Kauno technologijos universitetas

Informatikos fakultetas

P170B115 Skaitiniai metodai ir algoritmai

4 projektinė užduotis. Paprastųjų diferencialinių lygčių sprendimas.

Nedas Liaudanskis

Studentas

doc. Čalnerytė Dalia

Dėstytoja

KAUNAS, 2023

Turinys

Užduotis

Reikia sudaryti diferencialinę lygtį iš duotos fizikinio uždavinio sąlygos, naudojant fizikos dėsnius. Sudarytą lygtį išspręsti Eulerio ir IV eilės Rungės ir Kutos metodais. Keisdami metodo žingsnį turime įsitikinti, jog gavome tikslų sprendinį. Visus sprendinius atvaizduoti viename grafike ir palyginti jų gautus atsakymus. Keisdami metodo žingsnį nustatyti didžiausią žingsnį, su kuriuo metodas išlieka stabilus. Palyginti metodų stabilumo prasme ir atvaizduoti viename grafike sprendinius. Patikrinkite gautą sprendinį su MATLAB standartine funkcija ode45, Python scipy.integrate bibliotekos funkcija solve_ivp ar kitais išoriniais šaltiniais. Tame pačiame grafike turi būti pateikti realizacijose ir naudojant išorinius šaltinius gauti sprendiniai.

m_1 masės parašiutininkas su m_2 masės įranga iššoka iš lėktuvo, kuris skrenda aukštyje h_0. Po tg laisvo kritimo parašiutas išskleidžiamas. Oro pasipriešinimo koeficientas laisvo kritimo metu lygus k_1, o išskleidus parašiutą - k_2. Tariama, kad paliekant lėktuvą parašiutininko greitis lygus 0 m/s, o oro pasipriešinimas proporcingas parašiutininko greičio kvadratui.

- Raskite, kaip kinta parašiutininko greitis nuo 0 s iki nusileidimo.
- Kada ir kokiu greičiu parašiutininkas pasiekia žemę?
- Kokiame aukštyje išskleidžiamas parašiutas?

Varianto numeris	m_1, kg	m_2, kg	h_0, m	t_g, s	k_1, kg/m	k_2, kg/m
18	60	15	3500	25	0,1	7

Lygties sudarymas

Remiantis Niutono dėsniu sudaroma lygtis :

$F = Fgravitacija - Foro\ pasipriesinimas$

Diferencialinė lygties gavimas :

$$F = ma$$

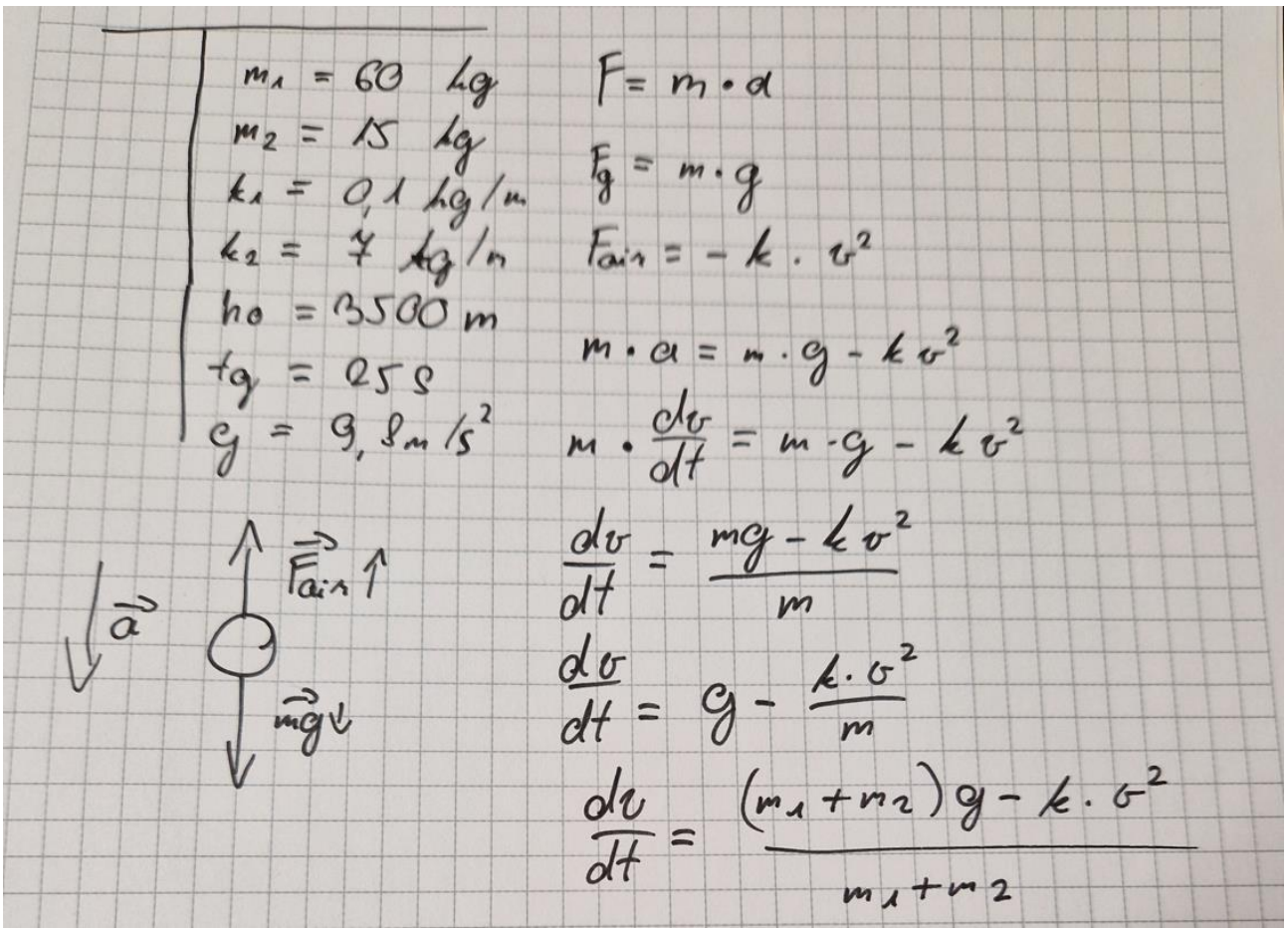
$$F_g = mg$$

$$F_{oro} = -kv^2$$

$$ma = mg - kv^2$$

$$m\frac{dv}{dt} = mg - kv^2$$

$$\frac{dv}{dt} = \frac{mg - kv^2}{m}$$



Sprendimas Eulerio metodu

Diferencialinė lygtis (DL) yra matematinė lygtis, kurioje nagrinėjamas kintamasis, priklausantis nuo šio kintamojo išvestinių. Ji aprašo, kaip kinta tam tikras dydis, kuris priklauso nuo kitų dydžių arba šio dydžio kitimo laike.

Eulerio metodas, pradeda skaičiavimus, nuo pradinės reikšmės, mūsų atveju tai visi duoti duomenys ir pradinio laiko momento. Tai yra paprastas skaitinis metodas diferencialinių lygčių sprendimui, jis kiekviename žingsnyje apskaičiuoja naują išvestinės aproksimaciją priklausančią nuo tam tikro dydžio, mūsų atveju tai laikas. Kiekvieną žingsnį apskaičiuotas naujas sprendinys būtų greitis. Šis aproksimavimas kartojasi, kol pasiekiamas norimas laiko intervalas.

Formulės:

$$y_{n+1} = y_n + dt * f(t_n, y_n)$$

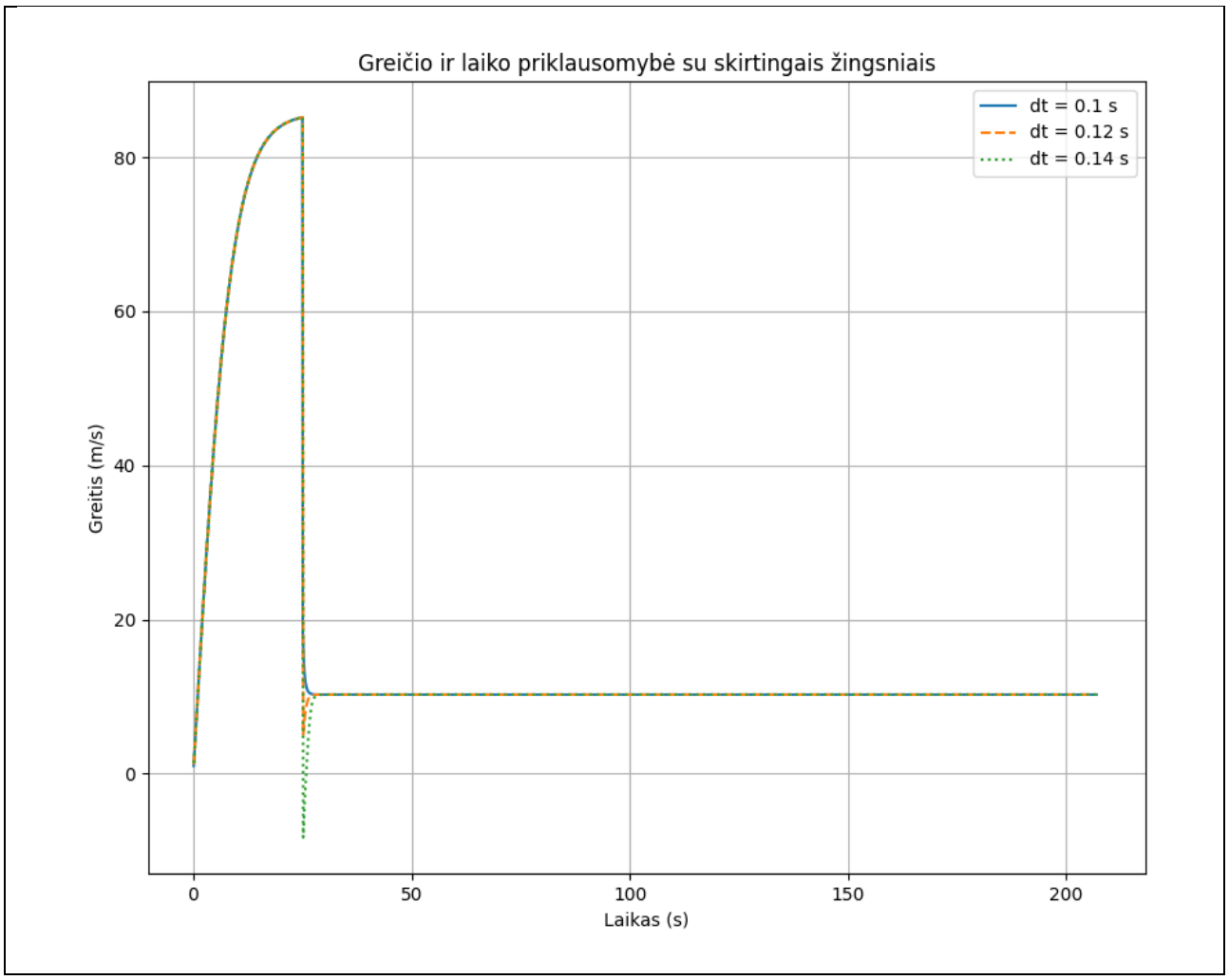
<u>Kodas:</u>

```python
import matplotlib.pyplot as plt

m1 = 60  # Parasiutininko mase
m2 = 15  # Parasiuto mase (kartu 75kg)
k1 = 0.1  # Oro pasipriesinimas laisvo kritimo metu
k2 = 7    # Oro pasipriesinimas iskleidus parasiuta
h0 = 3500 # Aukstis is kurio sokama
tg = 25   # Sekundes po kuriu iskleidziamas parasiutas

parachute_deployed_height = 0
uzfiksuotasAukstis = False

def dvdt(t, v, k):
    g = 9.8
    return (((m1 + m2) * g) - (k * v * abs(v))) / (m1 + m2)

def simulate_parachute(dt):
    global uzfiksuotasAukstis
    global parachute_deployed_height
    v0 = 0
    v = v0
    h = h0
    t = 0

    t_list = []
    h_list = []
    v_list = []

    while h > 0:
        if t < tg:
            v += dt * dvdt(t, v, k1)
        else:
            if not uzfiksuotasAukstis:
                parachute_deployed_height = h
                uzfiksuotasAukstis = True
            v += dt * dvdt(t, v, k2)

        h -= dt * abs(v)
        t += dt
```

```
        t_list.append(t)
        h_list.append(h)
        v_list.append(v)

    return t_list, h_list, v_list

# Simulate parachute descent with different time steps
t_values_01, h_values_01, v_values_01 = simulate_parachute(0.1)
t_values_05, h_values_05, v_values_05 = simulate_parachute(0.12)
t_values_1, h_values_1, v_values_1 = simulate_parachute(0.14)

# Plot all three graphs on one plot
plt.figure(figsize=(10, 8))
plt.plot(t_values_01, v_values_01, label='dt = 0.1 s')
plt.plot(t_values_05, v_values_05, label='dt = 0.12 s', linestyle='--')
plt.plot(t_values_1, v_values_1, label='dt = 0.14 s', linestyle=':')
plt.xlabel('Laikas (s)')
plt.ylabel('Greitis (m/s)')
plt.title('Greičio ir laiko priklausomybė su skirtingais žingsniais')
plt.legend()
plt.grid(True)
plt.show()

# Rest of the code remains the same for printing and displaying the results
```

Rezultatai:

Parašiutininkas pasiekia žemę per: 206.90 s
Parašiutininko greitis kai pasieka žemę: 10.25 m/s
Aukštis kuriame yra išskleidžiamas parašiutas: 1867.36 m



Sprendimas IV eilės Rungės ir Kutos metodu:

Kiekvieno žingsnio metu apskaičiuojamas tarpinės kintamųjų reikšmės, leidžiančios tiksliau aproksimuoti funkcijos kitimą per žingsnį. Procesas kartojamas tol, kol pasiekiama norima laiko momentų aibė arba kol pasiekiamas iš anksto nustatytas stabdymo kriterijus.

Naudojamos formulės:

$$k_1 = h * f(t_n, y_n)$$
$$k_2 = h * f(t_n + \frac{h}{2}, y_n + \frac{k_1}{2})$$
$$k_3 = h * f(t_n + \frac{h}{2}, y_n + \frac{k_2}{2})$$
$$k_4 = h * f(t_n + h, y_n + k_3)$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3\, k_4)$$

Kodas:

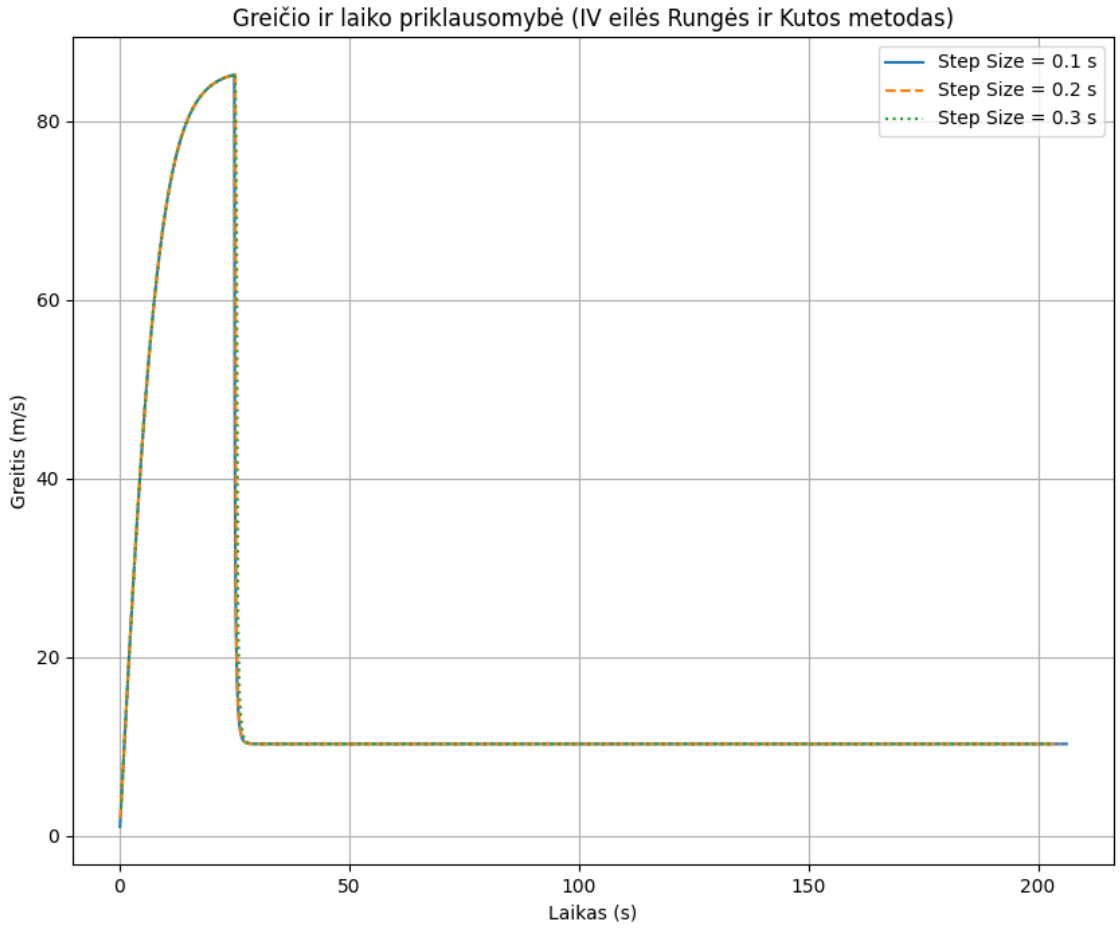
```python
import matplotlib.pyplot as plt

m1 = 60  # parasiutininko mase
m2 = 15  # parasiuto mase (kartu 75kg)
k1 = 0.1  # oro pasipriesinimas laisvo kritimo metu
k2 = 7  # oro pasipriesinimas iskleidus parasiuta
h0 = 3500  # aukstis is kurio sokama
tg = 25  # sekundes po kuriu iskleidziamas parasiutas

parachute_deployed_height = 0
uzfiksuotasAukstis = False

def dvdt(t, v, k):
    g = 9.8
    return (((m1 + m2) * g) - (k * v * abs(v))) / (m1 + m2)

def rk4_method():
    global uzfiksuotasAukstis
    global parachute_deployed_height
    v0 = 0
    v = v0
    dt = 0.3  # Žingsnis
    h = h0
    t = 0

    t_list = []
    h_list = []
    v_list = []
```

```
    while h > 0:
        if t < tg:
            k1v = dt * dvdt(t, v, k1)
            k2v = dt * dvdt(t + dt/2, v + k1v/2, k1)
            k3v = dt * dvdt(t + dt/2, v + k2v/2, k1)
            k4v = dt * dvdt(t + dt, v + k3v, k1)
            v += (k1v + 2*k2v + 2*k3v + k4v) / 6
        else:
            if not uzfiksuotasAukstis:
                parachute_deployed_height = h
                uzfiksuotasAukstis = True
            k1v = dt * dvdt(t, v, k2)
            k2v = dt * dvdt(t + dt/2, v + k1v/2, k2)
            k3v = dt * dvdt(t + dt/2, v + k2v/2, k2)
            k4v = dt * dvdt(t + dt, v + k3v, k2)
            v += (k1v + 2*k2v + 2*k3v + k4v) / 6

        h -= dt * abs(v)
        t += dt

        t_list.append(t)
        h_list.append(h)
        v_list.append(v)

    return t_list, h_list, v_list

t_values, h_values, v_values = rk4_method()
```

```
plt.figure(figsize=(8, 6))
plt.plot(t_values, v_values, label='IV eilės Rungės ir Kutos metodas')
plt.xlabel('Laikas (s)')
plt.ylabel('Greitis (m/s)')
plt.title('Greičio ir laiko priklausomybė (IV eilės Rungės ir Kutos metodas)')
plt.legend()
plt.grid(True)
plt.show()

touchdown_time = t_values[-1]
touchdown_speed = v_values[-1]

print(f"Parašiutininkas pasiekia žemę per: {touchdown_time:.2f} s")
print(f"Parašiutininko greitis kai pasieka žemę: {touchdown_speed:.2f} m/s")
print(f"Aukštis kuriame yra išskleidžiamas parašiutas: {parachute_deployed_height:.2f} m")
```

Rezultatai:

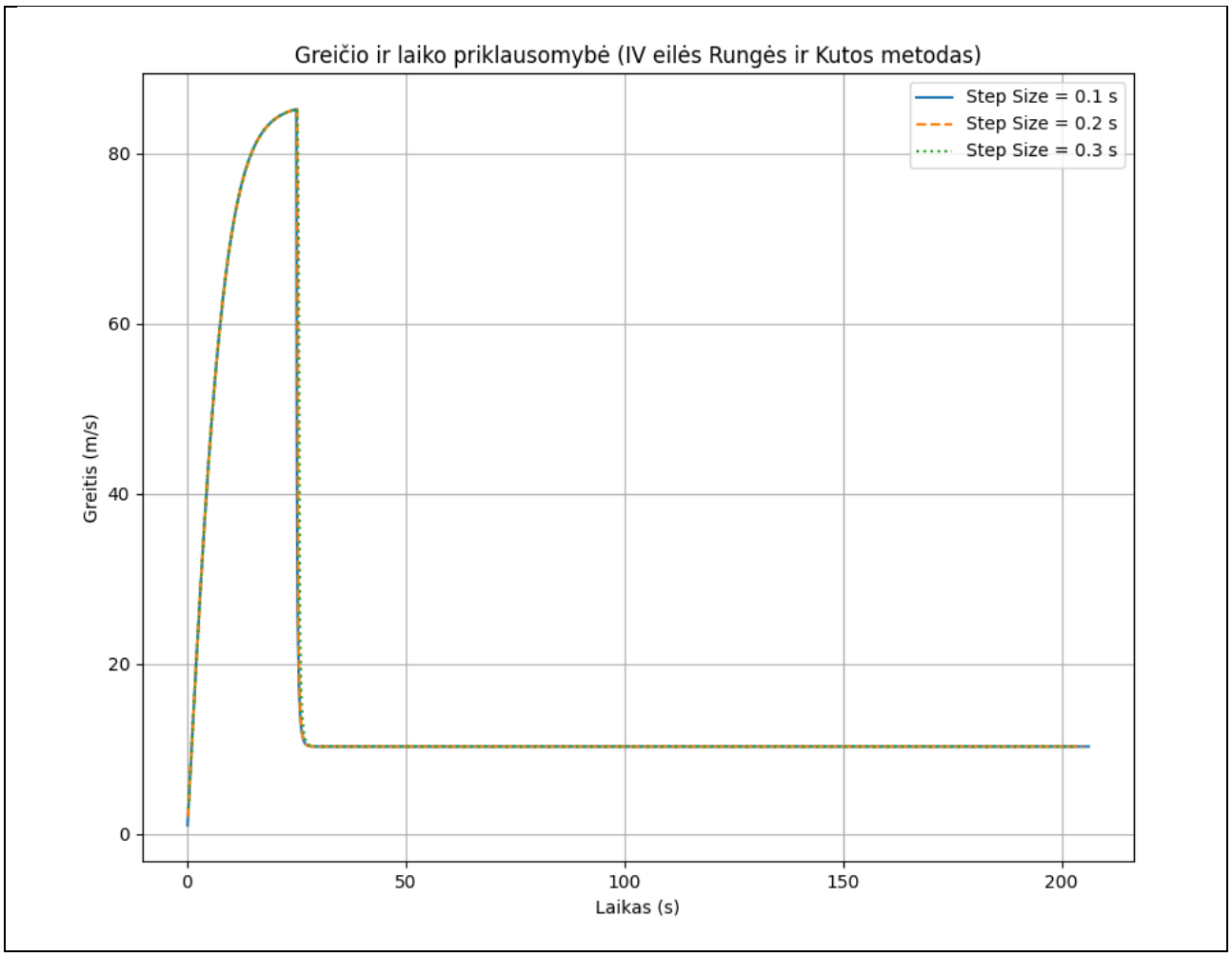
Parašiutininkas pasiekia žemę per: 202.80 s
Parašiutininko greitis kai pasieka žemę: 10.25 m/s
Aukštis kuriame yra išskleidžiamas parašiutas: 1844.35 m

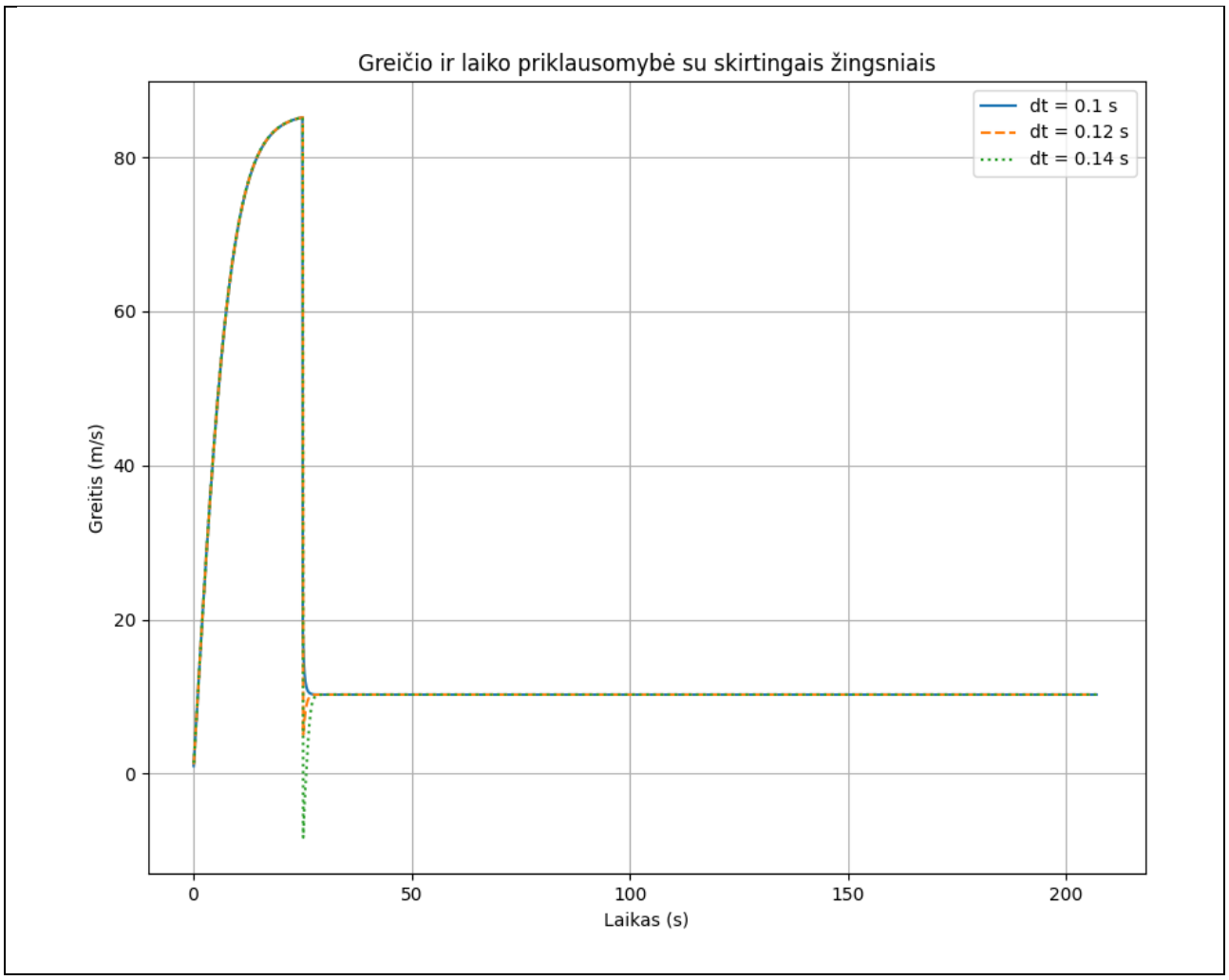


Žingsnių kitimo palyginimai

Kaip matome iš apačioje pateiktų grafų, naudojant Eulerio metodą ir keičiant žingsnį, grafikas pradeda nukrypti nuo pradinės reikšmės gan greitai, tačiau skaičiavimo atsakymai nepasikečia tol kol žingsnis nepasiekia 0.26. O naudojant IV RK metodą, darant minimalius žinginio pakeitimus nepakeičia grafiko, tuo labiau reikšmės.

<u>Rezultatai:</u>


Greičio ir laiko priklausomybė (IV eilės Rungės ir Kutos metodas)
Step Size = 0.1 s
Step Size = 0.2 s
Step Size = 0.3 s
Greitis (m/s)
0
20
40
60
80
0
50
100
150
200
Laikas (s)

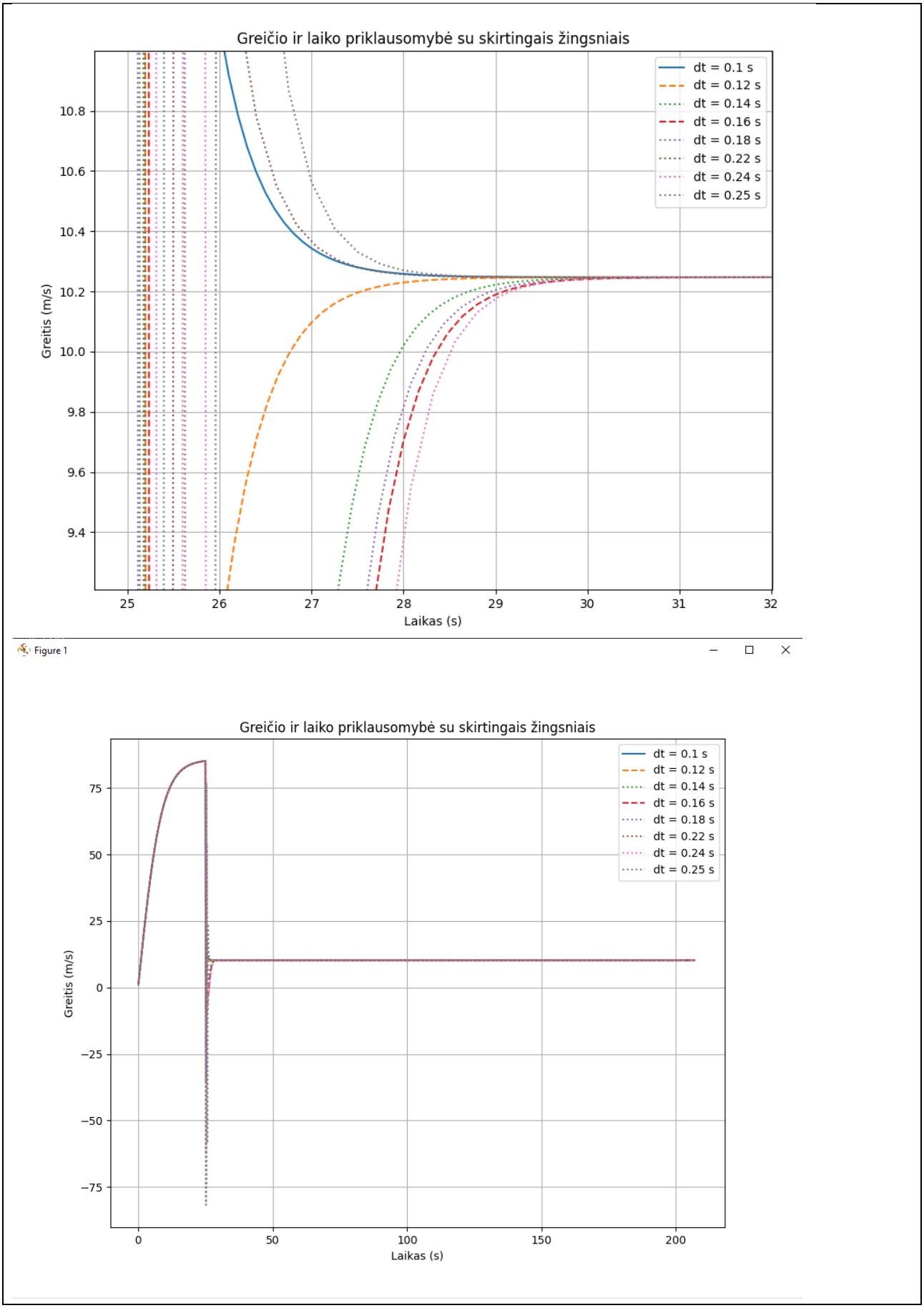


Didžiausias stabilus žingsnis

Norint nustatyti didžiausią stabilų žingsnį atlikau daug skirtingų skaičiavimų, su skirtingais žingsniais. Apačioje pateikti skaičiavimų grafikai, rodantys kiekvieną žingsnį ir grafiką. Gan lengva pastebėti, jog žingsnis gan greitai iškraipo grafiką, tačiau atsakymo nepakeičia iki tol, kol žingsnis netampa 0.26. Tada staiga ir grafikas ir atsakymai tampa nebelogiški ir neteisingi. Todėl galime teigti, jog su žingsniu 0.1 – 0.25 metodas dar yra stabilus, tačiau pasiekus žingsnį 0.26 metodas staiga lūžta.

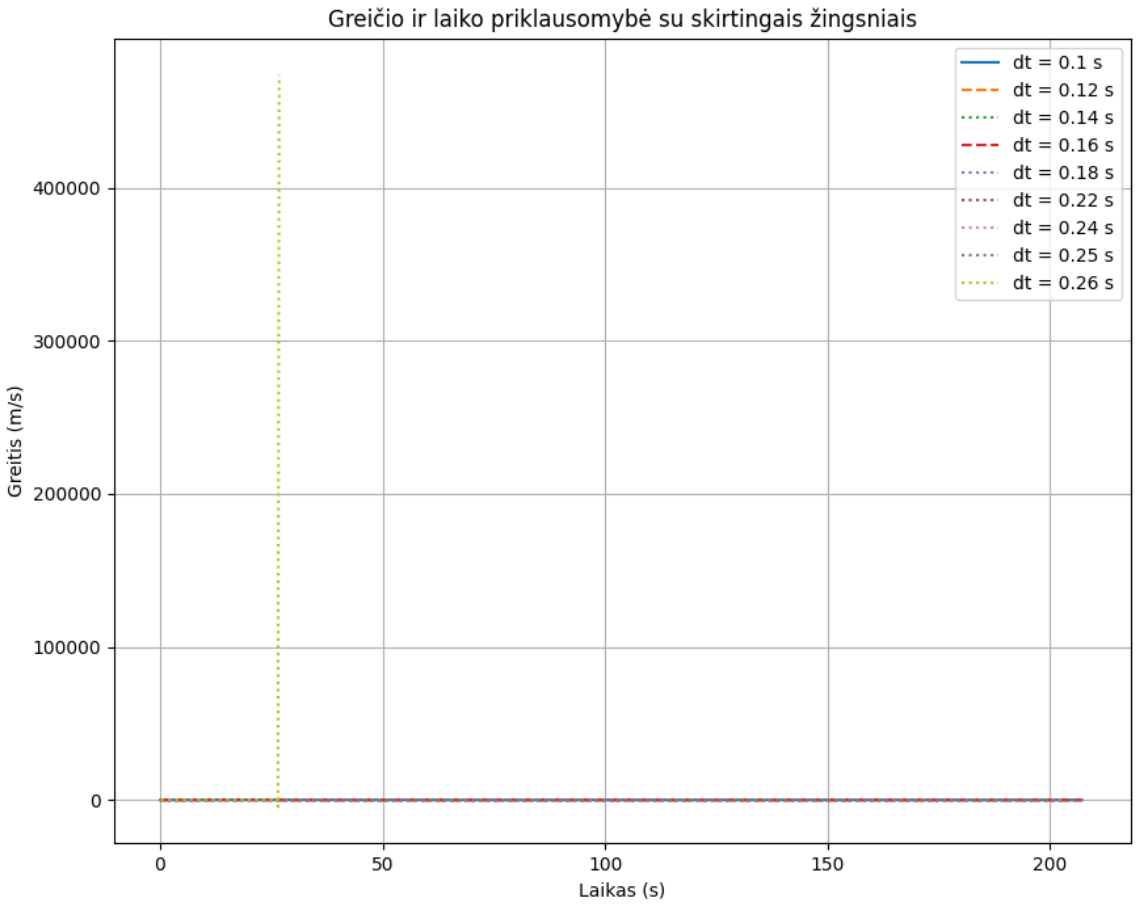
<u>Rezultatai:</u>

Parašiutininkas pasiekia žemę per: 207.36 s
Parašiutininko greitis kai pasieka žemę: 10.25 m/s
Aukštis kuriame yra išskleidžiamas parašiutas: 1867.36 m


Greičio ir laiko priklausomybė su skirtingais žingsniais
dt = 0.1 s
dt = 0.12 s
dt = 0.14 s
dt = 0.16 s
dt = 0.18 s
dt = 0.22 s
dt = 0.24 s
dt = 0.25 s
Greitis (m/s)
Laikas (s)
Figure 1
Greičio ir laiko priklausomybė su skirtingais žingsniais
dt = 0.1 s
dt = 0.12 s
dt = 0.14 s
dt = 0.16 s
dt = 0.18 s
dt = 0.22 s
dt = 0.24 s
dt = 0.25 s
Greitis (m/s)
Laikas (s)

Rezultatai:

Parašiutininkas pasiekia žemę per: 26.78 s
Parašiutininko greitis kai pasieka žemę: 473755.25 m/s
Aukštis kuriame yra išskleidžiamas parašiutas: 1867.36 m



Gautų sprendinių patikrinimas

Sprendimiams patikrinti naudojau scipy odeint.

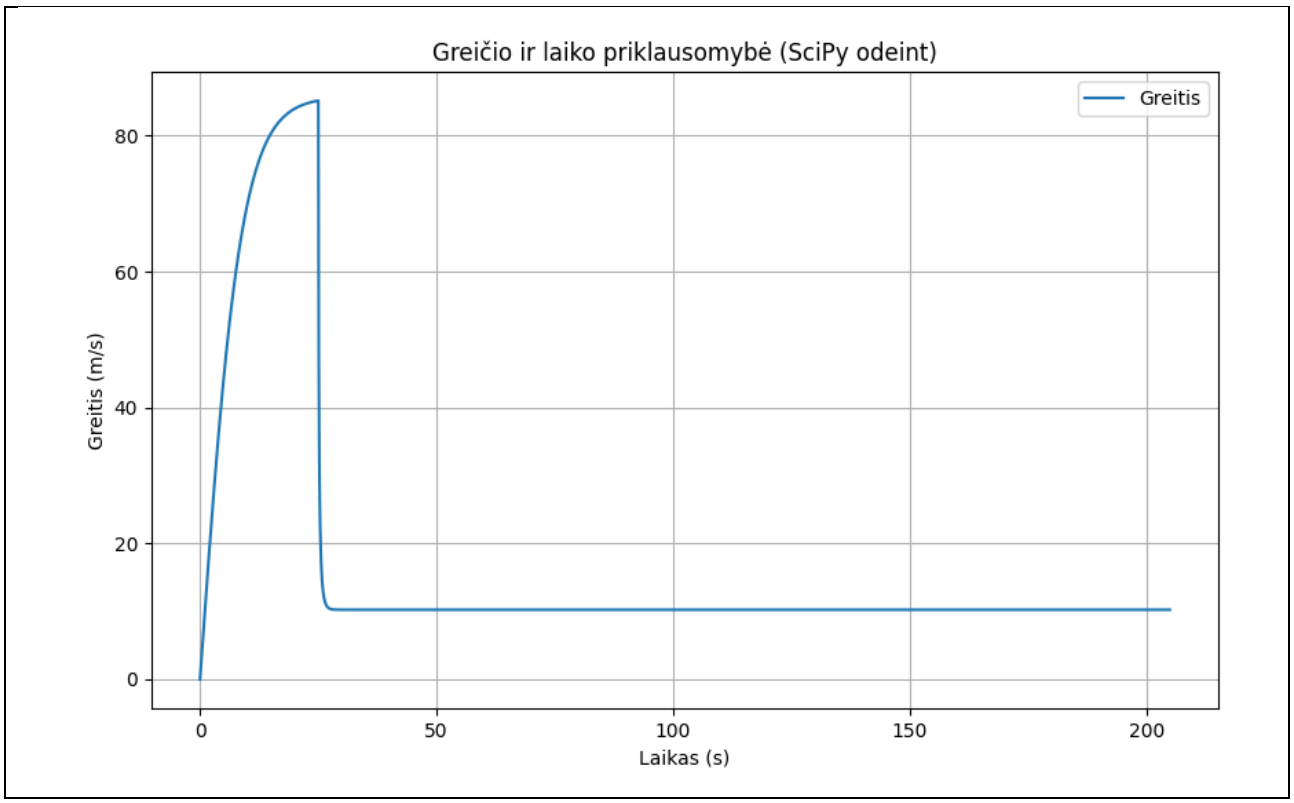
Kodas:

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

m1 = 60  # parasiutininko mase
m2 = 15  # parasiuto mase (kartu 75kg)
k1 = 0.1  # oro pasipriesinimas laisvo kritimo metu
k2 = 7  # oro pasipriesinimas iskleidus parasiuta
h0 = 3500  # aukstis is kurio sokama
tg = 25  # sekundes po kuriu iskleidziamas parasiutas

def dvdt(v, t, k):
    g = 9.8
    return (((m1 + m2) * g) - (k * v * abs(v))) / (m1 + m2)

def model(y, t, k):
    v = y[0]
    h = y[1]
    dydt = [dvdt(v, t, k), -abs(v)]
    return dydt

# Initial conditions
y0 = [0, h0]

# Time points for the first stage
t1 = np.arange(0, tg, 0.1)

# Solve ODE using odeint for the first stage
result1 = odeint(model, y0, t1, args=(k1,))
v_values_eulerio = result1[:, 0]

# Extract the final height and velocity from the first stage
final_height = result1[-1, 1]
final_velocity = result1[-1, 0]
```

```python
# Set initial conditions for the second stage
y0 = [final_velocity, final_height]

# Time points for the second stage
t2 = np.arange(tg, 100, 0.1)

# Continue solving ODE using odeint with k2 for the second stage
result2 = odeint(model, y0, t2, args=(k2,))
v_values_parachute = result2[:, 0]

# Concatenate the results from both stages
t = np.concatenate((t1, t2))
v_values = np.concatenate((v_values_eulerio, v_values_parachute))

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(t, v_values, label='Greitis')
plt.xlabel('Laikas (s)')
plt.ylabel('Greitis (m/s)')
plt.title('Greičio ir laiko priklausomybė (SciPy odeint)')
plt.legend()
plt.grid(True)
plt.show()

# Extract values for printing
touchdown_time = t[-1]
touchdown_speed = v_values[-1]
parachute_deployed_height = final_height

# Print the values
print(f"Parašiutininkas pasiekia žemę per: {touchdown_time:.2f} s")
print(f"Parašiutininko greitis kai pasieka žemę: {touchdown_speed:.2f} m/s")
print(f"Aukštis kuriame yra išskleidžiamas parašiutas: {parachute_deployed_height:.2f} m")
```

Rezultatai:

Parašiutininkas pasiekia žemę per: 204.90 s
Parašiutininko greitis kai pasieka žemę: 10.25 m
Aukštis kuriame yra išskleidžiamas parašiutas: 1882.61 m


Greičio ir laiko priklausomybė (SciPy odeint)
Greitis
Greitis (m/s)
0
20
40
60
80
0
50
100
150
200
Laikas (s)