Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки
Кафедра обчислювальної техніки

Лабораторна робота № 8

з дисципліни «Чисельні методи»

на тему

“Розв’язання задачі Коші”

Виконав:
студент гр. ІП-93
Завальнюк Максим

Викладач:
доц. Рибачук Л.В.

Київ – 2021

Зміст

Зміст 2
1 Постановка задачі 3
2 Розв’язок 4
3 Розв’язок за допомогою NumPy 6
4 Лістинг програми 8

1 Постановка задачі

Методами Рунге-Кутта та Адамса розв'язати задачу Коші. Для фіксованого h потрібно навести: значення наближеного розв'язку y(x) у тих самих точках, одержані обома методами, значення функції помилки ε(x) для обох методів, графіки обох наближених - на одному малюнку, обох помилок - на другому малюнку. Розв'язати задане рівняння за допомогою NumPy, порівняти із власними результатами. Розв'язати за допомогою NumPy систему рівнянь, побудувати графік y^{0} та фазовий портрет системи ($u^{<2>}$) $u^{<1>}$, зробити висновки щодо стійкості системи.

2 Розв’язок

Вихідне рівняння: $y^{`} = e^{-3x}(y^2 + 3)$

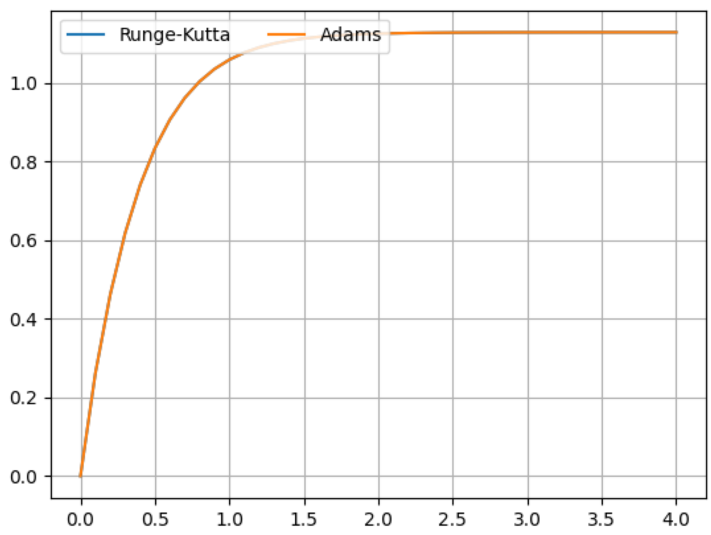
########## Runge-kutta method ##########

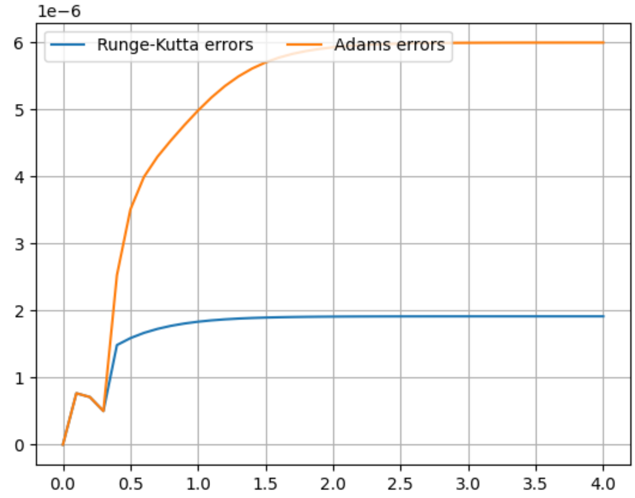
	x	y	Error
0	0	0	0
1	0.1	0.261146	7.66843e-07
2	0.2	0.461697	1.12847e-06
3	0.3	0.617817	1.34199e-06
4	0.4	0.739389	1.48552e-06
5	0.5	0.833555	1.58934e-06
6	0.6	0.905971	1.66731e-06
7	0.7	0.961262	1.72681e-06
8	0.8	1.00321	1.77242e-06
9	0.9	1.03486	1.80731e-06
10	1	1.05864	1.8339e-06
11	1.1	1.07644	1.85405e-06
12	1.2	1.08974	1.86927e-06
13	1.3	1.09965	1.8807e-06
14	1.4	1.10703	1.88927e-06
15	1.5	1.11251	1.89567e-06
16	1.6	1.11658	1.90044e-06
17	1.7	1.1196	1.90399e-06
18	1.8	1.12185	1.90664e-06
19	1.9	1.12351	1.9086e-06
20	2	1.12474	1.91005e-06
21	2.1	1.12565	1.91113e-06
22	2.2	1.12633	1.91194e-06
23	2.3	1.12683	1.91253e-06
24	2.4	1.1272	1.91297e-06
25	2.5	1.12748	1.9133e-06
26	2.6	1.12768	1.91354e-06
27	2.7	1.12784	1.91372e-06
28	2.8	1.12795	1.91385e-06
29	2.9	1.12803	1.91395e-06
30	3	1.12809	1.91402e-06
31	3.1	1.12814	1.91408e-06
32	3.2	1.12817	1.91412e-06
33	3.3	1.1282	1.91415e-06
34	3.4	1.12821	1.91417e-06
35	3.5	1.12823	1.91419e-06
36	3.6	1.12824	1.9142e-06
37	3.7	1.12825	1.91421e-06
38	3.8	1.12825	1.91421e-06
39	3.9	1.12826	1.91422e-06
40	4	1.12826	1.91422e-06

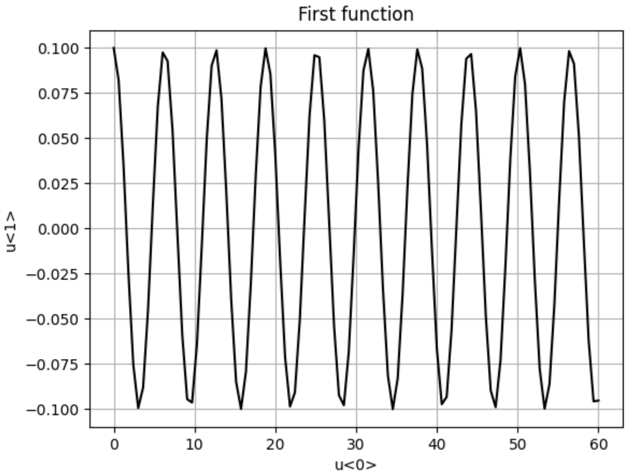
########## Adams method ##########

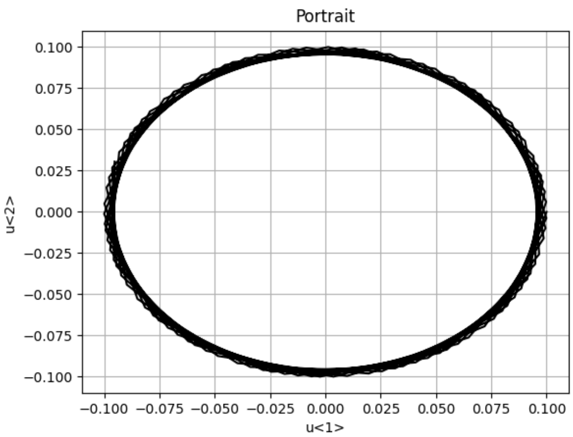
	x	y	Error
0	0	0	0
1	0.1	0.261146	7.66843e-07
2	0.2	0.461697	7.10012e-07
3	0.3	0.617817	5.01771e-07

4	0.4	0.73942	2.52777e-06
5	0.5	0.8336	3.5088e-06
6	0.6	0.906023	3.99331e-06
7	0.7	0.961319	4.29401e-06
8	0.8	1.00327	4.53847e-06
9	0.9	1.03492	4.76683e-06
10	1	1.05871	4.98253e-06
11	1.1	1.07651	5.17879e-06
12	1.2	1.08981	5.34964e-06
13	1.3	1.09973	5.49274e-06
14	1.4	1.1071	5.60899e-06
15	1.5	1.11259	5.70124e-06
16	1.6	1.11666	5.77317e-06
17	1.7	1.11968	5.82853e-06
18	1.8	1.12193	5.87071e-06
19	1.9	1.12359	5.90262e-06
20	2	1.12482	5.92663e-06
21	2.1	1.12574	5.94462e-06
22	2.2	1.12641	5.95806e-06
23	2.3	1.12691	5.96808e-06
24	2.4	1.12729	5.97554e-06
25	2.5	1.12756	5.98109e-06
26	2.6	1.12777	5.9852e-06
27	2.7	1.12792	5.98826e-06
28	2.8	1.12803	5.99053e-06
29	2.9	1.12811	5.99221e-06
30	3	1.12817	5.99346e-06
31	3.1	1.12822	5.99438e-06
32	3.2	1.12825	5.99507e-06
33	3.3	1.12828	5.99557e-06
34	3.4	1.1283	5.99595e-06
35	3.5	1.12831	5.99623e-06
36	3.6	1.12832	5.99643e-06
37	3.7	1.12833	5.99659e-06
38	3.8	1.12833	5.9967e-06
39	3.9	1.12834	5.99678e-06
40	4	1.12834	5.99685e-06









Як видно з графіків, тип коренів системи є чисто уявним, а особливою точкою є центр.

3 Розв’язок за допомогою NumPy

Нижче наведено розв’язок системи у NumPy:

```
results = odeint(dfunction, first_y, x_axis)
```

Результат:

########## SciPy solution ##########

	x	y
0	0	0
1	0.1	0.261134
2	0.2	0.461679
3	0.3	0.617795
4	0.4	0.739365
5	0.5	0.833529
6	0.6	0.905944
7	0.7	0.961234
8	0.8	1.00318
9	0.9	1.03483
10	1	1.05861
11	1.1	1.07641
12	1.2	1.08971
13	1.3	1.09962
14	1.4	1.107
15	1.5	1.11248
16	1.6	1.11655
17	1.7	1.11957
18	1.8	1.12181
19	1.9	1.12348
20	2	1.12471
21	2.1	1.12562
22	2.2	1.1263
23	2.3	1.1268
24	2.4	1.12717
25	2.5	1.12745
26	2.6	1.12765
27	2.7	1.1278
28	2.8	1.12792
29	2.9	1.128
30	3	1.12806
31	3.1	1.12811
32	3.2	1.12814
33	3.3	1.12817
34	3.4	1.12818
35	3.5	1.1282
36	3.6	1.12821
37	3.7	1.12822
38	3.8	1.12822
39	3.9	1.12823
40	4	1.12823

4 Лістинг програми

```
from tabulate import tabulate
import pandas as pd
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt
from string import Template
from math import e

# Constants
n = 5
a = b = 1 + 0.4 * n
interval = [0, 4]
h = 0.1
x0 = y0 = 0
epsilon = 10 ** (-1)
template = Template('#' * 10 + ' $string ' + '#' * 10)


def dfunction(y: float, x: float) -> float:
    """
    Main diff function
    :param x: x-argument
    :param y: y-argument
    :return: result
    """
    return e ** (-a * x) * (y ** 2 + b)


def system_function(y: list, x: float) -> list:
    """
    Main diff system
    :param y: y-argument
    :param x: x-argument
    :return: list of results, tow arguments
    """
    k = 10
    return [y[1], ((k - 10) / 10 * y[1]) - y[0]]


def show_plot(x_axis_1: list, y_axis_1: list, x_axis_2: list, y_axis_2: list,
labels: list) -> None:
    """
    Function for showing plot
    :param x_axis_1: x-values for the first function
    :param y_axis_1: y-values for the first function
    :param x_axis_2: x-values for the second function
    :param y_axis_2: y-values for the second function
    :param labels: labels on a plot
    :return: nothing to return
    """
    fig, ax = plt.subplots()
    ax.plot(x_axis_1, y_axis_1, label=labels[0])
    ax.plot(x_axis_2, y_axis_2, label=labels[1])
    ax.legend(loc='upper left', ncol=2)
    plt.grid()
    plt.show()


def show_plot_for_system(x_axis: list, y_axis: list, labels: list) -> None:
    """
    Function for showing plots for system
    :param x_axis: x-values of the function
    :param y_axis: y-values of the function
    :param labels: labels on a plot
```

```
    :return: nothing to return
    """
    plt.title(labels[2])
    plt.xlabel(labels[0])
    plt.ylabel(labels[1])
    plt.grid()
    plt.plot(x_axis, y_axis, 'k')
    plt.show()


def runge_kutte_method(limits: list, h_value: float, epsilon_value: float,
first_x: float, first_y: float) -> list:
    """
    Implementation of the Runge-Kutta method
    :param limits: limits of x-values
    :param h_value: step
    :param epsilon_value: value for controlling the fault
    :param first_x: known x-value
    :param first_y: known y-value
    :return: list with x and y values
    """
    results = []
    results.append([first_x, first_y, 0])
    current_x = first_x
    current_y = first_y
    while current_x < limits[1]:
        k1 = h_value * dfunction(current_y, current_x)
        k2 = h_value * dfunction(current_y + k1 / 2, current_x + h_value / 2)
        k3 = h_value * dfunction(current_y + k2 / 2, current_x + h_value / 2)
        k4 = h_value * dfunction(current_y + k3, current_x + h_value)
        delta_y = (1.0 / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
        current_x += h_value
        current_y += delta_y
        fault = abs((k2 - k3) / (k1 - k2))
        if fault > epsilon_value:
            h_value /= 2
        results.append([current_x, current_y])
    return results


def adams_method(limits: list, h_value: float, epsilon_value: float,
runge_kutta_results: list) -> list:
    """
    Implementation of the Adams method
    :param limits: limits of x-values
    :param h_value: step
    :param epsilon_value: alue for controlling the fault
    :param runge_kutta_results: known results from the previous method
    :return: list with x and y values
    """
    index = 3
    step_value = h_value
    number_of_steps = ((limits[1] - limits[0]) / h_value)
    while index < number_of_steps:
        k1 = dfunction(runge_kutta_results[index][1],
runge_kutta_results[index][0])
        k2 = dfunction(runge_kutta_results[index - 1][1],
runge_kutta_results[index - 1][0])
        k3 = dfunction(runge_kutta_results[index - 2][1],
runge_kutta_results[index - 2][0])
        k4 = dfunction(runge_kutta_results[index - 3][1],
runge_kutta_results[index - 3][0])
        extra_y = runge_kutta_results[index][1] + h_value / 24 * (55 * k1 - 59 *
k2 + 37 * k3 - 9 * k4)
        next_x = runge_kutta_results[index][0] + step_value
        intra_y = runge_kutta_results[index][1] + h_value / 24 * (9 *
```

```
dfunction(extra_y, next_x) + 19 * k1 - 5 * k2 + k3)
        fault = abs(intra_y - extra_y)
        if fault > epsilon_value:
            step_value / 2
        if extra_y == intra_y:
            runge_kutta_results.append([next_x, extra_y])
        else:
            runge_kutta_results.append([next_x, intra_y])
        index += 1
    return runge_kutta_results


def search_error_for_runge_kutta(runge_kutta_results: list, h_value: float) ->
list:
    """
    Function for calculating errors for Runge-Kutta method
    :param runge_kutta_results: results from this method
    :param h_value: step
    :return: list with errors
    """
    errors = []
    for index in range(len(runge_kutta_results) - 1):
        k1 = dfunction(runge_kutta_results[index][1],
runge_kutta_results[index][0])
        k2 = dfunction(runge_kutta_results[index][1],
runge_kutta_results[index][0] + h_value / 2)
        k3 = dfunction(runge_kutta_results[index][1],
runge_kutta_results[index][0] + h_value / 2)
        k4 = dfunction(runge_kutta_results[index][1],
runge_kutta_results[index][0] + h_value)
        delta_y = (k1 + 2 * k2 + 2 * k3 + k4) / 6
        right_part = (runge_kutta_results[index + 1][1] -
runge_kutta_results[index][1]) / h_value
        error = delta_y - right_part
        errors.append(error)
    return errors


def search_error_for_adams(adams_results: list, adams_results_less: list) ->
list:
    """
    Function for calculating errors for Adams method
    :param adams_results: results from this method
    :param adams_results_less: results from this method with divided step
    :return: list with errors
    """
    errors = []
    for index in range(len(adams_results)):
        error = (adams_results[index][1] - adams_results_less[index * 2][1]) /
(16 - 1)
        errors.append(error)
    return errors


def print_table(table: list, headers: tuple) -> None:
    """
    Function for printing the table
    :param table: values
    :param headers: headers of the table
    :return: nothing
    """
    dataframe = pd.DataFrame(table)
    format_style = 'github'
    print(tabulate(dataframe, headers=headers, tablefmt=format_style))
```

```
def scipy_solver(x_axis: list, first_y: float) -> list:
    """
    Solve the diff equation with SciPy
    :param x_axis: x-values
    :param first_y: known y-value
    :return: list with results
    """
    results = odeint(dfunction, first_y, x_axis)
    return results


def system_solver():
    y_axis = [0.1, 0]
    x_axis = np.linspace(0, 60, 100)
    results = odeint(system_function, y_axis, x_axis)
    first_y_results = results.transpose()[0]
    second_y_results = results.transpose()[1]
    show_plot_for_system(x_axis, first_y_results, ['u<0>', 'u<1>', 'First
function'])
    show_plot_for_system(x_axis, second_y_results, ['u<0>', 'u<2>', 'Second
function'])
    show_plot_for_system(first_y_results, second_y_results, ['u<1>', 'u<2>',
'Portrait'])


print(template.substitute(string='Runge-kutta method'))
runge_kutta_results = runge_kutte_method(interval, h, epsilon, x0, y0)
runge_kutta_results_less = runge_kutte_method(interval, h / 2, epsilon, x0, y0)
runge_kutta_errors = search_error_for_adams(runge_kutta_results,
runge_kutta_results_less)
for index in range(1, len(runge_kutta_errors)):
    runge_kutta_results[index].append(abs(runge_kutta_errors[index]))
print_table(runge_kutta_results, ('x', 'y', 'Error'))
print(template.substitute(string='Adams method'))
adams_results = adams_method(interval, h, epsilon, runge_kutta_results[:4])
adams_results_less = adams_method(interval, h / 2, epsilon,
runge_kutta_results_less[:4])
adams_errors = search_error_for_adams(adams_results, adams_results_less)
for index in range(len(adams_errors)):
    if index < 4:
        adams_results[index][2] = abs(adams_errors[index])
    else:
        adams_results[index].append(abs(adams_errors[index]))
print_table(adams_results, ('x', 'y', 'Error'))
x_axis = np.arange(interval[0], interval[1] + 0.1, h)
scipy_results = scipy_solver(x_axis, y0)
print(template.substitute(string='SciPy solution'))
print_table([[x_axis[i], scipy_results[i]] for i in range(len(scipy_results))],
('x', 'y'))
show_plot([el[0] for el in runge_kutta_results], [el[1] for el in
runge_kutta_results], [el[0] for el in adams_results], [el[1] for el in
adams_results], ['Runge-Kutta', 'Adams'])
show_plot([el[0] for el in runge_kutta_results], [el[2] for el in
runge_kutta_results], [el[0] for el in adams_results], [el[2] for el in
adams_results], ['Runge-Kutta errors', 'Adams errors'])
system_solver()
```