НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи

з дисципліни «АЛГОРИТМИ І СИСТЕМИ КОМП’ЮТЕРНОЇ МАТЕМАТИКИ 1.МАТЕМАТИЧНІ АЛГОРИТМИ»

на тему

“Чисельне диференціювання та інтегрування”

Виконав:	Перевірила:
студент групи КМ-03	Асистент кафедри ПМА
Шаповалов Г. Г.	Ковальчук-Химюк Л. О.

Київ — 2023

Зміст

Вступ

Метою роботи є ознайомитися з програмними засобами чисельного диференціювання й інтегрування функцій; практичне розв’язання задач з використанням СКМ, порівняльний аналіз методів чисельного диференціювання й інтегрування.

Основна частина

Варіант 7

Підінтегральна функція	Проміжок інтегрування	Кількість частин розбиття	Крок h	Первісна функція
$\frac{x^2}{2x+3}$	[1; 3]	100	0.02	$\frac{1}{8}(2x^2 - 6x + 9\ln(2x+3))$

Теоретичні відомості

Суть методу Сімпсона полягає в наближенні підінтегральної функції на відрізку [a,b] інтерполяційним многочленом другого ступеня p2(x), тобто наближення графіка функції на відрізку параболою.

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Рис. 1 – Формула Сімпсона

Для більш точного обчислення інтеграла, інтервал [a, b] розбивають на N елементарних відрізків однакової довжини і застосовують формулу Сімпсона на складових відрізках. Кожен відрізок складається з сусідньої пари елементарних відрізків. Значення вихідного інтеграла є сумою результатів інтегрування на складових відрізках:

$$\int_a^b f(x)dx \approx \frac{h}{3}\cdot\left(\frac{1}{2}f(x_0) + \sum_{k=1}^{N-1} f(x_k) + 2\sum_{k=1}^{N} f\left(\frac{x_{k-1}+x_k}{2}\right) + \frac{1}{2}f(x_N)\right)$$

Рис. 2 – Ітераційна формула Сімпсона

Похибка рахується шляхом віднімання по модулю результату за методом Сімпсона та результату за формулою Ньютона-Лейбніца.

Опис програми

Програма складається з двох частин – диференціювання та інтегрування.

Python

Для інтегрування використовувалась функція simps [2] із бібліотеки Sympy [1]. Математичні функції задаються у текстовому файлі, а потім зчитуються програмою.

Диференціювання здійснюється функцією diff [3] із бібліотеки Sympy [1]

Octave

Програма використовує Oct2py [4] бібліотеку для створення ‘мосту’ між пайтоном та октав. Вбудована функція eval() розбить із тексту математичної формули об’єкт октав.

Диференціювання здійснюється за допомогою правої границі [5]

Для інтегрування використовується метод Сімпсона. Нижня межа є статичною, а верхня змінюється з кроком h = 0.02

Висновки

Досягнуто таких результатів:

Похибка диференціювання Python:

- Мінімальна – 0
- Максимальна – 1.11022302462516e-16

Похибка диференціювання Octave:

- Мінімальна – 6.668748e-09
- Максимальна – 1.654807e-08

Похибка інтегрування Python:

- Мінімальна – 0
- Максимальна – 9.447105e-12

Похибка інтегрування Octave:

- Мінімальна – 0
- Максимальна – 4.440892e-16

Отримали хороші результати, оскільки похибки малі. Скріншоти з результатами наведені в Додатку А.

Відповіді на контрольні питання

1. ***Як залежить похибка усікання і похибка округлення від розміру кроку при чисельному диференціюванні ?*** – Похибка усікання і похибка округлення в чисельному диференціюванні залежать від розміру кроку. Похибка усікання зазвичай зменшується при зменшенні розміру кроку, оскільки більша кількість точок дозволяє краще апроксимувати функцію. Однак похибка округлення може збільшуватися при зменшенні розміру кроку через накопичення помилок округлення при виконанні більшої кількості обчислень.

2. ***Оцініть похибку заданої формули чисельного диференціювання за допомогою дослідження на апроксимацію*** – Оцінка похибки заданої формули чисельного диференціювання може бути виконана за допомогою дослідження на апроксимацію. Це включає в себе порівняння результатів, отриманих за допомогою формули, з точними значеннями, якщо вони доступні, або з результатами, отриманими за допомогою більш точних методів.

3. ***Проведіть порівняння точності формул трапецій і Сімпсона на підставі аналізу залишкових членів формул*** – Точність формул трапецій і Сімпсона можна порівняти на основі аналізу залишкових членів формул. Зазвичай формула Сімпсона дає більш точні результати, ніж формула трапецій, оскільки вона базується на квадратичному апроксимаційному поліномові, тоді як формула трапецій використовує лінійний апроксимаційний поліном.

4. ***У яких випадках доцільно використовувати квадратні формули Гауса ?*** – Квадратурні формули Гауса доцільно використовувати, коли функція є достатньо гладкою і не має особливостей в межах інтегрування. Вони також ефективні для інтегрування поліноміальних функцій, оскільки вони можуть надати точний результат для поліномів до певного степеня.

5. ***Які формули чисельного інтегрування зручніше програмувати для ЕОМ ?*** – Для програмування на ЕОМ зручно використовувати ті формули чисельного інтегрування, які легко реалізуються і не вимагають складних обчислень. Це можуть бути такі методи, як метод прямокутників, метод трапецій та метод Сімпсона. Однак оптимальний вибір методу залежить від конкретної задачі і вимог до точності.

Перелік посилань

1. Sympy - https://www.sympy.org/en/index.html
2. Метод Сімпсона - https://www.blacksacademy.net/texts/SimpsonMethod.pdf
3. Sympy.diff - https://docs.sympy.org/latest/tutorials/intro-tutorial/calculus.html
4. Oct2py - https://pypi.org/project/oct2py/
5. Права границя диференціювання - http://techtrend.com.ua/index.php?newsid=17216

Додаток А – Скріншоти роботи програми

```
Диференціювання за допомогою Python

Час виконання 0.634 сек

                    f(x)              (F(x)')               Різниця
1.00  0.200000000000000  0.200000000000000                     0
1.02  0.206428571428571  0.206428571428571                     0
1.04  0.212913385826772  0.212913385826772                     0
1.06  0.219453125000000  0.219453125000000                     0
1.08  0.226046511627907  0.226046511627907  2.77555756156289e-17
...                 ...                ...                   ...
2.92  0.964524886877829  0.964524886877829  1.11022302462516e-16
2.94  0.973378378378379  0.973378378378379  1.11022302462516e-16
2.96  0.982242152466369  0.982242152466368  1.11022302462516e-16
2.98  0.991116071428572  0.991116071428572  1.11022302462516e-16
3.00   1.00000000000000   1.00000000000000  2.22044604925031e-16
```

Рис. 1 – Диференціювання Python

```
Інтегрування за допомогою Python

Час виконання 0.21 сек

      м.Сімпсона  м.Ньютона-Лейбніца       Різниця
1.00    0.000000            0.000000  0.000000e+00
1.02    0.004064            0.004064  9.485896e-08
1.04    0.008258            0.008258  9.447105e-12
1.06    0.012581            0.012581  1.067231e-09
1.08    0.017036            0.017036  1.817894e-11
...          ...                 ...           ...
2.92    1.082680            1.082680  1.378571e-10
2.94    1.102059            1.102059  2.023537e-11
2.96    1.121615            1.121615  1.384137e-10
2.98    1.141349            1.141349  2.495293e-11
3.00    1.161260            1.161260  1.389457e-10
```

Рис. 2 – Інтегрування Python

```
Диференціювання за допомогою Octave

Час виконання 96.5611 сек

           f(x)     (F(x)'       Різниця
1.00  0.200000  0.200000  1.654807e-08
1.02  0.206429  0.206429  8.296550e-09
1.04  0.212913  0.212913  1.219655e-08
1.06  0.219453  0.219453  2.659049e-09
1.08  0.226047  0.226046  1.899277e-08
...        ...       ...           ...
2.92  0.964525  0.964525  1.751209e-08
2.94  0.973378  0.973378  2.268305e-08
2.96  0.982242  0.982242  4.622189e-08
2.98  0.991116  0.991116  6.668748e-09
3.00  1.000000  1.000000  6.077472e-09
```

Рис. 3 – Диференціювання Octave

```
Інтегрування за допомогою Octave

Час виконання 98.2354 сек

      м.Сімпсона  м.Ньютона-Лейбніца        Різниця
1.00    0.000000            0.000000  0.000000e+00
1.02    0.004064            0.004064  2.931683e-16
1.04    0.008258            0.008258  1.942890e-16
1.06    0.012581            0.012581  1.301043e-16
1.08    0.017036            0.017036  2.706169e-16
...          ...                 ...           ...
2.92    1.082680            1.082680  0.000000e+00
2.94    1.102059            1.102059  4.440892e-16
2.96    1.121615            1.121615  2.220446e-16
2.98    1.141349            1.141349  0.000000e+00
3.00    1.161260            1.161260  4.440892e-16
```

Рис. 4 – Інтегрування Octave

Додаток В – Код програми

Вміст файлу variant_7.txt :

1

3

100

x.^2 / (2*x + 3)

(2*x.^2 - 6*x + 9*log(2*x + 3)) / 8

Вміст файлу main.py :

```
import time
import numpy as np
import pandas as pd
import sympy as sp
from scipy.integrate import simps
from oct2py import Oct2Py


def get_func():
    '''Читання формули, первісної та меж інтегрування з файлу
    '''
    with open('variant_7.txt', 'r') as f:
        lines = f.readlines()
        lower_limit = float(lines[0].strip())
        upper_limit = float(lines[1].strip())
        n = float(lines[2].strip())
        func = lines[3].strip()
        antiderivative = lines[4].strip()
    return [lower_limit, upper_limit, n, func, antiderivative]

lower_limit, upper_limit, _, func, _ = get_func()

if lower_limit > upper_limit:
    print('lower_limit > upper_limit')


start_timer = time.time()
lower_limit, upper_limit, n, func_str, antiderivative_str = get_func()

try:
    func_str = func_str.replace('.', '')
    func_str = func_str.replace('^', '**')
    antiderivative_str = antiderivative_str.replace('.', '')
    antiderivative_str = antiderivative_str.replace('^', '**')
```

```python
    x = sp.symbols('x')
    func = sp.sympify(func_str)  #  Функція
    antiderivative = sp.sympify(antiderivative_str)  #  Первісна

    antiderivative = sp.diff(antiderivative, x)  # Обчислення формули похідної

    h = (upper_limit - lower_limit) / n  #  Крок
    x_values = np.arange(lower_limit, upper_limit+h, h)  # Створюємо масив точок на
інтервалі [a, b] з кроком h

    # Обчислення значень функцій у цих точках
    func_values = [func.subs(x, val).evalf() for val in x_values]
    antiderivative_values = [antiderivative.subs(x, val).evalf() for val in
x_values]

    # Обчислення різниці між значеннями функцій
    difference = np.array(func_values) - np.array(antiderivative_values)

    diff_py = pd.DataFrame({
        'f(x)': func_values,
        '(F(x)\')': antiderivative_values,
        'Різниця': np.abs(difference)
    }, index=x_values)

    print('\n\nДиференціювання за допомогою Python\n')
    print(f'Час виконання {round(time.time() - start_timer, 4)} сек\n')
    print(diff_py)
except:
    print('SYNTAX ERROR')




start_timer = time.time()
lower_limit, upper_limit, n, func_str, antiderivative_str = get_func()

try:
    func_str = func_str.replace('.', '')
    func_str = func_str.replace('^', '**')
    antiderivative_str = antiderivative_str.replace('.', '')
    antiderivative_str = antiderivative_str.replace('^', '**')

    x = sp.symbols('x')
    func = sp.sympify(func_str)  #  Функція
    antiderivative = sp.sympify(antiderivative_str)  #  Первісна

    int_func = sp.integrate(func, x)  #  Інтеграл від функції

    h = (upper_limit - lower_limit) / n  #  Крок
```

```
    x_values = np.arange(lower_limit, upper_limit+h, h)  # Створюємо масив точок на
інтервалі [a, b] з кроком h

    # Перетворюємо функцію та первісну у викликаємі функції
    func = sp.lambdify(x, func)
    antiderivative = sp.lambdify(x, antiderivative)

    y_values = [func(point) for point in x_values]  #  Створюємо масив значень
функції для кожної точки

    # Обчислюємо чисельне значення визначеного інтегралу для кожної точки
    simpson = [simps(y_values[:i+1], x_values[:i+1], dx=h) for i in
range(len(x_values))]

    # Обчислюємо чисельне значення визначеного інтегралу за допомогою методу
Ньютона-Лейбніца
    newton_leibniz = [antiderivative(point) - antiderivative(lower_limit) for point
in x_values]

    # Обчислюємо різницю між значеннями, отриманими двома методами
    difference = [abs(simpson[i] - newton_leibniz[i]) for i in
range(len(x_values))]

    int_py = pd.DataFrame({
        'м.Сімпсона': simpson,
        'м.Ньютона-Лейбніца': newton_leibniz,
        'Різниця': difference
    }, index=x_values)

    print('\n\nІнтегрування за допомогою Python\n')
    print(f'Час виконання {round(time.time() - start_timer, 4)} сек\n')
    print(int_py)
except:
    print('SYNTAX ERROR')



start_timer = time.time()
lower_limit, upper_limit, n, func_str, antiderivative_str = get_func()

try:
    oc = Oct2Py()
    oc.eval('f1 = @(x) {0};'.format(func_str))
    oc.eval('F = @(x) {0};'.format(antiderivative_str))

    # Обчислюємо похідну функції F
    oc.eval('f2 = @(x) diff(F(x))./diff(x);')

    h = (upper_limit - lower_limit) / n  #  Крок
    x_values = np.arange(lower_limit, upper_limit+h, h)  # Створюємо масив точок на
інтервалі [a, b] з кроком h

    # Обчислюємо значення функцій f1 та f2 для кожної точки в x_values
```

```
    results_f1 = []
    results_f2 = []
    delta = 1e-8  # Мале число для обчислення похідної
    for x in x_values:
        result_f1 = oc.eval('f1({0});'.format(x))
        #  Обчислюємо похідну за допомогою методу передньої різниці
        result_f2 = oc.eval('(F({0}+{1}) - F({0})) / {1};'.format(x, delta))
        results_f1.append(result_f1)
        results_f2.append(result_f2)

    # Створюємо DataFrame з результатами
    diff_oct = pd.DataFrame({
        'f(x)': results_f1,
        '(F(x)\'': results_f2,
        'Різниця': np.abs(np.array(results_f1) - np.array(results_f2))
    }, index=x_values)

    # Виводимо результати
    print('\n\nДиференціювання за допомогою Octave\n')
    print(f'Час виконання {round(time.time() - start_timer, 4)} сек\n')
    print(diff_oct)
except:
    print('SYNTAX ERROR')



start_timer = time.time()
lower_limit, upper_limit, n, func_str, antiderivative_str = get_func()

try:
    oc = Oct2Py()
    oc.eval('f1 = @(x) {0};'.format(func_str))
    oc.eval('f2 = @(x) {0};'.format(antiderivative_str))

    h = (upper_limit - lower_limit) / n  #  Крок
    x_values = np.arange(lower_limit, upper_limit+h, h)  # Створюємо масив точок на
інтервалі [a, b] з кроком h

    # Обчислюємо інтеграли за допомогою методу Ньютона-Лейбніца та методу Сімсона
для кожної точки в x_values
    results_simpson = []
    results_newton_leibniz = []
    for x in x_values:
        result_simpson = oc.eval('quad(f1, {0}, {1});'.format(lower_limit, x))
        result_newton_leibniz = oc.eval('f2({1}) - f2({0});'.format(lower_limit,
x))
        results_simpson.append(result_simpson)
        results_newton_leibniz.append(result_newton_leibniz)

    int_oct = pd.DataFrame({
        'м.Сімпсона': results_simpson,
        'м.Ньютона-Лейбніца': results_newton_leibniz,
```

```
        'Різниця': np.abs(np.array(results_simpson) -
np.array(results_newton_leibniz))
    }, index=x_values)

    # Виводимо результати
    print('\n\nІнтегрування за допомогою Octave\n')
    print(f'Час виконання {round(time.time() - start_timer, 4)} сек\n')
    print(int_oct)
except:
    print('SYNTAX ERROR')
```