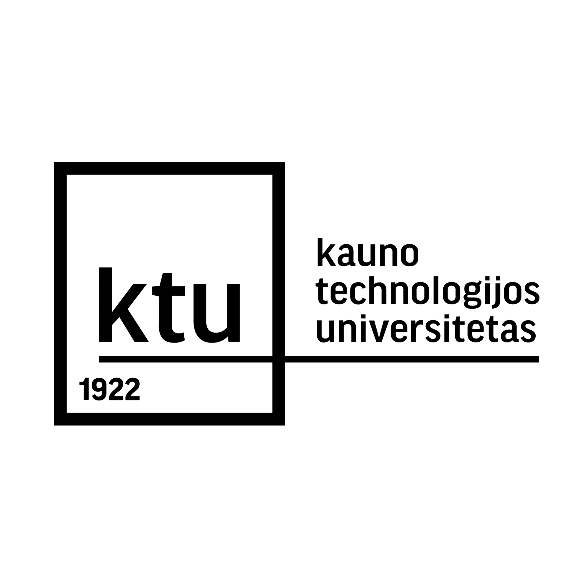
**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

**SKAITINIAI METODAI IR ALGORITMAI(P170B115)**

**4 LABORATORINIS DARBAS**

Varianto Nr. 15

**Atliko:**

IFF-1/8 gr. studentas

Matas Palujanskas**Priėmė:**

Prof. Rimantas Barauskas

Doc. Andrius Kriščiūnas

KAUNAS

2023

**Turinys**

[1. Užduotis 3](#_Toc152693052)

[1.1 Užduoties tikslas 3](#_Toc152693053)

[1.2 Užduoties sąlyga 3](#_Toc152693054)

[2. Užduoties sprendimas 4](#_Toc152693055)

[2.1. Teorinė dalis 4](#_Toc152693056)

[2.2. Sprendimas Eulerio metodu 5](#_Toc152693057)

[2.2.1 Kodas 5](#_Toc152693058)

[2.2.2 Testavimas 6](#_Toc152693059)

[2.2.3 Grafikai 7](#_Toc152693060)

[2.3. Sprendimas IV eilės Rungės ir Kutos metodus 7](#_Toc152693061)

[2.3.1 Kodas 7](#_Toc152693062)

[2.3.2 Testavimas 9](#_Toc152693063)

[2.3.3 Grafikai 10](#_Toc152693064)

[2.4. Tikrinimas su solve\_ivp 11](#_Toc152693065)

[2.4.1 Tikrinimo kodas 11](#_Toc152693066)

[2.4.2 Tikrinimo grafikai 12](#_Toc152693067)

[2.5. Bendros išvados 13](#_Toc152693068)

[3. Literatūros sąrašas 13](#_Toc152693069)

# **Užduotis**

## Užduoties tikslas

Remdamiesi pateiktų fizikinių dėsnių aprašymais, gautajam **15**-ąjam variantui sudaryti

diferencialinę lygtį arba lygčių sistemą bei ją paaiškinti. Diferencialinę lygtį arba lygčių sistemą

įspręsti skaitinių metodų pagalba tai yra Eulerio ir IV eilės Rungės ir Kutos metodais. Suprasti

kaip žingsnis gali daryti įtaką uždavinio rezultatų tikslumui. Palyginti metodų tikslumo prasmę

bei patikrinti gautą rezultatą su MATLAB standartine funkcija ode45 arba kitais išoriniais

šaltiniais.

## Užduoties sąlyga

Paveikslėlis, kuriame yra tekstas, ekrano kopija, skaičius, Šriftas

Automatiškai sugeneruotas aprašymas

pav. 1 Užduoties sąlyga



pav. 2 15 varianto duomenys

# Užduoties sprendimas

## Teorinė dalis

Pagal užduoties sąlygą kūnas juda su pagreičiu, todėl remsiuosi antruoju Niutono dėsniu, kuris teigia, kad pagreitis 𝑎⃗ , kuriuo juda kūnas yra tiesiogiai proporcingas kūną veikiančiai jėgai 𝐹⃗ A ir atvirkščiai proporcingas to kūno masei.

𝐹⃗ A = 𝑚𝑎⃗

Paveikslėlis, kuriame yra tekstas, ekrano kopija, diagrama, Šriftas

Automatiškai sugeneruotas aprašymas

pav. 3 Kūną veikiančių jėgų schema

Taipogi žinau, kad greitis yra pirmoji kelio funkcijos s(t) išvestinė, kaip pagreitis yra pirmoji greičio funkcijos išvestinė. Todėl gauname:

Paveikslėlis, kuriame yra Šriftas, skaičius, tekstas, ekrano kopija

Automatiškai sugeneruotas aprašymas

Remdamasis šiuo dėsniu susidarau lygtį, kuri rodo parašiutininką veikiančia gravitaciją bei oro pasipriešinimo jėgą.

Paveikslėlis, kuriame yra Šriftas, tipografija, tekstas, simbolis

Automatiškai sugeneruotas aprašymas

Visas kūnas kurį veikia jėga yra m1+m2 (parašiutininko masė bei pačio parašiuto), kadangi pasipriešinimo koeficientą turiu, galiu išsireikšti lygtį.

Paveikslėlis, kuriame yra tekstas, Šriftas, linija, skaičius

Automatiškai sugeneruotas aprašymas

Pasipriešinimo koeficientas po parašiuto išskleidimo pakis, todėl pradžioje jis yra **0.25**, o po parašiuto išskleidimo jis tampa **10**.

## Sprendimas Eulerio metodu

### 2.2.1 Kodas

**Eulerio.ipynb:**

import numpy as np  
import matplotlib.pyplot as plt  
  
def parašiutininko\_kritimas(X, t):  
 h, v = X  
 m1 = 120 *# parašiutininko masė (kg)* m2 = 10 *# įrangos masė (kg)* k1 = 0.25 *# oro pasipriešinimo koeficientas laisvo kritimo metu (kg/m)* k2 = 10 *# oro pasipriešinimo koeficientas išskleidus parašiutą (kg/m)* g = 9.81 *# gravitacijos pagreitis (m/s^2)* if t < 25:  
 *# Laisvas kritimas iki parašiuto išsiskleidimo* F = (m1 + m2) \* g - k1 \* v\*\*2 \* np.sign(v)  
 else:  
 *# Parašiutas išsiskleidęs* F = (m1 + m2) \* g - k2 \* v\*\*2 \* np.sign(v)  
  
 dhdt = -v  
 dvdt = F / (m1 + m2)  
  
 return np.array([dhdt, dvdt])  
  
*# Laiko nustatymai*viso\_laikas = 146 *# Visas skaičiavimo laikas (s)*dt = 0.1 *# Laiko žingsnis Eulerio metodu  
  
# Pradinės sąlygos*h0 = 2800 *# Pradinis aukštis (m)*v0 = 0 *# Pradinis greitis (m/s)  
  
# Rezultatų masyvas*N = int(viso\_laikas / dt) + 1  
t = np.linspace(0, viso\_laikas, N)  
rezultatas = np.zeros([2, N])  
rezultatas[:, 0] = np.array([h0, v0])  
  
*# Skaičiavimas naudojant Eulerio metodą*for i in range(N - 1):  
 išvestinė = parašiutininko\_kritimas(rezultatas[:, i], t[i])  
 rezultatas[:, i + 1] = rezultatas[:, i] + išvestinė \* dt  
  
  
*# Analizuojame rezultatus*pasiekimo\_momento\_indeksas = np.argmax(rezultatas[0] <= 0) *# Indeksas, kai aukštis tampa neigiamas*pasiekimo\_laikas = t[pasiekimo\_momento\_indeksas]  
pasiekimo\_greitis = rezultatas[1, pasiekimo\_momento\_indeksas]  
  
išskleidimo\_indeksas = np.argmax((t >= 25) & (rezultatas[0] > 0)) *# Indeksas, kai parašiutas išsiskleidžia*išskleidimo\_laikas = t[išskleidimo\_indeksas]  
išskleidimo\_aukštis = rezultatas[0, išskleidimo\_indeksas]  
  
*# Spausdiname rezultatus*print(f"Parašiutininkas pasiekia žemę laiko t = {pasiekimo\_laikas:.2f} s metu su greičiu {pasiekimo\_greitis:.2f} m/s.")  
print(f"Parašiutas išsiskleidžiamas laiko t = {išskleidimo\_laikas:.2f} s metu, esant aukštyje {išskleidimo\_aukštis:.2f} m.")  
  
*# Braižome rezultatus*fig, ax = plt.subplots()  
ax.plot(t, rezultatas[0, :], label='Aukštis')  
ax.set\_xlabel('Laikas (s)')  
ax.set\_ylabel('Aukštis (m)')  
ax.legend()  
plt.show()

### 2.2.2 Testavimas

Sprendimas Eulerio metodu bus testuojamas 4 kartus su skirtingais žingsniais

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. | Žingsnis | Kada parašiutininkas pasiekia žemę?(s) | Kokiu greičiu jis pasiekia žemę?(m/s) | Kokiame aukštyje išskleidžiamas parašiutas?(m) |
| 1 | 0.01 | 145.24 | 11.29 | 1374.47 |
| 2 | 0.05 | 145.35 | 11.29 | 1375.04 |
| 3 | 0.1 | 145.60 | 11.29 | 1375.76 |
| 4 | 0.15 | 145.85 | 11.29 | 1372.92 |

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 4 Gauti programos rezultatai

### 2.2.3 Grafikai

Paveikslėlis, kuriame yra tekstas, linija, ekrano kopija, diagrama

Automatiškai sugeneruotas aprašymas

pav. 5 Eulerio aukščio ir laiko priklausomybė

Paveikslėlis, kuriame yra tekstas, ekrano kopija, linija, Grafikas

Automatiškai sugeneruotas aprašymas

pav. 6 Eulerio greičio ir laiko priklausomybė

**Išvada**: keičiant žingsnio dydį grafikas ir rezultatai skiriasi nežymiai, grafikai perdengia vienas kitą.

## Sprendimas IV eilės Rungės ir Kutos metodus

### 2.3.1 Kodas

**IV RK.ipynb:**

import numpy as np  
import matplotlib.pyplot as plt  
  
def parašiutininko\_kritimas(X, t):  
 h, v = X  
 m1 = 120 *# parašiutininko masė (kg)* m2 = 10 *# įrangos masė (kg)* k1 = 0.25 *# oro pasipriešinimo koeficientas laisvo kritimo metu (kg/m)* k2 = 10 *# oro pasipriešinimo koeficientas išskleidus parašiutą (kg/m)* g = 9.81 *# gravitacijos pagreitis (m/s^2)* if t < 25:  
 *# Laisvas kritimas iki parašiuto išsiskleidimo* F = (m1 + m2) \* g - k1 \* v\*\*2 \* np.sign(v)  
 else:  
 *# Parašiutas išsiskleidęs* F = (m1 + m2) \* g - k2 \* v\*\*2 \* np.sign(v)  
  
 dhdt = -v  
 dvdt = F / (m1 + m2)  
  
 return np.array([dhdt, dvdt])  
  
def runge\_kutta\_4(func, y0, t):  
 N = len(t)  
 y = np.zeros((len(y0), N))  
 y[:, 0] = y0  
  
 for i in range(N - 1):  
 h = t[i + 1] - t[i]  
 k1 = h \* func(y[:, i], t[i])  
 k2 = h \* func(y[:, i] + k1 / 2, t[i] + h / 2)  
 k3 = h \* func(y[:, i] + k2 / 2, t[i] + h / 2)  
 k4 = h \* func(y[:, i] + k3, t[i] + h)  
 y[:, i + 1] = y[:, i] + (k1 + 2 \* k2 + 2 \* k3 + k4) / 6  
  
 return y  
  
*# Laiko nustatymai*viso\_laikas = 170 *# Visas skaičiavimo laikas (s)*dt\_values = [0.01, 0.1, 0.5, 1.2] *# Laiko žingsniai Eulerio metodu  
  
# Pradinės sąlygos*h0 = 2800 *# Pradinis aukštis (m)*v0 = 0 *# Pradinis greitis (m/s)  
  
# Braižome rezultatus su skirtingais laiko žingsniais*fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))  
  
for dt in dt\_values:  
 *# Laiko vektorius* t = np.arange(0, viso\_laikas, dt)  
  
 *# Skaičiavimas naudojant Rungės-Kutos metodą* rezultatai\_rk4 = runge\_kutta\_4(parašiutininko\_kritimas, [h0, v0], t)  
  
 *# Analizuojame rezultatus* pasiekimo\_momento\_indeksas = np.argmax(rezultatai\_rk4[0] <= 0) *# Indeksas, kai aukštis tampa neigiamas* pasiekimo\_laikas = t[pasiekimo\_momento\_indeksas]  
 pasiekimo\_greitis = rezultatai\_rk4[1, pasiekimo\_momento\_indeksas]  
  
 išskleidimo\_indeksas = np.argmax((t >= 25) & (rezultatai\_rk4[0] > 0)) *# Indeksas, kai parašiutas išsiskleidžia* išskleidimo\_laikas = t[išskleidimo\_indeksas]  
 išskleidimo\_aukštis = rezultatai\_rk4[0, išskleidimo\_indeksas]  
  
 *# Braižome rezultatus aukščio grafike* ax1.plot(t, rezultatai\_rk4[0, :], label=f'Aukštis (dt = {dt:.2f})')  
  
 *# Braižome rezultatus greičio grafike* ax2.plot(t, rezultatai\_rk4[1, :], label=f'Greitis (dt = {dt:.2f})')  
  
*# Plot nustatymai aukščio grafikui*ax1.set\_xlabel('Laikas (s)')  
ax1.set\_ylabel('Aukštis (m)')  
ax1.legend()  
  
*# Plot nustatymai greičio grafikui*ax2.set\_xlabel('Laikas (s)')  
ax2.set\_ylabel('Greitis (m/s)')  
ax2.legend()  
  
plt.tight\_layout()  
plt.show()

### 2.3.2 Testavimas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nr. | Žingsnis | Kada parašiutininkas pasiekia žemę?(s) | Kokiu greičiu jis pasiekia žemę?(m/s) | Kokiame aukštyje išskleidžiamas parašiutas?(m) |
| 1 | 0.01 | 145.22 | 11.29 | 1374.33 |
| 2 | 0.1 | 145.30 | 11.29 | 1375.33 |
| 3 | 0.5 | 146.50 | 11.29 | 1374.33 |
| 4 | 1.2 | 147.60 | 11.29 | 1360.08 |

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, skaičius

Automatiškai sugeneruotas aprašymas

pav. 7 Programiškai gauti rezultatai

### 2.3.3 Grafikai

Paveikslėlis, kuriame yra tekstas, linija, ekrano kopija, Grafikas

Automatiškai sugeneruotas aprašymas

pav. 8 IV eilės Rungės ir Kutos aukščio ir laiko priklausomybės grafikai

Paveikslėlis, kuriame yra tekstas, ekrano kopija, linija, Grafikas

Automatiškai sugeneruotas aprašymas

pav. 9 IV eilės Rungės ir Kutos greičio ir laiko priklausomybės grafikai

**Išvada**: keičiant žingsnio dydį grafikas ir rezultatai skiriasi nežymiai, grafikai perdengia vienas kitą. Tačiau kai žingsnis didesnis 1.20 jau matosi nuokrypis.

## Tikrinimas su solve\_ivp

### 2.4.1 Tikrinimo kodas

**Tikrinimas.ipynb:**

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.integrate import solve\_ivp  
  
def parašiutininko\_kritimas(t, X):  
 h, v = X  
 m1 = 120 *# parašiutininko masė (kg)* m2 = 10 *# įrangos masė (kg)* k1 = 0.25 *# oro pasipriešinimo koeficientas laisvo kritimo metu (kg/m)* k2 = 10 *# oro pasipriešinimo koeficientas išskleidus parašiutą (kg/m)* g = 9.81 *# gravitacijos pagreitis (m/s^2)* if t < 25:  
 *# Laisvas kritimas iki parašiuto išsiskleidimo* F = (m1 + m2) \* g - k1 \* v\*\*2 \* np.sign(v)  
 else:  
 *# Parašiutas išsiskleidęs* F = (m1 + m2) \* g - k2 \* v\*\*2 \* np.sign(v)  
  
 dhdt = -v  
 dvdt = F / (m1 + m2)  
  
 return [dhdt, dvdt]  
  
*# Laiko nustatymai*viso\_laikas = 146 *# Visas skaičiavimo laikas (s)  
  
# Pradinės sąlygos*h0 = 2800 *# Pradinis aukštis (m)*v0 = 0 *# Pradinis greitis (m/s)  
  
# Laiko vektorius*t\_span = (0, viso\_laikas)  
  
*# Skaičiavimas naudojant solve\_ivp*rezultatai\_ivp = solve\_ivp(parašiutininko\_kritimas, t\_span, [h0, v0], t\_eval=np.linspace(0, viso\_laikas, 1000), method='RK45')  
  
*# Analizuojame rezultatus*pasiekimo\_momento\_indeksas = np.argmax(rezultatai\_ivp.y[0] <= 0) *# Indeksas, kai aukštis tampa neigiamas*pasiekimo\_laikas = rezultatai\_ivp.t[pasiekimo\_momento\_indeksas]  
pasiekimo\_greitis = rezultatai\_ivp.y[1, pasiekimo\_momento\_indeksas]  
  
išskleidimo\_indeksas = np.argmax((rezultatai\_ivp.t >= 25) & (rezultatai\_ivp.y[0] > 0)) *# Indeksas, kai parašiutas išsiskleidžia*išskleidimo\_laikas = rezultatai\_ivp.t[išskleidimo\_indeksas]  
išskleidimo\_aukštis = rezultatai\_ivp.y[0, išskleidimo\_indeksas]  
  
*# Spausdiname rezultatus*print(f"Parašiutininkas pasiekia žemę laiko t = {pasiekimo\_laikas:.2f} s metu su greičiu {pasiekimo\_greitis:.2f} m/s.")  
print(f"Parašiutas išsiskleidžiamas laiko t = {išskleidimo\_laikas:.2f} s metu, esant aukštyje {išskleidimo\_aukštis:.2f} m.")  
  
*# Braižome rezultatus*fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))  
  
*# Aukščio grafikas*ax1.plot(rezultatai\_ivp.t, rezultatai\_ivp.y[0], label='Aukštis (solve\_ivp)')  
ax1.set\_xlabel('Laikas (s)')  
ax1.set\_ylabel('Aukštis (m)')  
ax1.legend()  
  
*# Greičio grafikas*ax2.plot(rezultatai\_ivp.t, rezultatai\_ivp.y[1], label='Greitis (solve\_ivp)')  
ax2.set\_xlabel('Laikas (s)')  
ax2.set\_ylabel('Greitis (m/s)')  
ax2.legend()  
  
plt.tight\_layout()  
plt.show()

### 2.4.2 Tikrinimo grafikai

Paveikslėlis, kuriame yra linija, Grafikas, diagrama, ekrano kopija

Automatiškai sugeneruotas aprašymas

pav. 10 Tikrinimas su solve\_ivp aukščio ir laiko priklausomybė

Paveikslėlis, kuriame yra linija, ekrano kopija, Grafikas, diagrama

Automatiškai sugeneruotas aprašymas

pav. 11 Tikrinimas su solve\_ivp greičio ir laiko priklausomybė

**Išvada**: naudojant Pyhton scipy.integrate bibliotekos funkciją solve\_ivp buvo įsitikinta, jog buvo gauti teisingi rezultatai.

## **Bendros išvados**

**Išvada**: Darant šį darbą buvo įsisavinti paprastųjų diferencialinių lygčių sudarymo ir sprendimo metodai. Buvo prisimintas fizikos kursas. Išnagrinėti Eulerio ir 4 eilės Rungės ir Kutos metodai. Nustatytas pastarojo pranašumas tikslumo atžvilgiu prieš Eulerio metodą. To buvo galima tikėtis, nes skaičiuojant sprendinį 4 eilės Rungės ir Kutos metodu yra daromi papildomi tarpiniai skaičiavimai. Taip pat panaudota Pyhton scipy.integrate bibliotekos funkcija solve\_ivp patikrinti sprendinių tikslumą.

# Literatūros sąrašas

1. „Skaitiniai metodai ir algoritmai“ „Moodle“ aplinkoje [HTTPS://MOODLE.KTU.EDU/COURSE/VIEW.PHP?ID=7639](https://moodle.ktu.edu/course/view.php?id=7639)