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](#_Toc71578848)

[1 Постановка задачі 3](#_Toc71578849)

[2 Розв’язок 4](#_Toc71578850)

[3 Розв’язок за допомогою NumPy 6](#_Toc71578851)

[4 Лістинг програми 8](#_Toc71578852)

### 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 та фазовий портрет системи ( ) , зробити висновки щодо стійкості системи.

### 2 Розв’язок

Вихідне рівняння:

########## 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 |

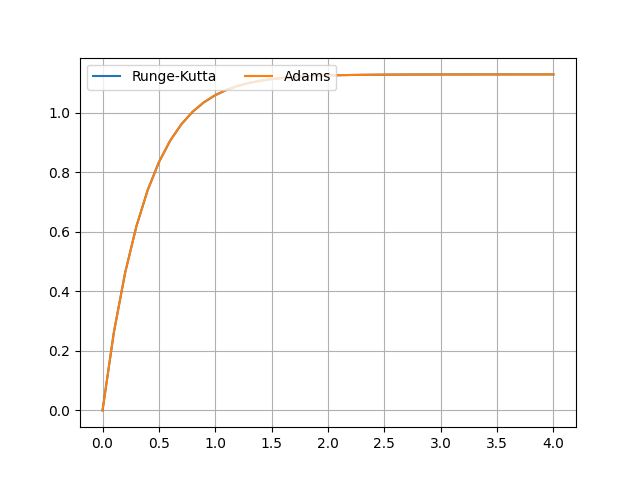
| 36 | 3.6 | 1.12832 | 5.99643e-06 |

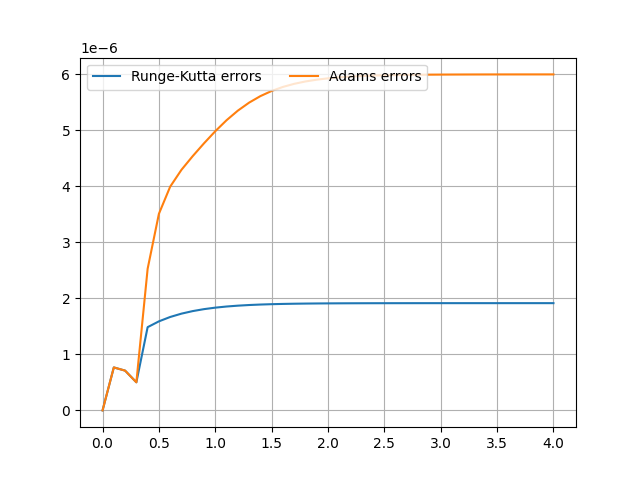
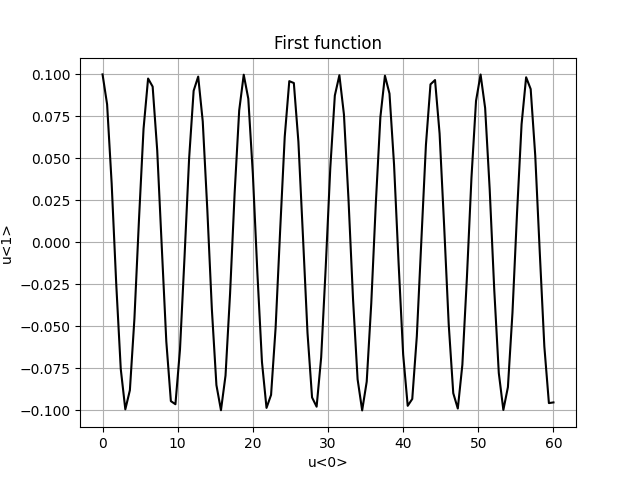
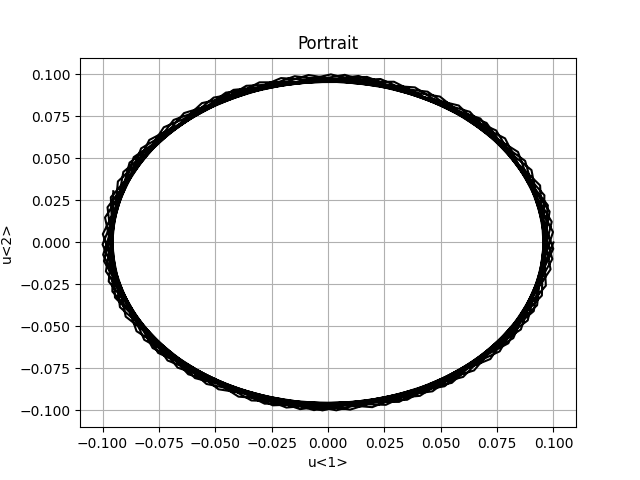
| 37 | 3.7 | 1.12833 | 5.99659e-06 |

| 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()