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

今回の課題では、python 言語を使って 2 × 2 または 3 × 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 × 2, 3 × 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 × 2, 3 × 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。

```python
import math
import cmath
from typing import Tuple, List
import numpy as np
import scipy

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams["savefig.bbox"] = "tight"

import numpy as np

def det_2x2or3x3(A: np.ndarray) -> float:

    # --- 行列サイズの確認 ---
    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。"

    # --- 2x2 の場合 ---
    if A.shape == (2, 2):
        det = A[0, 0]*A[1, 1] - A[0, 1]*A[1, 0]

    # --- 3x3 の場合 ---
    else:
        det = (
            A[0, 0]*(A[1, 1]*A[2, 2] - A[1, 2]*A[2, 1])
          - A[0, 1]*(A[1, 0]*A[2, 2] - A[1, 2]*A[2, 0])
          + A[0, 2]*(A[1, 0]*A[2, 1] - A[1, 1]*A[2, 0])
        )

    return float(det)



A = np.array([[1, 2],[3, 4]])
result = det_2x2or3x3(A)
```

```
print(result)  # -2.0


A = np.array([[1, 2, 3],[0, 1, 4],[5, 6, 0]])

result = det_2x2or3x3(A)
print(result)  # 1.0


A = np.array([[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12],[13, 14, 15, 16]])
result = det_2x2or3x3(A)# AssertionError
```

Listing 1　課題 01-1

まず、2 × 2, 3 × 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、2x2,3x3 以外はエラーがでるようにしました。

行列式の計算は, 2x2 の行列は公式より、
A[0, 0]*A[1, 1] - A[0, 1]*A[1, 0]

3x3 の行列は
A[0, 0]*(A[1, 1]*A[2, 2] - A[1, 2]*A[2, 1]) - A[0, 1]*(A[1, 0]*A[2, 2] - A[1, 2]*A[2, 0]) + A[0, 2]*(A[1, 0]*A[2, 1] - A[1, 1]*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力されました。より、この関数はあっているといえます。

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。なた、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

```
import math
import cmath
from typing import Tuple, List
import numpy as np
```

```python
import scipy

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams["savefig.bbox"] = "tight"

def exp_maclaurin_partial(x: np.ndarray,i: int) -> np.ndarray:
    y = 0
    for n in range(i + 1):
        y += x ** n / math.factorial(n)

    return y

    """f_i(x)を計算する関数"""

def norm(x: np.ndarray, i: int) -> float:
    """‖f_i(x) - e^‖∞x を返す"""
    f_i = exp_maclaurin_partial(x, i)
    diff = np.abs(f_i - np.exp(x))
    return np.max(diff)


#グラフを出力する

from matplotlib import rcParams
rcParams["savefig.bbox"] = "tight"  # 周囲の余白を削除する

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)


fig = plt.figure()  # プロット用を作成figure
ax = fig.subplots()  # にを作成figureaxis


# には凡例用のラベル文字列を指定label
ax.plot(x, y1, label="f_5(x)")
ax.plot(x, y2, label="f_10(x)")
ax.plot(x, y3, label="f_15(x)")
ax.plot(x, y4, label="exp(x)")

ax.set_xlabel("x")  # 横軸のラベル
ax.set_ylabel("f_i(x) / exp(x)")  # 縦軸のラベル
ax.set_xlim(0,10)  # 横軸の表示範囲
ax.set_ylim(0)  # 縦軸の表示範囲
ax.grid(True)
ax.legend()  # 凡例を作成


# ノルム計算
```

```
i = np.arange(0, 20)
sup_norms = [norm(x, n) for n in i]

fig = plt.figure()  # プロット用を作成figure
ax = fig.subplots()  # にを作成figureaxis



#ここからプロットを作成するコードを書く

ax.plot(i, sup_norms, label="norm")
ax.set_yscale("log")  # 対数目盛
ax.set_xlabel("i")
ax.set_ylabel("norm")

ax.set_xlim(0,10)  # 横軸の表示範囲\textbf{}
ax.grid(True, which="both")\textbf{}
ax.legend()
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。関数については、

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

であるため入力値に (x,i) を用い、x の乗は $x**n$ で表現し、n の階乗は、math 関数を用い、$math.factral(n)$ を使いました。これらを n 回繰り返して足し合わせることによってマクローリン展開式を定義しました。
また、この関数を使って i が増え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: で行いました。この関数は、入力に (x: 配列, i) を必要とし、$f_i(x) - e(x)$ の絶対値の最大値を出力する構造になっています。
そして、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めす。

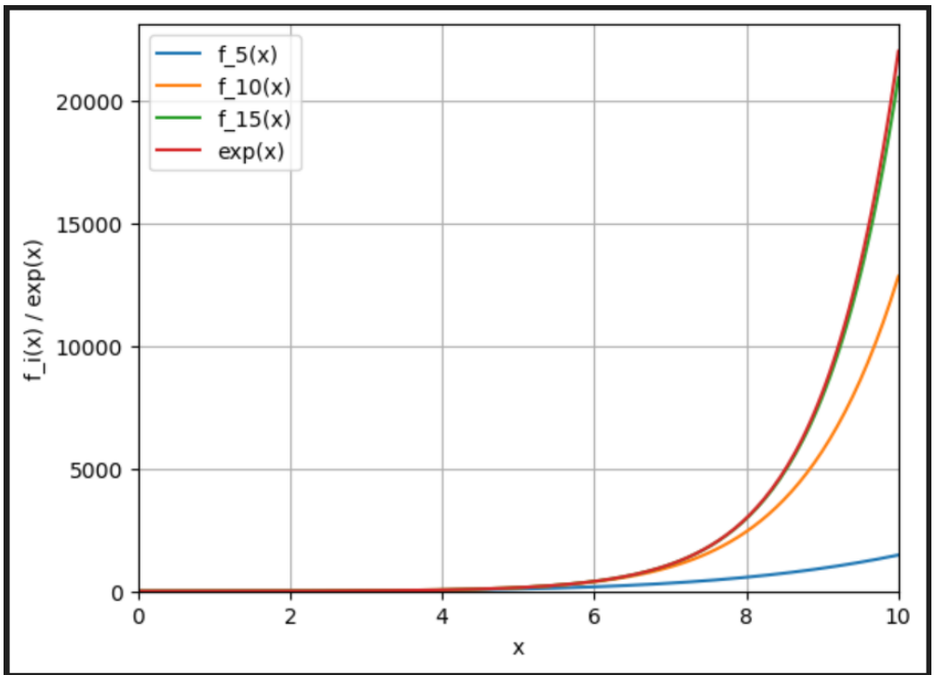

図 1　e^x と e^x をマクローリン展開

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以下にしめす。

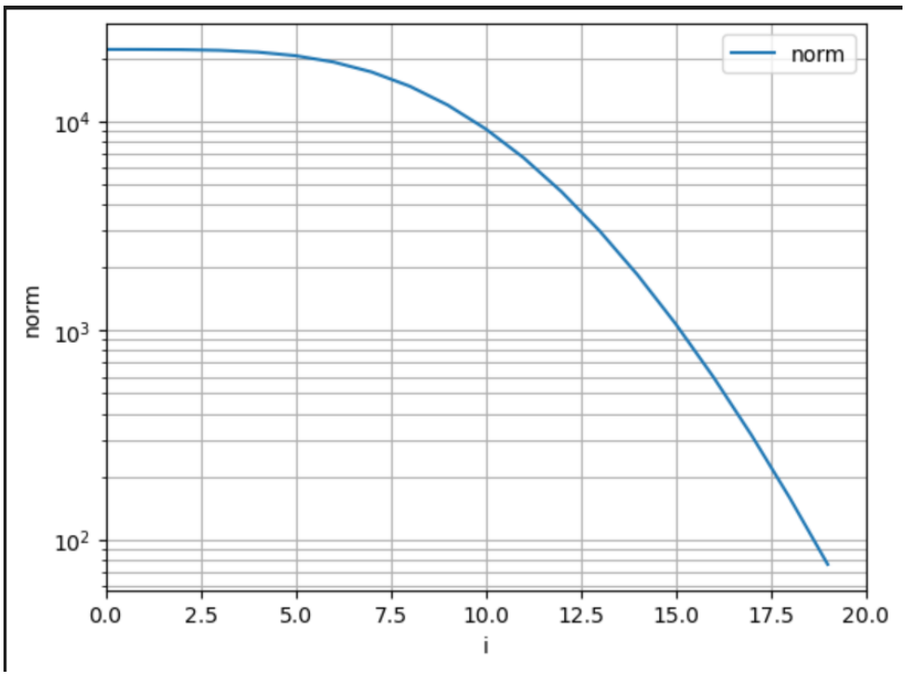

図 2　e^x と e^x をマクローリン展開

今回は i の範囲を 0 から 20 にしました。理由は上のグラフで i が大きくなればなるほど元の式に近似していくことがわかったため、i の幅を大きくすることでより違いが見えやすくなるのではないかと考えました。また、y 軸は対数グラフとしました。これにより、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。