J. Plasma Fusion Res. Vol.94, No.3 (2018) 150–161

講座 今日からはじめる Python

3．Python による科学技術計算

3. Scientific Computing in Python

3. 2　matplotlib の使い方

3.2 How to Use matplotlib

吉沼幹郎
YOSHINUMA Mikirou
核融合科学研究所

（原稿受付：2018年1月23日）

Python を用いて，データをグラフに描画するときに利用される matplotlib というモジュールの使い方を説明します．このモジュールを利用することで，Python を用いた行われた計算の途中や結果の数値を確認することが簡単にできるようになります．また，論文掲載用の図として利用できる品質のさまざまなグラフを描画することができます．ここでは，簡単な 2 次元データの散布図から，凡例表示やエラーバーの表示方法，3 次元データのヒートマップの表示について説明します．

Keywords:
Python, matplotlib, data visualization, graph, control plot

3. 2. 1　はじめに

Python を用いて，実験データの解析を行うとき，あるデータのセットに対して，Python で記述されたなにかの処理を適用して，結果を得るということを繰り返します．そのようなとき，得られた結果をグラフとして描画したいと思うことでしょう．数値をファイルに書き出し，グラフ描画ソフトウェアで読み込み，描画するという手順を踏んでもよいのですが，処理の途中経過などは，いちいちファイルに書き出すのを面倒に感じます．ここでは，そのような状況でも便利に利用できる，Python で取り扱っているデータから直接グラフを描画できる matplotlib モジュールの使い方を紹介します．matplotlib は，Python でグラフを描画するときに利用される標準的なモジュールとなっており，二次元の散布図からヒートマップなど様々なグラフを描画することができます．matplotlib のホームページのギャラリーでは，どのようなグラフが描画できるのか一覧できます．

matplotlib は，グラフ描画に使われる要素（オブジェクト），例えば軸や線やラベルといったものが詰まったもので，それらを適切に設定，操作することで，グラフ全体を構成します．matplotlib で，それらのオブジェクトを操作するには，オブジェクトごとに持っている操作手続き（メソッド）を直接呼び出す方法（オブジェクトインターフェース）と，オブジェクトの操作手続きを代行する関数を呼び出す方法（関数インターフェース）があります．前者の方法は，記述が増える代わりに，matplotlib のすべての機能を使うことができます．Python で構築するアプリケーションソフトウェアに matplotlib を組み込んで使う場合に便利です．後者の方法は，記述が少なくなるため，Jupyter などのインタラクティブな環境で使うときに便利です．

Python のモジュールを利用するためには，利用したいモジュールをインポートする必要がありました．matplotlib というモジュールは，複数のモジュールファイルで構成されており，どれをどのようにインポートしたらよいか分かりにくいかもしれません．グフラフ描画に利用されるものが matplotlib.pyplot というモジュールです．しばしば，スクリプトの冒頭で‘import matplotlib.pyplot as plt’と記述されインポートされています．これは，matplotlib.pyplot モジュールを plt という名前でインポートすることを意味します．このようにすると，matplotlib.pyplot に含まれる関数，例えば，plot()や show()などが，’plt.plot(...)’や’plt.show()’のように，モジュール名’plt.’をつけて記述することで呼び出すことができます．matplotlib には，MATLAB と互換性のある関数名を使いたい人のために matplotlib.mlab モジュールも用意されています．

ここではインタラクティブな環境を利用することを考え，pylab というモジュールをインポートして関数イン

National Institute for Fusion Science, Toki, GIFU 509-5292 Japan
author's e-mail: yoshinuma@nifs.ac.jp

ターフェースを使うスタイルで説明していきます．そこで，スクリプトの冒頭には，‘from pylab import *’と記述しています．このように pylab モジュールを import すると，matplotlib.pyplot,matplotlib.mlab,numpy モジュールに含まれる関数を，モジュール名無しに，関数名だけで使えるようになり，記述量が少なくなります．ただし，このようにモジュール名を省略すると，異なるモジュールにおいて同名の関数がある場合に上書きされてしまう問題が起こりますので注意が必要です．同名の関数や変数があるかどうかは，インタラクティブな環境でその名前を評価してみるとわかりますので，疑わしいときは確認するとよいでしょう．ここでは，気軽にスクリプトを入力して試していただくために，記述量が少なくなるスタイルをとりましたが，関数などの出所を明確にするためにも，モジュール名を省略しないスタイルをお勧めします．

matplotlib によるグラフ描画の説明には，プロットするデータが必要です．プロットするデータはすべてスクリプト内で手短に生成していますので，試すために，データファイルを用意する必要はありません．ぜひ，リストにあるスクリプトを入力，改変して，動きを確認していただきたいと思います．

3.2.2 はじめの一歩

まず，インタラクティブな環境を起動して **List 1** にあるものを実行してみましょう．説明のため，余計な線や文字が入っていますが，**図 1** に描画結果を示します．"a"という名前の配列に −1 から 1 まで 0.1 刻みの数値を設定（2 行目）し，その内容をグラフにプロット（3 行目）するものです．plot()関数に与えた a という配列の値が y 軸の値としてプロットされています．配列一つを plot()関数に与えると，x 軸の値は配列のインデックス番号が使われます．このように配列にどのような値がはいっているかを簡単にグラフにして確認することができます．数値列を生成する arange()関数は numpy モジュールによって提供されていますが，pylab モジュールを 1 行目の記述で import したことで，モジュール名を省略して利用できるようになっています．

List 1：pylab のインポートと配列内容のプロット

```
1 from pylab import *
2 a = arange(-1, 1.1, 0.1)
3 plot(a)
4 show()
```

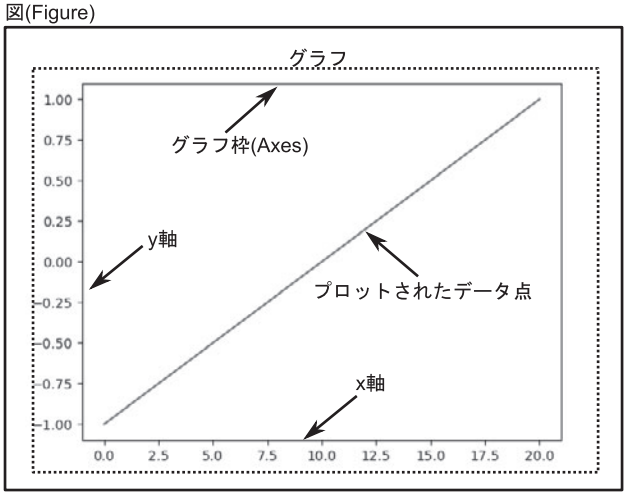

図 1　List 1 による描画結果と図（Figure）の構成要素．

図 1 を見ながら，グラフを描画するときのいくつかの重要な構成要素について説明します．グラフが描画されている領域全体が Figure と呼ばれる要素です．本文では「図(Figure)」と書くことにします．matplotlib ではこの図(Figure) にグラフを描画し，それを表示させます．図(Figure) の中には，x 軸，y 軸で囲まれた四角の領域があります．本文では，「グラフ枠(Axes)」と書くことにします．グラフ枠（Axes）は x 軸，y 軸によって構成され，その中にデータ点がプロットされます．グラフ枠（Axes）は，図 (Figure) の中に一つ以上配置することができます．関数を呼ぶスタイルで matplotlib を利用する場合，plot()関数によって，グラフ枠（Axes）内にデータ点をプロットし，show()関数によって，グラフ枠が配置された図(Figure) を描画します．図（Figure）やグラフ枠（Axes）は自動的に作成され，操作の対象となる図（Figure）やグラフ枠（Axes）も自動的に決められます．関数を呼び出した結果は，現在対象になっている図（Figure）やグラフ枠（Axes）に現れます．ですから，現在対象になっている図(Figure) やグラフ枠（Axes）というものを意識して操作するとよいでしょう．

3.2.3 (x, y)データをプロットする

ここでは単純なxy散布図のグラフ描画を通して，matplotlibの各種操作を説明していきます．

3.2.3.1 データ点をプロットしてみる

単純なガウス分布関数をプロットしてみましょう．**List 2**を実行してみてください（**図2**）．

List 2：ガウス分布関数

```
1 from pylab import *
2 xdata = arange(-2, 2.2, 0.2)
3 ydata = exp(-xdata**2)
4 plot(xdata, ydata, 'o')
5 show()
```

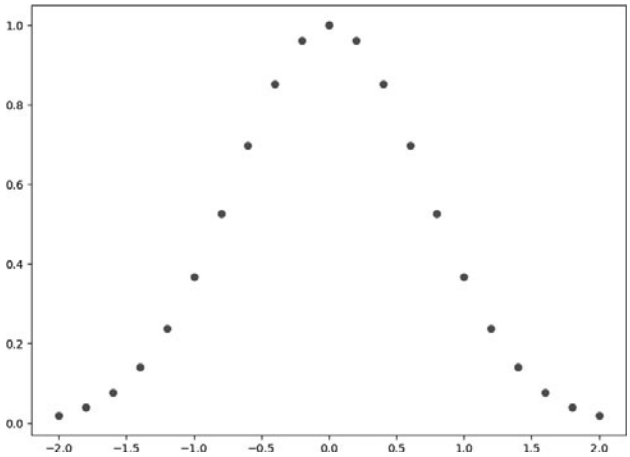

図2 List 2による描画結果．

2行目，3行目でプロットするデータの配列を作り，xdata，ydataとしています．それらをplot()関数の1番目の引数，2番目の引数としてそれぞれ与えます．plot()関数に渡す3番目の引数で，プロットに用いるマーカーを指定します．アルファベット小文字の‘o’を渡すことで，塗りつぶされた丸記号を指定しています．注意していただきたいのは，plot()関数の引数は，前の章で配列を一つ与えたときとは異なっています．このように，同じ関数でも，引数の与え方に応じて適切に振る舞いが変化するものもあります．

3.2.3.2 マーカーや線種の指定とオーバープロット

plot()関数に渡す三番目の引数でマーカーを変更することができました．この指定に使われる文字は，インタラクティブな環境で‘help(plot)’を実行すると表示される説明で読むことができます．丸●，三角▲，四角■，ダイヤ◆の記号は，それぞれ‘o’，‘^’，‘s’，‘D’で指定します．実線，破線，点線，一点鎖線は，それぞれ‘-’，‘--’，‘:’，‘-.’で指定します．マーカーで点を打ち，さらに線で結びたいときは，これらを組み合わせます．例えば，丸でプロットして，点線で結びたいときは，‘o:’という文字の列を三番目の引数に渡します．マーカーのサイズは，markersize（あるいはms）というキーワード引数を使って指定できます．キーワード引数とは，Pythonの関数で用いられる引数の種類の一つで，順不同に渡せるばかりでなく，渡さなかった場合にデフォルトの値が使われるため，指定しなくてもよい引数です．線の太さは，linewidth(lw)というキーワード引数を使って指定できます．いくつかマーカーとマーカーのサイズ，線種を変えてプロットしてみます．plot()関数の呼び出しを重ねるとデータをオーバープロットできます（**List 3**，**図3**）．

List 3：マークを変えてオーバープロット

```
1 from pylab import *
2 xdata = arange(-2, 2.2, 0.2)
3 ydata = exp(-xdata**2)
4 plot(xdata, ydata, 'o--', markersize=4, linewidth=2)
5 plot(xdata, ydata*0.7, 's', ms=6)
6 plot(xdata, ydata*0.5, '^', ms=8)
7 plot(xdata, ydata*0.3, 'o-', ms=8, lw=4)
8 show()
```

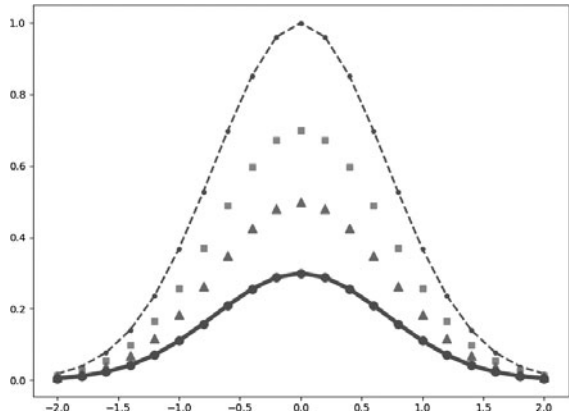

図 3　List 3 による描画結果.

3. 2. 3. 3　色の指定

plot()関数を繰り返し呼び出してオーバープロットすると色が自動的に変わっていきます．自分で色を指定したいときは，plot()関数を呼び出すときに color（あるいは c）というキーワード引数に色を示す値を渡します．色を示す値には，色の名前，色を表す 1 文字，RGB（A）の16進数文字列，RGB（A）の値をもつ tuple などいろいろあります．ここで（A）と書きましたが，これは透明度を指定するアルファチャンネルの値も指定できるからです．色を表す 1 文字には，‘b’，‘g’，‘r’，‘c’，‘m’，‘y’，‘k’，‘w’の 8 文字があり，それぞれ青（blue），緑（green），赤（red），水色（cyan），赤紫（magenta），黄（yellow），黒（key），白（white）に対応します．色の名前は多くありますが，実際の色をイメージすることは難しいので，先の 8 文字以外の色を使いたい場合は，RGB の16進数文字列で指定することをお勧めします．RGB の16進数文字列とは，00（10進数で 0）から FF（10進数で255）の大きさで赤（Red），緑（Green），青（Blue）の各色の強さを指定する方法です．先頭に‘#’をつけて，例えば赤なら‘#FF0000’となります．白は，RGB 各色が最大値ですので，‘#FFFFFF’となります．一方，黒は各色が最小値の‘＃000000’となります．**List 4** にいろいろな方法で色を指定した例を示します．最後の例のように色を 1 文字で指定する場合，スタイル指定の文字列に含めてしまうこともできます．手早く色を変えてプロットしたいときは，こちらの指定が楽でしょう．マーカー内部の色，境界の色はそれぞれ markerfacecolor（mfc），markeredgecolor（mec）というキーワード引数に色を指定する値を渡すことで個別に設定できます．これを用いて，白抜きや中抜きのマーカーを利用することもできます（**List 4**，**図 4**）.

List 4：さまざまな色指定の方法

```
1  from pylab import *
2  xdata = arange(-2, 2.2, 0.2)
3  ydata = exp(-xdata**2)
4  plot(xdata, ydata, 's--', color='#FF0000') # 16進数文字列で指定
5  plot(xdata, ydata*0.8, 's--', c=(0, 0, 1)) # tupleで指定
6  plot(xdata, ydata*0.6, 's--', c='y') # 色の文字で指定
7  plot(xdata, ydata*0.5, 's--', c='m') # 色の文字で指定
8  plot(xdata, ydata*0.4, 's--', c='magenta') # 色の名前で指定
9  plot(xdata, ydata*0.3, 's--', c='bisque') # 色の名前で指定
10 plot(xdata, ydata*0.2, 's--g')
11 show()
```

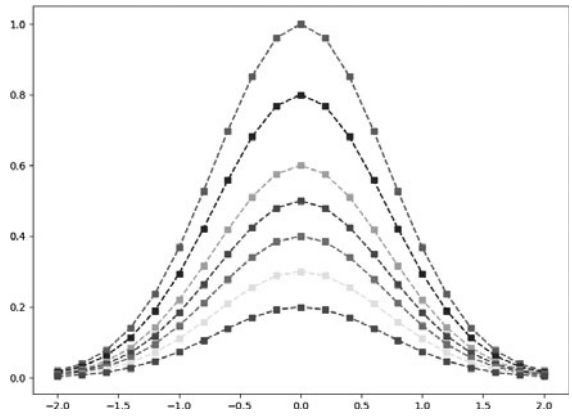

図 4　List 4 による描画結果.

3. 2. 3. 4 軸の設定

リスト **List 5** と**図 5** に，軸の目盛りや範囲を設定する例を示します．matplotlib では，図（Figure）をディスプレイに表示する前，すなわち show()関数を呼ぶ前に設定を済ませておく必要があります．

4 行目の tick_params()関数によって，軸に目盛りや目盛りのラベル描画をさせるかの設定ができます．right キーワード引数，top キーワード引数に True を設定することで，x 軸，y 軸の対向側にも目盛りを振るようにしています．direction キーワード引数に'in'を設定することで，目盛りをグラフ内側に向けて描画するようにしています．5 行目の grid()関数によって，目盛りにあわせて格子状の線を描画させます．6 行目，8 行目の xticks()関数，yticks()関数によって，目盛りを振る場所を指定できます．第一引数には，目盛りを振る位置を数値の配列にして渡します．第二引数に，文字列のリストを渡すと，目盛りのラベルにその文字列が使われます．例では，目盛り位置は，arange()関数を用いて生成しています（6 行目，7 行目）．y 軸のラベルは，目盛り位置（yticks_positions）の数値を'%4.3f'（小数点以下 3 桁）でフォーマットした文字列のリストを第二引数に渡しています．9，10行目の xlim()，ylim()関数によって，それぞれ x 軸の範囲，y 軸の範囲を設定します．第一引数に下限値，第二引数に上限値を指定します．軸の方向は下限値，上限値の大小関係によって適切に変化します．各関数のパラメータの詳細については，plot()関数のときと同様に，インタラクティブ環境で 'help（関数名）' を実行することで読むことができます．

List 5：軸の範囲とラベルの設定

```
1  from pylab import *
2  xdata = arange(-2, 2.2, 0.2)
3  ydata = exp(-xdata**2)
4  tick_params(right=True, top=True, direction='in')
5  grid(True)
6  xticks(arange(-3.0, 3.1, 0.5))
7  yticks_positions = arange(0.0, 3.1, 0.25)
8  yticks(yticks_positions, ["%4.3f" % v for v in yticks_positions])
9  xlim(-2.5, 2.5)
10 ylim(0, 1.5)
11 plot(xdata, ydata, 's-', ms=8, c='r')
12 show()
```

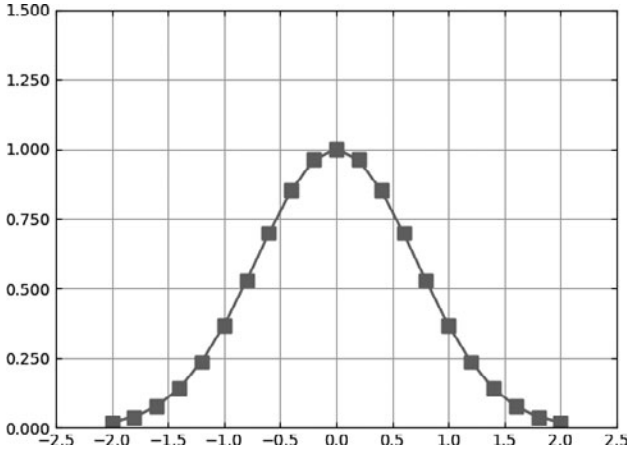

図 5　List 5 による描画結果．

3. 2. 3. 5 タイトルと凡例の表示

List 6 と**図 6** に，グラフタイトル，軸のラベルや凡例の表示を行う例を示します．4，5 行目の xlabel()，ylabel()関数によって，それぞれ x 軸のラベル，y 軸のラベルを設定します．ラベルとして書き出す文字列を第一引数に指定します．6，7 行目でデータをプロットしています．プロットされたデータと凡例に表示する文字を一致させるために，plot()関数を呼び出すときに，キーワード引数 label に凡例として表示される文字列を設定しています．8 行目の legend()関数で凡例を表示させる指示を行いますが，凡例として使用される文字列が指定された後，すなわち plot()を行った後に行います．9 行目で title()関数によってタイトルの文字列を設定しています．文字列には，TEX 形式の数式表記も利用できます．**List 6** では，タイトルの文字列として，r'Function…'のように文字'r'をつけています．この例では使用していませんが，TEX 表記ではしばしば'\'文字（機種によっては'¥'文字）を利用します．Python では文字列の表記中の'\'文字は，特別な文字（例えば改行やタブ文字）を表記するために使われます．また，'\'文字を表記するためにも使われるため，文字列データの中に'\'文字を入れたいときは'\\'のように二回書く必要があります．文字列の前に'r'文字をつけると書いたものをそのまま Python の文字列データとすることができます．ですからこのような記述をしておくと，TEX 表記の文字列を書きやすくなります．文字の大きさは，xlabel()，ylabel()，title()，legend()関数の fontsize キーワード引数に数値を設定する

ことで，変更することができます．legend()関数には，表示設定のためのキーワード引数が多く存在します．例では numpoints＝1 を指定して，凡例におけるマーカーの数を１つに設定しています．

List 6：凡例を表示

```
1  from pylab import *
2  xdata = arange(-2, 2.2, 0.2)
3  ydata = exp(-xdata**2)
4  xlabel('x-axis', fontsize=14)
5  ylabel('y-axis', fontsize=14)
6  plot(xdata, ydata, 'o-', ms=8, c='r', label='A=1.0')
7  plot(xdata, ydata*0.5, 's-', ms=8, c='b', label='A=0.5')
8  legend(fontsize=16, numpoints=1)
9  title(r'Function␣of␣$y=Ae^{-x^2}$', fontsize=20)
10 show()
```

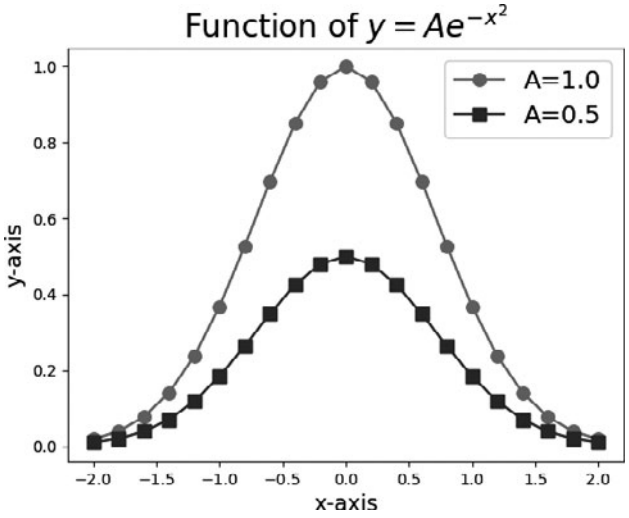

図 6　List 6 による描画結果

3. 2. 3. 6　y 2 軸の利用

異なる値を同じ x 軸上にプロットしたい場合，x 軸を共通にして，y 軸をもうひとつ増やしたいことがあります．もう一つの縦軸をy2軸とここでは呼ぶことにします．グラフ枠の左側の枠線が y 軸に用いられ，y2 軸は y 軸の反対側（右側）の枠線に描かれます．

List 7 および**図 7** に y2 軸を利用する例を示します．5 行目から 9 行目までは，これまで行ってきた x-y 軸へのプロットです．11行目で twinx()関数を呼び出しています．twinx()関数を用いると，x軸を共有した‘新しいy軸’（＝y2軸）をもったグラフ枠（Axes）ができ，操作対象が新しくできたグラフ枠(Axes)に変化します．y2 軸は twinx()を行った後で作られますので，その範囲設定，ラベル設定，y2 軸上へのプロット，レジェンドの描画は，その後で行います．

List 7：y 2 軸を使う

```
1  from pylab import *
2  xdata = arange(-2, 2.2, 0.2)
3  ydata = exp(-xdata**2)
4
5  ylim(0, 1.5)
6  xlabel('x-axis', fontsize=14)
7  ylabel('y-axis', fontsize=14)
8  plot(xdata, ydata, 'o--', ms=8, c='r', label='on␣x-y')
9  legend(fontsize=16, loc='upper␣left')
10
11 twinx()
12 ylim(0, 3)
13 ylabel('y2-axis', fontsize=14)
14 plot(xdata, ydata, 's--', ms=8, c='b', label='on␣x-y2')
15 legend(fontsize=16, loc='upper␣right')
16 show()
```

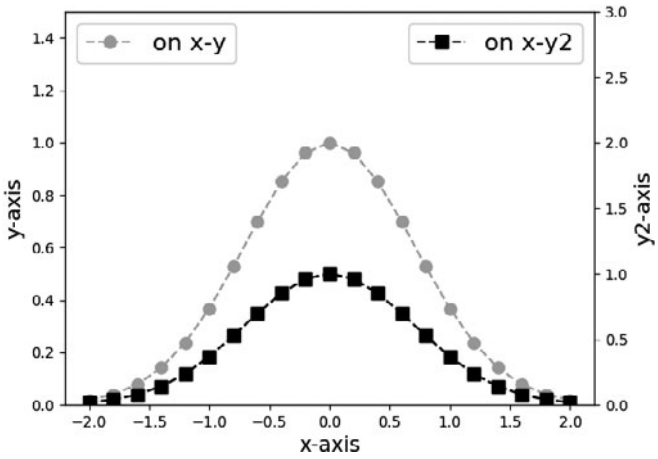

図 7　List 7 による描画結果.

3. 2. 3. 7　一つの図に複数のグラフを描く

しばしば，複数のグラフを用いて異なる物理量を同時に載せることもあります．グラフを一つ一つ作成しておいて，あとで合わせることもできますが，ここではmatplotlib で一つの図に複数のグラフを描く方法を説明します．

複数のグラフを作る場合，それぞれにグラフ枠(Axes)を作成する必要があります．これまで行った twinx()関数も新たにグラフ枠（Axes）を作成する効果がありました．ここでは subplot()関数を用いて新たにグラフ枠（Axes）を作成します．3 行 2 列，合計 6 枚のグラフ枠（Axes）を並べたい場合，subplot()関数の第 1 引数に行数の 3 を，第 2 引数に列数の 2 を与え，第 3 引数に，どの位置のグラフ枠（Axes）を作成するかをインデックス番号で指定します．インデックス番号は**図 8** のように，まず行方向に変化します．4 番の位置にプロットしたいなら，subplot(3, 2, 4)とすると，その位置に描画されるグラフ枠（Axes）が作成され，それが操作対象になります．また，行数，列数，インデックスが10より小さい場合，それらを一つの引数にまとめて subplot（324）のように指定することもできます．

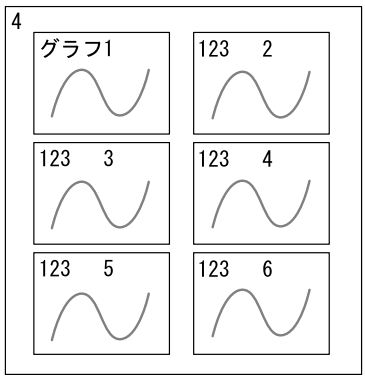

図 8　subplot()関数によるレイアウトとインデックス番号.

List 8 および**図 9** に 1 行 2 列のレイアウトでプロットする例を示します．5 行目で左側のグラフ枠（Axes）を作成しています．6 行目で枠の上と右側にも目盛りを振るように，tick_params()関数で設定しています．11行目で右側のグラフ枠（Axes）を作成しています．こちらの枠は，左側の目盛りの数値を表示させないように，12行目の tick_params()関数で labelleft=False を設定しています．15行目の tight_layout()関数でグラフ枠の間隔を調整しています．この tight_layout()関数を使うと，各グラフの間隔を大まかに変えることができます．h_pad，w_pad というキーワード引数にフォントサイズを基準にした値を設定します．たとえば，横方向の間隔を 2 文字分ほど広げたいなら w_pad＝2.0 とします．位置の調整には，subplots_adjust()関数も使うことができます．subplots_adjust（wspace＝0.0，hspace＝0.0）とするとグラフ同士が密着します．ここでキーワード引数 wspace,hspace には，それぞれグラフの x 軸の幅，y 軸の幅を基準とした値を渡します．

List 8：subplot()関数を用いた 1 行 2 列のレイアウト

```
from pylab import *
xdata = arange(-2, 2.2, 0.2)
ydata = exp(-xdata**2)

subplot(1, 2, 1)
tick_params(top=True, right=True, direction='in')
plot(xdata, ydata, 'o-', ms=8, c='r')
xlabel('x-axis', fontsize=14)
ylabel('y-axis', fontsize=14)

subplot(1, 2, 2)
tick_params(top=True, right=True, labelleft=False, direction='in')
plot(xdata, ydata, 's-', ms=8, c='b')
xlabel('x-axis', fontsize=14)
tight_layout(w_pad=0.0)
show()
```

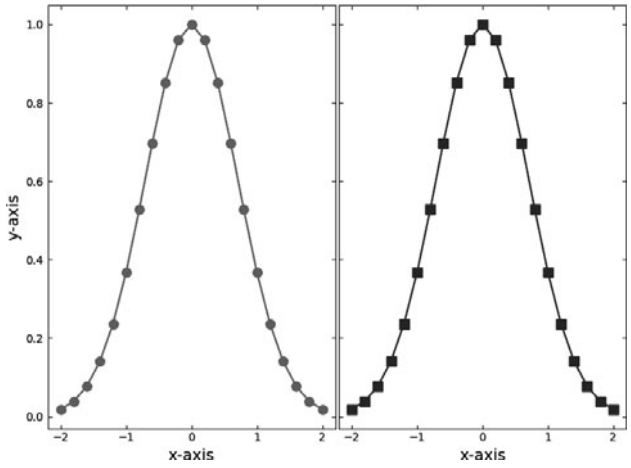

図 9　List 8 の描画結果.

3. 2. 3. 8　図(Figure)の大きさの変更と保存 Change size of figure and save to file

これまで自動的に作成されるデフォルト設定の図(Figure)を利用してきましたが，figure()関数を用いると，大きさを指定して図(Figure)を作成することができます．この関数を使うと，操作対象が新たに生成された図(Figure)に切り替わります．**List 9** には，8 インチ× 6 インチの画像を 100 dpi で png (Portable Network Graphics) 形式で保存し，また表示する例を示します．4 行目の figure()関数の figsize キーワード引数にサイズを表す(8, 6) のタプルを設定しています．dpi キーワード引数に，100を設定しています．6 行目の savefig()関数によって，画像ファイルとして保存しています．第一引数に出力ファイル名を指定します．拡張子によって自動的に画像形式が変換されます．拡張子は，'.png'，'.svg'，'.eps'というところがよく使われるものでしょう．保存と表示を行いたい場合は，savefig()関数を使ってから show()関数を使ってください．例では，カレントディレクトリ（現在の作業ディレクトリ）に'output_gauss.png'という名前で画像ファイルが作成されますので，同名のファイルがないことを確認してから実行してください．

List 9：図の保存

```
from pylab import *
xdata = arange(-2, 2.2, 0.2)
ydata = exp(-xdata**2)
figure(figsize=(8, 6), dpi=100)
plot(xdata, ydata, 'o-', ms=8, c='r')
savefig('output_gauss.png', dpi=100)
show()
```

3. 2. 4　いろいろなプロット

ここまで，matplotlib で単純な散布図を描画して，軸の範囲を設定したり，ラベルを付けたりしてきました．次に，誤差棒のついたプロットやヒートマップをプロットする方法を紹介します．

3. 2. 4. 1　誤差棒のついたグラフ Plot with error bars

誤差棒のついたグラフを描画するには errorbar()関数を使います．第 1 引数，第 2 引数にそれぞれ x，y のデータを渡すのは，plot()関数と同じです．errorbar()関数には誤差の値を渡すキーワード変数 xerr と yerr があります．x 軸方向の誤差は xerr に，y 軸方向の誤差は yerr に渡します．また，マーカーの指定には，キーワード変数 fmt を用います．

誤差棒の描き方を指定するためにキーワード変数 capsize, capthick,elinewidth があります．工字型の誤差棒を表示したい場合は，capsize を適切に指定してください．capsize で誤差棒両端の線の長さを，capthick で誤差棒両端の線の太さを，elinewidth で誤差棒の太さを指定できます．**List 10** に例を示します．

List 10：誤差棒のついたグラフ

```
1 from pylab import *
2 xdata = arange(-2, 2.2, 0.2)
3 ydata = exp(-xdata**2)
4 yedata = 0.1*ones(xdata.size)
5 errorbar(xdata, ydata, yerr=yedata, fmt='o', ms=8, c='r', capsize=5)
6 show()
```

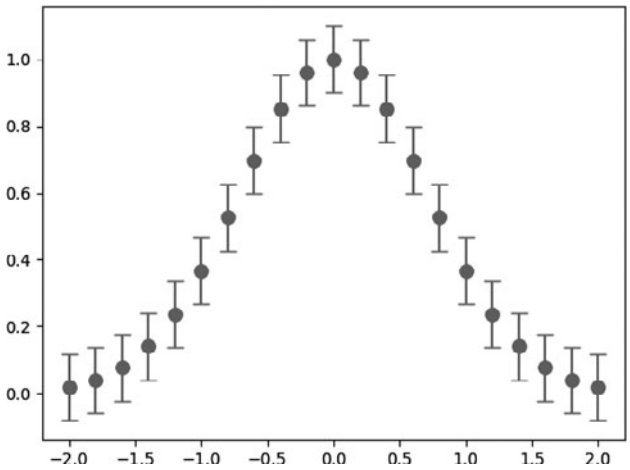

図10　List 10 の描画結果．

3. 2. 4. 2　等高線のプロット　Contour plot

ここから 3 次元のデータ（二次元の配列に大きさが入ったもの）のプロットをしていきます．3 次元プロットのためのデータを二次元配列で与えるときに混乱しやすいのは，x 軸，y 軸方向と 2 次元配列の行方向，列方向の関係です．向きが分かりやすいように，x 軸方向に非対称，y 軸方向に対称な $f(x,y)=\sin(x)+\cos(y)$ という関数の値を z としてプロットしてみます．

等高線は，contour()関数を用いて描きます．等高線の間を塗りつぶす contourf()関数もありますが，おおよその使い方は contour()関数と同じです．contour()関数には 2 次元の配列を一つ渡すと，x 軸，y 軸はそれぞれは列方向，行方向のインデックス番号が使われます．2 次元配列のデータを確認するときには便利かと思います．よく行うのは，x，y，z の 3 次元データを与えて，等高線を描かせることでしょう．

式（1），（2）で示される x,y のリストに対して，z の二次元配列をどのような並びで作成するかを式（3）に示します．配列 x，配列 y の要素数がそれぞれ z の列数，z の行数に一致する必要があります．すなわち行方向が y 軸の方向，列方向が x 軸の方向になります．contour()関数の x と y には，式（4），（5）に示す xm，ym のような 2 次元配列を渡しても問題ありません．

List 11にプロットした例を示します．contour に与えるデータを作成する箇所が重要な場所です．x，y のリストから xm，ym のようなリストを作成する関数が meshgrid() です．levels キーワード引数には，線を引くレベルを指定できます．

等高線にラベルをつけるには，プロットした等高線オブジェクトを clabel()関数に渡す必要があります．contour()関数は，等高線をプロットした後，等高線オブジェクトを返してきますので，それを cntr 変数に保持して（6 行目），clabel()関数に渡しています（7 行目）．ラベルの書式の設定は clabel()関数のキーワード引数 fmt に，Python の書式指定文字列（C 言語と似ています）を渡すことで行います．例では，小数点以下 2 桁の表示を指定しています．

contour()関数には，他にも設定があります．colors キーワード引数に色の指示値を与えると，その色が順番に使われますが，一つだけ渡すと単色で線を引くことができます．詳しくは，matplotlib ホームページのギャラリーを見て調べるとよいでしょう．

$$x=[x_0,x_1,\cdots,x_n] \tag{1}$$

$$y=[y_0,y_1,\cdots,y_m] \tag{2}$$

$$\begin{aligned} z=[&[f(x_0,y_0),f(x_1,y_0),\\ &[f(x_0,y_1),f(x_1,y_1),\\ &\cdots\\ &[f(x_0,y_m),f(x_1,y_m),\cdots,f(x_n,y_m)]] \end{aligned} \tag{3}$$

$$\begin{aligned} xm=[&[x_0,x_1,\cdots,x_n],\\ &[x_0,x_1,\cdots,x_n],\\ &\cdots\\ &[x_0,x_1,\cdots,x_n]] \end{aligned} \tag{4}$$

$$ym = [[y_0, y_0, \cdots, y_0],\ \cdots \\ [y_1, y_1, \cdots, y_1],\ [y_m, y_m, \cdots, y_m]] \quad (5)$$

List 11：等高線のグラフ

```
1 from pylab import *
2 x=arange(-3, 3.1, 0.1)
3 y=arange(-3, 3.1, 0.1)
4 xm, ym = meshgrid(x, y)
5 zm = sin(xm) + cos(ym)
6 cntr = contour(x, y, zm, levels=[-1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5])
7 clabel(cntr, fmt='%.2f')
8 show()
```

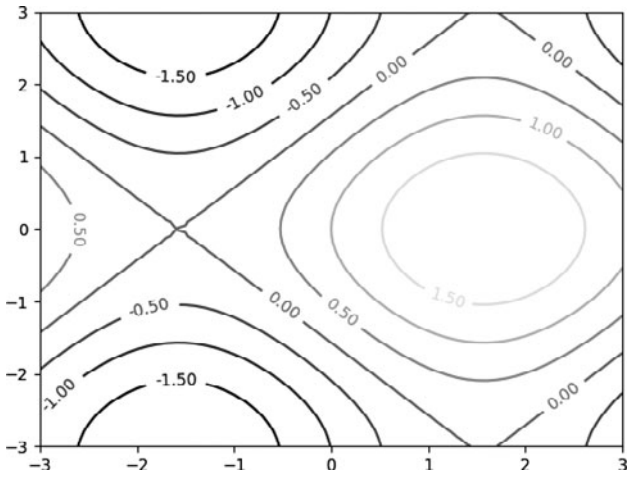

図11　List 11 の描画結果.

3. 2. 4. 3　ヒートマップのプロット　Heat map

(x, y, z)の値の分布を色の変化で示したい場合にヒートマップが用いられます．ヒートマップのプロットには，pcolormesh()関数を使います．例を **List 12**および描画結果を**図12**に示します．ほぼ，前節のcontour()関数の代わりに，pcolormesh()関数を使うだけです．キーワード引数 cmap に，カラーマップ（値と色を対応させるもの）を指定します．カラーマップの名前は，よく使われるもので'jet', 'hot','rainbow'などがあります．色と値の対応を示すために，同時にカラーバーを描画することが一般的です．カラーバーは，colorbar()関数に，対象となるヒートマップを渡すことで描画できます．対象となるヒートマップは，pcolormesh()関数で描画したときに得られます．7 行目で，pcolormesh()で得られたヒートマップを hmap に保存しておき，8 行目で colorbar()関数に渡しています．colorbar()関数からは，描画したカラーバーが得られます．9 行目では，8 行目で得られたカラーバー cbar からカラーバー自身のグラフ枠（Axes）を取り出して，y 軸のラベルを設定する機能を呼び出してカラーバーにラベルを設定しています.

このプロットでは気が付かないかもしれませんが，描画範囲を狭めてメッシュが拡大されると，色の塗り方に問題が現れます．わかりやすいように，関数を $f(x,y) = e^{-(x^2+y^2)}$ をzとしてpcolormesh()関数でヒートマップを描画しました（**図13**左，**List 13**）．同時に等高線図も描画し，$(x,y)=(0,0)$ 付近を拡大しています．この図のように，色と等高線がずれて見えます．これは，メッシュ内の色が，そのメッシュの左下の値に応じた色で塗られているためです．メッシュの値と塗りの中心をあわせたい場合には，塗りに使われるメッシュを計算に使われたメッシュサイズの半分だけシフトさせる必要があります．**List 13** の 6 行目，7 行目でずらしたメッシュ位置を作成し，20行目の pcolormesh()関数に渡しています．注意していただきたいのは，この pcolormesh()に渡している shifted_x と shifted_y は，pcolormesh()でヒートマップの表示をずらして適切にするために渡しているだけで，ここで渡している zm の値は，あくまでも xm，ym の位置で計算したものであることです．（ずらした後の位置（shifted_x，shifted_y）で計算したものではありません.）

6 行目，7 行目のずらした位置を得るところについて少し説明します．'r_'は，numpy モジュールが提供するオブジェクトで，r_[a, b, c]とするとオブジェクトの機能（メソッド）によって，a，b，c，を行方向へ連結した配列を返してくれます．式を見ると，x(y)の 2 番目の要素から最後まで，間隔の半分を差し引いてずらしています．その先頭と最後に，x(y)の先頭の要素 x[0](y[0])と最後の要素 x[-1](y[-1])を連結させ付け加えています.

List 12：ヒートマップによる表示

```
from pylab import *
x=arange(-3, 3.1, 0.1)
y=arange(-3, 3.1, 0.1)
xm, ym = meshgrid(x, y)
zm = sin(xm) + cos(ym)
xlim(-3.0, 3.0)
hmap = pcolormesh(x, y, zm, vmin=-3, vmax=3, cmap='jet')
cbar = colorbar(hmap)
cbar.ax.set_ylabel('colorbar label')
show()
```

List 13：メッシュをずらす

```
from pylab import *
x=arange(-0.4, 0.41, 0.2)
y=arange(-0.4, 0.41, 0.2)
xm, ym = meshgrid(x, y)
zm = exp(-1*(xm**2 + ym**2))
shifted_x = r_[x[0], x[1:] - diff(x)*0.5, x[-1]]
shifted_y = r_[y[0], y[1:] - diff(y)*0.5, y[-1]]

subplot(121) # for using original mesh (left figure)
xlim(-0.5, 0.5)
ylim(-0.5, 0.5)
title('use x, y')
pcolormesh(x, y, zm, vmin=0, vmax=1, cmap='jet')
contour(x, y, zm, levels=[0.9, 0.95], colors='k')

subplot(122) # for using shifted mesh (right figure)
xlim(-0.5, 0.5)
ylim(-0.5, 0.5)
title('use shifted x, y')
pcolormesh(shifted_x, shifted_y, zm, vmin=0, vmax=1, cmap='jet')
contour(x, y, zm, levels=[0.9, 0.95], colors='k')

tight_layout(w_pad=3)
show()
```

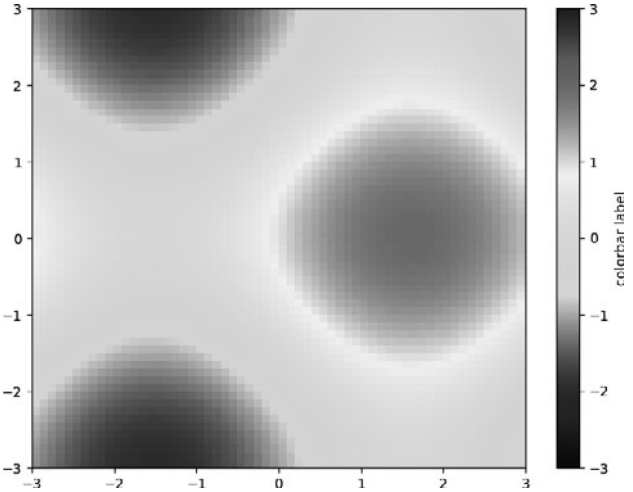

図12 List 12 の描画結果.

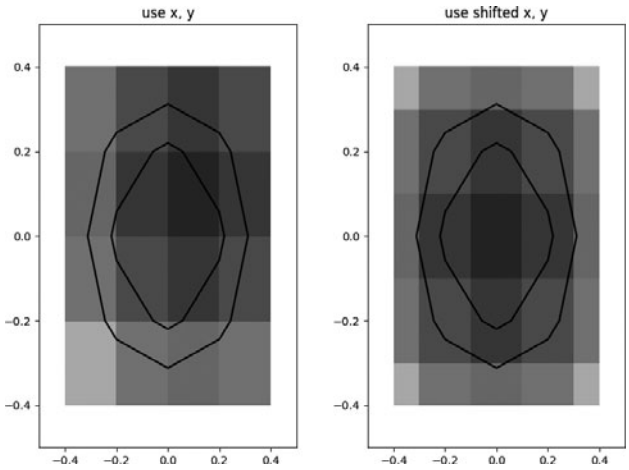

図13 List 13 の描画結果.

3. 2. 4. 4 色付き散布図のプロット Scatter plot with color depend on values

z の値に応じて，色を変化させた散布図を描きたいこともあります．その場合は，scatter()関数を使います．例を List 14に示します．プロットする位置の座標 x, y をそれぞれ第 1 引数，第 2 引数に渡します．キーワード引数 c に z を渡します．キーワード引数 marker にマーカーの種類を渡します（図14）．マーカーに渡す値については，plot()関数を参考にしてください．例では，‘o’を渡して，丸を用いています．マーカーの大きさは，キーワード引数 s に point の二乗の単位で指定します．すなわち，10point のマーカーなら100を s に渡します．範囲やカラーマップについては，pcolormesh()関数と同じです．

List 14：色付きの散布図による表示

```
from pylab import *
x = -3.0 + 6.0*random(100)
y = -3.0 + 6.0*random(100)
z = sin(x) + cos(y)
sct = scatter(x, y, c=z, s=100, marker='o', vmin=-3, vmax=3, cmap='jet')
cbar = colorbar(sct)
cbar.ax.set_ylabel('colorbar␣label')
show()
```

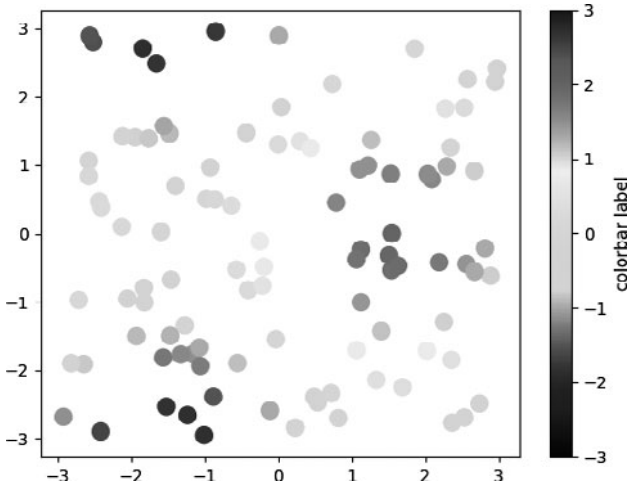

図14　List 14 の描画結果.

3.2.5　最後に

matplotlib の関数インターフェースを用いて，よく利用されるグラフを描画できるように紹介してきました．matplotlib は，オブジェクトの集合として構成されているため，関数インターフェースを使っていても，オブジェクトを関数に渡したり，オブジェクトの機能を呼び出したりと，どうしてもオブジェクトを介した記述が出てきてしまいました．オブジェクトについては，Python とそのモジュールを利用していくと自然とわかってくるかと思います．また，matplotlib の公式ページには，多くのプロット例があり，それを描画したスクリプトを見ることができます．公式のページ以外にも，インターネットでアクセスできる情報も豊富にありますので，検索してみると，解決策が見つかることでしょう．

さて，matplotlib を使ってみた感想はいかがでしょうか．望みのグラフを得るためには，いちいち設定しないといけないと，面倒に感じたでしょうか．そのような場合にこそ，いろいろな設定の手間なく，最低限のデータさえ与えれば望むようなプロットができる自分用の plot 関数を Python で書くときです．ぜひ，Python と matplotlib を活用してください．