

# 2025 年度 科学技術計算

学生番号 36714143  
情報工学科知能情報分野  
和田小百合

2025 年 10 月 21 日

## 目次

1	課題の概要	2
2	課題 01-1	2
2.1	課題の説明 . . . . .	2
2.2	コードの説明 . . . . .	2
2.3	実行結果 . . . . .	3
3	課題 01-5	3
3.1	課題の説明 . . . . .	3
3.2	コードの説明 . . . . .	3
3.3	実行結果 . . . . .	6

## 1 課題の概要

今回の課題では、python 言語を使って  $2 \times 2$  または  $3 \times 3$  の行列積を計算するプログラムと、指数関数とその関数のマクローリン展開がどれほど近似しているのかの可視化をグラフを用いておこないました。

## 2 課題 01-1

### 2.1 課題の説明

$2 \times 2$ ,  $3 \times 3$  の行列式を計算する関数を作成します。また、行列の次元の入力は  $2 \times 2$ ,  $3 \times 3$  のサイズのみを計算し、それ以外の行列サイズが入力された場合はエラーが出るようにします。

### 2.2 コードの説明

以下に実行コードを示す。

```
1 import math
2 import cmath
3 from typing import Tuple, List
4 import numpy as np
5 import scipy
6
7 import matplotlib.pyplot as plt
8 from matplotlib import rcParams
9 rcParams["savefig.bbox"] = "tight"
10
11 import numpy as np
12
13 def det_2x2or3x3(A: np.ndarray) -> float:
14
15     # --- 行列サイズの確認 ---
16     assert A.shape in [(2, 2), (3, 3)], "行列のサイズは 2x2 または 3x3 のみ対応しています。"
17
18     # --- 2x2 の場合 ---
19     if A.shape == (2, 2):
20         det = A[0, 0]*A[1, 1] - A[0, 1]*A[1, 0]
21
22     # --- 3x3 の場合 ---
23     else:
24         det = (
25             A[0, 0]*(A[1, 1]*A[2, 2] - A[1, 2]*A[2, 1])
26             - A[0, 1]*(A[1, 0]*A[2, 2] - A[1, 2]*A[2, 0])
27             + A[0, 2]*(A[1, 0]*A[2, 1] - A[1, 1]*A[2, 0])
28         )
29
30     return float(det)
31
32
33
34 A = np.array([[1, 2],[3, 4]])
35 result = det_2x2or3x3(A)
```

```

36 print(result) # -2.0
37
38
39 A = np.array([[1, 2, 3],[0, 1, 4],[5, 6, 0]])
40
41 result = det_2x2or3x3(A)
42 print(result) # 1.0
43
44
45 A = np.array([[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12],[13, 14, 15, 16]])
46 result = det_2x2or3x3(A) # AssertionError

```

Listing 1 課題 01-1

まず、 $2 \times 2$ ,  $3 \times 3$  の行列式の計算を行う関数を `det_2x2or3x3` で定義します。関数の入力値は `np.ndarray` により配列の形を受理します。また、この関数では行列のサイズは `A.shape` を用い判別し、 $2 \times 2, 3 \times 3$  以外はエラーがでるようにしました。

行列式の計算は、 $2 \times 2$  の行列は  
 $A[0, 0] \cdot A[1, 1] - A[0, 1] \cdot A[1, 0]$

$3 \times 3$  の行列は  
 $A[0, 0] \cdot (A[1, 1] \cdot A[2, 2] - A[1, 2] \cdot A[2, 1]) - A[0, 1] \cdot (A[1, 0] \cdot A[2, 2] - A[1, 2] \cdot A[2, 0]) + A[0, 2] \cdot (A[1, 0] \cdot A[2, 1] - A[1, 1] \cdot A[2, 0])$   
 なのでそれぞれ対応した関数を `if(条件分岐)` で表現しました。

## 2.3 実行結果

課題に書かれている入力例を張り付けて実行したところ同じ値が出てきました。よってこの関数はあっています。

## 3 課題 01-5

### 3.1 課題の説明

指数関数  $e^x$  のマクローリン展開は

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

であり、マクローリン展開で近似する関数列  $f_i(x)$  を作り、それぞれのグラフを同じグラフに表示します。また、近似誤差を測るために、関数のノルムを作り、区間  $S$  上で  $f(x)$  の絶対値が最も大きい値を、関数全体の大きさ (ノルム) とする。このノルムをグラフに表示する。

### 3.2 コードの説明

以下に実行コードを示す

```

1 import math
2 import cmath
3 from typing import Tuple, List
4 import numpy as np

```

```

5 import scipy
6
7 import matplotlib.pyplot as plt
8 from matplotlib import rcParams
9 rcParams["savefig.bbox"] = "tight"
10
11 def exp_maclaurin_partial(x: np.ndarray, i: int) -> np.ndarray:
12     y = 0
13     for n in range(i + 1):
14         y += x ** n / math.factorial(n)
15
16     return y
17
18     """f_i(x)を計算する関数"""
19
20 def norm(x: np.ndarray, i: int) -> float:
21     """ || f_i(x) - e^x ||_{\infty} を返す """
22     f_i = exp_maclaurin_partial(x, i)
23     diff = np.abs(f_i - np.exp(x))
24     return np.max(diff)
25
26
27 # グラフを出力する
28
29 from matplotlib import rcParams
30 rcParams["savefig.bbox"] = "tight" # 周囲の余白を削除する
31
32 x = np.linspace(0, 10, 200)
33 y1 = exp_maclaurin_partial(x, 5)
34 y2 = exp_maclaurin_partial(x, 10)
35 y3 = exp_maclaurin_partial(x, 15)
36 y4 = np.exp(x)
37
38
39 fig = plt.figure() # プロット用を作成figure
40 ax = fig.subplots() # にを作成figureaxis
41
42
43 # には凡例用のラベル文字列を指定label
44 ax.plot(x, y1, label="f_5(x)")
45 ax.plot(x, y2, label="f_10(x)")
46 ax.plot(x, y3, label="f_15(x)")
47 ax.plot(x, y4, label="exp(x)")
48
49 ax.set_xlabel("x") # 横軸のラベル
50 ax.set_ylabel("f_i(x) / exp(x)") # 縦軸のラベル
51 ax.set_xlim(0, 10) # 横軸の表示範囲
52 ax.set_ylim(0) # 縦軸の表示範囲
53 ax.grid(True)
54 ax.legend() # 凡例を作成
55
56
57 # ノルム計算

```

```

58 i = np.arange(0, 10)
59 sup_norms = [norm(x, n) for n in i]
60
61 fig = plt.figure() # プロット用を作成figure
62 ax = fig.subplots() # にを作成figureaxis
63
64
65
66 #ここからプロットを作成するコードを書く
67
68 ax.plot(i, sup_norms, label="norm")
69 ax.set_yscale("log") # 対数目盛
70 ax.set_xlabel("i")
71 ax.set_ylabel("norm")
72
73 ax.set_xlim(0,10) # 横軸の表示範囲\textbf{}
74 ax.grid(True, which="both")\textbf{}
75 ax.legend()
76 plt.show()

```

Listing 2 課題 01-5

マクローリン展開の近似式  $f_i(x)$  は `exp_maclaurin_partial(x: np.ndarray, i: int)` で定義し、`x` を配列で入力されてこの関数が受理します。この関数を使って `i` が増えるるとどのように近似されるのかをグラフで表示するために

```

x = np.linspace(0, 10, 200)
y1 = exp_maclaurin_partial(x, 5)
y2 = exp_maclaurin_partial(x, 10)
y3 = exp_maclaurin_partial(x, 15)
y4 = np.exp(x)

```

で `i = 5, 10, 15` で表示させました。次に近似誤差 (ノルム) の関数定義は、`def norm(x: np.ndarray, i: int) -> float:` で行い、グラフで出力する際、`i = np.arange(0, 10)` `sup_norms = [norm(x, n) for n in i]` と配列にすることによってグラフを表示させました。

### 3.3 実行結果

$e^x$  と  $e^x$  のマクローリン展開した関数  $f_i(x)$  が同じグラフで生成されたものを以下にします。

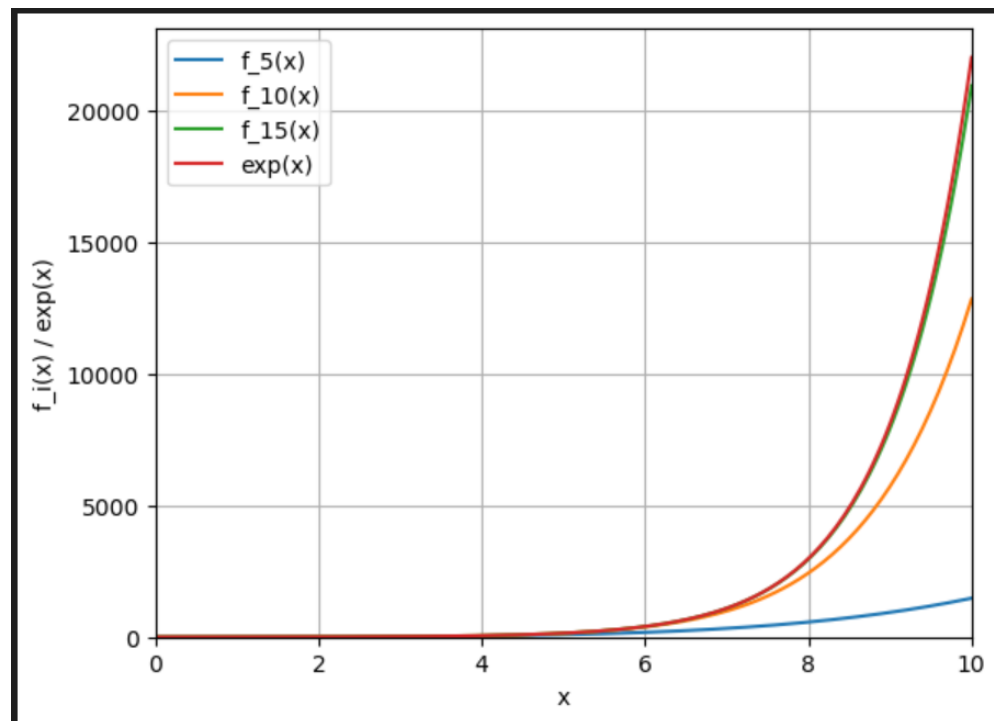


図1  $e^x$  と  $e^x$  をマクローリン展開

グラフを見ると、 $e^x$  についてマクローリン展開した関数は  $i$  が多くくなればなるほどもとの指数関数  $e^x$  に近づいていることがわかる。このグラフから次に表示される近似誤差 (ノルム) のグラフが横軸  $i$  において大きくなればなるほど近似誤差が小さくなっていくことから右肩下がりのグラフが生成されることが考えられる。

また、近似誤差のグラフをいかにしめす。

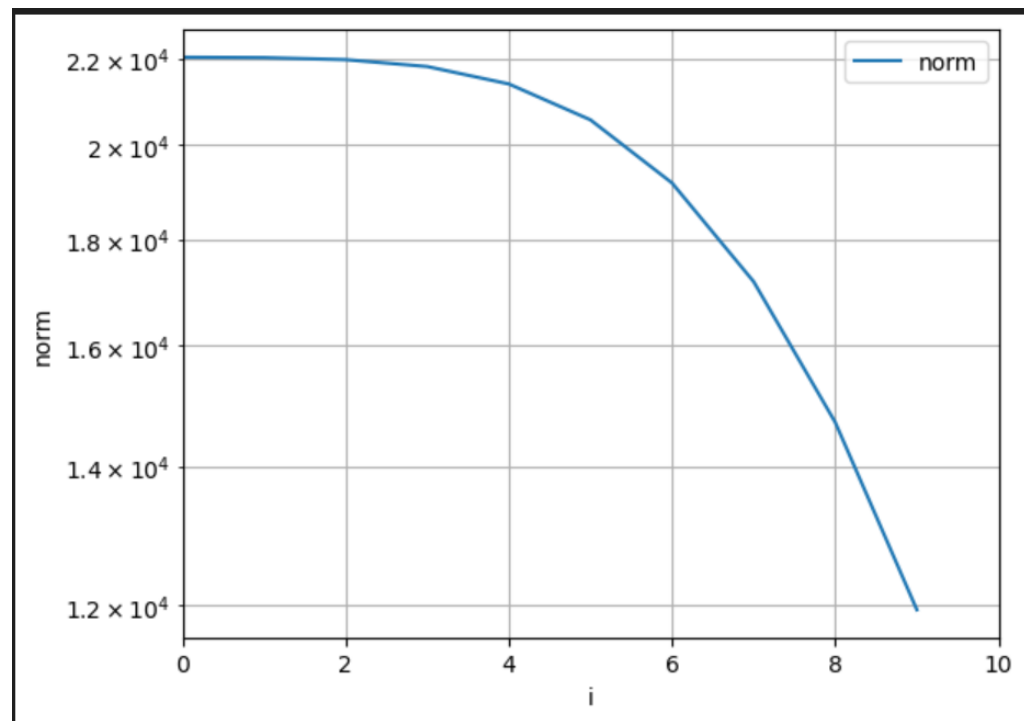


図2  $e^x$  と  $e^x$  をマクローリン展開

こちらのグラフを見ると近似誤差は、 $i$ が増えるごとに指数関数的に下がっていることがわかりました。