Zadanie Numeryczne 13

Mateusz Wojtyna

1. Wstęp

Należało obliczyć całkę

$$\int_0^\infty \sin\left(\pi\frac{1+\sqrt{x}}{1+x^2}\right)e^{-x}dx$$

z dokładnością 10^{-7} używając metody Romberga z metodą trapezów.

2. Opis

Całkę możemy podzielić na „głowę" i „ogon":

$$\int_0^A \sin\left(\pi\frac{1+\sqrt{x}}{1+x^2}\right)e^{-x}dx + \int_A^\infty \sin\left(\pi\frac{1+\sqrt{x}}{1+x^2}\right)e^{-x}dx$$

a następnie zażądać, żeby ogon był $< 10^{-7}$. Rozwiązując równanie $e^{-A} < 10^{-7}$ otrzymujemy $A > 16.1181$. Wybieramy $A = 17$.

2.1. Metoda Romberga

W metodzie Romberga wyznaczamy kolejne przybliżenia całki:

- Na początku definiujemy $A_{0,0}$ jako wynik metody trapezów z jednym podziałem.
- Następnie $A_{0,k+1}$ obliczamy ze złożonej metody trapezów z 2^{k+1} podprzedziałami
- Potem wypełniamy kolejne $A_{1,k+1}, A_{2,k+1}...A_{k+1,k+1}$ elementy tabeli zgodnie ze wzorem:

$$A_{n,k} = \frac{4^n A_{n-1,k+1} - A_{n-1,k}}{4^n - 1}$$

Ze względu na implementację, w kodzie indeksowanie tabeli jest odwrócone, tzn. `A[k][n]` $= A_{n,k}$. Program zatrzymujemy jeśli $|A_{k,k} - A_{k-1,k-1}| \leq \varepsilon$ lub gdy przekroczono limit iteracji.

3. Kod

Program napisano w Pythonie z użyciem pakietów *NumPy*.

```python
import numpy as np
from numpy.typing import NDArray

num = np.float64
array = NDArray[num]


def f(x: num) -> num:
    return np.sin(np.pi * (1 + np.sqrt(x)) / (1 + x * x)) * np.exp(-x)


def trapezoidal_method(a: num, b: num, h: num) -> num:
    sum = f(a) / 2 + f(b) / 2
    while a + h < b:
        sum += f(a + h)
        a += h
    sum *= h
    return sum


def romberg(a: num, b: num, eps: num, limit: int) -> tuple[array, int]:
    # Wynik zawierający element diagonalny tabelki Romberga w każdym wierszu
    diag = []

    h = (b - a) / 2

    # indeksy odwrócone: A[k][n] = A[n][k] z wykładu
    # base case: A[0][0] = 1 podział
    prev = [h * (f(a) + f(b))]
    diag.append(prev[0])

    for k in range(1, limit + 1):
        h /= 2

        # A[k+1][0] = złożona metoda trapezów
        cur = [trapezoidal_method(a, b, h)]

        # Wypełnienie wiersza korzystając ze wzoru na A[k][n]
        factor = 1
        for n in range(1, k + 1):
            factor *= 4
            cur.append((factor * cur[n - 1] - prev[n - 1]) / (factor - 1))
```

```
        diag.append(cur[k])

        if np.abs(cur[k] - prev[k - 1]) <= eps:
            return (diag, k)

        prev = cur

    return (diag, limit)


def main():
    np.set_printoptions(suppress=True)

    # e^(-A) < 10^(-7) => A > 16.1181
    diag, steps = romberg(0, 17, 1e-7, 25)
    for n in diag:
        print(n)
    print(steps)


if __name__ == "__main__":
    main()
```

4. Wyniki

Otrzymano zbieżność po 18 krokach do wartości -0.217275.

k	$\boldsymbol{A_{k,k}}$
0	0
1	0.0392695
2	0.3649093
3	0.1910063
4	−0.1383214
5	−0.1783485
6	−0.2064431
7	−0.2135060
8	−0.2159571
9	−0.2168117
10	−0.2171117
11	−0.2172174
12	−0.2172547
13	−0.2172678
14	−0.2172725
15	−0.2172741
16	−0.2172747
17	−0.2172749
18	−0.2172750