Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

ЗВІТ

Про виконання лабораторної роботи №1

З дисципліни «Комп’ютерне моделювання та оптимізація»

**Моделювання стаціонарних систем**

TI-92 Черноусова Дениса

Перевірив проф. д.т.н. Шушура О. М.

Київ – 2021



g = 3

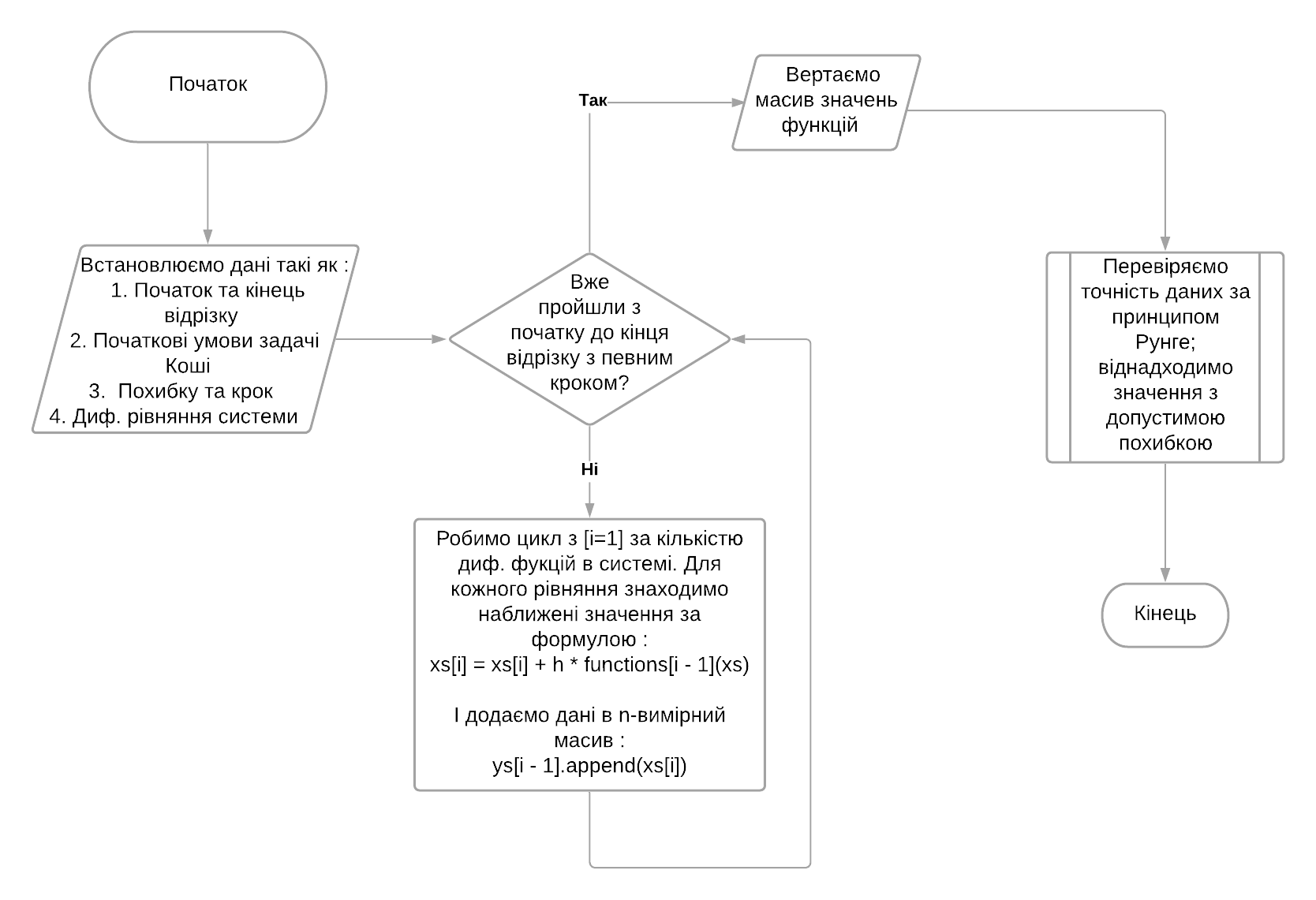
k = 2

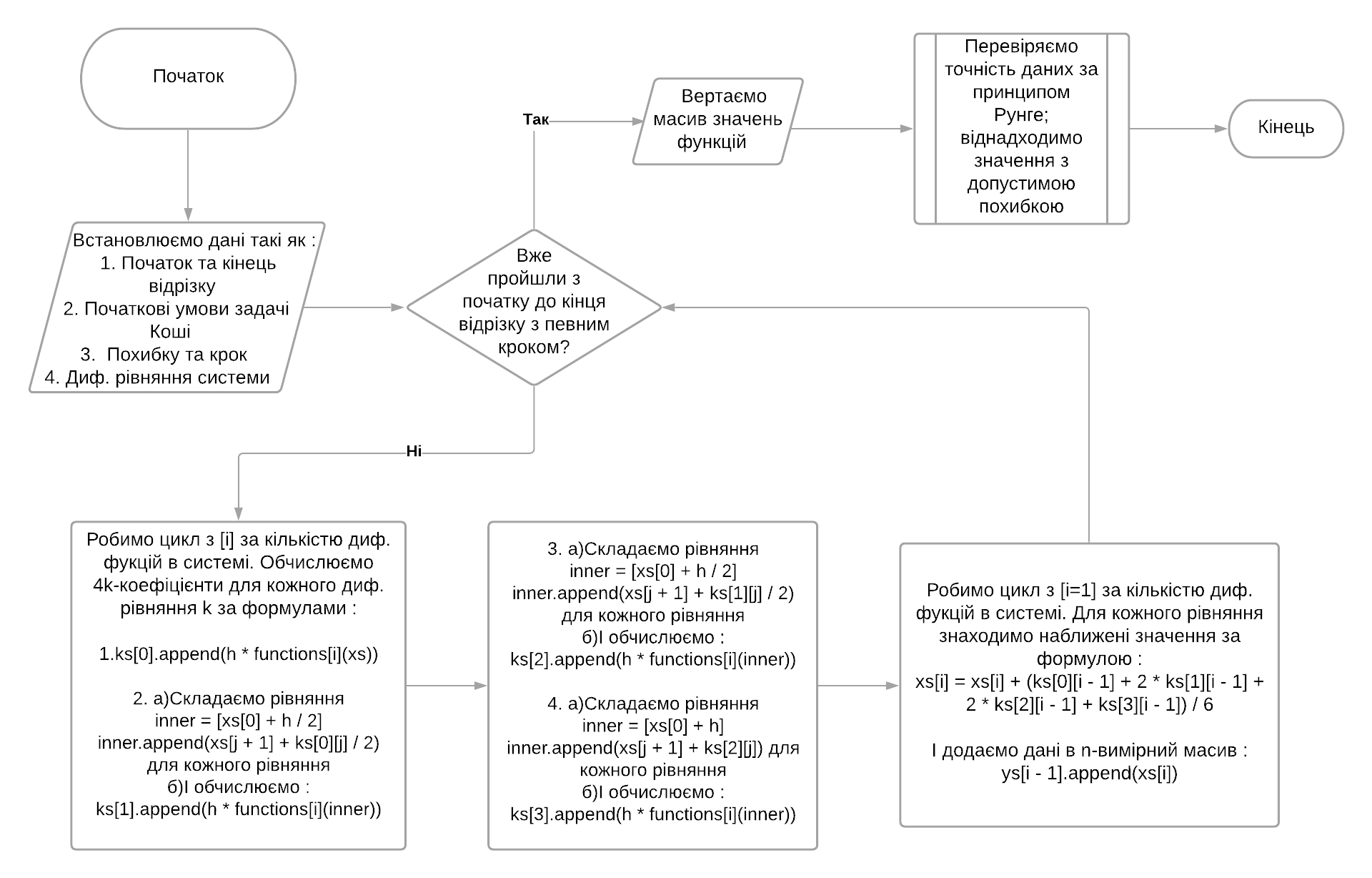
g – остання цифра у номері студентського квитка, а k – передостання.

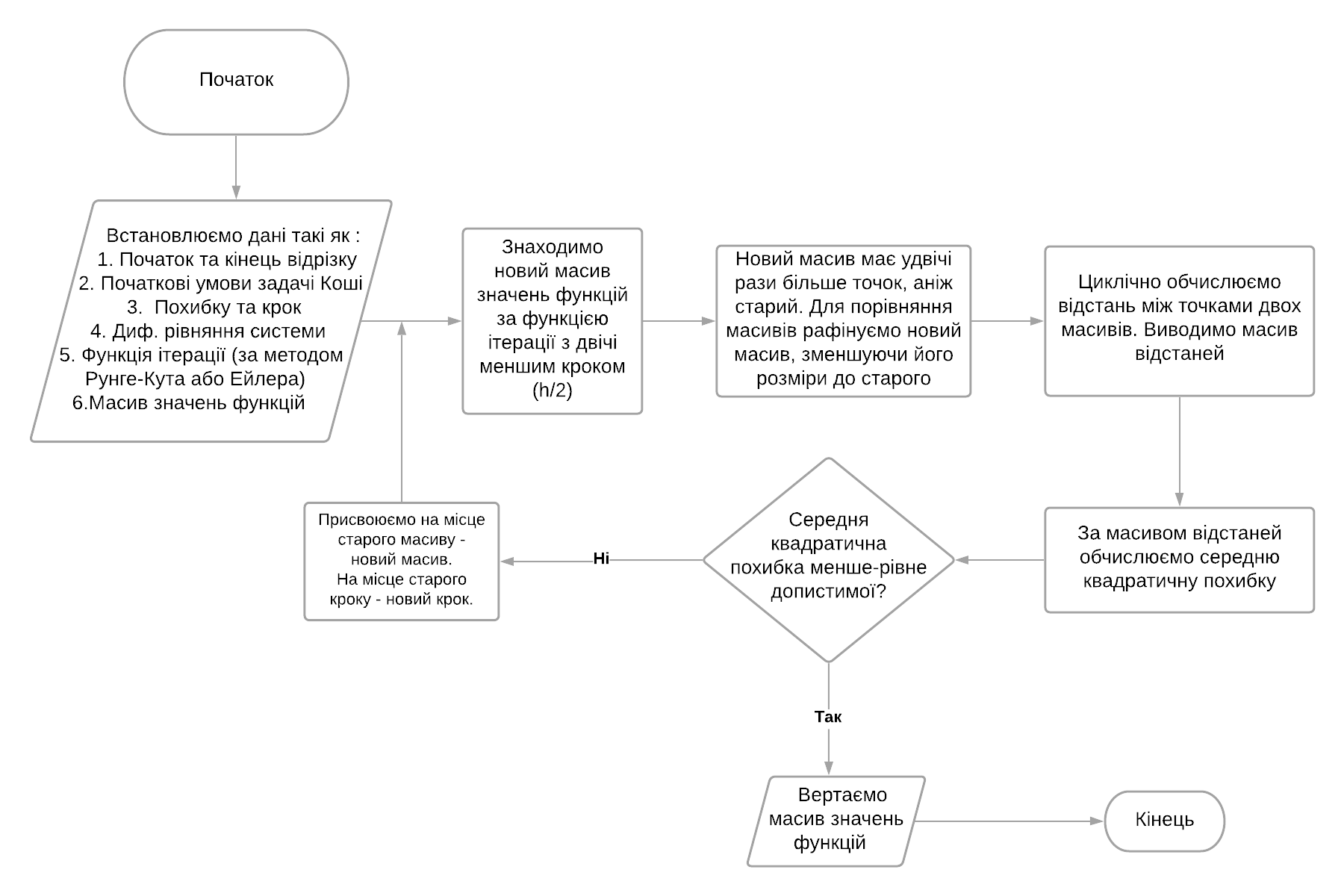
Розробити алгоритми та програмне забезпечення для розв’язку наведеної задачі Коші методам Ейлера та методом Рунге-Кута 4-го порядку точності. Алгоритми представити у вигляді блок-схем або діаграм діяльності UML. Програмне забезпечення розробити на будь-якій сучасній мові програмування. Середньоквадратична загальна точність пошуку має дорівнювати 0,1. Порівняти розв’язки задачі, знайдені вказанами вище методами. Знайти точний розв’язок задачі та порівняти з отриманими результатами.

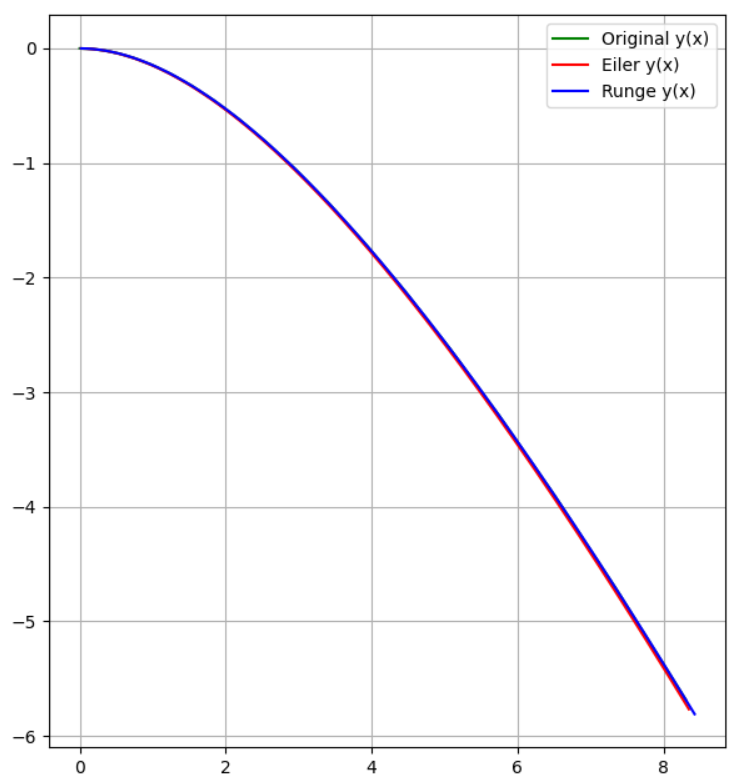
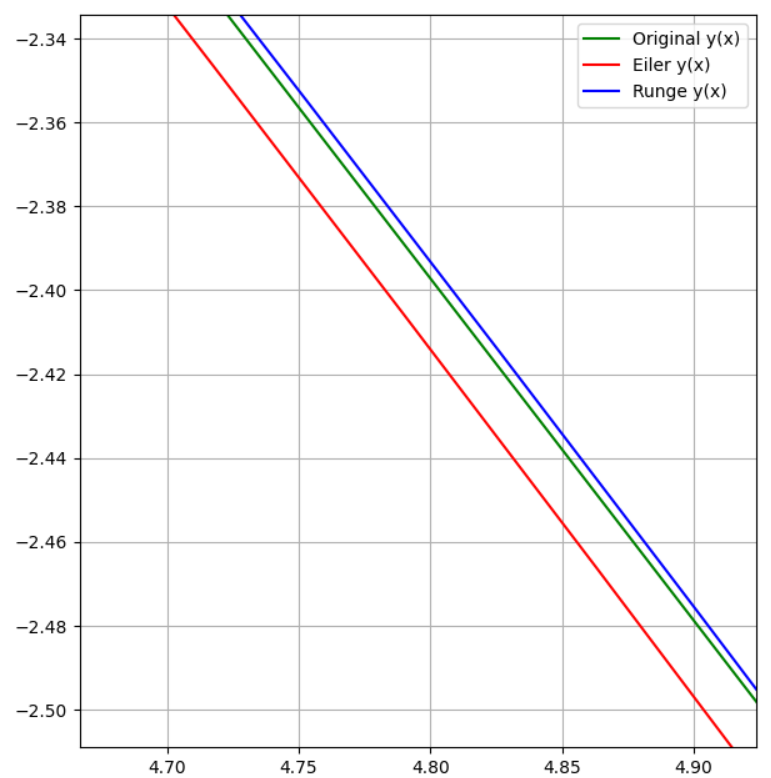
Початкові умови: *x(0)=0, y(0)=0,* на відрізку [0,1]

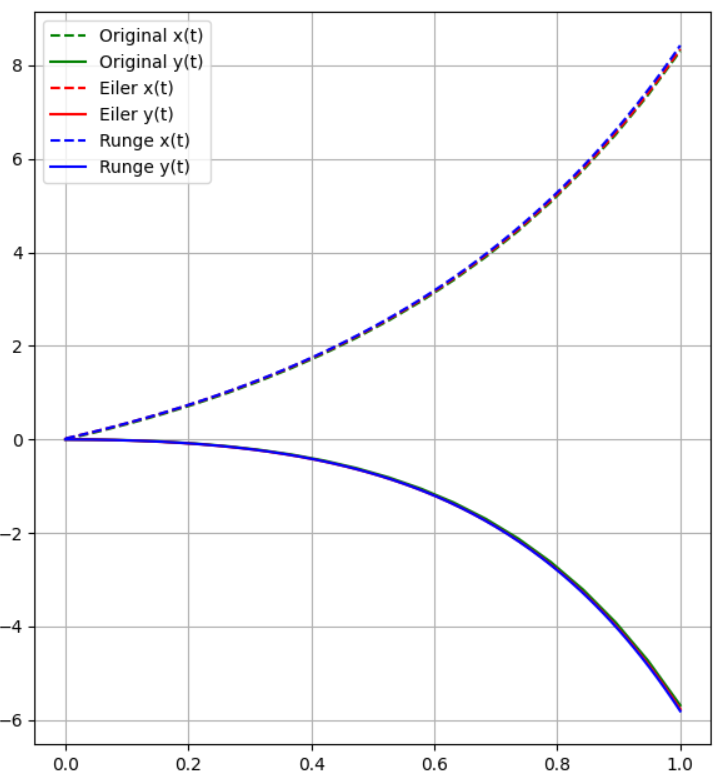
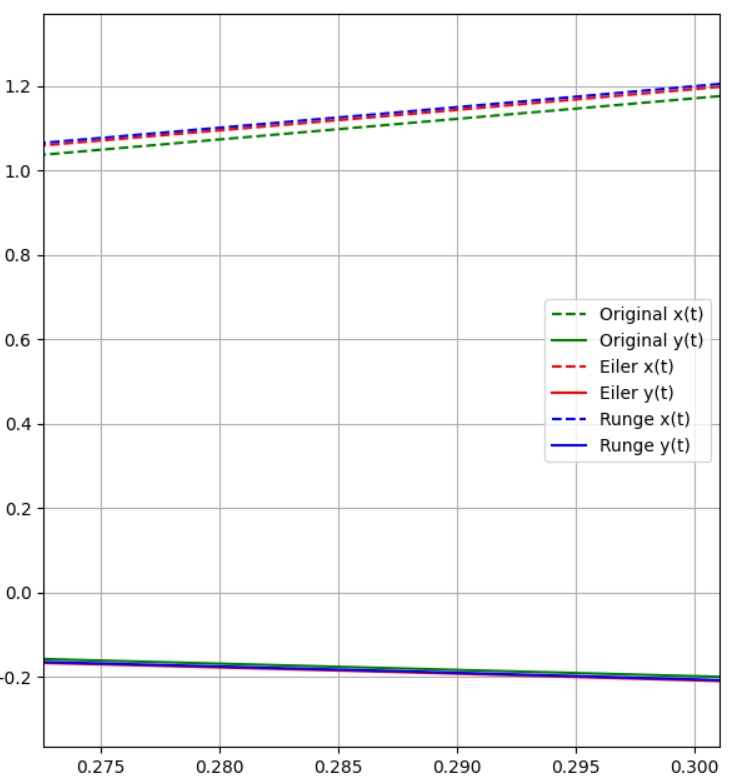
**Алгоритми**

****Метод Ейлера

Метод Рунге-Кутта

Принцип Рунге

**Результати роботи**



**Лістинг**

Main.py

import numpy as np  
from Eiler import \*  
from Runge\_Kutt import Runge\_Kutt  
from system\_of\_functions import \*  
from scipy.integrate import odeint  
  
  
Eiler = eiler\_method([x\_der, y\_der], 0, 0, 0, 0.2, 0, 1)  
  
print("Метод Ейлера\n", np.matrix(Eiler))  
  
Runge = Runge\_Kutt([x\_der, y\_der], 0, 0, 0, 0.2, 0, 1)  
  
print("Метод Рунге-Кута\n", np.matrix(Runge))  
  
t = np.linspace(0, 1, 20)  
xs = [0, 0]  
original = odeint(F, xs, t)  
  
print("Original\n", np.matrix(original))  
  
fig, (ax1, ax2) = plt.subplots(1, 2)  
  
  
ax1.plot(original[:, 0], original[:, 1], 'g-', label="Original y(x)")  
ax1.plot(Eiler[0], Eiler[1], 'r-', label="Eiler y(x)")  
ax1.plot(Runge[0], Runge[1], 'b-', label="Runge y(x)")  
  
ax2.plot(t, original[:, 0], 'g--', label="Original x(t)")  
ax2.plot(t, original[:, 1], 'g-', label="Original y(t)")  
  
EilerT = np.linspace(0, 1, len(Eiler[0]))  
  
ax2.plot(EilerT, Eiler[0], 'r--', label="Eiler x(t)")  
ax2.plot(EilerT, Eiler[1], 'r-', label="Eiler y(t)")  
  
RungeT = np.linspace(0, 1, len(Runge[0]))  
ax2.plot(RungeT, Runge[0], 'b--', label="Runge x(t)")  
ax2.plot(RungeT, Runge[1], 'b-', label="Runge y(t)")  
  
ax1.legend()  
ax2.legend()  
ax1.grid()  
ax2.grid()  
plt.show()

Eiler.py

import numpy as np  
  
from runge\_principle import runge\_principle  
import matplotlib.pyplot as plt  
  
  
def eiler\_method(functions, t, x, y, h, start, end):  
 ys = iteration(functions, t, x, y, h, start, end)  
  
 return runge\_principle(ys, iteration(functions, t, x, y, h / 2, start, end), iteration, functions, t, x, y, h / 2,  
 start, end)  
  
  
def iteration(functions, t, x, y, h, start, end):  
 xs = [t, x, y]  
 ys = [[], []]  
 times = (end - start) / h  
 for l in range(int(times) + 1):  
  
 for i in range(1, len(functions) + 1):  
 xs[i] = xs[i] + h \* functions[i - 1](xs)  
 ys[i - 1].append(xs[i])  
  
 xs[0] += h  
  
 return ys

runge\_princeple.py

import matplotlib.pyplot as plt  
  
ERROR = 0.1  
  
  
def runge\_principle(ys\_old, ys\_new, iteration, functions, t, x, y, h, start, end):  
 print("H:", h)  
 ys\_refined = refine\_array\_over2(len(ys\_old[0]), len(ys\_old), ys\_new)  
  
 differences = calculate\_distance\_of\_arrays(ys\_old, ys\_refined)  
  
 error = calculate\_error\_by\_array(differences)  
  
 if error <= ERROR: return ys\_new  
 return runge\_principle(ys\_new, iteration(functions, t, x, y, h / 2, start, end), iteration, functions, t, x, y,  
 h / 2, start,  
 end)  
  
  
def calculate\_distance\_of\_arrays(ys\_old, ys\_new):  
 distances = []  
  
 for i in range(len(ys\_old[0])):  
 sum = 0  
 for j in range(len(ys\_old)):  
 sum += (ys\_old[j][i] - ys\_new[j][i]) \*\* 2  
 distances.append(sum \*\* 0.5)  
  
 return distances  
  
  
def refine\_array\_over2(cols, rows, ys\_new):  
 ys\_refined = [[], []]  
  
 for i in range(rows):  
 for j in range(cols):  
 ys\_refined[i].append(ys\_new[i][j \* 2])  
  
 return ys\_refined  
  
  
def calculate\_error\_by\_array(arr):  
 error = 0  
 for i in range(len(arr)):  
 error += arr[i] \*\* 2  
  
 return (error / (len(arr) - 1)) \*\* 0.5

system\_of\_functions

G = 3  
K = 2  
  
  
def x\_der(xs):  
 t = xs[0]  
 x = xs[1]  
 y = xs[2]  
 return K \* t + x - y + G  
  
  
def y\_der(xs):  
 x = xs[1]  
 y = xs[2]  
 return -x + K \* y  
  
  
def F(xs, t):  
 x = xs[0]  
 y = xs[1]  
 x\_der\_t = K \* t + x - y + G  
 y\_der\_t = -x + K \* y  
 return [x\_der\_t, y\_der\_t]

Runge\_Kutt.py

from runge\_principle import runge\_principle  
  
  
  
def Runge\_Kutt(functions, t, x, y, h, start, end):  
 ys = iteration(functions, t, x, y, h, start, end)  
  
 return runge\_principle(ys, iteration(functions, t, x, y, h / 2, start, end), iteration, functions, t, x, y, h / 2,  
 start, end)  
  
  
def iteration(functions, t, x, y, h, start, end):  
 xs = [t, x, y]  
 ys = [[], []]  
  
 times = (end - start) / h  
  
 for l in range(int(times) + 1):  
  
 ks = calculate\_Ks(functions, xs, h)  
  
 for i in range(1, len(functions) + 1):  
 xs[i] = xs[i] + (ks[0][i - 1] + 2 \* ks[1][i - 1] + 2 \* ks[2][i - 1] + ks[3][i - 1]) / 6  
 ys[i - 1].append(xs[i])  
  
 xs[0] += h  
  
 return ys  
  
  
def calculate\_Ks(functions, xs, h):  
 ks = [[], [], [], []]  
 for i in range(len(functions)):  
 ks[0].append(h \* functions[i](xs))  
  
 for i in range(len(functions)):  
 inner = [xs[0] + h / 2]  
 for j in range(len(functions)):  
 inner.append(xs[j + 1] + ks[0][j] / 2)  
  
 ks[1].append(h \* functions[i](inner))  
  
 for i in range(len(functions)):  
 inner = [xs[0] + h / 2]  
 for j in range(len(functions)):  
 inner.append(xs[j + 1] + ks[1][j] / 2)  
  
 ks[2].append(h \* functions[i](inner))  
  
 for i in range(len(functions)):  
 inner = [xs[0] + h]  
 for j in range(len(functions)):  
 inner.append(xs[j + 1] + ks[2][j])  
  
 ks[3].append(h \* functions[i](inner))  
  
 return ks

**Висновки**

Під час цієї лабораторної роботи були розроблені два алгоритми за методом Ейлера та методом Рунге-Кута 4-го порядку точності для розв’язування системи диференційних рівнянь задачі Коші. Також, точність обчислень забезпечувалася реалізованим принципом Рунге, що орієнтувався на середньоквадратичну загальну точність. Згідно спостереженням, Метод Рунге-Кутта є більш точним, аніж метод Ейлера, що можна помітити на графіках, порівнюючи з оригінальною системою диференційних рівнянь, яка також приставлена на обох малюнках.