Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Лабораторна робота #4

Обчислення інтегралів за складеними квадратурними формулами трапецій і парабол

Виконав:

Захаров Дмитро Олегович

Група МП-31

Харків – 2023

Зміст

1 Постановка задачі **2**

2 Опис методу **3**

2.1 Складова квадратурна формула трапеції 3

2.2 Складова квадратурна формула парабол 3

3 Текст програми **5**

3.1 Складова квадратурна формула трапеції 5

3.2 Складова квадратурна формула парабол 6

3.3 Програма запуску 7

4 Результати **9**

5 Висновки **10**

1 Постановка задачі

Обчислити заданий інтеграл з точністю $\varepsilon = 10^{-6}$ за квадратурними формулами трапецій та парабол, використовуючи двійний перерахунок і оцінку за Рунге.

На друк вивести наближене значення інтегралу.

Варіант 5.

$$\int_{[0,1]} \frac{1 - \cos x}{x} dx$$

2 Опис методу

2.1 Складова квадратурна формула трапеції

Нехай треба знайти значення інтегралу

$$\mathcal{I} = \int_{[\alpha,\beta]} f(x)dx$$

Інтеграл обчислюємо за допомогою наступного алгоритму:

Algorithm 1 Складова квадратурна формула трапеції, спосіб #1

$h \leftarrow \frac{\beta-\alpha}{2}$
$\mathcal{I}_h \leftarrow \frac{f(\alpha)+f(\beta)}{2} \cdot h$
$\rho \leftarrow \infty, k \leftarrow 1$
for $|\rho| > \varepsilon$ **do**
  $n \leftarrow 2^k$
  $h_{2n} \leftarrow \frac{h}{2n}$
  $\mathcal{I}_{2n} \leftarrow \frac{1}{2}\mathcal{I}_h + h_{2n} \sum_{i=1}^{n} f\left(\alpha + (2i-1)h_{2n}\right)$
  $k \leftarrow k + 1$
  $\rho \leftarrow \frac{\mathcal{I}_h - \mathcal{I}_{2n}}{3}$
end for
  return $\mathcal{I}_{2n} + \rho$

Також, як альтернатива, ми використаємо наступний алгоритм:

Algorithm 2 Складова квадратурна формула трапеції, спосіб #2

$h \leftarrow \beta - \alpha,\ k \leftarrow 1,\ \delta \leftarrow +\infty,\ \mathcal{I} \leftarrow 0$
for $|\delta| > \varepsilon$ **do**
  $n \leftarrow 2^k$ ▷ Розбиваємо відрізок на 2^k частин
  $h_n \leftarrow \frac{h}{n}$ ▷ Знаходимо розмір інтервалів
  $\{x_j\}_{j=0}^{n} \leftarrow \{\alpha + jh_n\}_j$ ▷ Задаємо вузли
  $\mathcal{I}_n \leftarrow \frac{h_n}{2} \sum_{j=1}^{n} \{f(x_{k-1}) + f(x_k)\}$ ▷ Значення інтегралу для 2^k розбиттів
  $\delta = \mathcal{I}_n - \mathcal{I}$ ▷ Знаходимо різницю між сусідніми апроксимаціями
  $\mathcal{I} \leftarrow \mathcal{I}_n$
end for
  return $\mathcal{I}$

2.2 Складова квадратурна формула парабол

Тут ми застосовуємо наступний алгоритм:

Algorithm 3 Складова квадратурна формула парабол, спосіб #1

$h \leftarrow \frac{\beta-\alpha}{2}, s_0 \leftarrow f(\alpha)+f(\beta), s_1 \leftarrow f(\alpha+h), s_2 \leftarrow 0$
$\mathcal{I}_h \leftarrow (s_0+4s_1+2s_2)\frac{h}{3}$
$\rho \leftarrow \infty, k \leftarrow 1$
for $|\rho| > \varepsilon$ **do**
  $n \leftarrow 2^k$
  $h_{2n} \leftarrow \frac{h}{2n}$
  $\mathcal{I}_{2n} \leftarrow \frac{h_{2n}}{3}\left(s_0 + 4\sum_{i=1}^{n} f\left(\alpha+(2i-1)h_{2n}\right) + 2\sum_{i=1}^{n-1} f\left(\alpha+2ih_{2n}\right)\right)$
  $k \leftarrow k+1$
  $\rho \leftarrow \frac{\mathcal{I}_h-\mathcal{I}_{2n}}{15}$
end for
  return $\mathcal{I}_{2n}+\rho$

Також, як альтернатива, ми використаємо наступний алгоритм:

Algorithm 4 Складова квадратурна формула парабол, спосіб #2

$h \leftarrow \beta-\alpha,\ k \leftarrow 1,\ \delta \leftarrow +\infty,\ \mathcal{I} \leftarrow 0$
for $|\delta| > \varepsilon$ **do**
  $n \leftarrow 2^k$ ▷ Розбиваємо відрізок на 2^k частин
  $h_n \leftarrow \frac{h}{n}$ ▷ Знаходимо розмір інтервалів
  $\{x_j\}_{j=0}^{n} \leftarrow \{\alpha + jh_n\}_j$ ▷ Задаємо вузли
  $\mathcal{I}_n \leftarrow \frac{h_n}{3}\left[f(x_0) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + f(x_n)\right]$
  $\delta = \mathcal{I}_n - \mathcal{I}$ ▷ Знаходимо різницю між сусідніми апроксимаціями
  $\mathcal{I} \leftarrow \mathcal{I}_n$
end for
  return $\mathcal{I}$

3 Текст програми

Повний текст програми можна знайти за цим посиланням (← напис клікабельний) на *GitHub* сторінку.

3.1 Складова квадратурна формула трапеції

Створимо файл `integrators.py` та реалізуємо алгоритм з розділу 2.1 у класі `TrapezoidalIntegrator`. При ініціалізації ми будемо вказувати функцію, що будемо інтегрувати, а також додамо метод `evaluate`, що приймає інтервал для інтегрування, а також бажану точність ε:

```python
from typing import Callable, Tuple, TypeAlias
from abc import ABC, abstractmethod

# Define type alias for an interval type
Interval: TypeAlias = Tuple[float, float]

class Integrator(ABC):
    """
    Abstract class for classes that implement function integration
    """

    def __init__(self, fn: Callable[[float], float]) -> None:
        """ Initialize the integrator with a function to integrate

        Args:
            fn (Callable[[float], float]): Function to integrate
        """
        self._fn = fn
        pass

    @abstractmethod
    def evaluate(self, interval: Interval, accuracy: float = 1e-6) -> float:
        """
        Evaluate the integral of the function over the interval

        Args:
            interval (Interval): Interval to integrate over
            accuracy (float, optional): Desired accuracy of the integral. Defaults to `1e-6`.
        """
        pass

class TrapezoidalIntegrator(Integrator):
    """
    Class for trapezoidal integration
    """
```

```

    # Maximum number of iterations
    MAX_ITERATIONS: int = 20

    def __init__(self, fn: Callable[[float], float]) -> None:
        super().__init__(fn)

    def evaluate(self, interval: Interval, accuracy: float = 1e-6) -> float:
        MAX_VALUE: float = 1e12
        interval_size: float = interval[1] - interval[0]
        estimate: float = MAX_VALUE
        error: float = MAX_VALUE
        k: int = 1 # Number of iterations

        while abs(error) > accuracy:
            intervals_number: int = 2**k
            step_size: float = interval_size / intervals_number
            k = k + 1

            nodes = [interval[0] + i * step_size for i in range(0, intervals_number+1)]
            current_esimate = (step_size / 2) * sum([(self._fn(nodes[i]) + self._fn(nodes[i+1])) for i in range(0, intervals_number)])

            error = current_esimate - estimate
            estimate = current_esimate
            if k > TrapezoidalIntegrator.MAX_ITERATIONS:
                raise RuntimeError("Maximum number of iterations reached")

        return estimate
```

Лістинг 1: Реалізація складової квадратурної формули трапеції

3.2 Складова квадратурна формула парабол

Ідея така сама, як в попередньому пункті.

```
class SimpsonIntegrator(Integrator):
    """
    Class for Simpson integration
    """

    # Maximum number of iterations
    MAX_ITERATIONS: int = 20

    def __init__(self, fn: Callable[[float], float]) -> None:
        super().__init__(fn)

```

```
    def evaluate(self, interval: Interval, accuracy: float = 1e-6) -> float:
        MAX_VALUE: float = 1e12
        interval_size: float = interval[1] - interval[0]
        estimate: float = MAX_VALUE
        error: float = MAX_VALUE
        k: int = 1 # Number of iterations

        while abs(error) > accuracy:
            intervals_number: int = 2**k
            step_size: float = interval_size / intervals_number
            k = k + 1

            nodes = [interval[0] + i * step_size for i in range(0, intervals_number+1)]
            current_esimate = (step_size / 3) * (
                self._fn(nodes[0]) +
                4*sum([self._fn(nodes[i]) for i in range(1, intervals_number, 2)]) +
                2*sum([self._fn(nodes[i]) for i in range(2, intervals_number, 2)]) +
                self._fn(nodes[-1])
            )

            error = current_esimate - estimate
            estimate = current_esimate
            if k > SimpsonIntegrator.MAX_ITERATIONS:
                raise RuntimeError("Maximum number of iterations reached")

        return estimate
```

Лістинг 2: Реалізація складової квадратурної формули парабол

3.3 Програма запуску

Програма для запуску зовсім проста: створюємо `TrapezoidalIntegrator` та `SimpsonIntegrator`, а потім проганяємо нашу функцію $f(x) = \frac{1-\cos x}{x}$ через неї:

```
from typing import Callable
from rich import print
from math import cos
from integrators import Interval, TrapezoidalIntegrator, SimpsonIntegrator

if __name__ == '__main__':
    fn: Callable[[float], float] = lambda x: (1 - cos(x)) / x
    # We define the left endpoint of the interval to be EPSILON instead of 0 to avoid division by zero
    EPSILON: float = 1e-12
```

```python
    interval: Interval = (EPSILON, 1.0)

    trapezoidal_integrator: TrapezoidalIntegrator = TrapezoidalIntegrator(fn)
    trapezoidal_integral_estimate = trapezoidal_integrator.evaluate(interval, accuracy=1e-6)

    simpson_integrator: SimpsonIntegrator = SimpsonIntegrator(fn)
    simpson_integrator_estimate = simpson_integrator.evaluate(interval, accuracy=1e-6)

    print(f"Trapezoidal integral estimate: {trapezoidal_integral_estimate}")
    print(f"Simpson integral estimate: {simpson_integrator_estimate}")
```

Лістинг 3: Перевірка результатів

4 Результати

Після запуску програми, отримуємо два майже однакових значення, що приблизно дорівнюють:

$$\int_{[0,1]} \frac{1-\cos x}{x} dx \approx 0.239812$$

Різниця у значеннях починається з 7 знака після коми. Більш конкретні значення наводимо нижче:

Формула трапеції	0.23981159166750359857
Формула парабол	0.23981174863719770252
Wolfram Mathematica	0.23981174200056472594

У *Wolfram Mathematica* ми використали наступну команду для приблизного обчислення:

```
NIntegrate[(1 - Cos[x])/x, {x, 0, 1}, AccuracyGoal -> 30,
    WorkingPrecision -> 20]
```

Лістинг 4: Обрахунок інтегралу в *Wolfram Mathematica*

5 Висновки

В цій лабораторній роботі ми:

- навчилися чисельно інтегрувати функцію на заданому відрізку $\int_{[\alpha,\beta]} f(x)dx$.
- навчилися писати комп'ютерну програму (на прикладі мови `Python`), що приймає на вхід функцію $f(x)$ та інтервал $[\alpha, \beta]$, і інтегрує її.
- Порівняли отримані результати з математичним пакетом *Wolfram Mathematica*.

При заданій точності, інтеграли збігаються дуже швидко ($1 - 2$ кроки), оскільки ми проходимось по ступеням двійки. При цьому, точність досяглася більше, використовуючи формулу парабол, якщо порівнювати з результатом в пакеті *Wolfram Mathematica*. Отже, ми спробували два точних і відносно швидких методи для знаходження приблизних значень визначених інтегралів.