第 7 章

基盤モジュール: Matplotlib

「百聞は一見に如かず」という諺がある通り，物事を伝える際には文章や言葉の説明より，写真やイラストを見せる方が，理解してもらいやすいものです。特に，現代のように，コンピュータが大量のデータを瞬時に処理でき時代においては，数字の羅列を目で追うことは不可能で，リアルタイムにグラフ化し，データの変化を目で見ることで理解することが重要です。Python では多種多様なグラフを生成することのできる Matplotlib パッケージが用意されていますので，データサイエンスでは欠かせない可視化ツールを Python で簡単に作ることが出来ます。この章では，膨大な Matplotlib の機能のほんの一部を使ってみることにします。

7.1 最初の関数グラフ

まず，$x \in [a, b]$ における関数 $y = \sin(x)$ のグラフを描いてみましょう。描き方としては

1. $[a, b]$ を n 分割した時の分割幅 $x_h := (b - a)/n$ を計算する
2. $x_i := a + ih (i = 0, 1, ..., n)$ を計算し，リスト $x = [x_0, x_1, ..., x_n]$ を作る
3. $y := \sin(x) = [\sin(x_0), \sin(x_1), ..., \sin(x_n)]$ を作る
4. ax.plot(x, y) を実行し，横軸が x，縦軸が y の値となる 2 次元折れ線グラフを描く

となります。

ソースコード 7.1 plot_sine.py

```
1  # plot_sine.py: サインカーブを描く
2  import matplotlib.pyplot as plt # Matplotlib
3  import numpy as np # NumPy
4
5  # x の範囲[a, b]を指定
6  a = 0
7  b = 4 * np.pi
8
9  # x 区間の分割数を指定
10 n = 10
11
12 # x の配列を生成
13 x_h = (b - a) / n # 小区間幅
14 x = [ a + i * x_h for i in range(n + 1)]
15
16 # x ごとに y の値を計算
```

```
y = np.sin(x)

print(x)
print(y)

# 2次元グラフを描画
# (1) Figure オブジェクト(fig)と
# その上にAxes オブジェクト(ax)を生成
fig, ax = plt.subplots()

# (2) 折れ線グラフをax 上に描画
ax.plot(x, y)

# (3) グラフタイトル,x 軸タイトル，y 軸タイトルを指定
ax.set_title('y␣=␣sin(x)')
ax.set_xlabel('x')
ax.set_ylabel('y')

# (4) グリッドを描画
ax.grid()

# (5) グラフを表示
plt.show()
```

これを実行すると，図 7.1 のように，Windows 上にグラフが描画されます。プログラムを終了したい時にはこのグラフウィンドウを閉じて下さい。

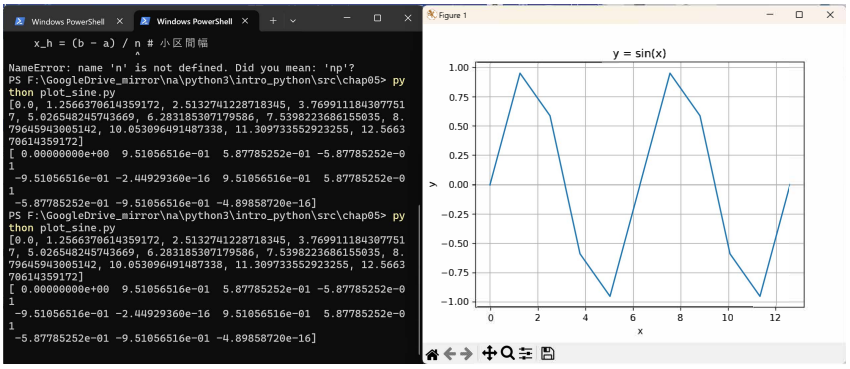

図 7.1　plot_sine.py 実行画面

問題 7.1

plot_sine.py に基づいて次の変更を行った plot_cosine.py を作成せよ。

1. なめらかな曲線に見えるよう，分割数 n を大きくする
2. $y = \cos(x)$ のグラフを描画する
3. グラフタイトルを関数式に変更する

7.2 複数グラフを並べる

Matplotlib は相当複雑なグラフを自在に描くことができます。例えば，複数のグラフをまとめて表示することも朝飯前です。

下記の例は，左側に三角関数，右側に指数関数と対数関数のグラフを描いたものになります。

ソースコード 7.2 plots_functions.py

```
1  # plot_functions.py: 複数のグラフを並べる
2  import matplotlib.pyplot as plt # Matplotlib
3  import numpy as np # NumPy
4
5  # x 区間の分割数を指定
6  n = 100
7
8  # Figure オブジェクト(fig)と
9  # その上にAxes オブジェクト(ax, ax2)を
10 # 1行 2列に生成し,サイズを 10x5 とする
11 fig, (ax, ax2) = plt.subplots(1, 2, figsize=(10, 5))
12
13 # 三角関数のグラフを描く
14 # x の範囲[a, b]を指定
15 a = 0
16 b = 4 * np.pi
17 x_h = (b - a) / n # 小区間幅
18 x = [ a + i * x_h for i in range(n + 1)]
19
20 # x ごとに y の値を計算
21 y_sin = np.sin(x)
22 y_cos = np.cos(x)
23
24 # 三角関数グラフを描画
25 ax.plot(x, y_sin, label='sin(x)')
26 ax.plot(x, y_cos, label='cos(x)')
27 ax.set_title('y␣=␣sin(x),␣cos(x)')
28 ax.set_xlabel('x')
29 ax.set_ylabel('y')
30 ax.set_ybound(lower=-3, upper=3)
31 ax.legend()
32 ax.grid()
33
34 # 指数関数を対数関数をfig 上に描画
35 # x の配列を生成し，exp(x)とlog(x)を計算
36 a = -3
37 b = 3
38 x_h = (b - a) / n # 小区間幅
39 x = [ a + i * x_h for i in range(n + 1)]
40 y_exp = np.exp(x)
41 y_log = np.log(x) # x < の時はNaN を含む
42
43 # 折れ線グラフをax2 上に描画
44 ax2.plot(x, y_exp, label='exp(x)')
45 ax2.plot(x, y_log, label='log(x)')
```

```
ax2.set_title('y␣=␣exp(x),␣log(x)')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_ybound(lower=-3, upper=4)
ax2.legend()
ax2.grid()

# グラフを表示
plt.show()
```

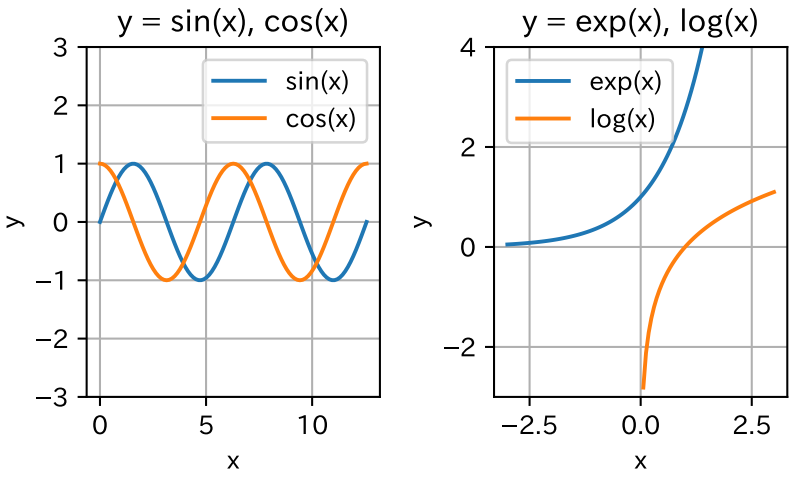

図 7.2　複数のグラフを描画する

問題 7.2

plot_functions.py の右のグラフに $y = \tan(x)$ のグラフを描画し，結果について考察せよ。

7.3　2 つの y 軸を持つグラフを描画する

一枚のグラフに，2 つの異なるスケールを持つ値をプロットしたい時があります。例えば，$f(x)$ の関数値と共に，数値微分の相対誤差を表示したい時，同じ x の値に対して，左縦軸は $f(x)$ のスケールで，右縦軸は log スケールにしておき，適切なスケールで値を描画すると，関数の値と誤差の変動がどのように連動しているかが一目で分かります。

このグラフを描画するスクリプトを下記に示します。

ソースコード 7.3　plot_num_diff.py

```
# plot_num_diff.py: 数値微分と相対誤差の描画
import numpy as np # NumPy
import matplotlib.pyplot as plt # Matplotlib

# 元の関数
def func(x):
    return np.exp(np.cos(x)) - x ** 3
```

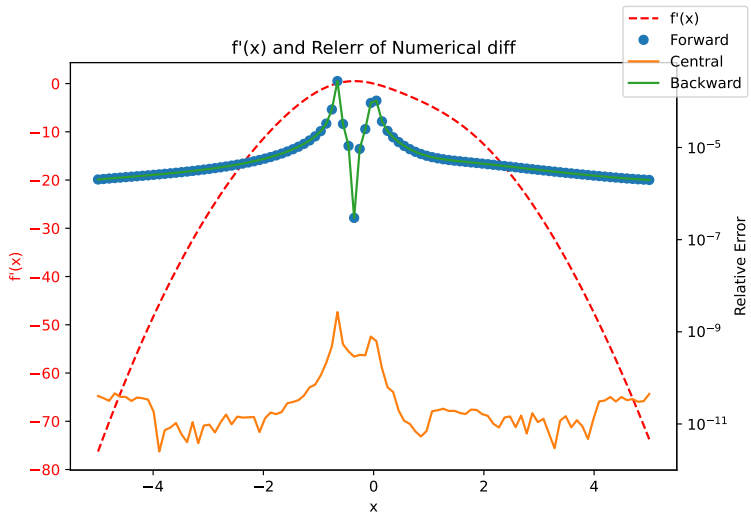

図 7.3　関数グラフ (左軸) と数値微分の相対誤差 (右軸)

```
8
9  # 真の導関数
10 def true_dfunc(x):
11     return -np.exp(np.cos(x)) * np.sin(x) - 3 * x ** 2
12
13 # 前進差分商: (f(x + h) - f(x)) / h
14 def forward_diff(x, h, f):
15     return (f(x + h) - f(x)) / h
16
17 # 中心差分商: (f(x + h) - f(x - h)) / 2h
18 def central_diff(x, h, f):
19     return (f(x + h) - f(x - h)) / (2.0 * h)
20
21 # 後退差分商: (f(x) - f(x - h)) / h
22 def backward_diff(x, h, f):
23     return (f(x) - f(x - h)) / h
24
25 # x = [-5, 5]
26 x = np.linspace(-5, 5, 100)
27 h = 10.0**(-5)
28
29 # 前進差分商,中心差分商,後退差分商
30 fdiff = forward_diff(x, h, func)
31 cdiff = central_diff(x, h, func)
32 bdiff = backward_diff(x, h, func)
33
34 # 相対誤差チェック
35 reldiff_f = np.abs((fdiff - true_dfunc(x)) / true_dfunc(x))
36 print('Forward␣diff␣relerr␣=␣', reldiff_f)
37 reldiff_c = np.abs((cdiff - true_dfunc(x)) / true_dfunc(x))
38 print('Central␣diff␣relerr␣=␣', reldiff_c)
39 reldiff_b = np.abs((bdiff - true_dfunc(x)) / true_dfunc(x))
```

```
print('Backword␣diff␣relerr=␣', reldiff_b)

# 導関数のグラフ描画
true_dfunc_y = true_dfunc(x)
print('x␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=', x)
print('True␣dfunc␣␣␣␣␣␣␣␣␣=', true_dfunc_y)

fig, ax1 = plt.subplots()

# 関数グラフのy 軸（左）を設定
ax1_color = 'red' # 赤
ax1.set_title('f\'(x) and Relerr of Numerical diff')
ax1.set_xlabel('x')
ax1.set_ylabel('f\'(x)', color=ax1_color)
ax1.plot(x, true_dfunc_y, '--', label='f\'(x)',␣color=ax1_color)␣#␣赤の点線
ax1.tick_params(axis='y',␣labelcolor=ax1_color)␣#␣y 軸に目盛を入れる

#␣2番目のy 軸（右）を設定
ax2␣=␣ax1.twinx()␣#␣x 軸は共通
ax2.set_ylabel('Relative Error')
ax2.set_yscale('log')␣#␣log スケールに変更
ax2.plot(x,␣reldiff_f,␣'o',␣label='Forward')␣#␣○プロット
ax2.plot(x,␣reldiff_c,␣label='Central')
ax2.plot(x,␣reldiff_b,␣label='Backward')
ax2.tick_params(axis='y')

#␣凡例
fig.legend()

#␣グラフ描画
plt.show()
```

問題 7.3

問題 5.3 に導関数と数値微分の相対誤差のグラフを描画する機能を付加した `plot_diff_poly.py` を作れ。

7.4 常微分方程式の解の描画

常微分方程式のソルバーは SciPy.integrate パッケージにあります。初期値問題の場合は `solve_ivp` 関数をソルバーとして使うのが標準で，独立変数 x は `t` としてソルバーでは使用されます。

以下のスクリプトでは，ソルバーに積分区間内の評価点を決めさせる方法（デフォルト）と，ユーザが必要となる評価点を指定する方法で，それぞれ数値解を求め，そのグラフを描いています。後者のようにしておくと，積分区間におけるグラフを滑らかに描くことができるようになります。

```
# plot_ode_ivp.py: 常微分方程式の初期値問題と解プロット
import numpy as np
import scipy.integrate as scint # ODE ソルバー
import matplotlib.pyplot as plt # グラフ描画

# 陽的形式の右辺
# y' = func(t, y) = y
def func(t, y):
```

```
  return y

# 初期値
# y(0) = 1
y0 = [1.0]

# t = [0, 1]
t_interval = [0.0, 1.0]
print(t_interval)

# 常微分方程式を解く
ret = scint.solve_ivp(func, t_interval, y0) # 評価点tが可変になる
ret_fix = scint.solve_ivp(
  func, t_interval, y0,
  t_eval=np.linspace(t_interval[0], t_interval[1], 10)
) # 評価点tが固定化される

# 結果を表示
print(ret)

# yを表示
print(ret.y)

# t-yグラフを描画
# グラフ初期化
figure, axis = plt.subplots()

# 値をセット
axis.plot(ret.t, ret.y[0, :], label='Automatic')
axis.plot(ret_fix.t, ret_fix.y[0, :], label='t_eval')

# x軸，y軸，グラフタイトルをセット
axis.set(xlabel='t', ylabel='y', title='y\' = y')

#␣グリッドを描画
axis.grid()

#␣凡例
axis.legend()

#␣グラフ保存ファイル名
figure.savefig('ode.png')

#␣グラフを画面に描画
plt.show()
```

問題 7.4

1. 上記のスクリプトに問題 6.4 のグラフを描く機能を追加して実行せよ。
2. 上記のスクリプトに真の解を計算する関数を追加し，数値解の相対誤差を求めてグラフとして描画せよ。

7.5　ワードクラウドを作る

ワードクラウド (word cloud) は，テキストファイルに含まれる単語全ての出現回数をカウントし，大きい順に大きく表示し，雲のようにまとめて表示したものです。下記に青空文庫 (https://www.aozora.gr.jp/) に登録されている 2 作品のワードクラウドを表示したものを示します。

図 7.4　ワードクラウドの例:「本はどのように消えていくのか」(左) と「吾輩は猫である」(右)

ソースコード 7.4　wordcloud.py

```
# wordcloud.py: WordCloud の生成
# https://github.com/WorksApplications/sudachi.rs/tree/develop/python
# WordCloud: https://amueller.github.io/word_cloud/
# Requests: https://requests.readthedocs.io/en/latest/
import sudachipy as sdc
import requests # Requests
import re # Regular expression
import string # string.split
import numpy as np # wordcloud では必須
import wordcloud
import matplotlib.pyplot as plt # グラフ描画

# 青空文庫にアクセス
#富田倫生「青空のリスタート」
#url = 'https://www.aozora.gr.jp/cards/000055/files/698_51256.html'
# 夏目漱石「吾輩は猫である」
url = 'https://www.aozora.gr.jp/cards/000148/files/789_14547.html'
#url = 'https://www.aozora.gr.jp/'
r = requests.get(url)
if r.status_code != 200: # 正常アクセス時のみ
    print(url +'cannot be read...')
    exit # 終了

#r.encoding = 'utf-8' #強制的にUTF-8 encoding
r.encoding = 'shift_jis' # Shift-jis encoding
print('encoding = ', r.encoding)
#print('r.text = ', r.text)

# 正規表現ですべてのタグ<*>を消去
# https://stackoverflow.com/questions/9662346/python-code-to-remove-html-tags-from-a-
    string
```

```
plaintext = re.sub(re.compile(r'<[^>]+>'), '', r.text)
print('plaintext␣=␣', plaintext)

# 一行ごとに分割
plaintext_lines = plaintext.split('\n')

# SudachiPy による分かち書きと品詞同定
tokenizer_obj = sdc.Dictionary().create()
#morphemes = tokenizer_obj.tokenize("吾輩は猫である。名前はまだない。")
#morphemes = tokenizer_obj.tokenize(plaintext)

# "普通名詞"のみ取り出し
noun = []

for line in plaintext_lines:
    morphemes = tokenizer_obj.tokenize(line)

    for m in morphemes:
        #print(
        # 'm = ',
        # m.surface(), # 単語
        # m.reading_form(), # 単語読み(カタカナ)
        # m.part_of_speech() # 品詞情報
        #)

        info = m.part_of_speech()
        #print('info = ', info[0], info[1], info[2])
        if info[0] == "名詞" and info[2] == "一般":
            noun.append(m.surface()) # 単語追加

# 名詞&一般のみ表示
#print('noun = ', noun)

# 辞書型
noun_count = {}
for word in noun:
    # 初見の単語数はゼロリセット
    if word not in noun_count:
        noun_count[word] = 0

    noun_count[word] += 1

# 単語数の多い順に逆順ソート
sorted_noun_count = sorted(noun_count.items(), reverse=True, key=lambda x:x[1])
print(sorted_noun_count)

# 名詞を半角スペースで接続
max_word_num = 100 # 100単語まで
word_list = ""
word_count = 0
for item in sorted_noun_count:
    word_list += item[0] + "␣"
    word_count += 1
    if word_count >= max_word_num:
```

```
85          break
86
87  print(word_list)
88
89  # Word cloud生成
90  # https://moji.or.jp/ipafont/ipaex00401/
91  ttf_path = './ipaexg.ttf'
92  #ttf_path = 'C:/windows/fonts/gothic.ttf'
93
94  word_list_cloud = wordcloud.WordCloud(font_path=ttf_path,background_color="white").
        generate(word_list)
95  plt.imshow(word_list_cloud, interpolation='bilinear')
96  plt.axis('off') # 軸表示なし
97
98  # WordCloud 表示
99  plt.show()
```

問題 7.5

青空文庫から好きな作品を一つ選び，上記のスクリプトを使ってワードクラウドを作れ。