НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи

із дисципліни «АЛГОРИТМИ І СИСТЕМИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ 1.МАТЕМАТИЧНІ АЛГОРИТМИ»

на тему:

Розв'язання звичайних диференціальних рівнянь

Виконав:
студент групи КМ-13
Онищенко В.С.

Перевірила:
асистент кафедри ПМА
Ковальчук-Химюк Л.О.

Київ — 2024 року

Зміст

ВСТУП

Мета роботи: дослідження чисельних методів розв’язання звичайних диференціальних рівнянь, зокрема методів Рунге-Кутта. Розробка програмного забезпечення для чисельного розв’язання заданих рівнянь із використанням математичних бібліотек. Порівняння точності чисельних рішень із аналітичними розв’язками та аналіз впливу параметрів, таких як крок інтегрування, на точність результатів.

Для розробки використовувати мови Python та Octave[1]

1 ПОРЯДОК ВИКОНАННЯ РОБОТИ

1) Одержати варіант завдання: диференціальне рівняння, метод інтегрування, використовану підпрограму СКМ.

2) Скласти блок-схему алгоритму розв’язання диференціального рівняння.

3) Скласти програму розв’язання диференціального рівняння, одержати розв’язок за допомогою розробленої програми і за допомогою підпрограми СКМ.

4) Результати обчислень оформити у вигляді графіків.

5) Визначити близькість отриманого заданим методом розв’язку до точного значення. Як значення використовувати розв’язок, отриманий за допомогою СКМ.

2 ОСНОВНА ЧАСТИНА

2.1 Вихідні дані

Диференційне рівняння:

$$y'' + y = x^2 - x + 2$$

Початкова умова:

$$\begin{cases} y(0) = 1; \\ y'(0) = 0 \end{cases}$$

Проміжок інтегрування: $[0; 1]$

Крок: $0{,}1$

2.2 Типовий перелік вимог до ПЗ

Програмне забезпечення, розроблюване в рамках виконання кожної лабораторної роботи, повинно задовольняти низку вимог, які можна розділити на *обов'язкові* (які ПЗ повинно задовольняти незалежно від лабораторної роботи) та *варіативні* (які для кожної лабораторної роботи унікальні). До обов'язкових вимог належать:

– У програмі повинно бути передбачено перевірки на некоректне введення для всіх полів введення, зокрема:

1) порожнє введення;

2) синтаксично некоректне введення (наприклад, літери в полі для числових коефіцієнтів);

3) уведення спеціальних символів;

4) уведення чисел, які перевищують максимальний розмір для чисел відповідного типу даних (для перевірки на переповнення розрядної сітки).

У випадку некоректного введення повинно з'являтися діалогове вікно з відповідним повідомленням.

– Для всіх полів введення повинно бути визначено гранично допустиму кількість символів (для числових полів — гранично допустимі значення).

– У програмі повинно відслідковуватися переповнення розрядної сітки під час виконання обчислень. У випадку переповнення повинно з'являтися діалогове вікно з відповідним повідомленням.

– У графічному інтерфейсі користувача повинно бути передбачено можливість гнучкого налаштування розмірності розв'язуваної задачі (наприклад, можливість зміни розмірності матриці чи кількості складів у транспортній задачі).

До варіативних вимог належать вимоги щодо перевірки на коректне опрацювання виключних ситуацій, які можуть виникати під час застосування

заданого методу до розв’язання поставленої задачі (наприклад, коли сума заявок не збігається з сумою ресурсів у транспортній задачі, нижня межа інтегрування перевищує верхню тощо).

2.3 Опис методів

2.3.1 Методи Рунге-Кутта

Методи Рунге-Кутта[2] належать до однокрокових чисельних методів розв'язання задачі Коші для звичайних диференціальних рівнянь. Ці методи дозволяють обчислювати значення розв'язку з високою точністю, обчислюючи наближення у наступній точці на основі середньозваженого значення функції $f(x, y)$ на поточному проміжку.

Задача Коші

Методи застосовуються для розв'язання задачі Коші:

$$y'(x) = f(x, y), \quad y(x_0) = y_0,$$

де x_0 — початкове значення аргументу, y_0 — початкове значення функції, а $f(x, y)$ — права частина рівняння.

Метод Рунге-Кутта 2-го порядку

Для другого порядку точності використовуються такі формули:

$$k_1 = h \cdot f(x_n, y_n),$$

$$k_2 = h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right),$$

$$y_{n+1} = y_n + k_2,$$

де h — крок інтегрування, k_1 і k_2 — допоміжні значення, які враховують зміну функції на поточному кроці.

Метод також відомий як метод середньої точки. Його локальна похибка має порядок $O(h^3)$, а глобальна похибка $O(h^2)$.

Метод Рунге-Кутта 4-го порядку

Метод четвертого порядку є найпоширенішим серед методів Рунге-Кутта завдяки його точності і простоті. Формули для обчислення:

$$k_1 = h \cdot f(x_n, y_n),$$

$$k_2 = h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right),$$

$$k_3 = h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right),$$

$$k_4 = h \cdot f(x_n + h, y_n + k_3),$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Метод забезпечує локальну похибку $O(h^5)$ та глобальну похибку $O(h^4)$. Він дозволяє знайти значення функції $y(x)$ з високою точністю, навіть для великих h, якщо функція $f(x, y)$ є гладкою.

Переваги методів Рунге-Кутта

– **Висока точність:** Методи забезпечують точні результати для невеликих кроків h.

– **Універсальність:** Можуть бути застосовані до широкого класу диференціальних рівнянь.

– **Відсутність похідних:** Для обчислень потрібна лише функція $f(x, y)$, без її похідних.

Приклад

Для рівняння

$$y' = -2xy, \quad y(0) = 1,$$

розглянемо чисельне розв'язання методом Рунге-Кутта 4-го порядку на відрізку $[0, 1]$ з кроком $h = 0.1$. Допоміжні значення k_1, k_2, k_3, k_4 на кожному кроці обчислюються за формулами:

$$k_1 = h \cdot f(x_n, y_n),$$

$$k_2 = h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right),$$

$$k_3 = h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right),$$

$$k_4 = h \cdot f(x_n + h, y_n + k_3),$$

після чого наближене значення y_{n+1} знаходиться як:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Методи Рунге-Кутта, зокрема 4-го порядку, широко використовуються у практичних задачах математичного моделювання, інженерії та фізики завдяки їх надійності, точності та простоті реалізації.

2.4 Блок-схеми алгоритмів методів

На рис. 2.1 можна побачити діграму роботи методу Рунге-Кутта.

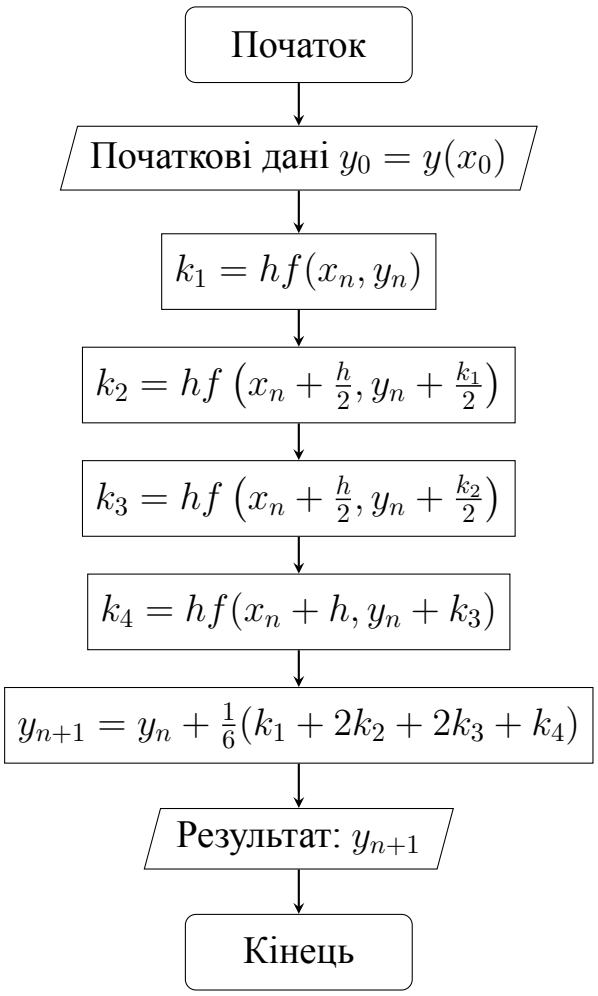

Рисунок 2.1 – діаграма роботи методу Рунге-Кутта

2.5 Верифікація розробленої програми

2.5.1 Python

Для забезпечення коректності результатів чисельного розв'язання задачі Коші важливо перевірити правильність вхідних даних. Було реалізовано функцію `validate_input`, яка виконує перевірку наступних аспектів: типи даних, діапазони значень та коректність роботи функцій.

Перевірка типів даних

На першому етапі перевіряється, чи вхідні функції $f_1(x, y, z)$, $f_2(x, y, z)$ та аналітичне розв'язання $y_{\text{анал.}}(x)$ є *викликаютьсями* (callable). Також перевіряється, чи числові параметри $(x_0, y_0, z_0, t_{\text{end}}, h)$ є дійсними числами (`int` або `float`). У разі невідповідності повертається помилка.

```
if not callable(f1) or not callable(f2) or not callable(y_analytical):
    raise ValueError("f1, f2, і y_analytical маютьбутифункціями .")
if not isinstance(x0, (int, float)) or not isinstance(y0, (int, float)) \
   or not isinstance(z0, (int, float)) or not isinstance(t_end, (int, float)) \
   or not isinstance(h, (int, float)):
    raise ValueError("x0, y0, z0, t_end і h маютьбутичислами .")
```

Ця перевірка запобігає помилкам, пов'язаним із некоректними типами вхідних даних.

Перевірка діапазонів значень

Далі перевіряється, чи значення $t_{\text{end}} > x_0$, а також чи $h > 0$ і чи $h \leqslant t_{\text{end}} - x_0$. Це забезпечує правильність математичної постановки задачі.

```
if t_end <= x0:
    raise ValueError("t_end маєбутибільшимза    x0.")
if h <= 0 or h > (t_end - x0):
```

```
    raise ValueError(”Крок h маєбутидодатнімінеперевищувати      t_end - x0.”)
```

Ці обмеження гарантують, що розрахунки виконуються на правильно заданому інтервалі з відповідним кроком.

Перевірка роботи функцій

На завершальному етапі функція виконує тестові виклики переданих функцій (f_1, f_2, $y_{\text{анал.}}$) із початковими значеннями x_0, y_0, z_0. Якщо функції повертають некоректні значення (наприклад, нечислові), виводиться відповідна помилка.

```
try:
    f1(x0, y0, z0)
    f2(x0, y0, z0)
    y_analytical(x0)
except Exception as e:
    raise ValueError(f”Помилка підчасвикликуфункцій   : {e}”)
```

Цей крок дозволяє виявити проблеми у логіці реалізації функцій.

Висновок

Функція `validate_input` забезпечує всебічну перевірку вхідних даних, включаючи їх типи, діапазони значень та коректність роботи. Це дозволяє уникнути некоректних результатів і робить алгоритм більш надійним. Реалізована перевірка є важливим етапом перед чисельним розв’язанням задачі Коші.

2.5.2 Octave

Для програми написаної на GNU Octave не передбачено перевірки на правильність введення.

2.6 Приклад роботи програми

2.6.1 Python

Після обробки початкових функцій, до тих на які можна застосовувати метод Рунге-Кутта отримуємо:

Результати роботи програми для варіанту 15 зображено на рисунку 2.2

2.6.2 Octave

Результати роботи програми для варіанту 15 зображено на рисунку 2.3

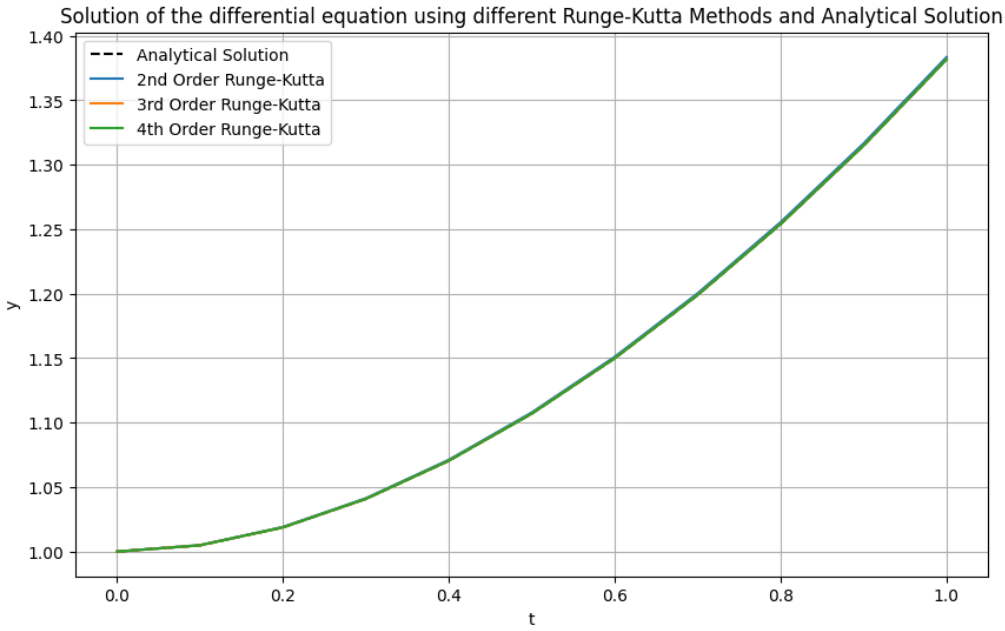

Рисунок 2.2 – графіки отримані при розв’язанні завдання мовою програмування Python

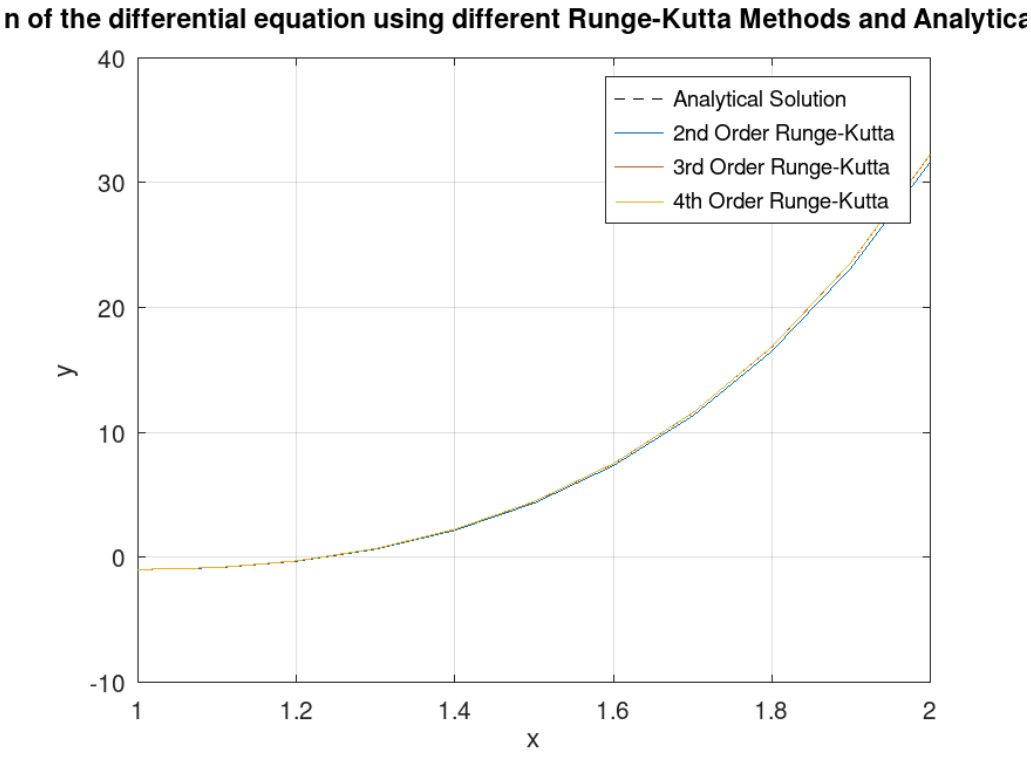

Рисунок 2.3 – графіки отримані при розв’язанні завдання мовою програмування Octave

ВИСНОВКИ

У даній лабораторній роботі було досліджено застосування методів Рунге-Кутти другого, третього та четвертого порядків для чисельного розв’язання диференціальних рівнянь. Було розглянуто аналітичний розв’язок для перевірки точності чисельних методів. У результаті виконання лабораторної роботи було досягнуто поставленої мети — реалізація та порівняння точності чисельних методів.

Під час виконання роботи були отримані такі результати:

1) **Помилки чисельних методів (Python):**

– Метод Рунге-Кутти другого порядку: **0.003519546436234152**

– Метод Рунге-Кутти третього порядку: **3.7709643942973794e-05**

– Метод Рунге-Кутти четвертого порядку: **1.5143067769622113e-06**

2) **Помилки чисельних методів (Octave):**

– Метод Рунге-Кутти другого порядку: **0.928995**

– Метод Рунге-Кутти третього порядку: **0.046178**

– Метод Рунге-Кутти четвертого порядку: **0.001835**

Аналізуючи результати, можна зробити висновок, що методи Рунге-Кутти вищих порядків (третій та четвертий) демонструють значно кращу точність у порівнянні з методом другого порядку. У Python точність була значно вищою, що свідчить про ефективність реалізації алгоритмів у цій середовищі. В Octave, хоча точність також зростає з підвищенням порядку методу, похибки вищі, що може бути пов’язано з внутрішніми обчислювальними особливостями середовища або з іншим вибором числових параметрів.

Таким чином, виконання лабораторної роботи продемонструвало ефективність методів Рунге-Кутти для чисельного розв’язання задач і підтвердило

теоретичні висновки про залежність точності від порядку методу. Мету роботи досягнуто.

ПЕРЕЛІК ПОСИЛАНЬ

1. Eaton, J. W., Bateman, D., Hauberg, S. & Wehbring, R. *GNU Octave 4.0 Reference Manual: Free Your Numbers* (Free Software Foundation, 2015). ISBN: 978-9888381050. https://docs.octave.org/octave-4.0.0.pdf.

2. Костюшко, І. А., Любашенко, Н. Д. & Третиник, В. В. *Методи обчислень* Бібліографія: с. 241–242, 243 (Вид-во «Політехніка», КПІ ім. Ігоря Сікорського, Київ, 2021).

Додаток А

Лістинги програм

А.1 Програма мовою програмування Python

Лістинг файлу main.py

```
import numpy as np
import matplotlib.pyplot as plt
import math

def validate_input(f1, f2, y_analytical, x0, y0, z0, t_end, h):
    try:
        # Перевірка, чи f1 і f2 є callable функціями()
        if not callable(f1) or not callable(f2):
            raise ValueError("f1 і f2 маютьбутифункціями  .")

        # Перевіркатипівпочатковихзначень

if not isinstance(x0, (int, float)) or not isinstance(y0, (int, float)) or not isinstance(
            raise ValueError("x0, y0 і z0 маютьбутичислами  .")

        # Перевіркатипу  t_end і h
        if not isinstance(t_end, (int, float)) or not isinstance(h, (int, float)):
            raise ValueError("t_end і h маютьбутичислами  .")

        # Перевіркадіапазонів  t_end і x0
        if t_end <= x0:
            raise ValueError("t_end маєбутибільшимза    x0.")

        # Перевірка, чи h додатнійінеперевищує    (t_end - x0)
        if h <= 0 or h > (t_end - x0):
            raise ValueError("Крок h маєбутидодатніміменшимаборівним
t_end - x0.")

        # Перевіркааналітичноїфункції
        if not callable(y_analytical):
            raise ValueError("y_analytical маєбутифункцією  .")

        # Тестовийвикликфункцій
        try:
            f1_result = f1(x0, y0, z0)
            f2_result = f2(x0, y0, z0)
            y_analytical_result = y_analytical(x0)
        except Exception as e:
            raise ValueError(f"Помилка підчасвикликуфункцій   : {e}")

        # Перевірка, чирезультатифункційчислові
        if not isinstance(f1_result, (int, float)):
            raise ValueError("f1 маєповертатичисловезначення   .")
        if not isinstance(f2_result, (int, float)):
            raise ValueError("f2 маєповертатичисловезначення   .")
        if not isinstance(y_analytical_result, (int, float)):
            raise ValueError("y_analytical маєповертатичисловезначення   .")

        print("Усі вхідніданікоректні  .")
        return True

    except ValueError as e:
        print(f"Помилка вхіднихданих : {e}")
        return False

def f1(x, y, z):
    return z
```

```
def f2(x, y, z):
    return x**2 - x + 2 - y

# Analytical solution for comparison
def y_analytical(x):
    result = (x-1) * x + np.sin(x) + np.cos(x)
    return result

x0 = 0
y0 = 1
z0 = 0
t_end = 1
h = 0.1


# Викликперевірки
validate_input(f1, f2, y_analytical, x0, y0, z0, t_end, h)

"""Основна частина"""

# Runge-Kutta methods for a system of ODEs
def runge_kutta_2nd_order(f1, f2, x0, y0, z0, h, x_end):
    n = int((x_end - x0) / h) + 1
    x = np.linspace(x0, x_end, n)
    y = np.zeros(n)
    z = np.zeros(n)

    y[0] = y0
    z[0] = z0

    for i in range(1, n):
        k1y = h * f1(x[i-1], y[i-1], z[i-1])
        k1z = h * f2(x[i-1], y[i-1], z[i-1])

        k2y = h * f1(x[i-1] + h, y[i-1] + k1y, z[i-1] + k1z)
        k2z = h * f2(x[i-1] + h, y[i-1] + k1y, z[i-1] + k1z)

        y[i] = y[i-1] + (k1y + k2y) / 2
        z[i] = z[i-1] + (k1z + k2z) / 2

    return x, y

def runge_kutta_3rd_order(f1, f2, x0, y0, z0, h, x_end):
    n = int((x_end - x0) / h) + 1
    x = np.linspace(x0, x_end, n)
    y = np.zeros(n)
    z = np.zeros(n)

    y[0] = y0
    z[0] = z0

    for i in range(1, n):
        k1y = h * f1(x[i-1], y[i-1], z[i-1])
        k1z = h * f2(x[i-1], y[i-1], z[i-1])

        k2y = h * f1(x[i-1] + h/2, y[i-1] + k1y/2, z[i-1] + k1z/2)
        k2z = h * f2(x[i-1] + h/2, y[i-1] + k1y/2, z[i-1] + k1z/2)

        k3y = h * f1(x[i-1] + h, y[i-1] - k1y + 2 * k2y, z[i-1] - k1z + 2 * k2z)
        k3z = h * f2(x[i-1] + h, y[i-1] - k1y + 2 * k2y, z[i-1] - k1z + 2 * k2z)

        y[i] = y[i-1] + (k1y + 4 * k2y + k3y) / 6
        z[i] = z[i-1] + (k1z + 4 * k2z + k3z) / 6

    return x, y

def runge_kutta_4th_order(f1, f2, x0, y0, z0, h, x_end):
    n = int((x_end - x0) / h) + 1
    x = np.linspace(x0, x_end, n)
    y = np.zeros(n)
    z = np.zeros(n)
```

```
    y[0] = y0
    z[0] = z0

    for i in range(1, n):
        k1y = h * f1(x[i-1], y[i-1], z[i-1])
        k1z = h * f2(x[i-1], y[i-1], z[i-1])

        k2y = h * f1(x[i-1] + h/2, y[i-1] + k1y/2, z[i-1] + k1z/2)
        k2z = h * f2(x[i-1] + h/2, y[i-1] + k1y/2, z[i-1] + k1z/2)

        k3y = h * f1(x[i-1] + h/2, y[i-1] + k2y/2, z[i-1] + k2z/2)
        k3z = h * f2(x[i-1] + h/2, y[i-1] + k2y/2, z[i-1] + k2z/2)

        k4y = h * f1(x[i-1] + h, y[i-1] + k3y, z[i-1] + k3z)
        k4z = h * f2(x[i-1] + h, y[i-1] + k3y, z[i-1] + k3z)

        y[i] = y[i-1] + (k1y + 2*k2y + 2*k3y + k4y) / 6
        z[i] = z[i-1] + (k1z + 2*k2z + 2*k3z + k4z) / 6

    return x, y


# Compute numerical and analytical solutions using different methods
methods = [runge_kutta_2nd_order, runge_kutta_3rd_order, runge_kutta_4th_order]
method_names = ['2nd Order Runge-Kutta', '3rd Order Runge-Kutta', '4th Order Runge-Kutta']
y_values_analytical = [y_analytical(t) for t in np.linspace(x0, t_end, int((t_end - x0) /

plt.figure(figsize=(10, 6))
plt.plot(np.linspace(x0, t_end, int((t_end - x0) / h) + 1), y_values_analytical, 'k--', la

for method, name in zip(methods, method_names):
    x_values, y_values = method(f1, f2, x0, y0, z0, h, t_end)
    plt.plot(x_values, y_values, label=name)

plt.xlabel('t')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.title("Solution of the differential equation using different Runge-Kutta Methods and A
plt.show()

# Calculate and output the error for each method
for method, name in zip(methods, method_names):
    x_values, y_values = method(f1, f2, x0, y0, z0, h, t_end)
    errors = np.abs(y_values - np.array(y_values_analytical))
    error_norm = np.linalg.norm(errors)
    print(f"Error for the {name}: {error_norm}")

def format_input(data):
    """Форматує вхідніданідлякомпактноговідображення    ."""
    formatted = []
    for value in data:
        if callable(value):
            formatted.append(value.__name__)  # Імя' функції
        else:
            formatted.append(repr(value))  # Представленняіншихзначень
    return ", ".join(formatted)

def run_tests():
    test_cases = [
        # 1. f1 i f2 неєфункціями
        {
            "input": [None, f2, y_analytical, x0, y0, z0, t_end, h],
            "expected_error": "f1 i f2 маютьбутифункціями  ."
        },
        {
            "input": [f1, None, y_analytical, x0, y0, z0, t_end, h],
            "expected_error": "f1 i f2 маютьбутифункціями  ."
        },
        # 2. x0, y0, z0 неєчислами
        {
            "input": [f1, f2, y_analytical, "0", y0, z0, t_end, h],
            "expected_error": "x0, y0 i z0 маютьбутичислами  ."
```

```
        },
        {
            "input": [f1, f2, y_analytical, x0, "1", z0, t_end, h],
            "expected_error": "x0, y0 і z0 маютьбутичислами  ."
        },
        {
            "input": [f1, f2, y_analytical, x0, y0, "z", t_end, h],
            "expected_error": "x0, y0 і z0 маютьбутичислами  ."
        },
        # 3. t_end і h неєчислами
        {
            "input": [f1, f2, y_analytical, x0, y0, z0, "1", h],
            "expected_error": "t_end і h маютьбутичислами  ."
        },
        {
            "input": [f1, f2, y_analytical, x0, y0, z0, t_end, "0.1"],
            "expected_error": "t_end і h маютьбутичислами  ."
        },
        # 4. t_end <= x0
        {
            "input": [f1, f2, y_analytical, x0, y0, z0, -1, h],
            "expected_error": "t_end маєбутибільшимза    x0."
        },
        # 5. h <= 0 або h > (t_end - x0)
        {
            "input": [f1, f2, y_analytical, x0, y0, z0, t_end, 0],
            "expected_error": "Крок h маєбутидодатніміменшимаборівним
t_end - x0."
        },
        {
            "input": [f1, f2, y_analytical, x0, y0, z0, t_end, 2],
            "expected_error": "Крок h маєбутидодатніміменшимаборівним
t_end - x0."
        },
        # 6. y_analytical неєфункцією
        {
            "input": [f1, f2, None, x0, y0, z0, t_end, h],
            "expected_error": "y_analytical маєбутифункцією  ."
        },
        # 7. Помилкивикликуфункцій
        {
            "input": [f1, f2, lambda x: x / 0, x0, y0, z0, t_end, h],
            "expected_error": "Помилка підчасвикликуфункцій   "
        },
        # 8. f1 або f2 повертаютьнечисловезначення
        {
            "input": [lambda x, y, z: "not a number", f2, y_analytical, x0, y0, z0, t_end,
            "expected_error": "f1 маєповертатичисловезначення   ."
        },
        {
            "input": [f1, lambda x, y, z: None, y_analytical, x0, y0, z0, t_end, h],
            "expected_error": "f2 маєповертатичисловезначення   ."
        },
        # 9. y_analytical повертаєнечисловезначення
        {
            "input": [f1, f2, lambda x: "not a number", x0, y0, z0, t_end, h],
            "expected_error": "y_analytical маєповертатичисловезначення   ."
        }
    ]

    for i, test_case in enumerate(test_cases, 1):
        print("\n" + "=" * 50)
        print(f"Тест {i}")
        print(f"Вхідні дані: {format_input(test_case['input'])}")
        print("-" * 50)
        try:
            result = validate_input(*test_case["input"])
            if result:
                print(f"Результат: Тестпровалено . Очікуваласьпомилка
: '{test_case['expected_error']}', алепомилокнебуло   .")
        except ValueError as e:
            error_message = str(e)
            print(f"Результат: {error_message}")
            if test_case["expected_error"] in error_message:
```

```
                print(f"Тест успішний.")
            else:
                print(f"Тест провалено. Очікуваласьпомилка
: '{test_case['expected_error']}', алеотримано : '{error_message}'.")
        print("=" * 50)

run_tests()
```

А.2 Програма мовою програмування Octave

```
function main()
  % Main script to solve the differential equation
  x0 = 1;
  y0 = -1;
  z0 = 0;
  x_end = 2;
  h = 0.1;

  methods = {@runge_kutta_2nd_order, @runge_kutta_3rd_order, @runge_kutta_4th_order};
  method_names = {'2nd Order Runge-Kutta', '3rd Order Runge-Kutta','4th Order Runge-Kutta

  x_values_analytical = linspace(x0, x_end, (x_end - x0) / h + 1);
  y_values_analytical = arrayfun(@main_y_analytical, x_values_analytical);

  figure;
  plot(x_values_analytical, y_values_analytical, 'k--', 'DisplayName', 'Analytical Solutio
  hold on;

  for i = 1:length(methods)
    [x_values, y_values] = methods{i}(@main_f1, @main_f2, x0, y0, z0, h, x_end);
    plot(x_values, y_values, 'DisplayName', method_names{i});
  end

  xlabel('x');
  ylabel('y');
  legend;
  grid on;
  title('Solution of the differential equation using different Runge-Kutta Methods and Ana
  saveas(gcf, 'runge_kutta_comparison.png');

  for i = 1:length(methods)
    [x_values, y_values] = methods{i}(@main_f1, @main_f2, x0, y0, z0, h, x_end);
    errors = abs(y_values - y_values_analytical);
    error_norm = norm(errors);
    fprintf('Error for %s: %f\n', method_names{i}, error_norm);
  end
end

function dzdt = main_f1(x, y, z)
  dzdt = z;
end

function dydt = main_f2(x, y, z)
  dydt = 2 * exp(e) + 2 * z;
end

function result = main_y_analytical(x)
  result = 0.5 * (exp(e) * (1 - 2 * x) + exp(2 * x + e - 2) - 2);
end

function [x_values, y_values] = runge_kutta_2nd_order(f1, f2, x0, y0, z0, h, x_end)
  n = (x_end - x0) / h + 1;
  x = linspace(x0, x_end, n);
```

```
  y = zeros(1, n);
  z = zeros(1, n);
  y(1) = y0;
  z(1) = z0;

  for i = 2:n
    k1y = h * f1(x(i-1), y(i-1), z(i-1));
    k1z = h * f2(x(i-1), y(i-1), z(i-1));
    k2y = h * f1(x(i-1) + h, y(i-1) + k1y, z(i-1) + k1z);
    k2z = h * f2(x(i-1) + h, y(i-1) + k1y, z(i-1) + k1z);
    y(i) = y(i-1) + (k1y + k2y) / 2;
    z(i) = z(i-1) + (k1z + k2z) / 2;
  end

  x_values = x;
  y_values = y;
end

function [x_values, y_values] = runge_kutta_3rd_order(f1, f2, x0, y0, z0, h, x_end)
  n = (x_end - x0) / h + 1;
  x = linspace(x0, x_end, n);
  y = zeros(1, n);
  z = zeros(1, n);
  y(1) = y0;
  z(1) = z0;

  for i = 2:n
    k1y = h * f1(x(i-1), y(i-1), z(i-1));
    k1z = h * f2(x(i-1), y(i-1), z(i-1));
    k2y = h * f1(x(i-1) + h/2, y(i-1) + k1y/2, z(i-1) + k1z/2);
    k2z = h * f2(x(i-1) + h/2, y(i-1) + k1y/2, z(i-1) + k1z/2);
    k3y = h * f1(x(i-1) + h, y(i-1) - k1y + 2 * k2y, z(i-1) - k1z + 2 * k2z);
    k3z = h * f2(x(i-1) + h, y(i-1) - k1y + 2 * k2y, z(i-1) - k1z + 2 * k2z);
    y(i) = y(i-1) + (k1y + 4 * k2y + k3y) / 6;
    z(i) = z(i-1) + (k1z + 4 * k2z + k3z) / 6;
  end

  x_values = x;
  y_values = y;
end

function [x_values, y_values] = runge_kutta_4th_order(f1, f2, x0, y0, z0, h, x_end)
  n = (x_end - x0) / h + 1;
  x = linspace(x0, x_end, n);
  y = zeros(1, n);
  z = zeros(1, n);
  y(1) = y0;
  z(1) = z0;

  for i = 2:n
    k1y = h * f1(x(i-1), y(i-1), z(i-1));
    k1z = h * f2(x(i-1), y(i-1), z(i-1));
    k2y = h * f1(x(i-1) + h/2, y(i-1) + k1y/2, z(i-1) + k1z/2);
    k2z = h * f2(x(i-1) + h/2, y(i-1) + k1y/2, z(i-1) + k1z/2);
    k3y = h * f1(x(i-1) + h/2, y(i-1) + k2y/2, z(i-1) + k2z/2);
    k3z = h * f2(x(i-1) + h/2, y(i-1) + k2y/2, z(i-1) + k2z/2);
    k4y = h * f1(x(i-1) + h, y(i-1) + k3y, z(i-1) + k3z);
    k4z = h * f2(x(i-1) + h, y(i-1) + k3y, z(i-1) + k3z);
    y(i) = y(i-1) + (k1y + 2*k2y + 2*k3y + k4y) / 6;
    z(i) = z(i-1) + (k1z + 2*k2z + 2*k3z + k4z) / 6;
  end

  x_values = x;
  y_values = y;
end
```