第5回 グラフの作成

第5回では、Matplotlibというパッケージを使ったグラフの作成についてご紹介します。

また、最終回となる次回の第6回ではみなさんにプログラムの作成・実行にチャレンジしていただきますので、そこに向けた準備も行っていきます。

はじめに、本講義で使用するファイルを皆さんの環境にダウンロードするため、次のコードを実行してください。

In [ ]:

```
!wget https://github.com/tendo-sms/python_seminar_2022/raw/main/lecture5/files.zip .
!unzip files.zip
!mv files/* .
```

Matplotlibの紹介

Matplotlibは、Pythonで様々なグラフを作成することができるパッケージです。

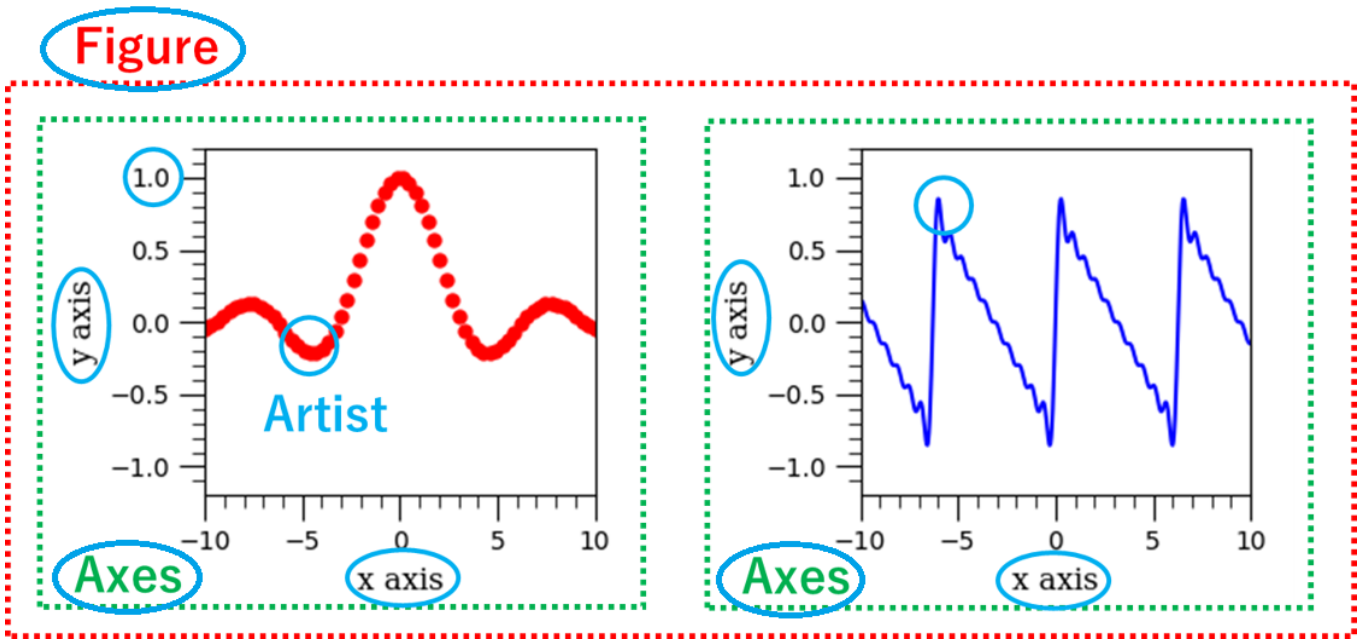
Matplotlibの構成要素

Matplotlibについて学ぶにあたり、基礎知識として、Matplotlibで作成するグラフの構成要素を説明します。

Matplotlibでグラフを作成する際の構成要素として、次のものがあります。

- 1つ以上のグラフを描画する領域全体のことを「**Figure**」と呼びます。
- Figure内に個々のグラフを描画する領域のことを「**Axes**」と呼びます。
- グラフを構成する部品(目盛りラベル、グラフ自体、軸のラベルなど)それぞれのことを「**Artist**」と呼びます。なお、FigureやAxesもArtistです。

FigureとAxesは階層構造になっていて、Figureの中に複数のAxesを持っています。



グラフの作成

それでは早速、Matplotlibを使ったグラフ作成の手順をご紹介します。

最初は、シンプルな折れ線グラフを1つ作成する例でご説明します。

折れ線グラフは、データ構造化プログラムにおいてスペクトルデータの可視化などによく利用されますので、ぜひ覚えておきましょう。

全体の流れは、次のとおりです。

1. モジュールのインポート
2. Figureの作成
3. Axesの作成
4. 折れ線グラフのプロット
5. グラフの画面表示
6. グラフのクローズ

モジュールのインポート

Matplotlibを利用するときは、import文でモジュールをインポートします。

Matplotlibパッケージにはいくつかのモジュールがありますが、もっともよく使用されるのが「**matplotlib.pyplotモジュール**」です。

次のようにインポートします。

```
imoprt matplotlib.pyplot as plt
```

第3回では、numpyモジュールをインポートするときに慣例として「np」という別名を付けること、およびpandasモジュールをインポートするときに慣例として「pd」という別名を付けることをご紹介しました。

matplotlib.pyplotモジュールをインポートするときも、慣例として「plt」という別名を付けるます。これもPythonプログラマの共通認識のようなものですので、慣例に倣うようにしてください。

【注意事項】

Google ColaboratoryやAnacondaでは、Matplotlibが最初からインストールされているのでimport文だけで利用できます。公式版Pythonでは最初はMatplotlibがインストールされていませんので、プログラムを作成する前に、あらかじめpipコマンドを使ってインストールしてください。

```
pip install matplotlib
```

Figureの作成

ここから、実際にソースコードを記述していきます。

最初は、グラフを描画する領域全体であるFigureを作成します。

Figureの作成には、matplotlib.pyplotモジュールの「**figure関数**」を使用します。基本的な構文は次の通りです。

```
変数 = plt.figure()
```

matplotlib.pyplotモジュールをpltという別名でインポートしていますので、「matplotlib.pyplot.figure()」ではなく「plt.figure()」とする点に注意してください。

これにより、作成されたFigureオブジェクトが変数に格納されます。

Axesの作成

次に、Figure内に個々のグラフを描画する領域であるAxisを作成します。

Axesの作成には、Figureオブジェクトの「**subplotsメソッド**」を使用します。基本的な構文は次のとおりです。

```
変数 = Figureオブジェクト.subplots()
```

これにより、作成されたAxesオブジェクトが変数に格納されます。

折れ線グラフのプロット

次に、メインとなる折れ線グラフのプロットを行います。

折れ線グラフのプロットには、Axesオブジェクトの「**plotメソッド**」を使用します。基本的な構文は次の通りです。

```
Axesオブジェクト.plot(X軸の配列, Y軸の配列)
```

X軸の配列およびY軸の配列には、リスト、pandasの1次元データ配列であるSeries、(第3回でちょっとだけご説明した)NumPyのndarrayなどを指定します。

グラフの画面表示

plotメソッドを実行しただけでは、グラフが画面に表示されません。

グラフを表示するには、matplotlib.pyplotモジュールの「**show関数**」を使用します。基本的な構文は次のとおりです。

```
plt.show()
```

グラフのクローズ

グラフの表示が終わったら、クローズ処理を行ってください。

グラフをクローズするには、matplotlib.pyplotモジュールの「**close関数**」を使用します。基本的な構文は次のとおりです。

```
plt.close(Figureオブジェクト)
```

グラフのクローズ処理を行わないと、メモリ等のマシンリソースを無駄に消費し続けてしまいます。第3回の「標準的なファイルの読み書き」で、オープンしたファイルを最後にクローズするようご説明しましたが、それと同様です。

小規模なプログラムでは実害がないためクローズを忘れてしまいがちですが、忘れずにクローズするクセを付けておきましょう。

折れ線グラフを表示するソースコードの例

ここまで、折れ線グラフの表示に必要な関数およびメソッドを順に紹介してきました。

早速、これらの関数およびメソッドを使ってグラフを作成してみましょう。

次のソースコードは、ここまでにご紹介した関数およびメソッドを順に実行しています。実際に動かして、結果を確認してみましょう。

In [ ]:

```
import matplotlib.pyplot as plt

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
ax.plot([1, 2, 3], [10, 100, 1000])

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

無事、はじめてのグラフが書けましたね。

【注意事項】

前述のとおり、close関数を実行しなくてもグラフの作成はできてしまいます。

それに加えてGoogle ColaboratoryやJupyter Notebookでは、show関数とclose関数を両方とも実行しない場合でも、グラフの作成ができてしまいます。一方、show関数を実行せずにclose関数を実行すると、グラフが作成されません・・・・

いつもGoogle ColaboratoryやJupyter Notebookだけを使っていて間違った知識を付けてしまう、といったことがないよう、気を付けるようにしてください。

スペクトルデータのグラフ化

はじめてのグラフとしてごく簡単な折れ線グラフを作成しましたが、リアリティのないデータで、あまり面白くありませんよね。

そこで、スペクトルをイメージしたデータを折れ線グラフにしてみましょう。

次のような内容のCSVファイル(spectrum1.csv)を読み込んで、グラフを作成します。

Wavelength	Intensity
300	409

Wavelength	Intensity
310	447
320	383
330	563
以下略	以下略

CSVデータの取り扱いということで・・・そうです、pandasを使いましょう。

pandasでは次のように記述することで、カラムをSeries(1次元の配列データ)として抽出できます。

    データフレーム[カラム名]

Seriesについては、第4回の講義資料「pandasで取り扱うデータ形式」にて復習しておきましょう。

第4回講義資料のURL：

https://colab.research.google.com/drive/1SgoL61X2fP7KLDFz2Npx9VaEHiSt_IZI?usp=sharing
(https://colab.research.google.com/drive/1SgoL61X2fP7KLDFz2Npx9VaEHiSt_IZI?usp=sharing)

次のとおり指定すれば、カラム名Wavelengthのカラム、およびカラム名Intensityのカラムが、それぞれSeriesとして抽出できます。

    データフレーム["Wavelength"]
    データフレーム["Intensity"]

前の例では、plotメソッドの引数としてX軸となるリストとY軸となるリストを渡しました。

    ax.plot([1, 2, 3], [10, 100, 1000])

これらの引数に、今度はpandasのSeriesを指定します。次のソースコードを実行してみてください。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum1.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
# 引数にはpandasのSeriesを指定する
ax.plot(df["Wavelength"], df["Intensity"])

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

複数本の折れ線を描画する

一つのグラフに複数本の折れ線を描画する場合、plotメソッドを次のとおり実行します。

```
Axesオブジェクト.plot(X軸の配列1, Y軸の配列1)
Axesオブジェクト.plot(X軸の配列2, Y軸の配列2)
                    :
                    :
```

このとき、同一のAxesオブジェクトに対して複数回plotメソッドを実行するという点に注意してください。

先ほどの折れ線グラフに、もう1本折れ線を追加してみましょう。

次のような内容のCSVファイル(spectrum2.csv)を読み込んで、2種類のスペクトルを描画するグラフを作成します。

Wavelength	Intensity1	Intensity2
300	409	298
310	447	349
320	383	393
330	563	480
以下略	以下略	以下略

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む ★読み込むファイルを変更
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット ★変更
ax.plot(df["Wavelength"], df["Intensity1"])
ax.plot(df["Wavelength"], df["Intensity2"])

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

複数のグラフを並べて表示する

先ほどのように2つのスペクトルデータを重ねて表示するのではなく、2つのグラフにして並べて表示してみましょう。

ここでは、1つのFigureの中に2つのAxesを作成し、それぞれのAxesにグラフを描画します。

まず、subplotsメソッドでAxesオブジェクトを作成する記述を次のようにします。

```
変数 = Figureオブジェクト.subplots(縦サイズ, 横サイズ)
```

変数には、Axesオブジェクトのリストが格納されます。リストの形式は、次のとおりとなります。

- 縦サイズまたは横サイズが1の場合、1次元のリストになります。
  - 縦サイズを2、横サイズを1とすると、Axesが縦に2つ並んで作成されます。変数名[0]が上のAxes、変数名[1]が下のAxesとなります。
- 縦サイズと横サイズが両方とも2以上の場合、2次元のリストになります。
  - 縦サイズを2、横サイズを2とすると、Axesが左上／右上／左下／右下と並んで4つ作成されます。変数名[0][0]が左上のAxes、変数名[0][1]が右上のAxes、変数名[1][0]が左下のAxes、変数名[1][1]が右下のAxesととなります。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成 ★引数を変更
ax = fig.subplots(2, 1)

# 折れ線グラフのプロット ★2つのプロットに変更
ax[0].plot(df["Wavelength"], df["Intensity1"])
ax[1].plot(df["Wavelength"], df["Intensity2"])

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

グラフが2つ、縦に並んで表示されましたね。確認できたら、プログラムを修正して、横に2つ並べてみてください。

グラフの装飾

グラフには、様々な装飾をつけることもできます。よく使う装飾を、いくつかご紹介します。

折れ線の種類を変更する

ここまでの例では、シンプルな直線の折れ線グラフを作成してきました。ここでは、線の種類を変更するための機能をいくつかご紹介します。

線の種類を変更するには、plotメソッドに引数を追加します。

```
Axesオブジェクト.plot(X軸の配列, Y軸の配列, 引数名1=値1, 引数名2=値2, ・・・)
```

よく使われる引数の例を、次に示します。

引数	意味	例
linestyle	実線・点線などを指定する	linestyle="--"
linewidth	線の太さを指定する	linewidth=2.0
color	線の色を変更する	color="red"
maker	線にマーカーを付ける	marker="s"

ここでは、以下の装飾をします。

- Intensity1の線を赤色にします。
- Intensity2の線を点線にします。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット ★引数を追加
ax.plot(df["Wavelength"], df["Intensity1"], color="red")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

グラフにタイトルを付ける

グラフにタイトルを付けるには、Axesオブジェクトの「**set_titleメソッド**」を実行します。基本的な構文は次のとおりです。

Axesオブジェクト.set_title(ラベル文字列)

前述した2つのスペクトルを重ねたグラフに、タイトルを付けてみましょう。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
ax.plot(df["Wavelength"], df["Intensity1"], color="red")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--")

# グラフのタイトルを設定 ★追加
ax.set_title("Spectrum Graph")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

set_titleメソッドの実行は、Axesオブジェクトを作成後、show関数を実行するまでのどこで行っても構いません。

上記の例ですと、plotメソッドの前で実行しても同じ結果となります。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# グラフのタイトルを設定 ★位置を変更
ax.set_title("Spectrum Graph")

# 折れ線グラフのプロット
ax.plot(df["Wavelength"], df["Intensity1"], color="red")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

軸にラベルを付ける

軸にラベルを付けるには、Axesオブジェクトの「**set_xlabelメソッド**」および「**set_ylabelメソッド**」を実行します。基本的な構文は次のとおりです。

```
Axesオブジェクト.set_xlabel(ラベル文字列)
Axesオブジェクト.set_ylabel(ラベル文字列)
```

先ほどのグラフに、軸のラベルを付けてみましょう。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
ax.plot(df["Wavelength"], df["Intensity1"], color="red")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--")

# グラフのタイトルを設定
ax.set_title("Spectrum Graph")

# 軸のラベルを設定 ★追加
ax.set_xlabel("Wavelength")
ax.set_ylabel("Intensity")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

set_xlabelメソッドおよびset_ylabelメソッドの実行は、set_titleメソッドと同様に、Axesオブジェクトを作成後、show関数を実行するまでのどこで行っても構いません。

凡例を付ける

グラフに凡例を付けるには、Axesオブジェクトの「**legendメソッド**」を実行します。基本的な構文は次のとおりです。

```
Axesオブジェクト.legend()
```

また、凡例で表示するラベルは、plotメソッドの引数にlabelを追加して指定します。

```
Axesオブジェクト.plot(X軸の配列, Y軸の配列, label=ラベル)
```

先ほどのグラフに、凡例を付けてみます。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット ★引数を変更
ax.plot(df["Wavelength"], df["Intensity1"], color="red", label="Intensity1")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--", label="Intensity2")

# グラフのタイトルを設定
ax.set_title("Spectrum Graph")

# 軸のラベルを設定
ax.set_xlabel("Wavelength")
ax.set_ylabel("Intensity")

# 凡例を設定 ★追加
ax.legend()

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

set_titleメソッドなどとは異なり、legendメソッドは、plotメソッドの後で実行する必要があります。注意してください。

グラフの補助線を引く

グラフに補助線を付けるには、Axesオブジェクトの「**gridメソッド**」を実行します。基本的な構文は次のとおりです。

```
Axesオブジェクト.grid()
```

先ほどのグラフに、補助線を付けてみます。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
ax.plot(df["Wavelength"], df["Intensity1"], color="red", label="Intensity1")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--", label="Intensity2")

# グラフのタイトルを設定
ax.set_title("Spectrum Graph")

# 軸のラベルを設定
ax.set_xlabel("Wavelength")
ax.set_ylabel("Intensity")

# 凡例を設定
ax.legend()

# 補助線を設定 ★
ax.grid()

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

[ご参考] 日本語の使用について

Matplotlibは、標準のフォントが日本語に対応していないため、タイトルや軸のラベルなどに日本語を表示することができません。

グラフ内に日本語を表示するには、日本語対応のフォントに変更する必要があります。

以下では、Googleによって開発されたフリーフォントである「Noto Sans JP」を使用して日本語を表示します。

(ただし、凡例については特別なやり方が必要なので、ここでは日本語化していません。)

フォントファイルは「NotoSansJP-Medium.otf」というファイル名で、講義の最初に必要ファイルとしてダウンロードしてあります。

このファイルは以下のページからも取得出来ます。

https://fonts.google.com/noto/specimen/Noto+Sans+JP?query=noto+sans+jp
(https://fonts.google.com/noto/specimen/Noto+Sans+JP?query=noto+sans+jp)

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# フォントマネージャーのインポート ★追加
from matplotlib import font_manager

# フォントファイルをMatplotlibに認識させる ★追加
fp = font_manager.FontProperties(fname="NotoSansJP-Medium.otf")

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
ax.plot(df["Wavelength"], df["Intensity1"], color="red", label="Intensity1")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--", label="Intensity2")

# フォントの設定 ★
hfont = {"fontproperties": fp, "size": 15}

# グラフのタイトルを設定
ax.set_title("スペクトルのグラフ", **hfont)

# 軸のラベルを設定
ax.set_xlabel("波長", **hfont)
ax.set_ylabel("強度", **hfont)

# 凡例を設定
ax.legend()

# 補助線を設定
ax.grid()

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

これで、日本語を表示することができました。

今回はフリーフォントをダウンロードしてそれを使用しましたが、そのシステムにインストール済みの日本語対応フォントをMatplotlibが認識してくれていることもあります。

その場合はset_titleメソッドなどの引数にフォントを指定するだけで日本語が表示できます。

指定できるフォント一覧は、以下のようなコードで確認することが出来ます。

In [ ]:

```
import matplotlib.font_manager as fm

font_list = [f.name for f in fm.fontManager.ttflist]

font_list
```

なお、ARIM事業のデータ構造化プログラムでは、例題でご紹介した方法で、「Noto Sans JP」を標準のフォントとして利用しています。

作成したグラフを保存する

Matplotlibでグラフをファイルとして保存するときは、Figureオブジェクトの「**savefigメソッド**」を使用します。

savefigメソッドでは、解像度(DPI)や出力フォーマットなどを指定できます。

```
Figureオブジェクト.savefig(出力ファイル名, dpi=解像度, format=出力フォーマット)
```

出力フォーマットは、"jpg"、"png"などといった文字列で指定します。

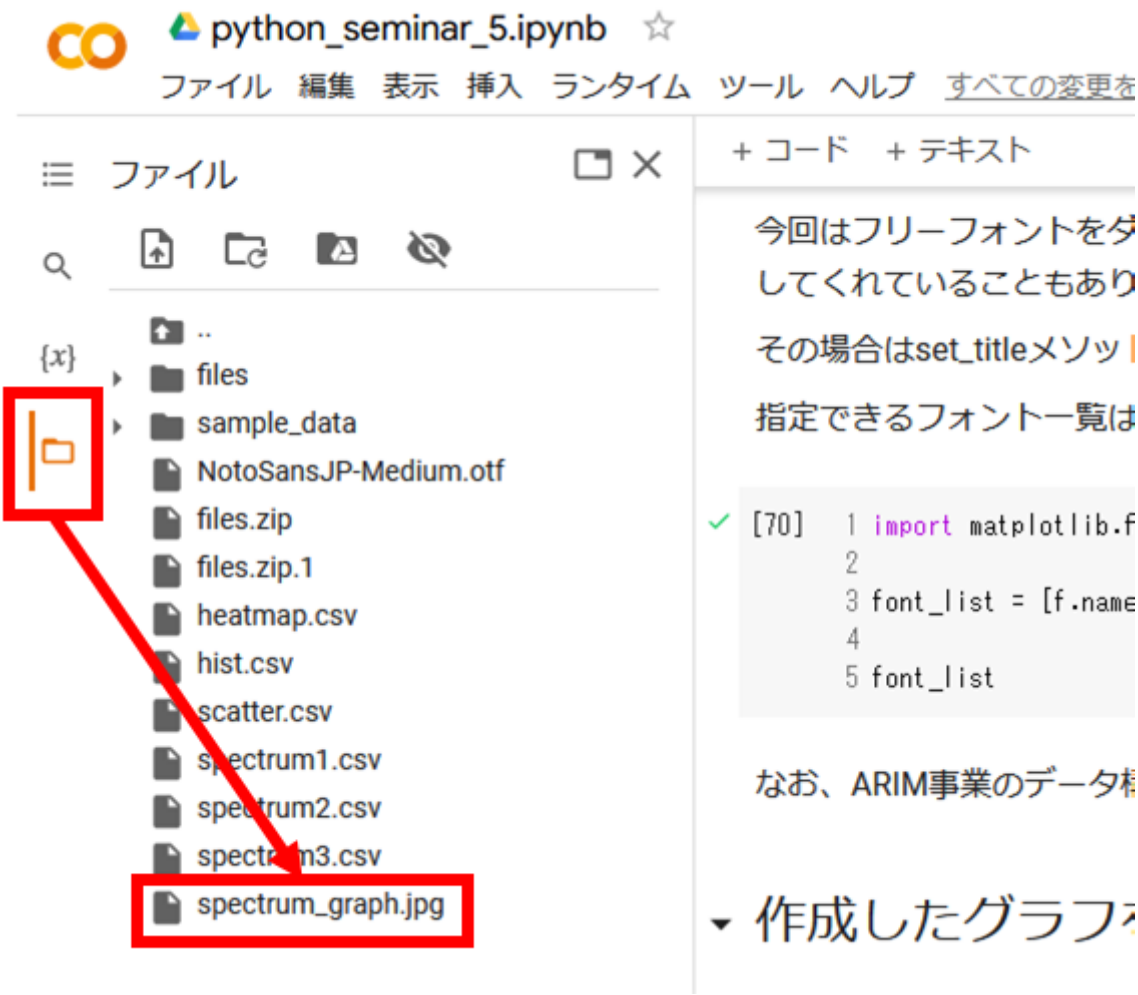
ここでは、DPIを100、出力フォーマットをJPEGとして、先ほどの装飾を施した折れ線グラフを画像として保存する例を示します。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
ax.plot(df["Wavelength"], df["Intensity1"], color="red", label="Intensity1")
ax.plot(df["Wavelength"], df["Intensity2"], linestyle="--", label="Intensity2")

# グラフのタイトルを設定
ax.set_title("Spectrum Graph")

# 軸のラベルを設定
ax.set_xlabel("Wavelength")
ax.set_ylabel("Intensity")

# 凡例を設定
ax.legend()

# 補助線を設定
ax.grid()

# グラフの画面表示
plt.show()

# グラフの保存 ★追加
fig.savefig("spectrum_graph.jpg", dpi=100, format="jpg")

# グラフのクローズ
plt.close(fig)
```

Google Colaboratory画面左のファイル一覧から、「spectrum_graph.jpg」が作成されていることを確認してみましょう。



練習問題

次のような内容のCSVファイル(spectrum3.csv)を読み込んで、1つのグラフ内に3種類のスペクトルを描画するグラフを作成してみましょう。

Wavelength	Intensity1	Intensity2	Intensity3
300	409	298	542
310	447	349	553
320	383	393	552
330	563	480	613
以下略	以下略	以下略	以下略

グラフのタイトル、軸のラベル、および凡例を次のとおり付けてください。

- グラフのタイトルを「Practice Graph」としてください。
- X軸のラベルを「WL」、Y軸のラベルを「Int」としてください。
- 凡例のラベルは、「Int1」「Int2」「Int3」としてください。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("spectrum2.csv")

# 続きのプログラムを作成してみましょう
```

様々なグラフのご紹介

ここからは、折れ線グラフ以外のグラフをご紹介します。本講義では、次のグラフについてご紹介します。

- 散布図
- ヒストグラム
- ヒートマップ

散布図

散布図の作成には、matplotlib.pyplotモジュールの「**scatterメソッド**」を使用します。基本的な構文は次のとおりです。

```
Axesオブジェクト.scatter(X軸の配列, Y軸の配列)
```

ここでは、ある成分の含有率と硬さの関係をイメージしたデータを用いて散布図を作成してみます。

次のような内容のCSVファイル(scatter.csv)を読み込んで、グラフを作成します。

Content	Hardness
0.51	23.3
0.18	20.8
0.46	23.9
0.5	24.6
以下略	以下略

例を見てみましょう。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("scatter.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 散布図のプロット ★線グラフのplotをscatterに変更
ax.scatter(df["Content"], df["Hardness"])

# グラフのタイトルを設定
ax.set_title("Sputter")

# 軸のラベルを設定
ax.set_xlabel("Content")
ax.set_ylabel("Hardness")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

正の相関が見られる散布図が作成されました。

ヒストグラム

ヒストグラムの作成には、matplotlib.pyplotモジュールの「**histメソッド**」を使用します。基本的な構文は次のとおりです。

Axesオブジェクト.hist(データの配列, bins=ビン数)

ここでは、粒子径分布をイメージしたデータを用いてヒストグラムを作成してみます。

次のような内容のCSVファイル(hist.csv)を読み込んで、グラフを作成します。

Particle size
48.1
49.7
2.9
28.5
以下略

例を見てみましょう。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df = pd.read_csv("hist.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# ヒストグラムのプロット ★線グラフのplotをhistに変更
ax.hist(df["Particle size"], bins=10)

# グラフのタイトルを設定
ax.set_title("Histgram")

# 軸のラベルを設定
ax.set_xlabel("Particle size (nm)")
ax.set_ylabel("Frequency")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

データを10個の範囲に区切り、それぞれの出現頻度を示すヒストグラムが作成されました。

ヒートマップ

ヒートマップの作成には、matplotlib.pyplotモジュールの「**imshowメソッド**」を使用します。基本的な構文は次のとおりです。

```
変数 = Axesオブジェクト.imshow(2次元配列)
```

変数には、「**AxesImageオブジェクト**」が格納されます。

また、ヒートマップにはカラーバーを付けることができます。

カラーバーは、Figureの部品(Artist)の一つとして作成されます。基本的な構文は次のとおりです。

```
Figureオブジェクト.colorbar(AxesImageオブジェクト)
```

ここでは、各座標の明るさを記録したイメージの2次元データを用いてヒートマップを作成してみます。

次のような内容のCSVファイル(heatmap.csv)を読み込んで、ヒートマップを作成します。

0	0	0	0	0	略
0	1	1	2	2	略
0	0	2	0	4	略
0	1	0	2	2	略

略　略　略　略　略　略

例を見てみましょう。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをpandasのデータフレームへ読み込む
df = pd.read_csv("heatmap.csv")

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# ヒートマップの作成
im = ax.imshow(df)

# カラーバーの作成
fig.colorbar(im)

# グラフのタイトルを設定
ax.set_title("Heatmap")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

カラーマップの変更

Matplotlibではカラーマップが豊富に用意されており、簡単に変更出来ます。カラーマップは、imshowメソッドの引数で指定します。

変数 = Axesオブジェクト.imshow(2次元配列, cmap=カラーマップ名)

カラーマップ名は、"binary"、"gray"などの文字列で指定します。

なお、引数にcmapを指定しない場合のデフォルトは、"viridis"というカラーマップ名です。

ここでは、コントラストが分かりやすいためよく使われる、「jet」というカラーマップに変更する例を示します。

次のソースコードを実行してみてください。先ほどのソースコードから変更されている箇所には、コメントに「★」を入れています。

In [ ]:

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import pandas as pd

# CSVファイルをpandasのデータフレームへ読み込む
df = pd.read_csv("heatmap.csv")

fig = plt.figure()
ax = fig.subplots()

# ヒートマップの作成 ★変更
im = ax.imshow(df, cmap="jet")

# カラーバーの作成
fig.colorbar(im)

# グラフのタイトルを設定
ax.set_title("Heatmap")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

以下のページにMatplotlibで選べるカラーマップ一覧がまとめられています。

表示するデータやその形式によっても、適切なカラーマップは変わるかもしれません。好みのカラーマップを探してみましょう。

https://matplotlib.org/stable/tutorials/colors/colormaps.html
(https://matplotlib.org/stable/tutorials/colors/colormaps.html)

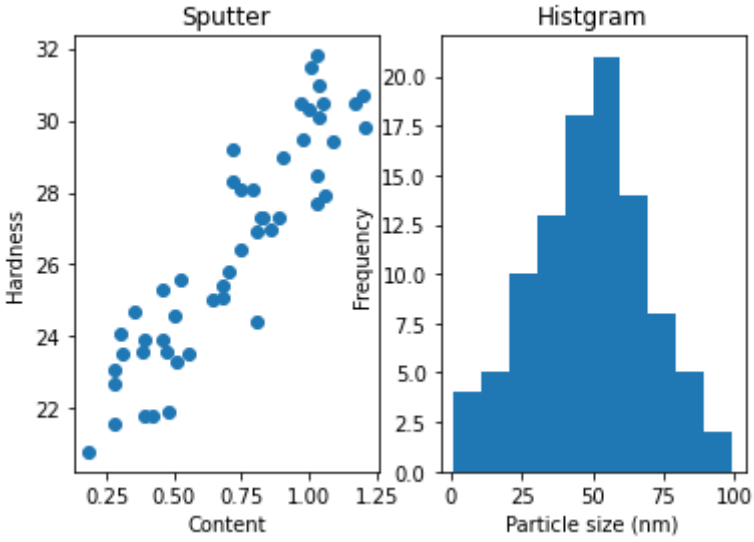
上記のソースコードで、"jet"の部分を色々と変えて動かしてみてください。たとえば下記などは、ちょっと面白い指定ですね。

"spring", "summer", "autumn", "winter"

ここまで、折れ線グラフ以外のいくつかのグラフについてご紹介しました。他にもMatplotlibには様々なグラフ作成機能がありますので、色々と調べて活用してみてください。

練習問題

散布図とヒストグラムを横に並べた、次のようなグラフを作成してみましょう。



散布図とヒストグラム、それぞれの内容は、前述の例と同じものです。

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd

# CSVファイルをデータフレームに読み込む
df1 = pd.read_csv("scatter.csv")
df2 = pd.read_csv("hist.csv")

# 続きのプログラムを作成してみましょう
```

Axesインターフェースとpyplotインターフェース

Matplotlibでグラフを作成する方法として、大きく分けると次の2つの方法があります。

- Axesインターフェース (オブジェクト指向インタフェース)
- pyplotインターフェース

ここまでご紹介してきた例は、すべて「Axesインターフェース」を使ってきました。

ごく簡単な折れ線グラフを例に、それぞれのインタフェースを使ったソースコードを比較してみます。

まずは、Axesインターフェースを示します。これまでご紹介してきた作成方法です。

In [ ]:

```
import matplotlib.pyplot as plt

# Figureの作成
fig = plt.figure()

# Axesの作成
ax = fig.subplots()

# 折れ線グラフのプロット
ax.plot([1, 2, 3], [10, 100, 1000])

# 軸のラベルを設定
ax.set_xlabel("X-Axis")
ax.set_ylabel("Y-Axis")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

次に、pyplotインタフェースのソースコードを示します。

pyplotインタフェースは、数値計算に特化したプログラミング言語である「MATLAB」に近いイメージで操作できるよう、開発されたものです。

In [ ]:

```
import matplotlib.pyplot as plt

# 折れ線グラフのプロット
plt.plot([1, 2, 3], [10, 100, 1000])

# 軸のラベルを設定
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig)
```

pyplotインタフェースでは、FigureやAxexを作成しなくても、同じグラフが書けました。軸のラベル設定もできています。

pyplotインタフェースの方が楽で良い！と思われたかもしれませんが・・・あくまで、1つのグラフをお手軽に作るためのインタフェースです。

Figure内に複数のグラフを書いたり、複雑な設定・装飾をする場合は、Axesインタフェースを使用する必要があります。

特に気を付けなければいけないのは、「**1つのプログラムで、2つのインタフェースを混在して使用しないこと**」です。これを行ってしまうと、ソースコードの読みやすさやメンテナンスのしやすさという観点で、非常に分かりづらいプログラムとなってしまいます。

次の例を見てみましょう。

In [ ]:

```
import matplotlib.pyplot as plt

# ★Axesインタフェース★で折れ線グラフを作成
fig1 = plt.figure()
ax1 = fig1.subplots()
ax1.plot([0, 1, 2], [10, 100, 1000])

# ★Axesインタフェース★で散布図を作成
fig2 = plt.figure()
ax2 = fig2.subplots()
ax2.scatter([0, 1, 2, 3, 4], [50, 20, 30, 10, 20])

# ★pyplotインタフェース★で軸にラベルを付与
plt.xlabel("Y-Axis")
plt.ylabel("Y-Axis")

# グラフの画面表示
plt.show()

# グラフのクローズ
plt.close(fig1)
plt.close(fig2)
```

xlabel関数およびylabel関数によるラベル付けは、最後に作成されたAxesである散布図に付与されました。

pyplotインタフェースによる操作は、暗黙的に「最後に作成したFigure/Axesオブジェクト」に対して行われます。このため、グラフ作成の順序を入れ替えたり、散布図作成の後に新しいグラフ作成を入れたりすると、ラベル付けをどのグラフに対して行うのかが変わってしまいます。

このようにプログラムが分かりづらいものとなりますので、インタフェースを混在させるのはやめましょう。

簡単なテスト用途など以外では、**基本的にAxesインタフェースを使用することをお勧めします**。Matplotlibのリファレンスでも、Axesインタフェースの使用が推奨されています。

[ご参考] Matplotlib以外でのグラフ作成

Pythonでは、Matplotlib以外にも様々なグラフ作成のためのパッケージがあります。

(とはいえ、裏でMatplotlibが動いているものがほとんどです。)

ここでは、代表的な下記のパッケージをご紹介します。それぞれについての詳しいご説明はしませんが、参考にしてください。

- pnadas
- seaborn
- Plotly
- PyQtGraph

pandas

実はpandasにもplotメソッドがあり、簡単な折れ線グラフを作成できます。

In [ ]:

```
import pandas as pd

df = pd.read_csv("spectrum1.csv")
df.plot("Wavelength", "Intensity", linestyle="--")
```

ただし、Google ColaboratoryやJupyter Notebookでは上記のソースコードでグラフを表示できますが、通常の環境では、結局Matplotlib.pyplotをインポートしてshow関数を使ったりしないとグラフが表示されません。

(つまり、裏でMatplotlibが動いているということですね。)

それであれば、グラフの作成もMatplotlibを使った方がよいでしょう。

Google ColaboratoryやJupyter Notebookでのごく簡単な動作確認程度であれば使い道はありますが、基本的には覚える必要はないかと思います。

seaborn

seabornは、Matplotlibをベースに、より見栄えのよいグラフを、より簡単に描画できるように作成されたパッケージです。

例を見てみましょう。この例では、データサイエンスの世界で非常に有名な「iris」データセットについて、すべてのカラムを組み合わせたグラフを描画してくれます。

In [ ]:

```
import seaborn as sns

iris = sns.load_dataset("iris")
sns.pairplot(iris)
```

Plotly

Plotlyは、2次元や3次元のインタラクティブなグラフを作成できるパッケージです。

例を見てみましょう。作成したグラフは、マウスドラッグで回転させたり、マウスホイールで拡大・縮小ができたりします。

In [ ]:

```
import plotly.graph_objs as go
import numpy as np

# プロットデータの作成
t = np.linspace(0, 10*np.pi, 1000)
x = np.cos(t)
y = np.sin(t)
z = t

# グラフを作成
fig = go.Figure(data=go.Scatter3d(x=x, y=y, z=z, mode='lines'))
fig.show()
```

PyQtGraph

PyQtGraphは、PythonでGUIプログラムを作成するのによく使われるパッケージであるPyQt(またはPySide)と連携して、グラフを作成できるパッケージです。

PyQtによるGUIプログラムの一部として利用しやすいことに加え、Matplotlibと比較してグラフの作成が高速であると言われています。

こちらはGoogle Colaboratoryでの動作が難しいためソースコードのご紹介はできませんが、しっかりとしたGUIプログラムを作成したい場合などは、調べてみてください。

第5回のメインであるグラフ作成についての講義は以上となります。

Matplotlibも、第4回のpandasと同様に、限られた講義時間ではお伝えしきれないほどたくさんの機能があります。

特に各関数およびメソッドの引数で設定できる装飾などは、ほんのごく一部しかご紹介できておりません。

ぜひ、ご自身で調べて使いこなしてみてください。

第6回に向けた準備

いよいよ、次の第6回が最終回となります。

第6回では、課題をご提示して実際にみなさんにプログラムを作成してもらいます。これまでの演習問題のような小規模なものではなく、比較的大きなプログラムとなります。

第5回までの講義では、みなさんの受講環境が様々であってもPython実行環境に関するトラブルが発生しないよう、Google Colaboratoryを使ってご説明してきました。

しかし、実際に本格的な構造化プログラムを作成するときは、次のような理由から、Google ColaboratoryやJupyter Notebookで開発するのは効率がよくありません。

- 1つのモジュール(.pyファイル)ではプログラムが完結せず、大きな機能ごとにモジュールを分割する必要がある。
- プログラムが上手く動作せず調査をするときに、プログラム実行途中の変数の値などを確認したいことがある。

そこで第6回の講義では、Google Colaboratoryを使用せず、第1回でご紹介した次の実行環境およびエディタを使って、プログラムの作成を行います。

- Python実行環境として公式版Pythonを使用する。
- ソースコードのエディタとしてVisual Studio Codeを使用する。

公式版PythonおよびVisual Studio Codeについては、第1回講義資料の「Pythonの実行環境・エディタの紹介」でおさらいしておきましょう。

https://colab.research.google.com/drive/17KmL3XUzeUwCQemdGrNbwroqHKy26K-p?usp=sharing
(https://colab.research.google.com/drive/17KmL3XUzeUwCQemdGrNbwroqHKy26K-p?usp=sharing)

公式版PythonおよびVisual Studio Codeのインストール

みなさんにプログラムを作成してもらうにあたり、可能な方は、第6回の講義開始までに公式版PythonおよびVisual Studio CodeをPCにインストールしておいていただけければと思います。

ただし、みなさんの所属機関のルールなどで、ソフトウェアをインストールできない方もおられると思います。また、みなさんのPC環境や操作の習熟度などから、インストールが難しい方もおられると思います。

そのため、公式版PythonおよびVisual Studio Codeのインストールは、みなさまのご判断・ご責任で実施をお願いします。例えば次のような方は、インストールをご検討いただければと思います。

- すでに、公式版PythonとVisual Studio Codeの環境構築・利用の経験がある
- 自由に試行錯誤できるPCを持っており、万が一トラブルがあっても業務・研究等に支障が生じない。

ご参考として、WindowsおよびUbuntu用のインストール手順書を配布します。ただし、みなさんのPC環境は様々ですので、配布する手順書はあくまでご参考としていただき、各自でご自身の環境に対応したセットアップ方法を調べて取り組んでいただきますよう、お願いいたします。

インストールが難しい方は、講義中はお手元のテキストエディタでソースコードの作成のみチャレンジして、実行については講師の解答例・実行デモをご覧いただいて学習するなど、工夫していただければと思います。

(講義中にどなたかを指名して質問したりなどはしないので、ご安心ください。)

それでは、最後の講義を楽しみにお待ちください。

【注意事項】

みなさまのPC環境やネットワーク環境等が多岐にわたることや、リモートでのご支援が難しいことから、公式版PythonおよびVisual Studio Codeのインストールについてはサポートすることができません。

申し訳ございませんが、ご理解のほどよろしくお願いいたします。