色を赤色にします.

7行目でgrid関数にTrueを代入して，グリッド線が描画されるようにします．以降は，画像ファイルの保存とインスタンスを閉じる処理です.

```
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [3, 5, 0]
fig = plt.figure()
plt.bar(x, y, color='r')
```

5行目

```
<BarContainer object of 3 artists>
```




```
plt.grid(True)
fig.savefig("bar2.png")
plt.close()
```

7行目

図22　同じ尺度のデータを比べやすい棒グラフ2…グラフの装飾

▶棒グラフの装飾その2

上記グラフのソースコードをベースにして，2つの要素を描画した棒グラフを作ります．

図23の1行目でmatplotlibライブラリのpyplotクラスを読み込みます．

2～3行目で1つ目の棒グラフのx軸とy軸のデータを変数に代入しています．

4～5行目でも2つ目のデータをそれぞれの変数に代入しています．これらのデータを使って棒グラフを描画します．

6行目で画像ファイルに書き込むため，インスタンスを作成して変数figに代入し，7行目で1つ目のバーを，8行目で2つ目のバーの描画を作成します(画面出力は省略している)．

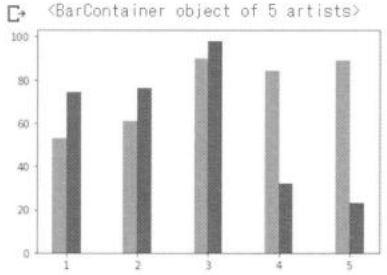
9行目でタイトル，10行目でx軸のラベル，11行目でy軸のラベルをグラフに描画しています．それぞれの関数を実行後，その下の行に返り値が表示されています．以降，画像ファイルの保存とインスタンスを閉じる処理です．

このようにすることで複数データのバーを描画した棒グラフを作ることができます．

この例では，バーのx軸座標をバーの幅を考慮して計算で求めるようにする応用が考えられます．また，x軸とy軸のデータ，バーの色などをパラメータとして，for文を使ってbar関数を繰り返し実行するようなプログラムに書き換えると，多数要素を描画するグラフを短いソースコードで記述できるかもしれません．

```
import matplotlib.pyplot as plt
x1 = [0.9, 1.9, 2.9, 3.9, 4.9]
y1 = [53, 61, 90, 84, 99]
x2 = [1.1, 2.1, 3.1, 4.1, 5.1]
y2 = [74, 76, 98, 32, 23]
fig = plt.figure()
plt.bar(x1, y1, color='r', width=0.2, align='center')
plt.bar(x2, y2, color='b', width=0.2, align='center')
```

```
<BarContainer object of 5 artists>
```




```
plt.title("Test score")
plt.xlabel("Subject")
plt.ylabel("Score")
fig.savefig("bar3.png")
plt.close()
```

グラフの定型デザインに入力データを当てはめて，プログラムで繰り返しグラフ生成できます．使い続けられるグラフ・デザインを考えてプログラムを作成します

図23　同じ尺度のデータを比べやすい棒グラフ3…複数のグラフ

10-5　円グラフ…ある量に占める内訳を可視化しやすい

● 円グラフの特徴

円グラフは，pie関数で描画できます．ある量に占める内訳や構成を直観的に知るために適した方法の1つです．

● pie関数の書式

pie関数の書式は次になります．

```
pie()←(引数は表4に示します)
```

matplotlib.pyplotクラスに含まれるpie関数には18種類の引数があります．関数の主要な引数を**表4**に示します．

● 実行例

▶基本の円グラフ

最もベーシックな円グラフを作成します(**図24**)．

図27の1～2行目でNumPyとmatplotlibライ

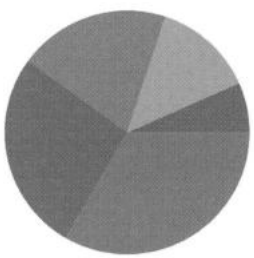


図24　デフォルトでは，値が反時計回りにプロットされる

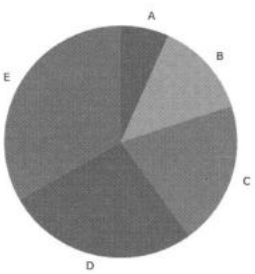


図25　ラベル付きの円グラフ

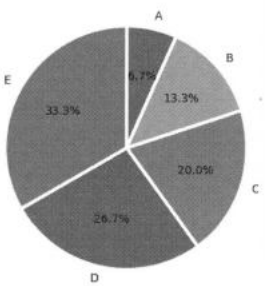


図26　円グラフを細かく調整できる

表4　pie関数の主要な引数

引　数	意　味
x	各要素の大きさを配列で指定
labels	各要素のラベル
colors	各要素の色を指定
labeldistance	ラベルを表示する位置．円の中心(0.0)から円周(1.0)を目安に指定(デフォルト値：1.1)
startangle	各要素の出力を開始する角度(デフォルト値：None)
counterclock	Trueに設定すると反時計回りで出力．Falseに設定すると時計回りで出力(デフォルト値：True)
wedgeprops	ウェッジ(くさび形の部分)に関する指定．枠線の太さなどを設定可能(デフォルト値：None)

ブラリを読み出します．

3行目でグラフにする値をNumPyの配列を作成して変数xに代入します．

4行目で画像ファイルに書き込むため，インスタンスを作成して変数figに代入します．

5行目でpie関数に変数xを渡して，プロット・オブジェクトが作成されると画面出力にメッセージが表示されます．

6行目でsavefig関数にファイルの出力先のパス(省略するとカレント・ディレクトリ)とファイル名を指定します．

7行目のclose関数でインスタンスを閉じます．

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([100, 200, 300, 400, 500])
fig = plt.figure()
plt.pie(x)

fig.savefig("pie1.png")
plt.close()
```

実行結果のメッセージは省略している

図27　内訳の比較が分かりやすい円グラフ1…基本の円グラフ

▶ラベル付き円グラフ

図24の円グラフにラベル名を付け，最初の値を円の上から描画するよう変更した円グラフを作成します(図25)．

図28の1～3行目は，上記のグラフと同様です．

4行目でラベル用として，文字列でリストを作り変数labelに代入します．

5行目で画像ファイルに書き込むため，インスタンスを作成して変数figに代入します．

6行目でpie関数に変数x，引数labelsに変数label，引数counterclockをFalseとして時計回りにプロットするようにします．

また，引数startangleに90を指定することで円の上(頂点)の位置から描画します．プロット・オブジェクトが作成されると画面出力にメッセージが表示されます．

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([100, 200, 300, 400, 500])
label = ["A", "B", "C", "D", "E"]
fig = plt.figure()
plt.pie(x, labels=label,
        counterclock=False, startangle=90)

fig.savefig("pie2.png")
plt.close()
```

4行目

1行に入力する

実行結果のメッセージは省略している

図28　内訳の比較が分かりやすい円グラフ2…ラベル付き円グラフ

▶円グラフの見た目を調整

図25の円グラフの見た目を次のように変更してみます(図26)．

- 構成割合をパーセントで表示
- 要素間に白い線を引く

変更点は図29の6行目のpie関数の引数です．上記の引数に加えて，引数autopctに%1.1f%%を設定して小数点第1位までのパーセント表示を行います．

さらに引数wedgepropsのlinewidthを3にして枠線の太さを設定し，引数edgecolorでwhiteを代入して枠線の色を白に設定します．

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([100, 200, 300, 400, 500])
label = ["A", "B", "C", "D", "E"]
fig = plt.figure()
plt.pie(x, labels=label, counterclock=False,
        startangle=90, autopct="%1.1f%%",
        wedgeprops={'linewidth': 3,
         'edgecolor':"white"})

fig.savefig("pie3.png")
plt.close()
```

6行目

1行に入力する

実行結果のメッセージは省略している

ウェブ・スクレイピングしたデータを使って，トレンドを把握したり，プレゼン資料を作成したりできます

図29　内訳の比較が分かりやすい円グラフ3…見た目を調整

◆参考文献◆

(1) Matplotlib.
https://matplotlib.org/

さとう・せい