Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота № 8

з дисципліни «Спеціальні розділи математики-2.
Чисельні методи»

Виконав:

студент гр. ІС-34

Колосов Ігор

Викладач:

доц. Рибачук Л.В.

Київ – 2024

Тема: Розв'язання задачі Коші

1. Постановка задачі

Методами Рунге-Кутта та Адамса розв'язати задачу Коші. На початку інтервалу у необхідній кількості точок значення для методу Адамса визначити методом Рунге-Кутта четвертого порядку.

Для фіксованого h потрібно навести:

- значення наближеного розв'язку $y(x)$ у тих самих точках, одержані обома методами;
- значення функції помилки $\varepsilon(x)$ для обох методів;
- графіки:
  - обох наближених – на одному малюнку;
  - обох помилок – на другому малюнку.

Розв'язати задане рівняння за допомогою Matchad, порівняти із власними результатами.

Розв'язати за допомогою Matchad систему рівнянь, побудувати графік y_0 та фазовий портрет системи $u^{<2>}(u^{<1>})$, зробити висновки щодо стійкості системи.

6 - 10	$y' = e^{-ax}(y^2 + b),\ a = 1{,}0 + 0{,}4n,\ n = №вар - 5,\ b = 1{,}0 + 0{,}4k,\ k = №вар - 5$

Взяти крок h = 0,1. Якщо вимоги на величину τ (див. метод Рунге-Кутта) для даного кроку не виконано, подрібнити крок. Початкові умови $y(0)=0$. Відрізок, що розглядається: [0; 1].

Розв'язати за допомогою Mathcad систему диференціальних рівнянь:

$$\begin{cases} y_0' = y_1 \\ y_1' = -y_0 + \dfrac{(k-10)}{10} \cdot y_1 \end{cases}$$

де $y_0(0) = 0.1$, $y_1(0) = 0$, k – № варіанту, тобто № у списку групи.

2. Вихідне рівняння

$$f(x,y) := \exp(-3 \cdot x)\left(y^2 + 3\right)$$

3. Значення наближеного розв'язку y(x) у тих самих точках одержані обома методами (Mathcad та власне рішення)

$M := 10 \quad start := 0$

$y0 := 0.1 \quad end := 1$

$f(x, y) := \exp(-3 \cdot x)\left(y^2 + 3\right) \qquad stepsize := \frac{end - start}{M} = 0.1$

u := rkfixed(y0, start, end, M, f)

u =

	0	1
0	0	0.1
1	0.1	0.364
2	0.2	0.57
3	0.3	0.733
4	0.4	0.861
5	0.5	0.96
6	0.6	1.037
7	0.7	1.096
8	0.8	1.141
9	0.9	1.175
10	1	1.201

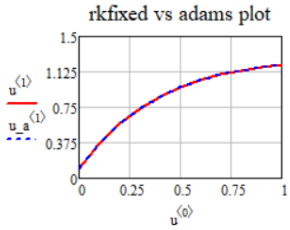
u_a := Adams(y0, start, end, M, f)

u_a =

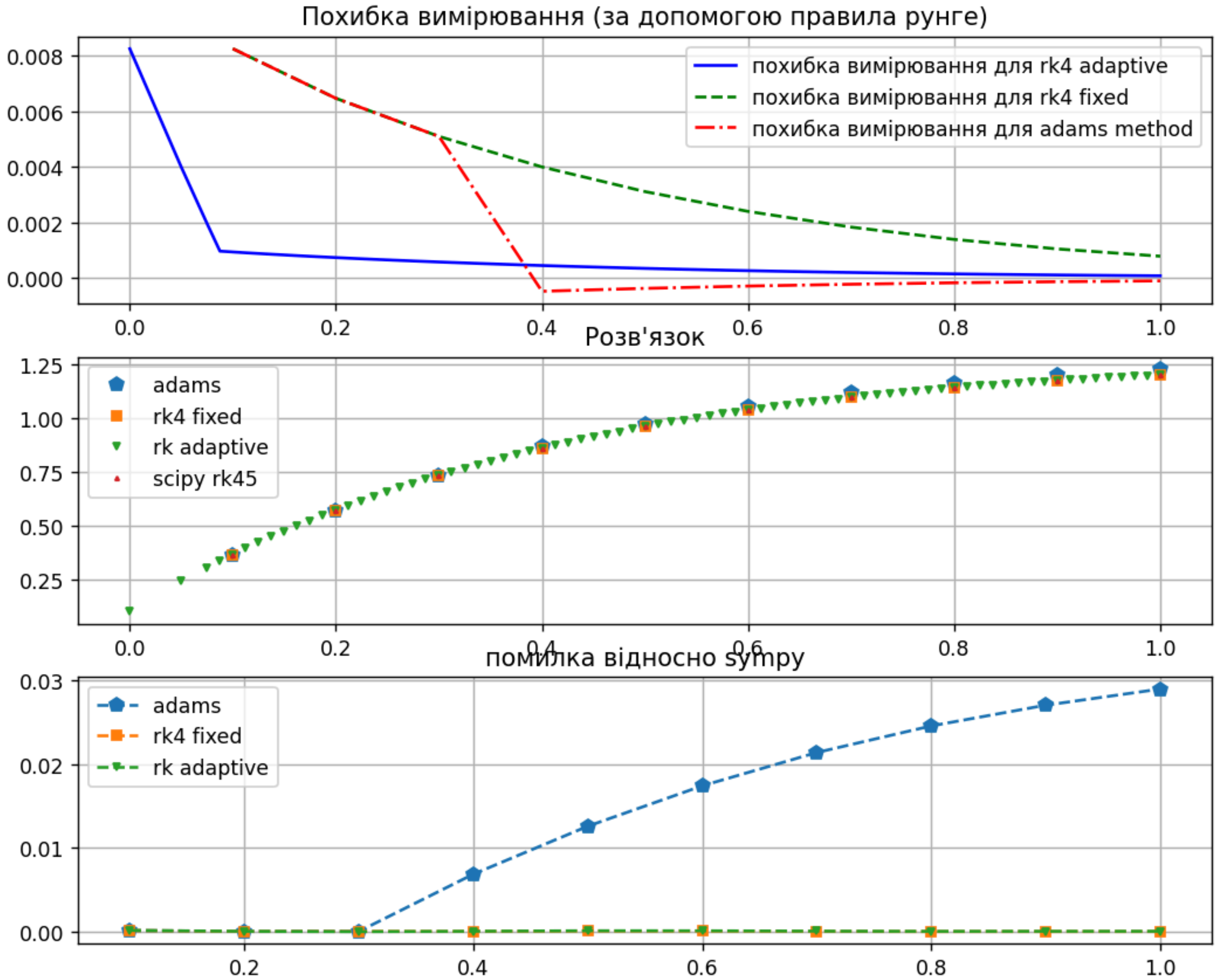
	0	1
0	0	0.1
1	0.1	0.364
2	0.2	0.57
3	0.3	0.733
4	0.4	0.861
5	0.5	0.96
6	0.6	1.037
7	0.7	1.096
8	0.8	1.141
9	0.9	1.175
10	1	1.201

Програмне рішення:

Time	rk_fixed_x	rk_adapt_x	adams	scipy rk45
0.10	0.3643	0.3643	0.3643	0.3645
0.20	0.5705	0.5705	0.5705	0.5705
0.30	0.7329	0.7329	0.7329	0.7329
0.40	0.8606	0.8606	0.8675	0.8606
0.50	0.9602	0.9602	0.9729	0.9603
0.60	1.0373	1.0373	1.0548	1.0374
0.70	1.0964	1.0964	1.1178	1.0964
0.80	1.1414	1.1413	1.1660	1.1414
0.90	1.1754	1.1754	1.2025	1.1754
1.00	1.2010	1.2010	1.2300	1.2010



4. Значення функції помилки для методів



5. Система диференціальних рівнянь: значення та фазовий портрет

$$D(t,y) := \begin{pmatrix} y_1 \\ -y_0 + \frac{k-10}{10}\cdot y_1 \end{pmatrix}$$

steps := 10
start := 0
end := 1

k := 10

$y0 := \begin{pmatrix} 0.1 \\ 0 \end{pmatrix}$

$stepsize := \frac{(end - start)}{steps} = 0.1$

Програмне рішення:

us := rkfixed(y0, start, end, steps, D)

us =

	0	1	2
0	0	0.1	0
1	0.1	0.1	$-9.983\cdot10^{-3}$
2	0.2	0.098	-0.02
3	0.3	0.096	-0.03
4	0.4	0.092	-0.039
5	0.5	0.088	-0.048
6	0.6	0.083	-0.056
7	0.7	0.076	-0.064
8	0.8	0.07	-0.072
9	0.9	0.062	-0.078
10	1	0.054	-0.084

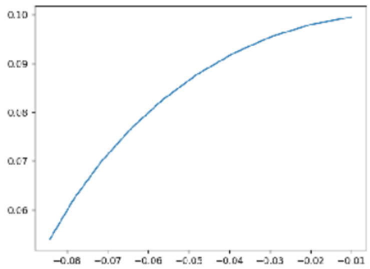
us_a := Adams(y0, start, end, steps, D)

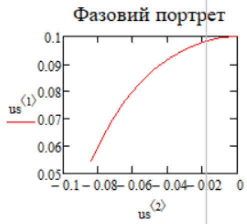
us_a =

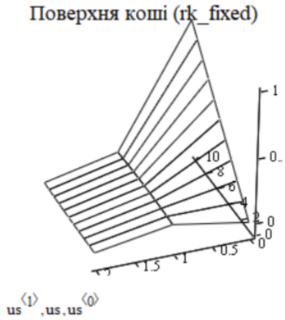
	0	1	2
0	0	0.1	0
1	0.1	0.1	$-9.983\cdot10^{-3}$
2	0.2	0.098	-0.02
3	0.3	0.096	-0.03
4	0.4	0.092	-0.039
5	0.5	0.088	-0.048
6	0.6	0.083	-0.056
7	0.7	0.076	-0.064
8	0.8	0.07	-0.072
9	0.9	0.062	-0.078
10	1	0.054	-0.084

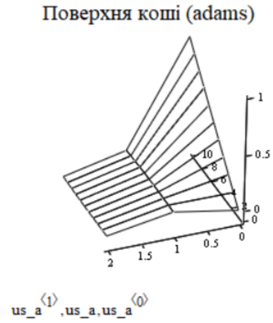
```
- RK4 FIXED STEPSIZE
 Time | y1      | y2
---------------------------------
 0.10 | 0.09950 | -0.00998
 0.20 | 0.09801 | -0.01987
 0.30 | 0.09553 | -0.02955
 0.40 | 0.09211 | -0.03894
 0.50 | 0.08776 | -0.04794
 0.60 | 0.08253 | -0.05646
 0.70 | 0.07648 | -0.06442
 0.80 | 0.06967 | -0.07174
 0.90 | 0.06216 | -0.07833
 1.00 | 0.05403 | -0.08415
```

власне рішення - фазовий портрет









6. Лістинг програми

```
import numpy as np
from typing import Callable
import warnings
warnings.filterwarnings("ignore")


def get_closest_values(source_times, source_values, target_times):
    closest_values = []
    for target_time in target_times:
        closest_idx = np.argmin(np.abs(source_times - target_time))
        closest_values.append(source_values[closest_idx])
    return np.array(closest_values)



def rk4_fixed(f, t0, x0, dt, total_steps,  tau_upper = 0.01):
    t = t0
    x = x0
    t_last = t0 + total_steps * dt

    t_hist = []
    x_hist = []
    errors = []

    current_step = 0
```

```
    while t < t_last:
        current_step += 1
        k1 = dt * f(t,           x)

        k2 = dt * f(t + dt / 2, x + k1 / 2)
        k3 = dt * f(t + dt / 2, x + k2 / 2)
        k4 = dt * f(t + dt,      x + k3)
        cur_step_size_x = x + (k1 + 2 * k2 + 2 * k3 + k4) / 6

        k2 = dt * f(t + dt / 4, x + k1 / 4)
        k3 = dt * f(t + dt / 4, x + k2 / 4)
        k4 = dt * f(t + dt / 2,      x + k3 / 2)
        smaller_step_size = x + (k1 + 2 * k2 + 2 * k3 + k4) / 12
        analytical_error = (cur_step_size_x - smaller_step_size) / (2**4 -
1)

        t += dt
        tau = np.max(np.abs(k2 - k3) / (k1 -  k2 + 1e-15))
        x = cur_step_size_x

        if tau > tau_upper:
            pass
            #print(f"(rk4 fixed): tau: {tau:3.5f} at t = {t:3.3f}")

        errors.append(analytical_error)
        t_hist.append(t)
        x_hist.append(x)
    return np.array(t_hist), np.array(x_hist), np.array(errors)


def rk4_adaptive(f, t0, x0, dt, total_steps, tau_upper = 0.01,  tau_lower
= 0.0001):
    t = t0
    x = x0
    t_last = t0 + total_steps * dt + dt


    t_hist = []
    x_hist = []
    errors = []
    steps_current = 0
    while t <= t_last :
        steps_current += 1
        k1 = dt * f(t,           x)

        # current step size s
```

```
        k2 = dt * f(t + dt / 2, x + k1 / 2)
        k3 = dt * f(t + dt / 2, x + k2 / 2)
        k4 = dt * f(t + dt,     x + k3)
        cur_step_size_x = x + (k1 + 2 * k2 + 2 * k3 + k4) / 6

        # double smaller
        k2 = dt * f(t + dt / 4, x + k1 / 4)
        k3 = dt * f(t + dt / 4, x + k2 / 4)
        k4 = dt * f(t + dt / 2,     x + k3 / 2)
        smaller_step_size = x + (k1 + 2 * k2 + 2 * k3 + k4) / 12

        # double bigger
        k2 = dt * f(t + dt,     x + k1)
        k3 = dt * f(t + dt,     x + k2)
        k4 = dt * f(t + dt * 2, x + k3 * 2)
        bigger_step_size = x + (k1 + 2 * k2 + 2 * k3 + k4) / 3

        x_new = cur_step_size_x
        tau = np.max(np.abs(k2 - k3) / (k1 -  k2 + 1e-15))

        analytical_error = (cur_step_size_x - smaller_step_size) / (2**4 -
1)

        if tau < tau_lower:
            dt *= 2
            x_new = bigger_step_size

        elif tau > tau_upper:
            dt /= 2
            x_new = smaller_step_size

        t_hist.append(t)
        x_hist.append(x)
        errors.append(analytical_error)
        x = x_new
        t += dt


    return np.array(t_hist), np.array(x_hist), np.array(errors)


def adams_method(f, x0, y0, dt, n):
    h = dt
    xf = n * dt + x0
    n = int(n)
    x = np.linspace(x0, xf, n + 1)
```

```
    y = np.zeros(n + 1)
    y[0] = y0

    for i in range(3):
        k1 = h * f(x[i], y[i])
        k2 = h * f(x[i] + h/2, y[i] + k1/2)
        k3 = h * f(x[i] + h/2, y[i] + k2/2)
        k4 = h * f(x[i] + h, y[i] + k3)
        y[i+1] = y[i] + (k1 + 2*k2 + 2*k3 + k4) / 6


    for i in range(3, n):
        y_pred = y[i] + h/24 * (55*f(x[i], y[i]) - 59*f(x[i-1], y[i-1]) +
                                37*f(x[i-2], y[i-2]) - 9*f(x[i-3],
y[i-3]))

        y[i+1] = y[i] + h/24 * (9*f(x[i+1], y_pred) + 19*f(x[i], y[i]) -
                                5*f(x[i-1], y[i-1]) + f(x[i-2], y[i-2]))
    return x, y


def adams(f, t, x, dt, steps):
    t_hist = []; x_hist = []; errors = []
    t_last = t0 + steps * dt
    t = t0
    x = x0
    current_step = 0

    for i in range(3):
        current_step += 1
        k1 = dt * f(t,          x)
        k2 = dt * f(t + dt / 2, x + k1 / 2)
        k3 = dt * f(t + dt / 2, x + k2 / 2)
        k4 = dt * f(t + dt,     x + k3)
        cur_step_size_x = x + (k1 + 2 * k2 + 2 * k3 + k4) / 6

        k2 = dt * f(t + dt / 4, x + k1 / 4)
        k3 = dt * f(t + dt / 4, x + k2 / 4)
        k4 = dt * f(t + dt / 2,     x + k3 / 2)
        smaller_step_size = x + (k1 + 2 * k2 + 2 * k3 + k4) / 12
        # runge rule
        analytical_error = (cur_step_size_x - smaller_step_size) / (2**4 -
1)
        t += dt
        x = cur_step_size_x
```

```
        errors.append(analytical_error)

        t_hist.append(t)
        x_hist.append(x)


    while t <= t_last:
        current_step+= 1
        extr = x + h/24 * (55 * f(t, x) - 59 * f(t_hist[-1], x_hist[-1]) +
37 * f(t_hist[-2], x_hist[-2]) - 9 * f(t_hist[-3], x_hist[-3]))
        inter = x + h/24 * (9 * f(t + dt, extr) + 19 * f(t, x) - 5 *
f(t_hist[-1], x_hist[-1]) + f(t_hist[-2], x_hist[-2]))

        cur_step = inter

        extr = x + h/24 * (55 * f(t, x) - 59 * f(t_hist[-1], x_hist[-1]) +
37 * f(t_hist[-2], x_hist[-2]) - 9 * f(t_hist[-3], x_hist[-3]))
        #                              dt / 2
        inter = x + h/24 * (9 * f(t + dt / 2, extr) + 19 * f(t, x) - 5 *
f(t_hist[-1], x_hist[-1]) + f(t_hist[-2], x_hist[-2]))

        smaller_step = inter
        analytical_error = (cur_step - smaller_step) / (2**4 - 1)
        """
        Наведені формули мають достатньо велику точність. Вони мають
похибку порядку ( O(h^4),
            але самі формули оцінки похибки достатньо складні, тому
використовують більш просте та
                загальне правило Рунге.
        """

        x = cur_step
        t += dt
        errors.append(analytical_error)
        t_hist.append(t)
        x_hist.append(x)

    return np.array(t_hist), np.array(x_hist), np.array(errors)



def f(x: float, y: float) -> float:
    a = 1.0 + 0.4 * (10 - 5)
    b = 1.0 + 0.4 * (10 - 5)
    return np.exp(-a*x) * (y**2 + b)
```

```
# Взяти крок h = 0,1.  Якщо вимоги на величину τ (див. метод Рунге-Кутта) для даного кроку не виконано, подрібнити крок. Початкові умови y(0)=0. Відрізок, що розглядається: [0; 1]

import matplotlib.pyplot as plt

# причому tau не повинно перевищувати декількох сотих, інакше крок потрібно зменшити.
t0, x0 = 0.0, 0.1
y0 = xf = x0
stepsize = dt = h = 0.1
last_t = 1.0
n = steps = int(last_t // stepsize)

rk_adapt_times, rk_adapt_x, errors_adapt = rk4_adaptive(f, t0, x0, dt, steps, 0.01)
rk_fixed_times, rk_fixed_x, errors_fixed = rk4_fixed(f, t0, x0, dt, steps, 0.01)
adams_times, adams_x, errors_adams = adams(f, t0, x0, dt, steps)

from scipy.integrate import solve_ivp
t_eval = np.arange(t0+dt, last_t+dt, dt)
sol = solve_ivp(f, [t0, last_t], [x0], method='RK45', t_eval=t_eval)
scipy_rk45_times = sol.t
scipy_rk45_x = sol.y[0]


interpolated_adapt_x = get_closest_values(rk_adapt_times, rk_adapt_x, rk_fixed_times)
interpolated_adapt_errors = get_closest_values(rk_adapt_times, errors_adapt, rk_fixed_times)
indices_to_print = np.linspace(0, len(rk_fixed_times)-1, 10, dtype=int)

print(f"  {'Time':<4} | {'rk_fixed_x':<10} | {'rk_adapt_x':<10} | {'adams':<10} | {'scipy rk45':<10}")
print(f"{'-' * (10+15+15+15 + 10 + 15 + 10)}")
for idx in indices_to_print:
    print(f"  {rk_fixed_times[idx]:<3.2f} | {rk_fixed_x[idx]:<10.4f} | {interpolated_adapt_x[idx]:<10.4f} | {adams_x[idx]:<10.4f} | {scipy_rk45_x[idx]:<10.4f}")


print(f"\n\n  {'Time':<4} | {'rk_fixed error':<10} | {'rk_adapt error':<15} | {'adams error':<15}")
print(f"{'-' * (10+15+15+15 + 10 + 15 + 10)}")
for idx in indices_to_print:
```

```
    print(f"  {rk_fixed_times[idx]:<3.2f} | {errors_fixed[idx]:<15.6f} |
{interpolated_adapt_errors[idx]:<15.6f} | {errors_adams[idx]:<15.6f}")


fig, ax = plt.subplots(3, 1, figsize=(10, 18))
ax[0].plot(rk_adapt_times, errors_adapt, label="похибка вимірювання для
rk4 adaptive", linestyle='-', color='blue')
ax[0].plot(rk_fixed_times, errors_fixed, label="похибка вимірювання для
rk4 fixed", linestyle='--', color='green')
ax[0].plot(adams_times, errors_adams, label="похибка вимірювання для adams
method", linestyle='-.', color='red')
ax[0].set_title("Похибка вимірювання (за допомогою правила рунге)")
ax[0].legend(loc='best', fontsize=10)
ax[0].grid(True)

ax[1].plot(adams_times, adams_x, marker="p", linestyle="", markersize=7,
alpha=1, label="adams")
ax[1].plot(rk_fixed_times, rk_fixed_x, marker="s", linestyle="",
markersize=5, alpha=1, label="rk4 fixed")
ax[1].plot(rk_adapt_times, rk_adapt_x, marker="v", linestyle="",
markersize=3, alpha=1, label="rk adaptive")
ax[1].plot(scipy_rk45_times, scipy_rk45_x, marker="^", linestyle="",
markersize=2, alpha=1, label="scipy rk45")
ax[1].set_title("Розв'язок")
ax[1].legend(loc='best', fontsize=10)
ax[1].grid(True)

diff_adams = np.abs(adams_x - scipy_rk45_x[:len(adams_x)])
diff_rk_adapt = np.abs(interpolated_adapt_x -
scipy_rk45_x[:len(interpolated_adapt_x)])
diff_rk_fixed = np.abs(rk_fixed_x - scipy_rk45_x[:len(rk_fixed_x)])

ax[2].plot(adams_times, diff_adams, marker="p", linestyle="--",
markersize=7, alpha=1, label="adams")
ax[2].plot(rk_fixed_times, diff_rk_fixed, marker="s", linestyle="--",
markersize=5, alpha=1, label="rk4 fixed")
ax[2].plot(rk_fixed_times, diff_rk_adapt, marker="v", linestyle="--",
markersize=3, alpha=1, label="rk adaptive")
ax[2].set_title("помилка відносно sympy")
ax[2].legend(loc='best', fontsize=10)
ax[2].grid(True)
plt.show()



# Розв’язати за допомогою Mathcad систему диференціальних рівнянь
def f2(x, y: np.ndarray) -> np.ndarray:
```

```
    y0, y1 = y
    return np.array([y1, -y0 + (10 - 10)/10 * y1])

t = 0.0
last_t = 1.0
steps = 9
step = 0.1


x = np.array([0.1, 0.0])
times, solution, errors = rk4_fixed(f2, t, x, step, steps)


print("\n\n - RK4 FIXED STEPSIZE")
print(f"  {'Time':<4} | {'y1':<7} | {'y2':<10}")
print(f"{'-' * (10+15+15+15 + 10 + 15 + 10)}")
for idx in range(len(times)):
    print(f"  {times[idx]:<3.2f} | {solution[:, 0][idx]:<5.5f} |
{solution[:, 1][idx]:<5.5f}")


plt.plot(solution[:, 1], solution[:, 0])
plt.show()
```