Google Colabで試せます

独自の描画ライブラリを作って研究効率をアップ
Matplotlibグラフ・マスタへの道

第1回　心電図で折れ線グラフを体験

辰岡 鉄郎

グラフ描画プログラミングをより効率的に

● 目的…独自の描画ライブラリを作る

連載では，カスタマイズできるグラフ描画ライブラリを作っていきます．これまで同じようなグラフ描画のコードを何度も書いているという経験はありませんか．凡例の有無，グリッド線の設定など多少の違いだけで，ライブラリ化したら効率的ではないかと思いつつ，ついコードをコピーして類似のコードを量産してしまってはいないでしょうか．本稿が独自ライブラリ作成の一助となり，グラフ描画プログラミングの効率化につながれば幸いです．

● Matplotlibの虎の巻を扱った記事

Interface 2021年6月号にて，Pythonのグラフ描画ライブラリであるMatplotlib（マットプロットリブ）を整理，解説した「虎の巻」記事を寄稿しました．コマンド一覧をExcelファイル(1)でも提供し，自分流にカスタマイズできることから，本家のCheat sheet(2)とは違った使いやすさが得られるのでは，と考えて執筆したものです．当時，声優さんに協力いただいた紹介動画(3)も作成していますので，本稿と合わせてご覧ください．

虎の巻　Cheat sheet　動画

作るのはこんなライブラリ

● 折れ線グラフを描画する

本稿では，1つのウィンドウに折れ線グラフ（**図1**）を描画する関数を持つライブラリ（正確にはモジュール）を作成します．実体はMatplotlibを呼び出すラッパ関数で，Matplotlibをさらに使いやすくしたものです．

さまざまなグラフのプロパティを引数として与えるだけでよく，直接メソッドを記述するよりもシンプルで分かりやすくなります．よく使うスタイルを初期値とした場合，コード量をさらに減らせるでしょう．グラフ出力は，画面表示とPNG形式でのファイル保存を選択できます．

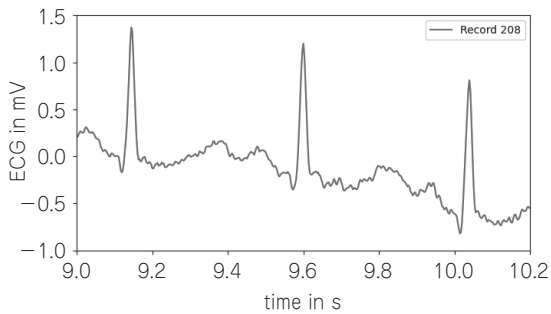

図1　タイトルやサイズ変更など，グラフの装飾をお手軽に

今後，m行n列のタイル状に複数の折れ線グラフを描画する関数や，振幅応答，位相応答を並べた周波数応答を描画する関数など，徐々に発展させていく構想です．

Matplotlibの基礎

● Pythonの可視化ライブラリ…超定番

Matplotlibは，2次元や3次元のグラフ，グラフ・アニメーションなど，さまざまな種類のグラフを作成できるPythonの定番可視化ライブラリです．今後の説明のため，Matplotlibで使われる用語（クラス名）を**図2**に示します．

● 2つのコーディング・スタイル

Matplotlibには，次の2つのコーディング・スタイルがあります．公式のリファレンス・サイト(4)では，簡便に使うときはpyplot-style，再利用を意図した本格的なスクリプトや関数などではOO-styleが推奨されています．本稿でも，OO-styleを使用します．

なお，下記のコードはコーディングスタイルの説明用のため，このままGoogle Colab.に入力してもエラーとなります．入力は少々堪えて先へお進みください．

▶ Object-oriented（OO）スタイル（OO-style）

Figure（グラフ全体）やAxes（1つのグラフ・エリ

プログラムは本誌サポート・ページから入手できます．
https://interface.cqpub.co.jp/2303py/

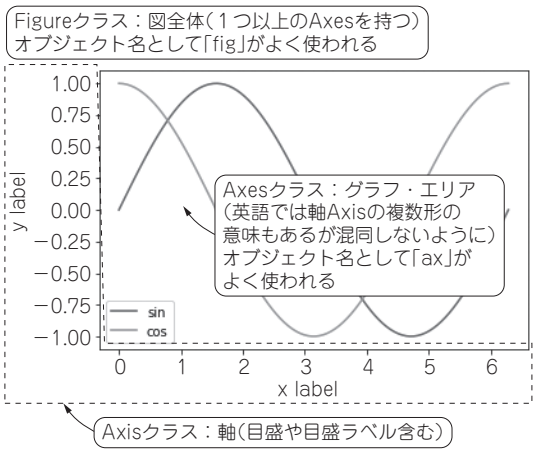

図2　Matplotlibの表示領域や軸を表す用語

ア）オブジェクトを生成し，ax.plot()などオブジェクトのメソッドで記述するスタイルです．

```
fig, ax = plt.subplots()
ax.plot(x, y) # Axesオブジェクトのメソッド
plt.show()
```

▶pyplotスタイル（pyplot-style）

plt.plot()など，pyplot関数で記述するスタイルです．（pyplotモジュールはpltとしてインポート）

```
plt.figure ()
plt.plot(x, y) # pyplot関数
plt.show()
```

これだけは知っておきたい… モジュール，パッケージ，ライブラリ

● モジュール…他から呼び出される拡張子.pyのファイル

モジュールはクラスや関数などを記述した，別のファイルから呼び出されるファイルのことです．実体は，拡張子.pyのファイルです．

● パッケージ…モジュールの集まり

パッケージはモジュールが格納されたフォルダです．フォルダに__init__.pyというファイルを置くことで，そのフォルダはパッケージと認識されます．__init__.pyは空のファイルでも構いませんし，パッケージの初期化コードなどを記述することもできます．

● ライブラリ…外部から呼び出して用いるコードやバイナリ

ライブラリは，Pythonのチュートリアル[5]では，標準ライブラリ，外部ライブラリといった使われ方はされているものの，明確な定義は見当たりません．使用方法からすると，外部から呼び出して用いるコードやバイナリを総称した用語と捉えて良いかと思います．本稿でも，ファイル1つのモジュールをライブラリと表現しています．

ライブラリのインポート方法

ライブラリのインポートの仕方をまとめておきます．下記のモジュールがPythonのカレント・ディレクトリに保存されているものとします．

```
[test_module.py]  ← 対象のモジュール
def func1():
    print('func1')

def func2():
    print('func2')
```

（モジュールの中身）

● モジュールのインポート①…基本形

「import モジュール名」の書式でインポートします．モジュール名は，拡張子.pyを除いたファイル名となります．

```
import test_module
```

▶関数の実行：モジュール名.関数名()

```
>>> test_module.func1()
```

（実行すると「func1」が表示される．以下省略．）

● モジュールのインポート②…関数名のみで実行

次のように記述すると，モジュールの関数を関数名だけで呼び出せるようになります（グローバル名前空間内の関数として読み込まれる）．

```
from test_module import func1, func2
```

▶関数の実行：関数名()

```
>>> func1()
```

上記は，アスタリスクを指定して全ての関数（アイテム）を読み込むこともできます．

```
from test_module import *
```

エイリアスを指定して，関数名を置き換えることも可能です（as以降の文字列に置き換わる）．

```
from test_module import func1 as f1
```

以降では，test_module.pyはカレント・ディレクトリ下のフォルダsamplelibに置かれているとします．

● モジュールのインポート③…サブフォルダの参照

「フォルダ名.モジュール名」の書式でインポートします．

```
import samplelib.test_module
```

▶関数の実行：フォルダ名.モジュール名.関数名()
```
>>> samplelib.test_module.func1()
```

● モジュールのインポート④…エイリアスを使用

エイリアスを指定してモジュール名を置き換えることができます（as以降の文字列に置き換わる）.

```
import samplelib.test_module as
test_mod
```

▶関数の実行：エイリアス名.関数名 ()
```
>>> test_mod.func1()
```

● モジュールのインポート⑤…パッケージの場合

フォルダsamplelib内に__init__.pyがある場合，samplelibはパッケージとして扱われます．このとき，test_moduleはパッケージの構成モジュールとなり，次の書式でインポートできます．

```
from samplelib import test_module
```

▶関数の実行：モジュール名.関数名()
```
>>> test_module.func1()
```

● 直接実行したときだけ動作するコードの記述

モジュールをインポートする際は，モジュール内のコードが実行されますが，インポート時は処理されず，直接スクリプトとして実行したときだけ処理されるコードを記述するには，下記のif文を使用します．

```
if __name__ == "__main__":
    main()
```

モジュールがトップレベルで実行された場合は，グローバル変数__name__には__main__が格納されているため，if文の論理式が真となる仕掛けです．

モジュールの検証用のスクリプトなどをif文以下に記述しておくと便利です．

Google Colab.でPythonプログラミング

皆さんがグーグルのアカウント（メール・アドレス）を持っていることを前提に解説します．ウェブ・ブラウザはGoogle Chromeを使います．

筆者提供の実行ファイル（xx.ipynb）は本誌ウェブ・ページから入手できます．

```
https://interface.cqpub.co.jp/2303py/
```

● モジュールをインポートする

test_module.pyが，

```
Google > マイドライブ >
        Colab Notebooks > samplelib
```

に置かれているものとします．

Google Colab.のノートブックからアクセスするには，カレント・ディレクトリをGoogleドライブに変更する必要があります．

まず，下記のコードを実行し，Googleドライブをマウントします．

```
from google.colab import drive
drive.mount('/content/drive')
```

アクセス権を許可するか確認するウィンドウが開くので「Googleドライブに接続」を選択します．「Google Drive for desktopが…リクエストしています」を許可します．

次に，カレント・ディレクトリを移動します．

```
import os
os.chdir('/content/drive/MyDrive/
Colab Notebooks')
```

あとは，デスクトップ環境と同じように，例えば下記のコードでインポートします．

```
from samplelib.test_module import
func1, func2
```

▶関数の実行：関数名()
```
>>> func1()
```

サンプル・データの概要と取得方法

● scipyが提供している心電図データを使用

例示する波形が正弦波などの人工波形では味気ないということで，今回はscipyが提供している心電図データ[6]を使用しました．有名な心電図のデータセットであるMIT-BIH不整脈データベース[7]のRecord 208のうち，ML Ⅱ誘導の1チャネル分のデータです．

ML誘導は，Mason-Likar誘導法という，標準12誘導の四肢の電極を体幹に変更した誘導法です．四肢誘導より体動ノイズの影響を受けにくい利点があります．

● データの取得方法

下記のコードで，サンプリング周波数360Hzで5分間，108,000点の1次元のデータが変数ecgに格納されます．

```
# 以下の2行は以降の処理のために必要
import numpy as np
from scipy import signal
# データ取得用のコードはここから
from scipy.misc import electro
cardiogram
ecg = electrocardiogram()
```

自作ライブラリの有り無しで比較

ライブラリを使用することで，グラフの描画コードがどのように変わるか見て行きましょう．

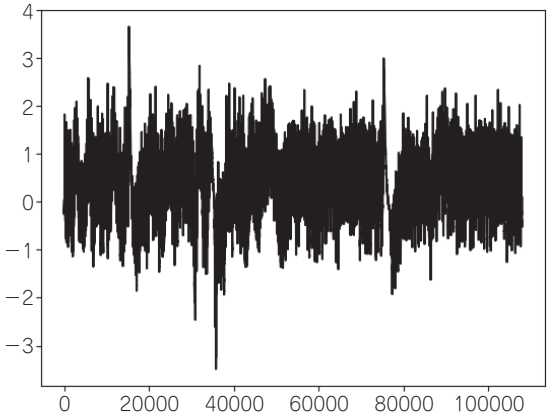

図3　全区間の心電図波形

リスト1　図3を描画するmatplotlibでのコード

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(ecg, 'k')
plt.show()
```

リスト2　図3を描画する自作ライブラリでのコード

```
from samplelib.sampleplot import plot_line
plot_line(ecg, 'k')
```

● 波形色を変更しただけのシンプルなグラフ

図3のような108,000点の波形をグラフの線の色をデフォルトから黒に変更しただけの場合を比較してみます．さすがに事例がシンプル過ぎるので，あまり恩恵を感じられないかもしれませんが，**リスト1**と**リスト2**を比較すると，自作ライブラリでは，Figureオブジェクトの生成やshow()メソッドが不要であること，plot()関数の引数と同一の書式であり，シンプルで理解しやすいことが分かります．

なお，コードは前述の通りOO-styleを採用しています．自作ライブラリはsampleplot.pyというファイル名のモジュールで，samplelibフォルダの下に置かれています．

プロパティを設定したグラフ

もう少し凝ったグラフで比較してみましょう．事前に各サンプルの時刻データと60Hzのノッチ・フィルタを適用した波形を作成しておきます．

```
fs = 360
time = np.arange(ecg.size) / fs
b, a = signal.iirnotch(60.0, Q=4.0,
fs=fs)
```

リスト3　図1を描画するmatplotlibでのコード

```
fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.plot(time, ecg, label='Record 208')
ax.set_title('ECG waveform')
ax.set_xlim([9, 10.2])
ax.set_ylim([-1, 1.5])
ax.set_xlabel('time in s')
ax.set_ylabel('ECG in mV')
ax.legend()
fig.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.plot(time, ecg, label='Original')
ax.plot(time, ecg_filt, label='Filtered')
ax.set_title('ECG waveform')
ax.set_xlim([9, 10.2])
ax.set_ylim([-1, 1.5])
ax.set_xlabel('time in s')
ax.set_ylabel('ECG in mV')
ax.legend()
fig.tight_layout()
plt.show()
```

リスト4　図1を描画する自作ライブラリでのコード

```
params = dict(
    fig_size = (7.0, 4.0),
    plot_kwargs = dict(label='Record 208'),
    title = 'ECG waveform',
    xlim = [9, 10.2],
    ylim = [-1, 1.5],
    xlabel = 'time in s',
    ylabel = 'ECG in mV',
    legend_enable = True,
    tight_layout = True,
)
plot_line(time, ecg, **params)

params['plot_kwargs'] = [dict(label='Original'),
                         dict(label='Filtered')]
plot_line([[time, ecg], [time, ecg_filt]], **params)
```

```
ecg_filt = signal.filtfilt(b, a,
ecg)
```

図1のような，ウィンドウ・サイズの変更やタイトルを付加したグラフを2点描画します（誌面ではフィルタ後の波形との違いが分かりにくいため，1つ目のグラフのみ掲載しています）．**リスト3**と**リスト4**とを比較すると，自作ライブラリでは繰り返しの設定が不要になるためコード量が少なくなっています．

もちろん，ライブラリと同様の関数をスクリプト内に作成すれば，呼び出し側のコードは同じように少なくできます．しかし，これを繰り返した結果，同じような関数を何回もコーディングし，似たような関数を持つファイルが量産されていたのです．別ファイルとしてライブラリ化することで，改善が期待できます．

リスト1やリスト2で呼び出しているグラフ描画ライブラリsamplelib.py

リスト1や**リスト2**で呼び出しているライブラリの中身について解説します．

リスト5　**sampleplot.pyの構造**

```
import os  # 画像ファイルを保存する際に使用
import matplotlib.pyplot as plt  # PyPlotのインポート

def plot_line(*args, **kwargs):   # 唯一の公開関数
    fig, ax = _subplots(kwargs.get('fig_size'))  # ①
    _plot(ax, *args, **kwargs)              # ②
    _set_ax_param(ax, *args, **kwargs)    # ③
    _set_fig_param(fig, **kwargs)     # ④
    _output_graph(fig, **kwargs)      # ⑤

def _subplots(fig_size=None): # ①
    ウィンドウ・サイズを指定してグラフオブジェクトを生成
    (戻り値：Figureオブジェクト，Axesオブジェクト)

def _plot(ax, *args, **kwargs):   # ②
    グラフの描画(_plot_entity()(⑥)の呼び出し)
    (*argsがリストの場合は，複数回plotする)

def _plot_entity(ax, scale_mode, *args, **kwargs):  # ⑥
    グラフの描画(ax.plot()またはax.semilogx()など)

def _set_ax_param(ax, *args, **kwargs):  # ③
    グラフ・エリア・プロパティの設定(ax.set_title()など)
    (タイトル，軸(範囲，ラベル等)，凡例，グリッド)

def _set_fig_param(fig, **kwargs):    # ④
    グラフ・プロパティの設定(fig.tight_layout()の設定)

def _output_graph(fig, **kwargs): # ⑤
    グラフ・ウィンドウの表示，または画像ファイルの保存

if __name__ == "__main__":
    検証用コード
```

● ライブラリの構造

ライブラリ(モジュール)sampleplot.py全体の構造は**リスト5**のようになっています．こちらのコードも，本誌サポート・ページから入手できます．

● 外部から呼び出される公開関数：plot_line()

外部に公開しているパブリックな関数はplot_line()関数のみで，そこから5つのプライベートな関数が呼ばれます．グラフの描画は次の5ステップに分けることができるため，各ステップを関数化しました．

[グラフ描画の5ステップ]
①グラフ・オブジェクトの生成(fig，ax)
②グラフの描画(ax.plot()など)
③グラフ・エリア(ax)の設定(ax.set_title()など)
④グラフ(fig)の設定(fig.tight_layout()など)
⑤グラフの出力(plt.show()またはfig.savefig())

● グラフ描画関数plot_line()の引数

plot_line(*args，*kwargs)は，可変長引数*argsと，可変長のキーワード引数**kwargsを取ります．

*argsには，描画データx，yを与えます．xを省略すると，ax.plot()と同様，横軸はyのデータ点数になります．描画データは，_plot()，_plot_entity()へと渡されax.plot()にて波形描画されます．

**argsのうち，plot_kwargsはそのままax.plot()のキーワード引数として与えられます．そのため格納される辞書型のキーおよび値は，pyplot.plot()のキーワード引数と同じである必要があります．

残りは，グラフのウィンドウ・サイズやグラフ・エリアのプロパティを設定するために使われます．これらはメソッド[ax.set_xlim()など]にて設定するため，キーワードの定義は本モジュール固有です．定義の具体例は後述します．

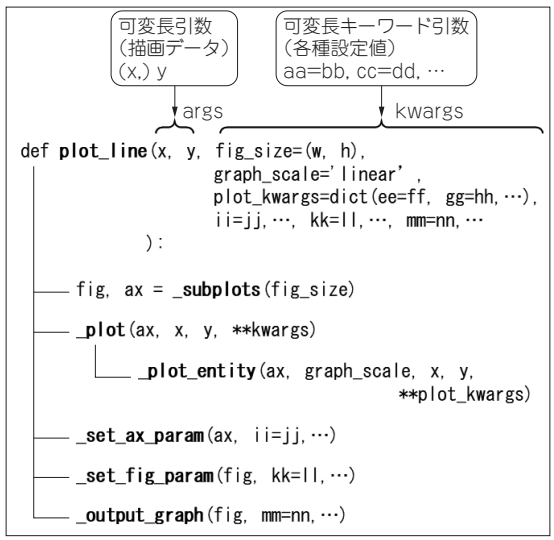

図4　plot_line関数の引数の流れ

リスト5のプライベート関数①～⑥に与える引数は，plot_line()に与えられた引数から展開されるものの他，最初に呼び出される_subplots()関数の戻り値であるオブジェクトfigおよびaxが使われます．引数の流れを**図4**に示します．

その他，モジュール内の各関数の処理について詳細は割愛しますが，特に難しい処理はありませんので，ソースコードをご確認ください．

APIリファレンス

plot_line(*args, **kwargs)
折れ線グラフを描画する

● シグニチャ

```
plot_line([x], y, [fmt], *, data=None, **kwargs)
plot_line([x], y, [fmt], [x2], y2, [fmt2], ...,
        **kwargs)
plot_line([[[x], y, [fmt]], [[x2], y2, [fmt2]], …],
        **kwargs)
```

(注) ax.plot() を複数回実行したい場合は描画データのリストを指定する.

● 引数

*args:

matplotlib.pyplot.plot() のリファレンス[8]参照

fig_size:	(float, float), default: (6.4, 4.8) グラフ・ウィンドウのサイズ
graph_scale:	{'linear','semilogx','semilogy', 'loglog'} グラフのリニア/ログ・スケールを選択
plot_kwargs:	{'label':str, 'color':str, …} pyplot.plot() のリファレンス[8]参照

(注) ax.plot() を複数回実行したい場合は辞書型変数のリストを実行回数分指定する.

title:	None or str グラフ・タイトル
title_size:	'large', etc. or float グラフ・タイトルのフォント・サイズ
xlim/ylim:	None or [float, float] X軸/Y軸の表示範囲
Xlabel/ylabel:	None or str X軸/Y軸のラベル
legend_enable:	bool 凡例の表示ON/OFF
Labels:	None, str or [str, str, …l 凡例のラベル
legend_loc:	{'best', 'ur', 'ul', 'll', 'lr', 'r', 'cl', 'cr', 'lc', 'uc', 'c'} 凡例の表示位置
grid_enable:	bool グリッド線の表示ON/OFF
grid_axis:	{'both', 'x', 'y'} グリッド線を表示する軸方向
grid_which:	{'both', 'major', 'minor'} グリッド線を表示する目盛
frame_disp:	{'full', 'noticks', 'frameonly', 'removeall'} (全表示, 目盛線削除, 軸線のみ, 全削除) 軸の枠線表示モード
tight_layout:	bool 余白調整のON/OFF
output_mode:	{'show', 'png'} (画面表示, 画像ファイル保存) グラフ出力方式の選択
dir_name:	str 画像ファイル保存時のフォルダパス
file_name:	str 画像ファイル保存時のファイル名

リスト6 plot_line() の使用方法

```
# ライブラリのインポート
from samplelib.sampleplot import plot_line
# グラフのプロパティを指定する辞書型変数の生成
params = dict(key1 = value1,…)
# グラフの描画(plot_line() 関数の呼び出し)
plot_line(x, y, **params)
```

●戻り値

なし

● 使用方法

▶準備 (ファイルの配置)

カレント・ディレクトリにsamplelibフォルダを作成し, その中にplot_line.pyを置きます.

▶描画データの準備

描画データをyまたはx, yを作成します. 形式はpyplot.plot() の描画データと同様です.

▶コードの記述

ライブラリのインポート, 辞書型の変数の生成, plot_line() 関数の呼び出し, の3段階でグラフを描画できます(**リスト6**).

◆参考・引用*文献◆

(1) Matplotlib 2Dグラフ虎の巻, Interface, 2021年6月号, pp.165-168, CQ出版社.
https://interface.cqpub.co.jp/2303py_matplot/

(2) matplotlib Cheatsheets.
https://github.com/matplotlib/cheatsheets

(3) 第2部第2章「お勧め可視化ライブラリMatplotlib」紹介, 特集2 C/C++でPython拡張, Interface, 2021年6月号, CQ出版社.
https://www.youtube.com/watch?v=oiJpJVFBQbg

(4) matplotlib Usage Guide.
https://matplotlib.org/stable/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py

(5) 6. モジュール (Python チュートリアル).
https://docs.python.org/ja/3/tutorial/modules.html

(6) scipy.misc.electrocardiogram.
https://docs.scipy.org/doc/scipy/reference/generated/scipy.misc.electrocardiogram.html

(7) MIT-BIH Arrhythmia Database.
https://physionet.org/content/mitdb/1.0.0/

(8) matplotlib.pyplot.plot().
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

たつおか・てつろう