Matplotlibの使い方：サンプルコードを交えた解説

はじめに

Matplotlibは、Pythonプログラミング言語において、静的、アニメーション、インタラクティブな可視化を作成するための包括的なライブラリです [1]。科学研究、データ分析、エンジニアリングなど、幅広い分野で利用されており、論文やレポート、Webアプリケーションなど、様々な場面で高品質なグラフを作成するために不可欠なツールとなっています [1]。本稿では、Matplotlibの基本的な使い方から、様々なグラフの作成、カスタマイズ、複数のグラフの表示、そしてグラフの保存方法までを、具体的なサンプルコードを交えながら解説します。さらに、より高度なグラフの種類やカスタマイズについても概要を紹介し、Matplotlibの持つ多様な機能の一端を示します。

Matplotlibの基本的な使い方

Matplotlibでグラフを作成する基本的な手順は、以下の通りです [3]。

1. 必要なモジュールのインポート: まず、matplotlib.pyplotモジュールをpltというエイリアスでインポートします。これは、グラフの作成と操作に必要な関数を提供します。多くの場合、数値計算のためにnumpyもインポートされます。
   Python
   import matplotlib.pyplot as plt
   import numpy as np

2. データの準備: グラフで表示するためのデータを用意します。Matplotlibは、NumPy配列やpandasのデータフレームと連携するように設計されており、表形式のデータから簡単にグラフを作成できます [3]。
3. **Figure**と**Axes**の作成: グラフを描画するための土台となるFigure（図）と、実際にデータをプロットするAxes（軸）を作成します。最も簡単な方法は、plt.subplots()関数を使うことです。
   Python
   fig, ax = plt.subplots()

   このコードは、一つのAxesを持つFigureを作成します。複数のAxesを配置したい場合は、plt.subplots(nrows, ncols)のように行数と列数を指定します [1]。
4. プロットの作成: Axesオブジェクトのメソッド（例：ax.plot()、ax.scatter()、ax.bar()など）を使用して、用意したデータをグラフ上に描画します [3]。
5. グラフのカスタマイズ **(**任意**)**: 必要に応じて、グラフにタイトル、軸ラベル、凡例などを追加したり、色や線のスタイルを変更したりすることで、グラフを見やすく、より情報豊かなものにすることができます [1]。
6. グラフの表示: 最後に、plt.show()関数を呼び出して、作成したグラフを表示します。Jupyter Notebookなどの環境では、plt.show()が不要な場合もあります [3]。

様々な種類のグラフ

Matplotlibでは、目的に応じて様々な種類のグラフを作成できます [1]。以下に、代表的なグラフの種類と、それぞれのサンプルコードを示します。

グラフの種類 **(Graph Type)**	主な用途 **(Primary Use Case)**	**pyplot** 関数 **(Function)**
折れ線グラフ (Line Plot)	連続データのトレンド表示 (Showing trends in continuous data)	plt.plot()
散布図 (Scatter Plot)	2変数間の関係表示 (Showing the relationship between two variables)	plt.scatter()
棒グラフ (Bar Chart)	カテゴリデータの比較 (Comparing categorical data)	plt.bar()
ヒストグラム (Histogram)	単一変数の分布表示 (Showing the distribution of a single variable)	plt.hist()

折れ線グラフ **(Line Plot)**
連続的なデータの変化やトレンドを示すのに適しています [1]。

Python

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], )
plt.ylabel('いくつかの数値')
plt.show()
```

散布図 **(Scatter Plot)**
2つの変数の間の関係や分布を視覚化するのに役立ちます [1]。

Python

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

data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
```

```
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100
plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('エントリー a')
plt.ylabel('エントリー b')
plt.show()
```

棒グラフ (Bar Chart)
カテゴリごとのデータの量を比較するのに適しています [1]。

Python

```
import matplotlib.pyplot as plt

names = ['group_a', 'group_b', 'group_c']
values =

plt.figure(figsize=(9, 3))
plt.subplot(131)
plt.bar(names, values)
plt.suptitle('カテゴリカルプロット')
plt.show()
```

ヒストグラム (Histogram)
単一の数値データの分布を視覚的に表現するのに使用されます [1]。

Python

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

# 正規分布に従うランダムなデータを生成
mu, sigma = 0, 1
data = np.random.normal(mu, sigma, 1000)

# ヒストグラムの作成
plt.hist(data, bins=30) # 30個のビンでヒストグラムを作成
plt.xlabel('値')
plt.ylabel('頻度')
plt.title('ヒストグラムの例')
```

```
plt.show()
```

グラフのカスタマイズ

Matplotlibは、グラフの外観を細かくカスタマイズするための豊富なオプションを提供しています [1]。タイトルや軸ラベルの追加、凡例の表示、色やマーカー、線種の変更など、様々なカスタマイズが可能です。

タイトルと軸ラベル

グラフの目的や内容を明確にするために、タイトルと軸ラベルを設定することは非常に重要です [4]。plt.title()関数でグラフ全体のタイトルを、plt.xlabel()関数でx軸のラベルを、plt.ylabel()関数でy軸のラベルを設定できます [5]。

Python

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], )
plt.title('基本的なプロット')
plt.xlabel('X軸')
plt.ylabel('Y軸')
plt.show()
```

凡例

複数のグラフを一つのAxesに描画した場合、それぞれのグラフが何を表しているかを区別するために凡例が必要です [1]。各プロット関数（例：plt.plot()）にlabel引数を指定し、その後plt.legend()関数を呼び出すことで凡例を表示できます [4]。凡例の位置は、loc引数で指定できます [4]。

Python

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], , label='データ1')
plt.plot([1, 2, 3, 4], , label='データ2')
plt.xlabel('X軸')
plt.ylabel('Y軸')
plt.title('凡例付きプロット')
plt.legend()
plt.show()
```

色、マーカー、線種

折れ線グラフや散布図では、色、マーカーの形状、線のスタイルなどをカスタマイズすることで、視覚的な表現力を高めることができます [4]。これらのプロパティは、plt.plot()関数などのプロット関数の引

数として指定できます。
フォーマット文字列を使うと、色、マーカー、線種をまとめて指定できます [5]。例えば、'ro--'は、赤い（'r'）丸（'o'）マーカーで、破線（'--'）の線を表します。

Python

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], , 'ro--') # 赤い丸マーカーと破線
plt.xlabel('X軸')
plt.ylabel('Y軸')
plt.title('色、マーカー、線種のカスタマイズ (フォーマット文字列)')
plt.show()
```

キーワード引数を使って、個別にこれらのプロパティを指定することも可能です [4]。

Python

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], , color='green', marker='^', linestyle='-.', linewidth=2, markersize=8)
plt.xlabel('X軸')
plt.ylabel('Y軸')
plt.title('色、マーカー、線種のカスタマイズ (キーワード引数)')
plt.show()
```

このように、フォーマット文字列は簡潔にスタイルを指定できる一方、キーワード引数はより詳細な設定や可読性の向上に役立ちます。状況に応じて使い分けることが推奨されます。

複数のグラフを一つの図に表示する

複数のグラフを一つの図に並べて表示することで、異なるデータセットの比較や、同じデータに対する異なる視点からの可視化が可能になります [1]。Matplotlibでは、subplot()とsubplots()という2つの主な方法でこれを実現できます。

subplot() の使い方

plt.subplot(nrows, ncols, index)関数は、図をnrows行ncols列に分割し、index番目のサブプロットをアクティブにします（インデックスは1から始まり、行優先で数えます） [5]。

Python

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

```
t = np.arange(0.0, 5.0, 0.1)
plt.figure() # 新しいFigureを作成
plt.subplot(2, 1, 1) # 2行1列の1番目のサブプロット
plt.plot(t, np.exp(-t))
plt.title('Subplot 1')
plt.subplot(2, 1, 2) # 2行1列の2番目のサブプロット
plt.plot(t, np.cos(2*np.pi*t))
plt.title('Subplot 2')
plt.xlabel('時間')
plt.show()
```

subplots() の使い方
plt.subplots(nrows, ncols)関数は、Figureオブジェクトと、Axesオブジェクトの配列（または単一のAxesオブジェクト）を返します [1]。これにより、個々のサブプロットに対してより直接的に操作を行うことができます。

Python

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
fig, axs = plt.subplots(2, 1) # 2行1列のAxesオブジェクトの配列を作成
axs.plot(x, y1)
axs.set_title('Sine Wave')
axs[1].plot(x, y2)
axs[1].set_title('Cosine Wave')
plt.tight_layout() # サブプロット間の間隔を自動調整
plt.show()
```

subplots()を使うと、返されるaxsオブジェクトが配列となるため、インデックスを使って個々のサブプロットにアクセスし、それぞれのプロットに対して個別の設定を行うことができます。また、plt.tight_layout()関数は、サブプロットのタイトルやラベルが重ならないように自動的にレイアウトを調整してくれる便利な機能です [1]。
一般的に、より複雑なレイアウトや個々のサブプロットに対する詳細な制御が必要な場合は、subplots()の使用が推奨されます。一方、単純なサブプロットの配置であれば、subplot()でも十分に対応できます。

作成したグラフを画像ファイルとして保存する

作成したグラフは、レポートやプレゼンテーションなどで利用するために、PNGやJPEGなどの画像

ファイルとして保存できます [1]。これには、plt.savefig()関数を使用します。
PNG形式で保存
グラフをPNG形式で保存するには、plt.savefig()関数の引数に、拡張子.pngが付いたファイル名を指定します。

Python

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], )
plt.savefig('your_plot.png')
```

JPEG形式で保存
同様に、JPEG形式で保存する場合は、拡張子を.jpgまたは.jpegにします。

Python

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], )
plt.savefig('your_plot.jpg')
```

plt.savefig()関数は、PNGやJPEGの他にも、PDF (.pdf)、SVG (.svg) など、様々なファイル形式をサポートしています。ファイル形式は、指定したファイル名の拡張子から自動的に判断されます。また、dpi（dots per inch）パラメータを指定することで、保存する画像の解像度を制御することも可能です [1]。高いDPI値を指定すると、より高解像度の画像が得られます。

より高度なグラフ

Matplotlibは、基本的なグラフの種類に加えて、より高度なグラフやカスタマイズの機能も提供しています。
3Dグラフの概要
Matplotlibでは、3次元のデータを可視化するための3Dグラフを作成することも可能です [1]。これには、mpl_toolkits.mplot3dモジュールからAxes3Dクラスをインポートする必要があります。基本的な手順としては、Figureを作成した後、projection='3d'というキーワード引数を指定してサブプロットを追加します。

Python

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')
# ここに ax.plot3D(), ax.scatter3D() などを使って3Dデータをプロットします。
plt.show()
```

3Dグラフの作成やカスタマイズには、plot3Dやscatter3Dといった専用の関数が用意されており、これらを活用することで、三次元空間におけるデータの分布や関係性を視覚的に捉えることができます。

その他の高度なカスタマイズ

Matplotlibでは、グラフのほぼすべての要素に対して、より細かく高度なカスタマイズを行うことができます [6]。

- ランタイムコンフィギュレーション **(rc**パラメータ**)**: matplotlib.rcParamsオブジェクトやmatplotlib.rc()関数を使用することで、線の太さ、線のスタイル、軸のプロパティなど、グラフの様々なデフォルト設定を動的に変更できます [6]。
- スタイルシート: matplotlib.style.use()関数を使うことで、あらかじめ定義されたスタイルシートを適用し、グラフの色やフォントなどの外観を簡単に変更できます。独自のスタイルシートを作成することも可能です [6]。
- **matplotlibrc**ファイル: Matplotlibの起動時に読み込まれる設定ファイルで、Figureのサイズ、DPI、線のスタイル、色、フォントなど、あらゆる種類のプロパティのデフォルト値を設定できます [6]。

これらの高度なカスタマイズオプションを利用することで、特定のニーズや出版物の要件に合わせた、洗練されたグラフを作成することができます。

まとめ

本稿では、Pythonの強力なグラフ描画ライブラリであるMatplotlibの基本的な使い方について、サンプルコードを交えながら解説しました。データの準備から始まり、基本的なグラフの作成、タイトルやラベル、凡例などのカスタマイズ、複数のグラフを一つの図に表示する方法、そして作成したグラフを画像ファイルとして保存する方法までを網羅しました。さらに、3Dグラフや高度なカスタマイズオプションについても触れ、Matplotlibの持つ幅広い機能を紹介しました。

Matplotlibは非常に多機能なライブラリであり、ここで紹介した内容はほんの一部です。より深くMatplotlibを理解し、その機能を最大限に活用するためには、公式ドキュメントやチュートリアルを参照することをお勧めします [7]。Matplotlibを使いこなすことで、データ分析や研究における可視化の可能性が大きく広がるでしょう。

引用文献

1. Matplotlib Tutorial - GeeksforGeeks, 3月 24, 2025にアクセス、https://www.geeksforgeeks.org/matplotlib-tutorial/
2. Matplotlib - PyPI, 3月 24, 2025にアクセス、https://pypi.org/project/matplotlib/
3. Quick start guide — Matplotlib 3.10.1 documentation, 3月 24, 2025にアクセス、https://matplotlib.org/stable/users/explain/quick_start.html
4. Introduction to Plotting with Matplotlib in Python | DataCamp, 3月 24, 2025にアクセス、https://www.datacamp.com/tutorial/matplotlib-tutorial-python
5. Pyplot tutorial — Matplotlib 3.10.1 documentation, 3月 24, 2025にアクセス、

https://matplotlib.org/stable/tutorials/pyplot.html
6. Customizing Matplotlib with style sheets and rcParams — Matplotlib ..., 3月 24, 2025にアクセス、https://matplotlib.org/stable/users/explain/customizing.html
7. Matplotlib 3.7.1 documentation, 3月 24, 2025にアクセス、https://matplotlib.org/3.7.1/
8. Matplotlib 3.1 documentation - DevDocs API Documentation, 3月 24, 2025にアクセス、https://devdocs.io/matplotlib~3.1/