計算物理学II

第9回
数値計算プログラミング入門（5）：
NumPyとSciPy

秋山 進一郎

2025年12月12日

授業日の確認

・全10回

- ・第1回：10月3日（金）
- ・第2回：10月10日（金）
- ・第3回：10月17日（金）
- ・第4回：10月24日（金）★
- ・第5回：10月31日（金）
- ・第6回：11月14日（金）★
- ・第7回：11月21日（金）
- ・第8回：12月5日（金） ★
- **・第9回：12月12日（金）**
- ・第10回：12月19日（金）★

・★の付いた授業にてレポート課題を配布予定

本日の演習内容

- NumPyを使ってみる
  - NumPy配列の型と形について理解する
  - ベクトルや行列，多次元配列(テンソル)が取り扱えることを理解する
- SciPyを使ってみる
  - SciPyを使うことで多種多様な科学計算が実行できることを認識する
- 計算時間の測定からNumPy，SciPyのadvantageを見る

NumPyとは？

- 数値計算（特に線形代数演算）を効率的に実行できるライブラリ
- NumPyの中身はC言語とFortranで書かれている
  - Pythonはスクリプト言語の一種であり，プログラムに書かれた内容を逐一機械語に翻訳しながら実行している
  - C言語やFortranはコンパイラ言語であり，プログラムに書かれた内容を事前に全て機械語に翻訳（コンパイル）し，その後一気に実行する
  - 一般に，コンパイラ言語の方が処理が高速
  - 加えて，Intel Math Kernel LibraryやOpen BLASと呼ばれる洗練された行列演算ライブラリが使用されているため、極めて処理が高速

配列とは?

- 配列とは, 同一種類の複数のデータ(数値)をまとめて管理するためのデータ構造のこと
- 配列に格納された数値には, その数値に対応する整数値(添え字)を通してアクセスする
  - 例えば, 一個の添え字iでラベルされた配列は一次元配列と呼ばれる
    - ベクトルv_iは一次元配列である
  - 二個の添え字i, jでラベルされる配列は二次元配列と呼ばれる
    - 行列M_{ij}は二次元配列である
  - N個の添え字でラベルされる配列はN次元配列と呼ばれる

NumPyをimportする

- 以下, ex_array.pyというファイルを作ってコードを書いていこう
- NumPyをimportする時は, npという略称をつけるのが慣習となっている
  - ex_array.pyの1行目に以下のコードを書こう

```
1    import numpy as np
```

NumPy用の配列を作る

- np.array()を使って, NumPy専用の配列を用意しよう
  - リストをnp.array()に渡せばよい
  - 以下のようなコードをex_array.pyに書き, 実行してみよう

```
1    import numpy as np
2
3    a = np.array([0,1,2])
4    print(a)
```

行列を作ってみる

・np.array()を使って, 行列を作ることができる

・リストを要素に持つリストをnp.array()に渡せばよい

・以下のようなコードをex_array.pyに書き, 実行してみよう

```
6    mat = np.array([[1,0],[0,1]])
7    print(mat)
```

・2行2列の行列が出力される

```
[[1 0]
 [0 1]]
```

要素が全てゼロの行列を作ってみる

- np. zeros()を使って, 要素が全てゼロの行列を作ることができる
  - 引数として行数と列数をタプルで渡す
  - 以下のようなコードをex_array.pyに書き, 実行してみよう

```
 9    mat = np.zeros((2,2))
10    print(mat)
```

  - 2行2列のゼロ行列が作られる

NumPy配列データの型

- NumPy配列に格納される数値は全て同じ型でないといけない
  - Pythonのリストのように, 複数種類の型を混在させることはできない
  - NumPy配列の型はdtypeで調べられる
  - 以下のようなコードをex_array.pyに書き, 実行してみよう

```
 9    mat = np.zeros((2,2))
10    print(mat)
11    print(mat.dtype)
```

- np.arrayで作った配列の型は, 渡されたリストの型から推定される
- np.zerosで作った配列の型は, デフォルトでfloat64
  - float64は倍精度浮動小数点数のこと

倍精度浮動小数点数とは？

- 64ビット（8バイト）のメモリ領域を使用して表現される浮動小数点数
  - 桁数はおよそ15桁
- 単精度浮動小数点数というものもある
  - 32ビット（4バイト）のメモリ領域を使用
  - 桁数はおよそ6桁
- 数値計算では倍精度の浮動小数点数を用いることが多い

NumPy配列の形

- NumPy配列には形があり, shapeを使って知ることができる
  - 以下のようなコードをex_array.pyに書き, 実行してみよう

```
 9    mat = np.zeros((2,2))
10    print(mat)
11    print(mat.dtype)
12
13    print(mat.shape)
```

  - shapeはタプルを返す

NumPy配列を作る他の方法

- NumPy配列の最も簡単な作り方はnp.array()にリストを渡す方法
- よく使う他の方法として, np.linspace()がある
  - 以下のようなコードをex_array.pyに書き, 実行してみよう

```
15    b = np.linspace(0,10,5)
16    print(b)
```

  - np.linspace(start, stop, num)とすることで, startからstopまでの値を等間隔でnum個に分割した数値を要素とするNumPy配列を作ることができる
    - 等差数列の交差を勝手に考えてくれている

NumPy配列同士の演算（1）：要素ごとの演算

- 形が同じNumPy配列同士は四則演算できる
  - ex_arithmetic.pyという名前のファイルを作り，右のコードを書いて実行してみよう
  - +, -, *, /は配列の要素ごとの足し算, 引き算, 掛け算, 割り算を意味することに注意
  - 特に，*は行列積ではないことに注意

```python
import numpy as np

a = np.array([[1,0],[0,1]])
b = np.array([[1,2],[3,4]])

c = a+b
print(c)

c = a+b
print(c)

c = a*b
print(c)

c = a/b
print(c)
```

NumPy配列同士の演算（2）：行列行列積

- 行列同士の積を計算する方法はいくつかある
  - ex_matprod.pyという名前のファイルを作り，右のコードを書いて実行してみよう
  - a, bが2次元配列(行列)の場合，これら4つは全て行列行列積を返す
- 多次元配列(テンソル)の場合，処理が異なるので注意が必要
  - このため，行列行列積には@演算子かmatmul()を使うのが好ましいとされている
  - 授業では扱わないが，味のある人は調べてみてください

```
1   import numpy as np
2
3   a = np.array([[1,0],[0,1]])
4   b = np.array([[1,2],[3,4]])
5
6   c = a@b
7   print(c)
8
9   c = np.matmul(a,b)
10  print(c)
11
12  c = np.dot(a,b)
13  print(c)
14
15  c = a.dot(b)
16  print(c)
```

NumPy配列の実体

- NumPy配列ではどんな形（ベクトル，行列，テンソル）を作ることができる
  - 実は計算機の中では，どんな形のNumPy配列であっても全て一次元配列として保存されている
  - reshapeを使うことで，データを変更せずに多次元配列へ変形することができる

一次元配列を多次元配列に変換する 1/2

- ex_reshape.pyというファイルを作り, 次のようなコードを書いて実行してみよう
  - 3行目のarangeは連番の要素を持つ1次元NumPy配列を作る
  - 6行目で配列aを2行4列の行列(二次元配列)にreshape
  - 9行目のshapeを使って, 行列bの行数(rows)と列数(cols)をタプルとして取得
  - 11行目以下で行列bのb_{ij}成分をprintする
    - b[i,j]で二次元配列bのi行j列成分にアクセスできる

```
1   import numpy as np
2
3   a = np.arange(8)
4   print(a)
5
6   b = a.reshape((2,4))
7   print(b)
8
9   rows, cols = b.shape
10
11  for i in range(rows):
12      for j in range(cols):
13          print(i,j,b[i,j])
14
```

一次元配列を多次元配列に変換する 2/2

- ex_reshape.pyに, 右の15行目以下を書き足し, 実行してみよう
  - 今度は, reshapeを使って一次元配列aを三次元配列cに変形する
  - 先ほど同様, shapeを使って, 三次元配列c_{ijk}の添え字i, j, kそれぞれのサイズをタプルとして取得
  - 最後に三次元配列c_{ijk}の成分をprintする
- reshapeを使うと, 一次元配列を好きなn次元配列に変形できる

```
1   import numpy as np
2
3   a = np.arange(8)
4   print(a)
5
6   b = a.reshape((2,4))
7   print(b)
8
9   rows, cols = b.shape
10
11  for i in range(rows):
12      for j in range(cols):
13          print(i,j,b[i,j])
14
15  c = a.reshape((2,2,2))
16  print(c)
17
18  dim1, dim2, dim3 = c.shape
19
20  for i in range(dim1):
21      for j in range(dim2):
22          for k in range(dim3):
23              print(i,j,k,c[i,j,k])
24
```

練習問題

- $n = 0, 1, \cdots, N-1$とし,n番目の要素が$\mathrm{e}^{2\pi \mathrm{i} n/N}$で与えられるNumPy配列を作ろう
- まずはこれまでに習ったことを組み合わせてコードを書いてみましょう
  - 例えば, $N = 10^4$とする
  - 次のページにサンプルコードがある

素朴なコード

- 例えば下のようなコードがあり得る
  - math.exp()の引数は実数でないとダメなのでsinとcosに分けた
    - cmathモジュールをimportしてcmath.exp()を使えば複素数の引数も渡せるが…

```
1   import math
2   import numpy as np
3
4   N = 10000
5   a = []
6
7   for i in range(N):
8       x = math.cos(2*math.pi*i/N)+1j*math.sin(2*math.pi*i/N)
9       a.append(x)
10
11  a = np.array(a)
12  print(a)
```

NumPy関数にはNumPy配列を直接渡すことができる

- 実は先ほどのコードは, NumPyだけを使って下のように書くことができる

```
1    import numpy as np
2
3    N = 10000
4    a = np.exp(2j*np.pi*np.arange(N)/N)
```

- NumPy関数（この例ではnp.exp()）にはNumPy配列（この例ではnp.arange()）を直接渡すことができる
- 先ほどの素朴なコードではfor文を使っていたが, こちらのコードにはない
  - コードがコンパクトになり読みやすい（コードの可読性が高まる）
  - 実はコードの実行速度もこちらの方が早い（後で見るようにPythonのfor文は遅い）

NumPy関数を積極的に利用しよう

- 多くのNumPy関数にはNumPy配列を直接渡すことができる
- NumPyには多くの数学関数が用意されている
- 今後コードで数学関数を使う場合には, 積極的にNumPy関数を使いましょう

演習課題：計算時間を測ってみよう

- 以下はNumPyを使った方が良いことを実感してもらうための演習です
- 行列行列積を具体例として, 自分で一から書いたコードとNumPyを使って書いたコードを作成し, 同じ問題を解くのにかかった時間を比較してみよう
- ここでは, timeモジュールを使って計算時間を測定してみる

timeモジュールを使った計算時間の測定方法

① timeモジュールをimportする

② 計算開始時と計算終了時の時刻をtime.time()で取得

③ 時刻の差分から実行時間が得られる

・例としてex_time.pyという名前で, 右のようなコードを書いて実行してみよう

  ・func_sampleの実行時間を測定するコード

  ・計算時間を測定したい部分の前後でtime.time()を使って時刻を取得すれば良い

  ・func_sampleの中で実行される演算回数はnum_term**2回なので, 計算時間もnum_termについて2乗で増大することに注意

```python
import time

def func_example(num_terms):
    num = 0
    for i in range(num_terms):
        for j in range(num_terms):
            num = num + i - 0.5*j

    return num

t1 = time.time()

sol = func_example(1000)
print(sol)

t2 = time.time()

elapsed_time = t2 - t1
print("Elapsed time: ",elapsed_time)

```

行列行列積を計算するコードを準備する

・正方行列a, bを引数で渡し, その行列積を返す関数を用意しよう

・行列のサイズはdim_matrixとした

```python
import time
import numpy as np

def matrix_multiplication(a,b,dim_matrix):

    sol = np.zeros((dim_matrix,dim_matrix))

    for i in range(dim_matrix):
        for j in range(dim_matrix):
            for k in range(dim_matrix):
                sol[i,j] = sol[i,j] + a[i,k] * b[k,j]

    return sol
```

二つの正方行列を用意する

- ここでは, NumPyに用意されている乱数生成器を使って二つの正方行列を用意する
- 下のようなコードを作成しよう

```
15    n_size = 100
16
17    array_a = np.random.rand(n_size, n_size)
18    array_b = np.random.rand(n_size, n_size)
```

- n_sizeが正方行列のサイズ(先ほどのdim_matrixと同じ)
- np.random.rand(n,m)で0~1の間の一様乱数を成分に持つn行m列の行列をNumPy配列として用意できる

計算時間を測定してみよう

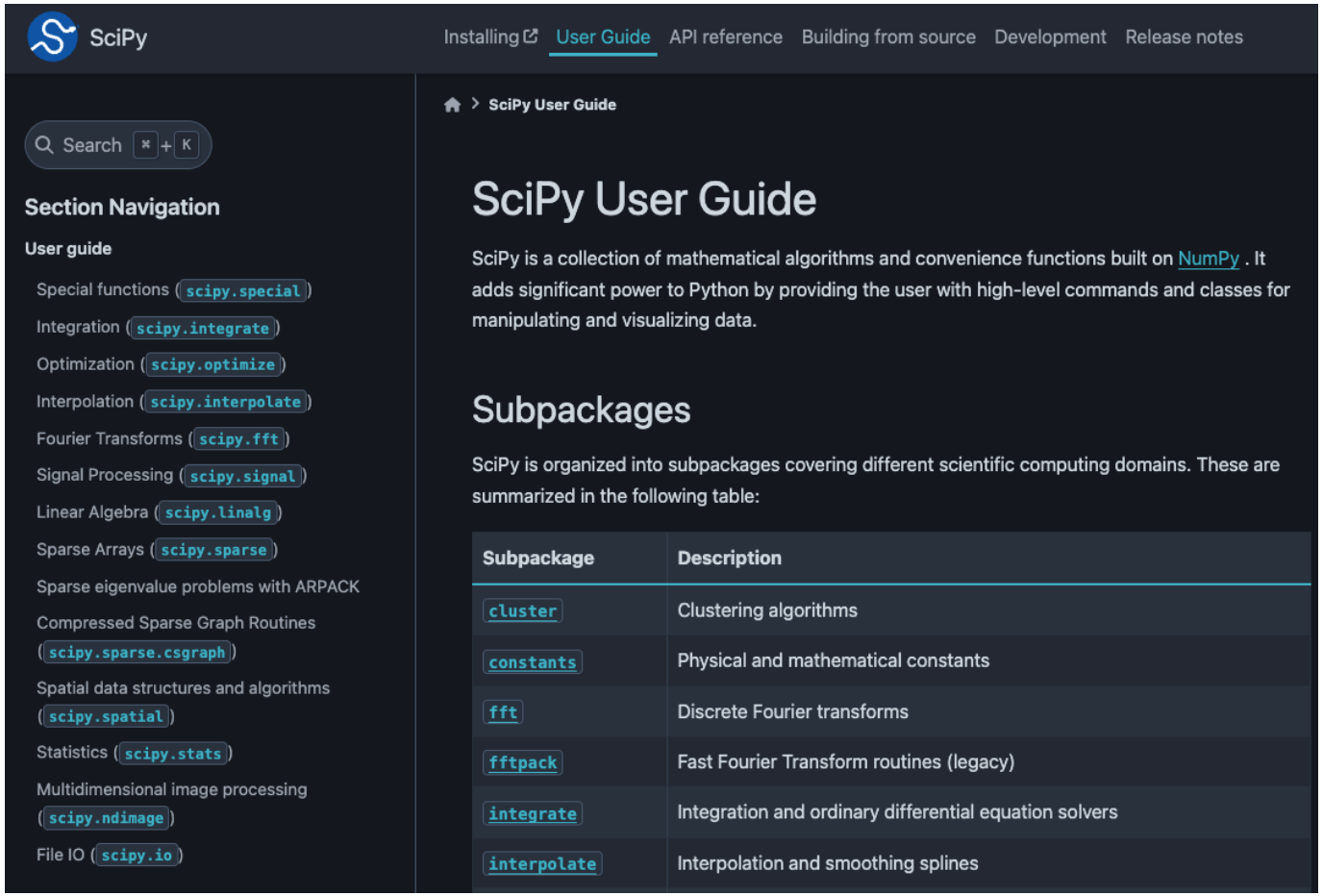
- 先ほど用意した関数matrix_multiplicationにarray_a, array_bを与え, その行列積の計算にかかった時間を出力するコードを書こう
- 続いて, np.matmulを使い, array_aとarray_bの行列積を計算するのにかかった時間を出力するコードを書こう
- R=(関数matrix_multiplicationの計算時間)/(np.matmulの計算時間)も出力させよう
  - 例えばR=2なら, np.matmulの方が2倍早いことを意味する
- n_sizeを変えてRを見てみよう
  - 関数matrix_multiplication中の演算回数はdim_matrix**3回なので，n_sizeの3乗で計算時間がスケールすることに注意(n_sizeが大きすぎると計算が終わらない)

教訓：for文を書くのはなるべく避け，ライブラリを使うべし

- for文やwhile文は遅い(特にPythonでは)
- 実際の数値計算では, for文やwhile文の使用はなるべく避け, NumPyやSciPyを使いましょう
- 同じアルゴリズムであっても, どのようなコードを書くか(実装方法)によって計算時間に大きな差が生まれるということを知っておきましょう
- 計算時間(計算量)がどのようにスケールするか予め見積もっておくのも重要
- [発展的な注]将来, 実践的な数値計算コードを書く必要に迫られた場合には,以下のような観点も重要
  - コードの中で一番計算時間を要する部分(ボトルネック)のはどこか?
  - 実際にコードを動かしてみて, 実行時間が見積もり通りにスケールしているか?

SciPyとは？

- NumPyベースの科学計算ライブラリ
  - 非常に多機能である
  - どんなことができるのか, 公式の Documentを一度眺めてみよう



SciPyの使用例（1）：固有値分解 1/2

- 線形代数演算を行うには, scipy.linalgというライブラリをimportする
- ex_evd.pyというファイルを作り, 次のようなコードを書いて実行してみよう
  - 例として, 行列 $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ の固有値分解を考える
  - 2行目でscipyからlinalgをimport
  - 7行目で行列aを固有値分解
  - wに固有値, vに対応する固有ベクトルが格納される

```
1    import numpy as np
2    from scipy import linalg
3
4    a = np.array([[1,2],[2,1]])
5    print(a)
6
7    w, v = linalg.eig(a)
8    print(w)
9    print(v)
```

SciPyの使用例（1）：固有値分解 2/2

- 先ほどの固有値分解の結果が正しいか確認してみよう
  - 計算機を使わずに, 自分で行列aを対角化してみる(各自やってみてください)
  - ここでは, 右のようにコードを書き足してみよう
    - 11行目で1次元配列wを対角行列に変換
    - 13行目で行列積を計算
      - .Tメソッドを使って行列vを転置できる
    - 13行目で得られる行列bは行列aと“一致”するはず
    - 16行目の出力を確認してみよう
      - 数値誤差の範囲内で行列aが再構成されるはずです

```
1   import numpy as np
2   from scipy import linalg
3
4   a = np.array([[1,2],[2,1]])
5   print(a)
6
7   w, v = linalg.eig(a)
8   print(w)
9   print(v)
10
11  D = np.diag(w)
12  b = v@D@v.T
13  print(b)
14
15  print(a-b)
16
```

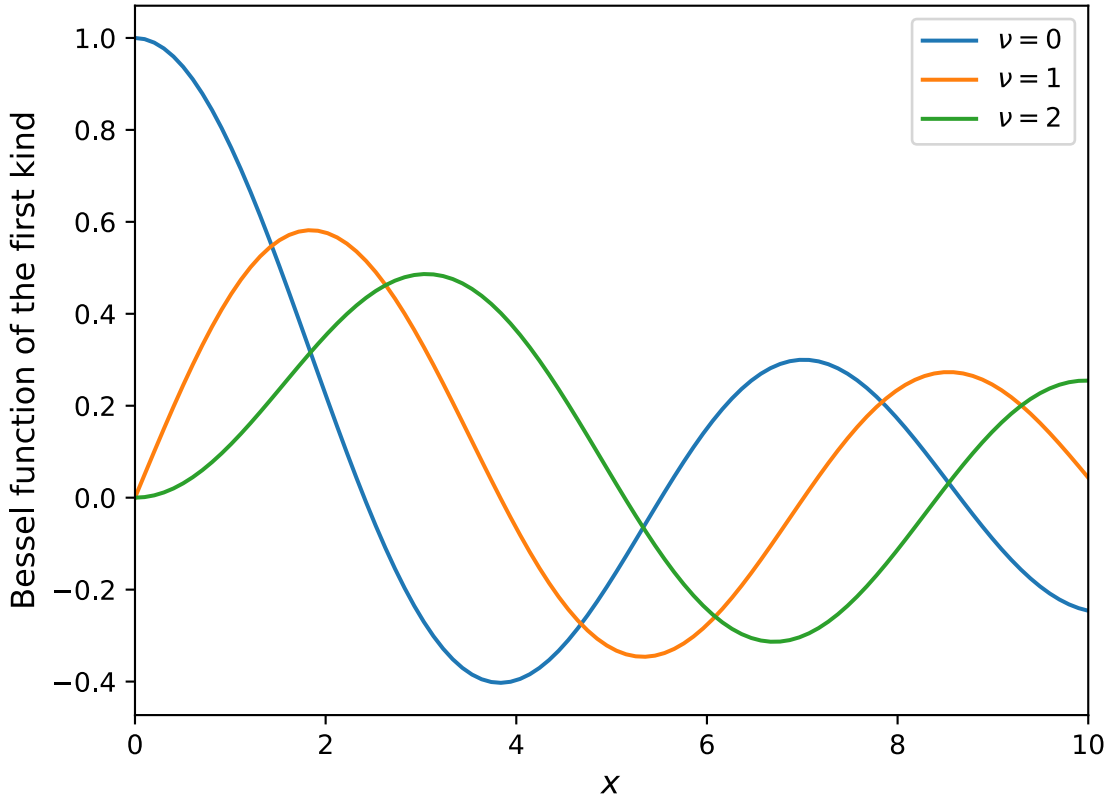
SciPyの使用例（2）：特殊関数 1/2

- 特殊関数を使うには，scipy.special というライブラリをimportする
- 右のコードを書いてみよう
  - 2行目でscipy.specialからjv (第一種Bessel関数)をimport
  - jv(v,x)でv番目の第一種Bessel関数のxにおける値が返ってくる
  - 今の場合，xがNumPy配列で，y0, y1, y2もNumPy配列
    - NumPy関数にはNumPy配列を直接渡すことができる
  - 14行目以下でmatplotlibによるグラフ作成

```
1   import numpy as np
2   from scipy.special import jv
3   import matplotlib.pyplot as plt
4
5   x = np.linspace(0, 10, 100)
6
7   y0 = jv(0,x)
8   y1 = jv(1,x)
9   y2 = jv(2,x)
10
11  plt.plot(x, y0, label=r"$\nu = 0$")
12  plt.plot(x, y1, label=r"$\nu = 1$")
13  plt.plot(x, y2, label=r"$\nu = 2$")
14  plt.legend()
15  plt.xlim(0,10)
16  plt.xlabel(r"$x$", fontsize=12)
17  plt.ylabel("Bessel function of the first kind", fontsize=12)
18  plt.savefig("bessel_jv.pdf")
```

SciPyの使用例（2）：特殊関数 2/2

- コードを実行し, bessel_jv.pdfを見てみよう
  - 以下のようなグラフができていればOK



SciPyの使用例（3）：数値積分

- 数値積分を行うには, scipy.integrateというライブラリをimportする
- $\int_0^2 \mathrm{d}x\sqrt{4-x^2}$の積分を考える(答えは$\pi$)
- 右のコードを書いて実行してみよう
- trapezoidは台形公式による数値積分
  - 積分区間[a,b]の分点数num_pointsを大きくすると精度が向上する
- quadはQUADPACKというFortranライブラリによる数値積分

```python
import numpy as np
from scipy.special import jv
from scipy import integrate

def func(x):
    return np.sqrt(4.0-x*x)

a, b = 0, 2
num_points = 100

x_mesh = np.linspace(a,b,num_points)
values_func = func(x_mesh)

res = integrate.trapezoid(values_func,x_mesh)
print(res)

res = integrate.quad(func,a,b)
print(res)
```

- ライブラリの使用にあたっては, 公式ドキュメントを確認し, 何を引数として渡す必要があるか, 出力結果のデータ形式は何か, などをその都度調べればOK