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

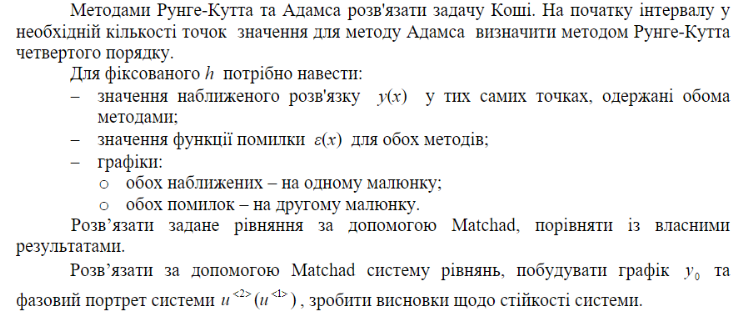
Викладач:

доц. Рибачук Л.В.

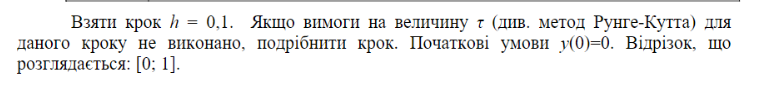
Київ – 2024

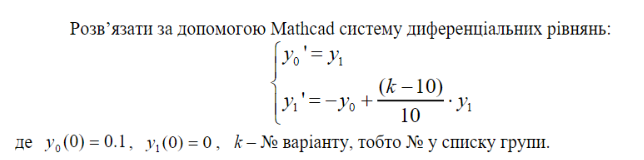
Тема: Розв’язання задачі Коші

1. Постановка задачі





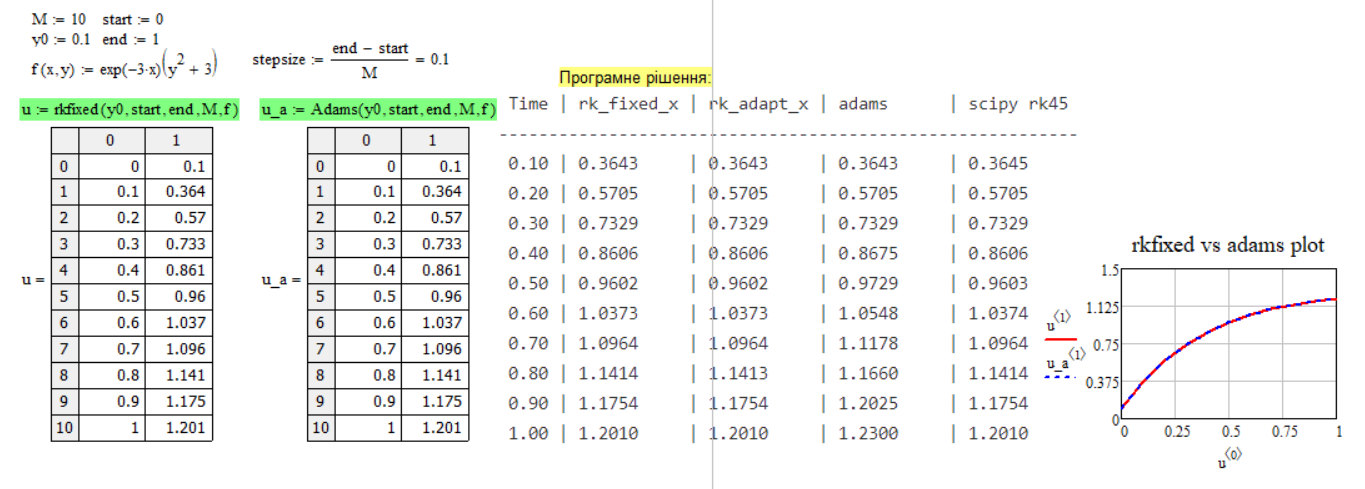




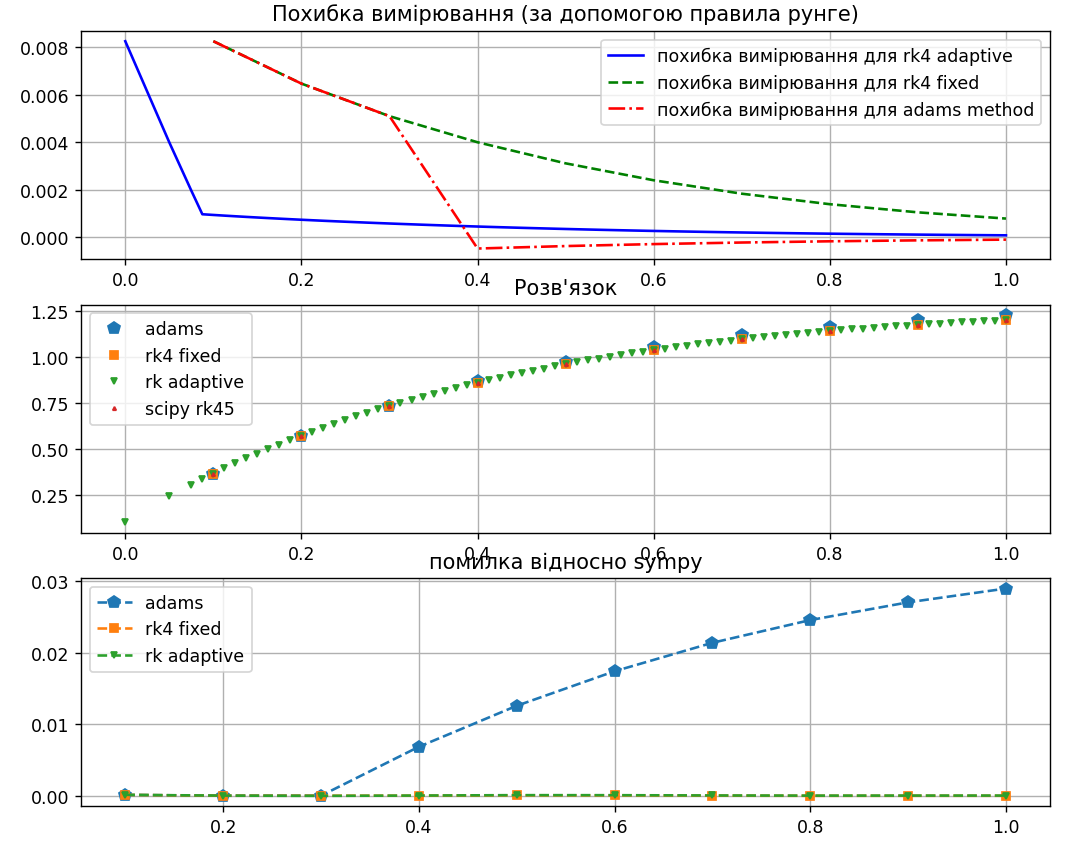
1. Вихідне рівняння



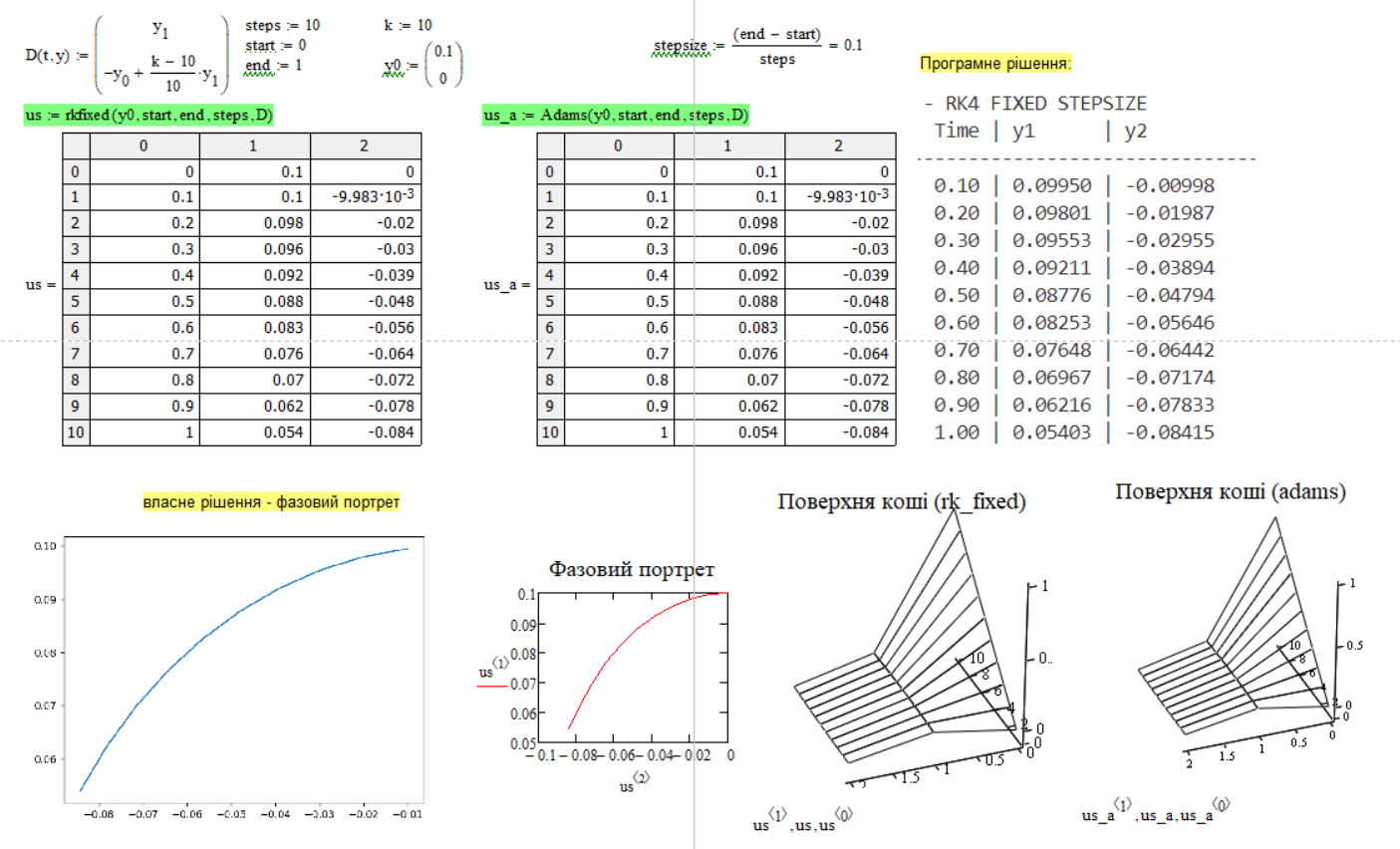
1. Значення наближеного розв’язку y(x) у тих самих точках одержані обома методами (Mathcad та власне рішення)



1. Значення функції помилки для методів



1. Система диференціальних рівнянь: значення та фазовий портрет



1. 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() |
| --- |