Antras laboratorinis darbas

Arnas Vaicekauskas

2024 m. spalio 14 d.

1 Uždavinys

Išspręsti pirmos eilės diferencialinę lygtį su Koši pradine sąlyga naudojanti Rungės-Kuto 3-pakopį ir 4-pakopį skaitinius modelius.

$$u' = x + 2x^2 \sin(u) \tag{1}$$
$$u(0) = -1 \tag{2}$$

2 Skaitiniai modeliai

Skaitiniai modeliai įgyvendinti naudojant python programavimo kalbą, numpy ir scipy paketus.

2.1 Rungė-Kuto 3-pakopis modelis

$$\begin{cases} k_1 & = f(x_n, y_n) \\ k_2 & = f(x_n + \tau, y_n + \tau k_1) \\ k_3 & = f(x_n + \frac{\tau}{2}, y_n + \frac{\tau}{2}\frac{k_1+k_2}{2}) \end{cases} \tag{3}$$
$$y_{n+1} = y_n + \frac{\tau}{6}(k_1 + k_2 + 4k_3). \tag{4}$$

2.2 Rungė-Kuto 4-pakopis modelis

$$\begin{cases} k_1 & = f(x_n, y_n) \\ k_2 & = f(x_n + \frac{\tau}{2}, y_n + \frac{\tau}{2}k_1) \\ k_3 & = f(x_n + \frac{\tau}{2}, y_n + \frac{\tau}{2}k_2) \\ k_4 & = f(x_n + \tau, y_n + \tau k_3) \end{cases} \tag{5}$$
$$y_{n+1} = y_n + \frac{\tau}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{6}$$

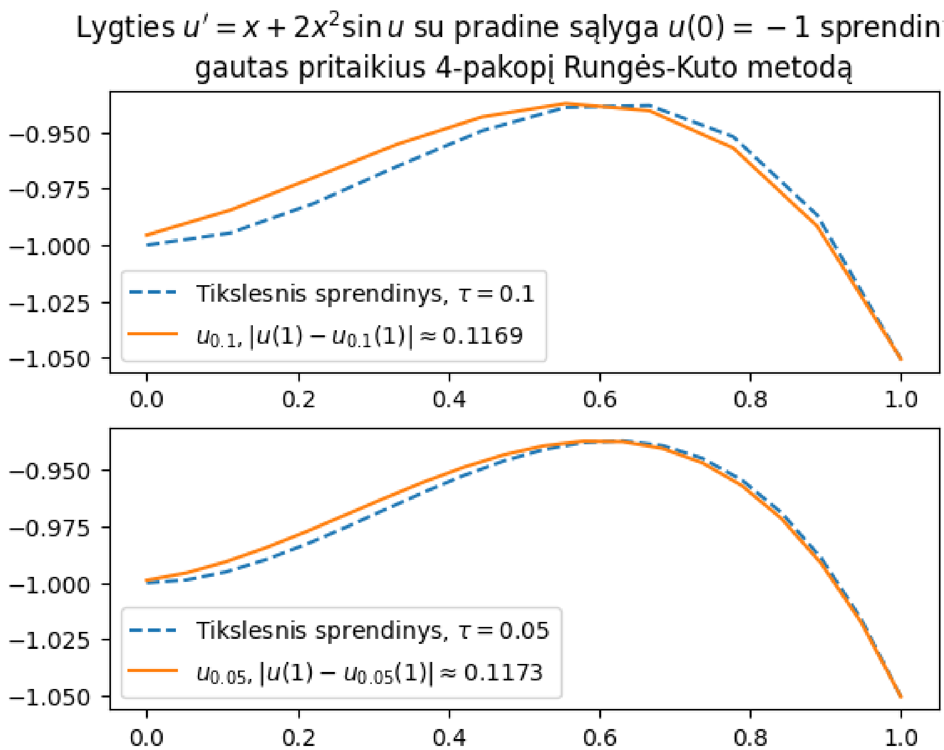
3 Rezultatai

3.1 Žymėjimas

x - laisvas kintamasis, u_τ - skaitinis sprendinys su žingsniu τ, $u_\tau(x)$ - skaitinio sprendinio su žingsniu τ reikšmė koordinatėje x, $u(x)$ - analitinio sprendinio reikšmė koordinatėje x.

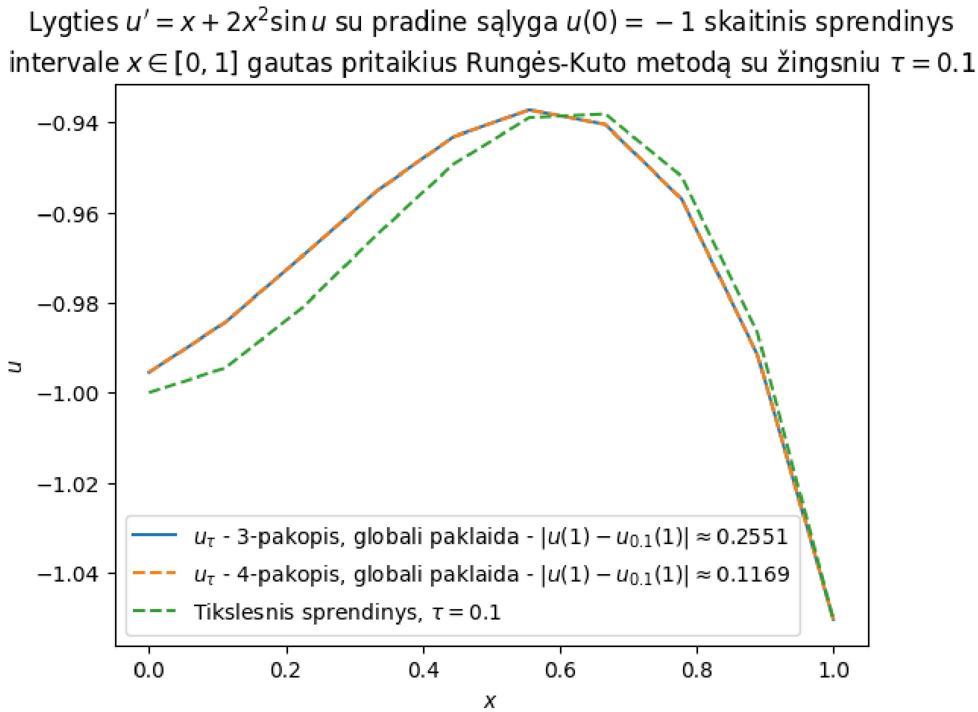
3.2 Sprendinių grafikai

3.2.1 (a) dalis



1 pav.: Skaitiniai lygties sprendiniai, kai $\tau = 0.1, 0.05$ lyginami su tikslesniu sprendiniu gautu naudojant scipy metodą odeint.

3.2.2 (b) dalis

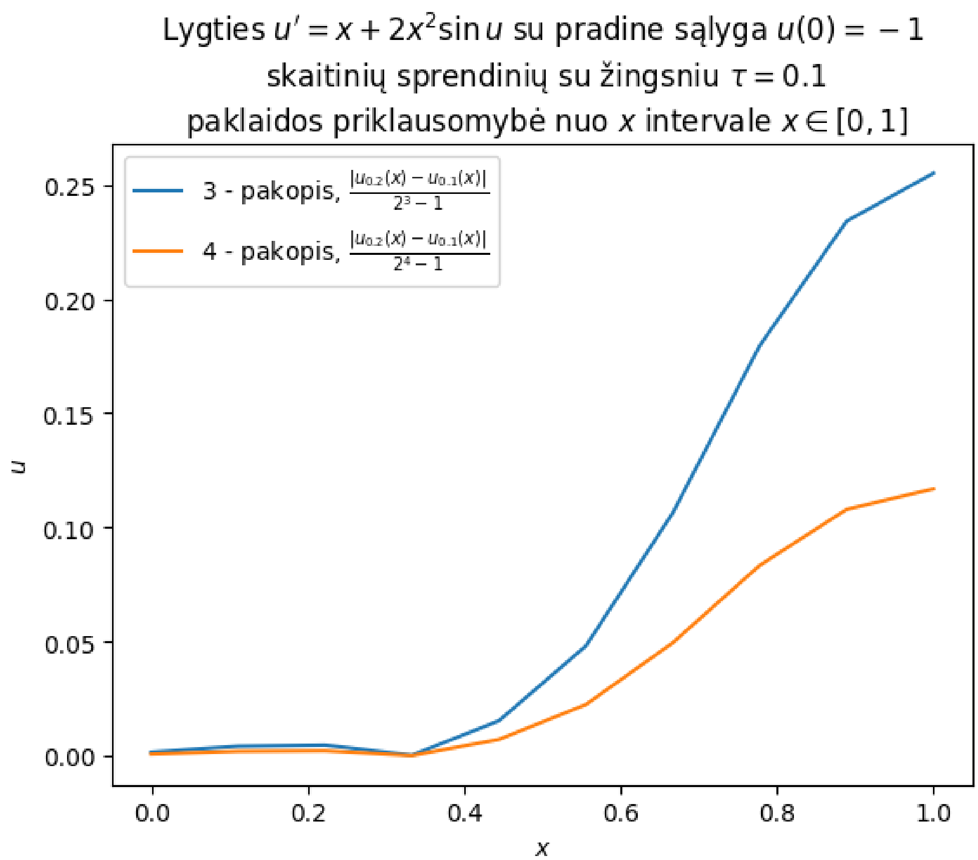


2 pav.: Skaitiniai lygties sprendiniai gauti skirtingais metodais lyginami su tikslesniu sprendiniu gautu naudojant scipy metodą odeint.

3.3 Paklaidos vertinimas

Rungės metodas paklaidai įvertinti

$$|u(x) - u_\tau(x)| \approx \frac{|u_{2\tau}(x) - u_\tau(x)|}{2^p - 1} \tag{7}$$



3 pav.: Skaitinių metodų paklaidos.

4 Priedai

Programos kodas naudotas sugeneruoti visus šiame dokumente esančius grafikus. Kode naudojamos antraštės išimtos, nes netelpa dokumente ir LaTeX kompiliatorius nesugeba sukompiliuoti simbolių eilučių, kuriose yra LaTeX kodo.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def rk3(f, t_0, y_0, tau, N):
    t_current = t_0
    y_current = y_0

    ys = np.zeros(N)

    for i in range(N):
        k1 = f(t_current, y_current)
        k2 = f(t_current + tau, y_current + tau * k1)
        k3 = f(t_current + tau/2, y_current + tau/2 * (k1 + k2)/2 )

        t_current = t_current + tau
        y_current = y_current + tau/6 * (k1 + k2 + 4*k3)
        ys[i] = y_current
    return ys

def rk4(f, t_0, y_0, tau, N):

    t_current = t_0
    y_current = y_0

    ys = np.zeros(N)

    for i in range(N):
        k1 = f(t_current, y_current)
        k2 = f(t_current + tau/2, y_current + tau/2 * k1)
        k3 = f(t_current + tau/2, y_current + tau/2 * k2)
        k4 = f(t_current + tau, y_current + tau * k3)

        t_current = t_current + tau
        y_current = y_current + tau/6 * (k1 + 2*k2 + 2*k3 + k4)
        ys[i] = y_current
    return ys

def f(t, u):
    return t + 2 * t**2 * np.sin(u)

u_0 = -1
```

```
x_0 = 0
x_end = 1
tau = [0.1, 0.05]

plt.xlabel('$x$')
plt.ylabel('$u$')

fig, axes = plt.subplots(2, sharey=True)

for index, tau_i in enumerate(tau):
    if index == 0:
        axes[index].set_title("")
    N = int((x_end - x_0) / tau_i)
    xs = np.linspace(x_0, 1, N)
    us = rk4(f, x_0, u_0, tau_i, N)
    us_real = odeint(lambda u, t: f(t, u), u_0, xs)
    axes[index].plot(xs, us_real,label="", linestyle='dashed')
    us_double_tau = rk4(f, x_0, u_0, 2 * tau_i, N)
    error = np.abs(us_double_tau[-1] - us[-1]) / (2**4 - 1)
    label = ""
    axes[index].plot(xs, us, label=label)
    axes[index].legend()
plt.show()

tau = 0.1
plt.title("")
plt.xlabel("")
plt.ylabel("")

N = int((x_end - x_0) / tau)
xs = np.linspace(x_0, 1, N)

us_3 = rk3(f, x_0, u_0, tau, N)
us_4 = rk4(f, x_0, u_0, tau, N)

us_3_double_tau = rk3(f, x_0, u_0, 2 * tau, N)
us_4_double_tau = rk4(f, x_0, u_0, 2 * tau, N)

error_3_global = np.abs(us_3_double_tau[-1] - us_3[-1]) / (2**3 - 1)
error_4_global = np.abs(us_4_double_tau[-1] - us_4[-1]) / (2**4 - 1)

label = ""
plt.plot(xs, us_3, label=label)

label = ""
plt.plot(xs, us_4, label=label, linestyle='dashed')

us_real = odeint(lambda u, t: f(t, u), u_0, xs)
```

```
plt.plot(xs, us_real,label="", linestyle='dashed')

plt.legend()
plt.show()

error_3_local = np.abs(us_3_double_tau - us_3) / (2**3 - 1)
error_4_local = np.abs(us_4_double_tau - us_4) / (2**4 - 1)

plt.title("")
label = ""
plt.plot(xs, error_3_local, label=label)
label = ""
plt.plot(xs, error_4_local, label=label)
plt.legend()
plt.xlabel("")
plt.ylabel("")
plt.show()
```