

10. 作図の基本

プログラミング・データサイエンス I

2023/6/22

1 今日の目的

今日の目的

- 作図の基本
 - 一枚の図を描く
 - 複数の図を描く
- 図形を描く

Python は、データ分析などで活用事例の多いプログラミング言語です。データから読み取れることを説明する際に、分析結果を図示することで、理解を助けることが可能です。今日は、データ作図の基本について説明します。matplotlib という作図ライブラリを使うことにします。マニュアルは以下の URL にあります。

<https://matplotlib.org/contents.html>

はじめに、一枚の図を描く方法を紹介します。データを点で描く、折れ線で結ぶなどの基本的な方法から始め、図の大きさ、フォントサイズ、作図範囲の指定などの書式設定までを扱います。

作図の際に指定できることは非常に多岐にわたります。それらの多くはデフォルト値、つまり指定しない場合の値が設定されており、必要に応じて設定することになります。それらの詳細は、上記 URL のマニュアルを見るのが基本ですが、インターネット上にも沢山の例があります。また、VSCode 内でも、ヒントとして表示されています。

二番目に、複数の図を描く方法を紹介します。異なる量を横軸や縦軸を揃えて見せたい時に便利です。一枚の図を描く方法とは、コマンド名が少し異なることに注意が必要です。

最後に、円や多角形などの図形の描き方を紹介します。

今日のサンプルプログラムをダウンロードしてください。

<https://github.com/first-programming-saga/plotSample>

2 データを一枚の図に作図する

データを一枚の図に作図する

- 折れ線
- 散布図
- 図の書式設定
 - － タイトル
 - － 軸の名前と範囲
 - － 凡例
- 図の表示と保存

2.1 作図の基本

作図の技術的内容の前に、基本的なことを確認しておきましょう。

図でデータを示すのは、それを使って読者に何かを伝えることが目的です。不適切な方法で図示すると、誤った情報や印象を伝えることになります。見た目を良くする工夫をしすぎたグラフを作成することで、特定の部分を強調してしまうこともあります。

また、図の種類は、データの性質に応じて選択するべきです。例えば、人口の時間的推移を表すならば横軸に時間を縦軸に人口をとって、折れ線グラフとするべきでしょう。あるいは、企業の従業員数と職員の平均年収の関係を示したいならば、データを点で表示する散布図を使います。各県の人口を比較したいならば、横軸を県名、縦軸を人口とした棒グラフを使います。

縦軸や横軸の範囲の選び方も重要です。人口の変化を表す際に、縦軸の範囲を狭くすると、大きく変動した印象を与えます。範囲を広くとると、変化がほとんどないように見えます。

もちろん、図にはタイトルを付け、横軸と縦軸にはラベルをつけなければいけません。単位がある量ならば、単位を示さなければいけません。見やすいように、色、線の太さ、点の大きさ、文字の大きさを調整することも必要です。黄色や黄緑色などの明るい色は、よく見えないことに注意してください。

2.2 簡単なグラフ

それでは、`simplePlot.ipynb` を開いてください。

ソースコード 2.1 ライブラリのインポート

```
1 import matplotlib.pyplot as plt
2 import japanize_matplotlib
```

最初の部分でライブラリをインポートしています (ソースコード 2.1)。`matplotlib` が最初に紹介した作図用ライブラリです。ライブラリの名前が長いので、プログラム中で `plt` と省略できるように、インポートしています。`japanize_matplotlib` は、図の中で日本語を使えるようにするライブラリです。

ソースコード 2.2 作図特性の設定

```
1 plt.rcParams["mathtext.fontset"] = 'cm'
2 plt.rcParams['mathtext.default'] = 'it'
3 plt.rcParams["font.size"] = 28#フォントサイズ
4 plt.rcParams['figure.subplot.bottom']=0.2#横軸の下のスペース
```

二番目のセル (ソースコード 2.2) は、作図の属性をしていします。数式を表示する際のフォントとフォントサイズです。最後の行では、横軸の名前が表示できるように、横軸の下スペースを拡げています。

三番目のセル、ソースコード 2.3 を見てください。5 行目の戻り値 `ax` は、`Axes` という型のオブジェクトで、ここに作図をしていきます。6 行目から 10 行目までが、図の書式を定めている部分です。上から、図のタイトル、 x 軸の範囲、 y 軸の範囲、 x 軸のラベル、そして y 軸のラベルです。 $\$x\$$ は、数式として x を表現することを示しています。数式表現は `LATEX` の表式を使うことができます*¹。

12 行目は、`xList` というデータリストを x 軸の値に、`yList` というデータリストを y 軸の値として、折れ線でデータを結びます。`label` という変数を指定すると、凡例にその文字列を表示します。`linewidth` は線の太さです。

*¹ `LATEX` は数式を使う科学技術論文用に開発された組版システムです。

ソースコード 2.3 作図を行う

```
1 def drawData(xList:list[float], yList:list[float], zList:list[float]):
2     """
3     xlist-ylist を折れ線で、xlist-zlist を点で作図する
4     """
5     fig, ax = plt.subplots(figsize = (10, 10),facecolor = 'white')
6     ax.set_title('グラフタイトル')
7     ax.set_xlim(0, 6)#x 軸の範囲
8     ax.set_ylim(0, 8)#y 軸の範囲
9     ax.set_xlabel('$x$')#x 軸のラベル
10    ax.set_ylabel('$y$')#y 軸のラベル
11
12    ax.plot(xList, yList, label = '理論', linewidth = 2)#折れ線
13    ax.scatter(xList, zList, label = 'データ',
14              color = 'red', marker = 's', linewidths = 5)#散布図
15
16    ax.legend(loc = 'upper right')#凡例の位置
17    plt.savefig('tmp.pdf')#ファイルへ出力
18    plt.show()
```

13 行目は、データを点で表します。color は点の色、marker は点の形、linewidth は点の大きさです。使える色と点の形は、以下の URL にあります。

color https://matplotlib.org/3.1.0/gallery/color/named_colors.html

marker https://matplotlib.org/3.1.0/api/markers_api.html?highlight=list%20markers

16 行目は、凡例を表示しています。英語で凡例を legend と言います。plot や scatter メソッドを使用した際に、label 引数に指定した文字列と作図のマークが対応つけて表示されます。loc='best' とすると、勝手に場所を決めてくれます。

作図した図をファイルに保存し (17 行目)、表示しています (21 行目)。ファイルへの保存の際には、拡張子に合わせた形式になります。ここでは、PDF という形式で保存しています。ファイルに出力した結果を図 1 に示します。

課題 2.1 ソースコード 2.3 において、以下の変更を行ってください。

1. *y* 軸の範囲を変更
2. 折れ線の色を設定
3. 散布図の点の形を変更

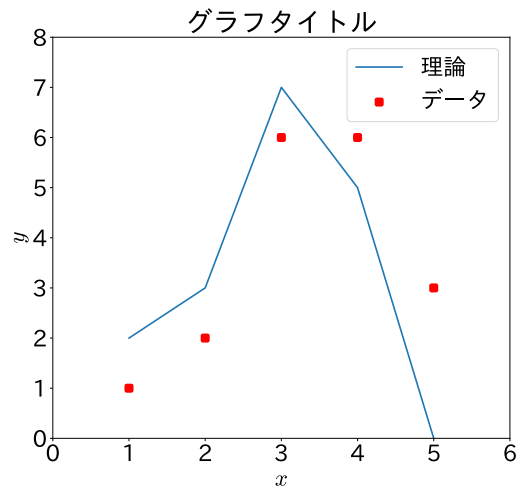


図1 ファイルへの出力結果

3 複数の図を描く

複数の図を描く

- 一つの図に、複数のグラフを描く
- x 軸や y 軸を共有する

3.1 軸を共有した複数の図の例 1

複数の図を縦軸、あるいは横軸を共有しながら描きたいことがあります。その場合は、`subplots()` に図の配置の情報を指定します。また、戻り値の `ax` は、指定した配置に応じた `Axes` という型のリストとなり、各要素が一つの図の領域を表します。

それでは、`multiPlot.ipynb` を見ましょう。作図をしている部分がソースコード 3.1 です。作図属性は別のセルに記述しています。`subplot` の最初の二つの引数 `1, 2` が一行二列の作図領域を作ることを表しています。つまり、横に 2 つ図が並びます。作図領域は `ax` として返ってきます。今回は、`ax[0]` と `ax[1]` です。最後の引数 `sharey = True` が y 軸を共有することを示しています。つまり、縦軸のラベルは左側だけに表示されます。

4 行目は、 y 軸の目盛り (`tick` と言います) とラベル設定です。-1 から 0.5 刻みで、1 ま

ソースコード 3.1 二つの図を左右に並べる

```
1 def drawData(xList:list[float], yList:list[float], zList:list[float]):
2     fig, ax = plt.subplots(1, 2, figsize = (10, 5),
3         sharey = True, facecolor = 'white')
4     plt.yticks([-1 + .5 * i for i in range(6)])#y 軸の目盛り
5     #二つの図に共通の設定
6     for a in ax:
7         a.set_xlim(-5, 5)
8         a.set_xlabel('$x$')
9         a.tick_params(axis = 'both', length = 15, labels=25)
10
11     #最初の図
12     ax[0].set_title('$\sin(x)$')
13     ax[0].plot(xList, yList)
14     #2 番目の図
15     ax[1].set_title('$\cos(x)$')
16     ax[1].plot(xList, zList)
17
18     plt.savefig('multiPlot.pdf')
19     plt.show()
```

で目盛りを作ります。

6 行目から 9 行目は、二つの図に共通の要素を指定しています。ax がリストであったことを思い出してください。x 軸の範囲、x 軸のラベル、そして目盛りの大きさです。

12 行目と 13 行目で ax[0] に、15 行目と 16 行目で ax[1] に、それぞれタイトルを付け、データを折れ線で描いています。

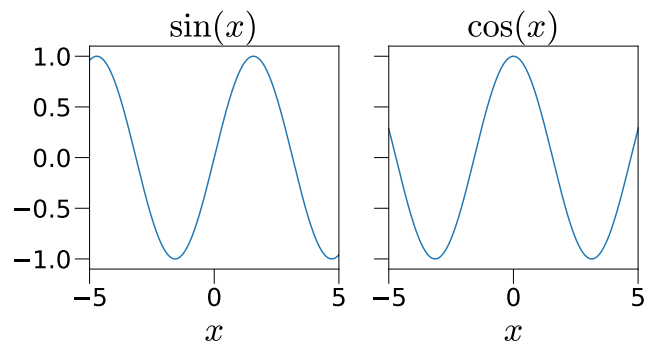


図 2 縦軸を共有した二つの図を描く

作図結果を図 2 に示します。変数名 x がきれいなイタリック体で、三角関数の名前がローマン体で表示されています。また、y 軸は、左の図だけに縦軸の数値が入っています。

3.2 軸を共有した複数の図の例 2

マルチプロット例

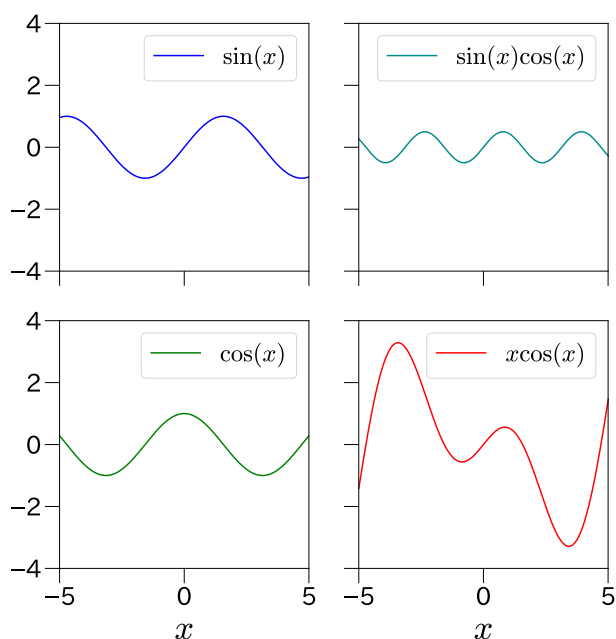


図 3 四つの図を描く

次は、4つの図を、 x 軸と y 軸を共通として描く例です。作図結果は図3のようになります。multiPlot2.ipynbを開けてください(ソースコード 3.2)。

3行目の最初の二つの引数2,2で、 2×2 という四つの図を作ること指定しています。最後の二つの引数が x 軸と y 軸を共通として描くことの指定です。結果として返ってくるaxも 2×2 のリストです。

四つの図に対する共通の設定を9行目から15行目で行っています。9行目の最初のforは、行に関する繰り返しです。10行目の内側のforは、列に関する繰り返しです。繰り返し方が違いますが、理解できますか。14行目では、行の番号を見て、一番最後の行に対してだけ、 x 軸のラベルを描いています。

17行目から、4つの図を折れ線で描いています。最後に、23行目からの二重forループで、凡例を作っています。

ソースコード 3.2 4枚の図を描く

```
1 def draw(x:list[float], y1:list[float], y2:list[float],
2         y3:list[float], y4:list[float]):
3     #xy 軸のスケールを揃える
4     fig, ax = plt.subplots(2, 2, figsize = (20, 20),
5                           sharex = True, sharey = True, facecolor = 'white')
6
7     fig.suptitle('マルチプロット例')
8     #全ての図に共通の設定
9     for i in range(len(ax)):
10        for b in ax[i]:
11            b.set_xlim(-5, 5)
12            b.set_ylim(-4, 4)
13            b.tick_params(axis = 'both', length = 15)
14            if i == 1:#2行目だけ x 軸ラベルを表示
15                b.set_xlabel('$x$')
16
17        ax[0,0].plot(x, y1, label = '$\sin(x)$', color = 'blue')
18        ax[1,0].plot(x, y2, label = '$\cos(x)$', color = 'green')
19        ax[0,1].plot(x, y3, label = '$\sin(x) \cos(x)$', color =
20        ↪ 'darkcyan')
21        ax[1,1].plot(x, y4, label = '$x \cos(x)$', color = 'red')
22
23        #全ての図の凡例
24        for a in ax:
25            for b in a:
26                b.legend(loc = 'best')
27
28        plt.savefig('multiPlot2.pdf')
29        plt.show()
```

4 その他の作図

その他の作図

- 棒グラフ
- 円グラフ
- ヒストグラム

4.1 棒グラフ

matplotlibでは、様々な型のグラフを描くことができます。ここでは、三種類を示します。

最初は、棒グラフです。plotBars.ipynbを見てください(ソースコード 4.1)。例として、九州各県の人口を棒グラフで表しましょう。

関数の最初のリスト population は、九州各県の 2015 年の人口です。棒グラフの高さに相当します。二番目のリスト prefecture は、九州各県の名前です。棒グラフの横軸に相当します。

15 行目から 17 行目に書いて、軸のラベルや目盛りを設定しています。

19 行目の bar() の最初の引数が横軸、二番目の引数が縦軸です。ただし、横軸は、リスト population の番号を示しているに過ぎません。リスト x は 10 行目で生成しています。横軸の名前は tick_label として渡しています。出力結果が図 4 です。

ソースコード 4.1 2015 年の九州各県人口を表す棒グラフ

```
1 def plotPopulation(prefectures:list[str],populations:list[int],
2   ↪ year=2015):
3     """
4     各県の人口を棒グラフで表す
5
6     Parameters
7     ---
8     prefectures: 県名のリスト
9     populations: 人口のリスト
10    """
11
12    x = [i for i in range(len(populations))]
13
14    fig, ax = plt.subplots(figsize = (10, 10), facecolor = 'white')
15
16    ax.set_title(f'九州の人口, {year}', fontsize = 28)
17    ax.set_xlabel('県', loc = 'right')
18    ax.set_ylabel('人口 (千人)', loc = 'top')
19
20    ax.tick_params(axis = 'both', length = 15)
21
22    ax.bar(x, populations, tick_label = prefectures)
23    plt.savefig('plotBar.pdf')
```

ソースコード 4.2 は、関数 plotPopulation() を呼び出すメイン部分です。

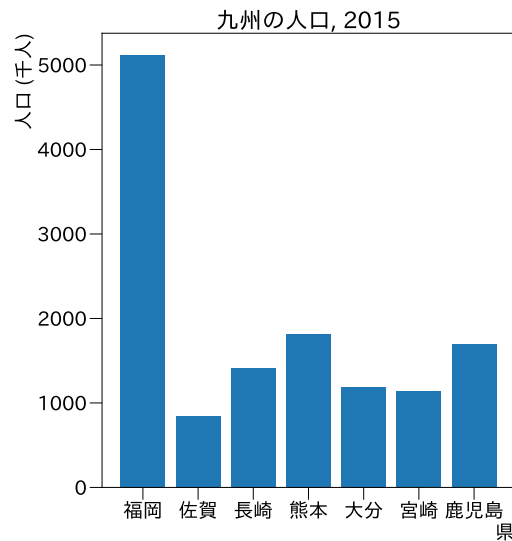


図4 棒グラフ

ソースコード 4.2 関数 `plotPopulation()` を呼び出すメイン部分

```

1 populations = [5120, 847, 1413, 1818, 1191, 1136, 1691]
2 prefectures = ['福岡', '佐賀', '長崎', '熊本',
3               '大分', '宮崎', '鹿児島']
4 plotPopulation(prefectures, populations)

```

4.2 円グラフ

次の例は、円グラフです。英語では、パイグラフと言います。お菓子のパイのことで、`piPlot.ipynb`を開けてください(ソースコード 4.3)。佐賀県の人口構成の変化を表すグラフです。人口を年少(14歳以下)、生産年齢(15歳以上、65歳未満)、そして老年(65歳以上)に分けて、1995年と2015年で比較します。

リスト `data` に、1995年と2015年の、三つの分類に相当する人口のリストを記述しています。リスト `years` は、タイトルに使う2つの年です。9行目で、一行二列の図を作っています。

13行目からの `for` ループで、二つの図を描いています。14行目が円グラフを描く命令です。最初の引数が描くデータ、二番目が各データに対応するラベルです。一番最後が、数値を描く書式の指定で、小数以下一桁まで百分率で表すことを指定しています。

ソースコード 4.3 円グラフ

```
1 data = [  
2     [160307, 566671, 157329], #1995  
3     [116122, 483019, 229335] # 2015  
4 ]  
5 label = ['年少', '生産年齢', '老年']  
6 color = ['y', 'c', 'r']  
7 year = [1995, 2015]  
8  
9 fig, ax = plt.subplots(1, 2, figsize = (20, 10), facecolor='white')  
10 plt.rcParams['font.size'] = 24 #文字の大きさ  
11 fig.suptitle('佐賀県の人口') #全体のタイトル  
12  
13 for k in range(2):  
14     ax[k].pie(data[k], labels = label, colors = color,  
15             startangle = 90, counterclock = False, autopct = '%1.1f%%')  
16     ax[k].set_title(str(year[k]))  
17  
18 plt.savefig('piPlot.pdf')  
19 plt.show()
```

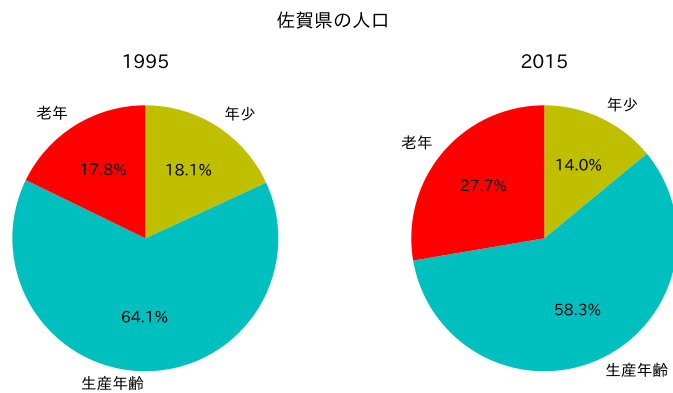


図5 佐賀県の人口構成を表す円グラフ

結果を図5に示します。matplotlibの円グラフのメソッドpie()は、データを与えると、自動的に百分率を計算してくれていることにも注意してください。

4.3 ヒストグラム

最後の例はヒストグラムです。histogram.ipynb を開いてください (ソースコード 4.4)。ヒストグラム (histogram) とは、頻度を表す図のことです。

ソースコード 4.4 ヒストグラム

```
1 #元データ
2 data = [2.5, 4., 5.5, 9.2, 0.7, 8.8, 6.1, 7.1, 4.3, 1.6,
3         3.4, 1.4, 7.7, 4.4, 5.5, 9.6, 6.8, 8.6, 2.1,
4         8.9, 2.9, 1.1, 2.1, 3.7, 2.0, 1.4, 0.5, 5.7]
5 fig, ax = plt.subplots(figsize = (10, 10), facecolor = 'w')
6 plt.rcParams['font.size'] = 24
7 ax.set_title('ヒストグラムの例')
8
9 numBins = 10 #binの数
10 rWidth = 0.9 #描く幅は、等間隔の幅に対して0.9倍
11 ax.hist(data, range=(0, 10), bins = numBins,
12         rwidth = rWidth, color = 'cyan')
13
14 plt.savefig('histogram.pdf')
15 plt.show()
```

1 行目の data には、0 以上 10 未満の小数のデータが入っています。このデータを 0.1 刻みの区間 (bin という) に分けて、各 bin に何個入っているかを図示しましょう。通常では、各 bin に入っている数を数えるプログラムを書く必要がありますが、matplotlib を使えば、自動的に数えてくれます。

9 行目の hist() メソッドを見てください。最初から、データ、データの区間、bin の数を指定します。rwidth は、作図する際に bin の間隔に対する棒グラフの相対的な大きさです。

結果は、図 6 のようになります。

5 図形を描く

図形を描く

- 既定の図を描く
- 点列から図を描く

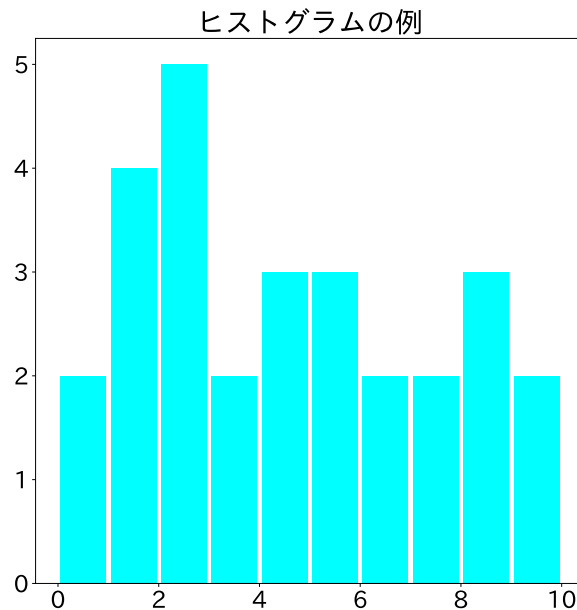


図6 ヒストグラム

作図の最後に、図を描くことを考えましょう。`drawShapes.ipynb`を開けてください。

図を描くには、二つの方法があります。一つ目は、既存の図形を描く方法です。ソースコード 5.1 の 6 行目から 11 行目は、円、四角形、正多角形という既存の図形を描いています。既存の図形の描き方は以下の URL にあります。

https://matplotlib.org/api/patches_api.html

もう一つの方法は、座標の列を使って定義する方法です。ソースコード 5.1 の 13 行目で、座標の列を与えています。注意が必要なのは、最後に出発点の座標を追加しておく必要があることです (11 行目)。

図を定義する過程を説明します。少しイメージしにくいかも知れませんが、昔、プリンターで図が描けなかった頃、図を描く専用のプロッター (plotter) という機材がありました。

<https://ja.wikipedia.org/wiki/%E3%83%97%E3%83%AD%E3%83%83%E3%82%BF%E3%83%BC>

プロッターの操作は、

`moveto` ペンを、線を描かずに指定した座標に移動する

ソースコード 5.1 図形を作図

```
1 def drawPatches(ax:plt.Axes):
2     """
3     図形を生成
4     """
5     patches=[]
6     #円
7     patches.append(pts.Circle((2,2),1,color='red'))
8     #四角
9     patches.append(pts.Rectangle((4,7),4.,2.,color='blue'))
10    #正多角形
11    patches.append(pts.RegularPolygon((7,3),5,2,color='yellow'))
12    #任意の点の列
13    verts = [(2.,8.), (3,4), (4,9), (6,2), (8,3)]
14    verts.append(verts[0])
15    path = createPath(verts)
16    patches.append(path)
17    #図形を追加
18    for p in patches:
19        ax.add_patch(p)
```

ソースコード 5.2 自分で点列を定義する

```
1 def createPath(vertList:list[tuple[float,float]])->pts.Patch:
2     """
3     頂点の列から多角形を生成
4     """
5     n = len(vertList)
6     #codes は、各座標への操作
7     codes = [Path.MOVETO] #最初は移動
8     for i in range(n-2): #残りは線を引く
9         codes.append(Path.LINETO)
10    codes.append(Path.CLOSEPOLY) #最後に閉じる
11    path = Path(vertList,codes) #座標と操作を与える
12    return pts.PathPatch(path,facecolor='orange',lw=2)
```

lineto ペンを線を引きながら、指定した座標に移動する

close 指定した座標に向かって図形を閉じる

という動作から構成します。これは、今でも、作図用のライブラリで共通に使われています。最後の動作は、最初の点に向けて図形を閉じるというものが多いのですが、matplotlibでは、最後の点も指定する必要があります。

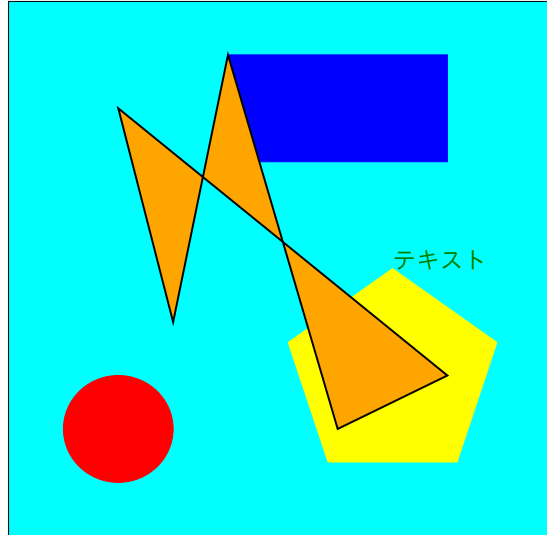


図7 作図

ソースコード 5.1 を見てください。点を結んで図形を描くために、13 行目で点のリストを定義します。14 行目では、始点を最後に追加しています。

このリストを使って図形を、ソースコード 5.2 で定義します。最初の点へ MOVETO した後、順次 LINETO し、最後に CLOSEPOLY します。

課題 5.1 quiz.ipynb にある課題です。関数 `plotLines(xlist, dataList)` に人口推移を作図するために必要なプログラムを追加し、作図しなさい。

6 次回

今回は、データをリストなどで作成し、作図を行いました。Excel では、内部で作図ができますね。Pandas のデータでも、matplotlib の機能と連携して、作図をすることができます。次回のテーマです。