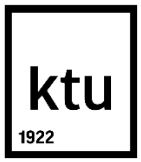


Kauno technologijos universitetas
Informatikos fakultetas

Skaitiniai metodai ir algoritmai

Paprastųjų diferencialinių lygčių sprendimas

Vytenis Kriščiūnas IFF-1/1

Studentas

doc. Kriščiūnas Andrius

Dėstytojas

Kaunas 2023

TURINYS

1. Užduotis

- Žemiau pateikti uždaviniai paprastųjų diferencialinių lygčių sistemų sprendimui. Remdamiesi tame pačiame faile pateiktų fizikinių dėsnių aprašymais, nurodytam variantui sudarykite diferencialinę lygtį arba lygčių sistemą. Lygties ar lygčių sistemos sudarymą paaiškinkite ataskaitoje.
- Diferencialinę lygtį (arba lygčių sistemą) išspręskite Eulerio ir IV eilės Rungės ir Kutos metodais.
- Keisdami metodo žingsnį įsitikinkite, kad gavote tikslų sprendinį. Atsakykite į uždavinyje pateiktus klausimus. Tuo pačiu metodu naudojant skirtingus žingsnius gautus sprendinius pavaizduokite viename grafike. Palyginkite metodus tikslumo prasme.
- Keisdami metodo žingsnį nustatykite didžiausią žingsnį, su kuriuo metodas išlieka stabilus. Tuo pačiu metodu naudojant skirtingus žingsnius gautus sprendinius pavaizduokite viename grafike. Palyginkite metodus stabilumo prasme.
- 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.

3 Uždavinys variantams 1-10

Sujungti m_1 ir m_2 masių objektai iššaunami vertikaliai į viršų pradiniu greičiu v_0. Oro pasipriešinimo koeficientas sujungtiems kūnams lygus k_s. Praėjus laikui t_s, objektai pradeda judėti atskirai. Oro pasipriešinimo koeficientai atskirai judantiems objektams atitinkamai yra k_1 ir k_2. Oro pasipriešinimas proporcingas objekto greičio kvadratui. Raskite, kaip kinta objektų greičiai nuo 0 s iki t_{max}. Kada kiekvienas objektas pasieks aukščiausią tašką ir pradės leistis?

1 Lentelė. Uždavinyje naudojami dydžiai.

Varianto numeris	m_1, kg	m_2, kg	v_0, m/s	k_s, kg/m	t_s, s	k_1, kg/m	k_2, kg/m	t_{max}, s
1	0,2	0,4	80	0,015	1	0,02	0,005	15
2	0,15	0,2	70	0,01	2	0,05	0,001	10
3	0,07	0,2	50	0,015	3	0,05	0,01	10
4	0,5	0,25	100	0,002	2	0,02	0,04	15
5	0,6	0,2	200	0,01	2	0,02	0,015	15
6	0,1	0,5	60	0,01	1	0,01	0,005	10
7	0,3	0,3	60	0,005	2	0,05	0,01	10
8	0,05	0,3	100	0,01	3	0,05	0,01	10
9	0,4	0,8	50	0,001	2	0,02	0,02	10
10	0,8	0,8	200	0,01	2	0,02	0,005	15

2. Teorinė dalis

2.1. Sudaryta lygtis

Jeigu t <= ts, tai $\frac{dv}{dt} = \frac{-g*(m1+m2)-sgn(v)*k_s*v^2}{m1+m2}$

Jeigu t > ts, tai $\frac{dv_1}{dt} = \frac{-g*m1-sgn(v_1)*k_1*{v_1}^2}{m1}; \frac{dv_2}{dt} = \frac{-g*m2-sgn(v_2)*k_2*{v_2}^2}{m2}$

Sudariau lygtį remdamasis sąlyga, kad kūnai juda kartu iki laiko ts ir vėliau atsiskiria ir juda atskirai iki laiko tmax. Rėmiausi Niutono dėsniais, kai kūnai juda kartu į viršų juos veikia gravitacijos jėga nukreipta žėmyn, o oro pasipriešinimo jėga visada veikia priešingai kūnų judėjimo krypčiai. Kai kūnai pradeda lesitis, gravitacijos jėga juos veikia judėjimo kryptimi. Taigi, reikėjo apskaičiuoti pirmąją kelio išvestinę pagal laiką, kad būtų galima rasti kada kūnai pasieks aukščiausią tašką ir pradės leistis.

2.2. Diferencialinės lygtys Eulerio ir IV Rungės ir Kutos metodais

Eulerio metodas:

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

g = 9.8 #laisvo kritimo pagreitis

m1 = 0.4 #pirmojo mase
m2 = 0.8 #antrojo mase
v0 = 50 #pradinis greitis
ks = 0.001 #abeju kunu koeficinetas
ts = 2 #laikas iki kurio juda abu kunai

k1 = 0.02 #pirmo kuno koeficinetas
k2 = 0.02 #antro kuno koeficinetas
tmax = 10 #maksimalus laikas

def calculate(deltat):
    v1 = 0
    v2 = 0
    t1 = 0
    t2 = 0
```

```
    v = v0
    t = 0
    times = []
    speeds1 = []
    speeds2 = []
    while (t <= tmax):
        if t <= ts:
            v += deltat * ((-g*(m1 + m2) - np.sign(v)*ks*v**2)/(m1+m2))
            v1 = v
            v2 = v
        else:
            v1 += deltat * ((-g*m1 - np.sign(v1)*k1*v1**2)/m1)
            v2 += deltat * ((-g*m2 - np.sign(v2)*k2*v2**2)/m2)
            if v1 <= 0 and t1 == 0:
                t1 = t
                print(f"Pirmas objektas pasieks auksciausia taska po: {t1} sec.")
            if v2 <= 0 and t2 == 0:
                t2 = t
                print(f"Antras objektas pasieks auksciausia taska po: {t2} sec.")
        times.append(t)
        speeds1.append(v1)
        speeds2.append(v2)
        t += deltat

   # Plot the results
   # plt.plot(times, speeds1, label="Object 1")
   # plt.plot(times, speeds2, label="Object 2")
   # plt.xlabel("Time (s)")
   # plt.ylabel("Speed (m/s)")
   # plt.legend()
   # plt.show()
    return times, speeds1, speeds2
```

```
calculate(0.01)
```

IV Rungės ir Kutos metodas:

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

g = 9.8 #laisvo kritimo pagreitis

m1 = 0.4 #pirmojo mase
```

```
m2 = 0.8 #antrojo mase
v0 = 50 #pradinis greitis
ks = 0.001 #abeju kunu koeficinetas
ts = 2 #laikas iki kurio juda abu kunai

k1 = 0.02 #pirmo kuno koeficinetas
k2 = 0.02 #antro kuno koeficinetas
tmax = 10 #maksimalus laikas

def dvdt(t, v1, v2):
    if t <= ts:
        v1 = (-g*(m1 + m2) - np.sign(v1)*ks*v1**2)/(m1+m2)
        v2 = v1

    else:
        v1 = (-g*m1 - np.sign(v1)*k1*v1**2)/m1
        v2 = (-g*m2 - np.sign(v2)*k2*v2**2)/m2

    return v1, v2

def calculate(deltat):
    v1 = 0
    v2 = 0
    t1 = 0
    t2 = 0

    v = v0
    t = 0
    times = []
    speeds1 = []
    speeds2 = []
    while (t <= tmax):
        if t <= ts:
            v += deltat * dvdt(t, v, 0)[0]
            func1 = v + (deltat / 2) * dvdt(t, v, 0)[0]
            func2 = v + (deltat / 2) * dvdt(t + (deltat / 2), func1, 0)[0]
            func3 = v + deltat * dvdt(t + (deltat / 2), func2, 0)[0]
            finalFunc = v + (deltat / 6) * (dvdt(t, v, 0)[0] + (2 * dvdt(t +
(deltat / 2), func1, 0)[0]) + (2 * dvdt(t + (deltat / 2), func2, 0)[0]) + dvdt(t
+ deltat, func3, 0)[0])
            finalFunc1 = finalFunc
            finalFunc2 = finalFunc
            v1 = v
            v2 = v
        else:
```

```
            v1 += deltat * dvdt(t, v1, 0)[0]
            v2 += deltat * dvdt(t, 0, v2)[1]
            func11 = v1 + (deltat / 2) * dvdt(t, v1, 0)[0]
            func12 = v2 + (deltat / 2) * dvdt(t, 0, v2)[1]
            func21 = v1 + (deltat / 2) * dvdt(t + (deltat / 2), func11, 0)[0]
            func22 = v2 + (deltat / 2) * dvdt(t + (deltat / 2), 0, func12)[1]
            func31 = v1 + deltat * dvdt(t + (deltat / 2), func21, 0)[0]
            func32 = v2 + deltat * dvdt(t + (deltat / 2), 0, func22)[1]
            finalFunc1 = v1 + (deltat / 6) * (dvdt(t, v1, 0)[0] + (2 * dvdt(t +
(deltat / 2), func11, 0)[0]) + (2 * dvdt(t + (deltat / 2), func21, 0)[0]) +
dvdt(t + deltat, func31, 0)[0])
            finalFunc2 = v2 + (deltat / 6) * (dvdt(t, 0, v2)[1] + (2 * dvdt(t +
(deltat / 2), 0, func12)[1]) + (2 * dvdt(t + (deltat / 2), 0, func22)[1]) +
dvdt(t + deltat, 0, func32)[1])

            if finalFunc1 <= 0 and t1 == 0:
                t1 = t
                print(f"Pirmas objektas pasieks auksciausia taska po: {t1} sec.")
            if finalFunc2 <= 0 and t2 == 0:
                t2 = t
                print(f"Antras objektas pasieks auksciausia taska po: {t2} sec.")
        speeds1.append(finalFunc1)
        speeds2.append(finalFunc2)
        times.append(t)
        t += deltat

   # Plot the results
   # plt.plot(times, speeds1, label="Object 1")
   # plt.plot(times, speeds2, label="Object 2")
   # plt.xlabel("Time (s)")
   # plt.ylabel("Speed (m/s)")
   # plt.legend()
   # plt.show()
    return times, speeds1, speeds2
```

```
calculate(0.01)
```

2.3. Tikslaus sprendinio gavimas

Eulerio metodas:

```
def zingsnioKeitimas(prad, galas, step):
```

```
    zingsniai = []
    pirmoObjgr = []
    antroObjgr = []

    for deltat in np.arange(prad, galas, step):
        times, v1, v2 = calculate(deltat)
        zingsniai.append(times)
        pirmoObjgr.append(v1)
        antroObjgr.append(v2)

    for i in range(len(zingsniai)):
        plt.plot(zingsniai[i], pirmoObjgr[i], label=f"Pirmas objektas - Δt,
iteracija {i}, zingsnis {prad + i * step}")
        plt.plot(zingsniai[i], antroObjgr[i], label=f"Antras objektas - Δt,
iteracija {i}, zingsnis {prad + i * step} ")


    plt.xlabel("Laikas")
    plt.ylabel("Greitis")
    plt.legend()
    plt.show()
```

```
zingsnioKeitimas(0.05, 0.1, 0.01) #3 salyga
```

IV Rungės ir Kutos metodas:

```
def zingsnioKeitimas(prad, galas, step):
    zingsniai = []
    pirmoObjgr = []
    antroObjgr = []

    for deltat in np.arange(prad, galas, step):
        times, v1, v2 = calculate(deltat)
        zingsniai.append(times)
        pirmoObjgr.append(v1)
        antroObjgr.append(v2)

    for i in range(len(zingsniai)):
        plt.plot(zingsniai[i], pirmoObjgr[i], label=f"Pirmas objektas - Δt,
iteracija {i}, zingsnis {prad + i * step}")
        plt.plot(zingsniai[i], antroObjgr[i], label=f"Antras objektas - Δt,
iteracija {i}, zingsnis {prad + i * step} ")
```

```
    plt.xlabel("Laikas")
    plt.ylabel("Greitis")
    plt.legend()
    plt.show()
```

```
zingsnioKeitimas(0.05, 0.1, 0.01) #3 salyga
```

2.4. Didžiausio žingsnio radimas

Eulerio metodas:

```
zingsnioKeitimas(1, 2, 0.1) #4 salyga
```

IV Rungės ir Kutos metodas:

```
zingsnioKeitimas(1, 2, 0.1) #4 salyga
```

2.5. Sprendinio patikrinimas

Eulerio metodas:

```
def system(t, y):
    v1, v2 = y
    if t <= ts:
        dv1dt = (-g * (m1 + m2) - np.sign(v1) * ks * v1 ** 2) / (m1 + m2)
        dv2dt = (-g * (m1 + m2) - np.sign(v2) * ks * v2 ** 2) / (m1 + m2)
    else:
        dv1dt = (-g * m1 - np.sign(v1) * k1 * v1 ** 2) / m1
        dv2dt = (-g * m2 - np.sign(v2) * k2 * v2 ** 2) / m2
    return [dv1dt, dv2dt]

def calculate_with_solve_ivp(dz):
    y0 = [v0, v0]  # pradiniai greičiai
    t_span = (0, tmax)
    t_eval = np.arange(0, tmax, dz)  # Šis sąrašas yra jūsų norimas žingsnių
dydis

    sol = solve_ivp(system, t_span, y0, t_eval=t_eval, method='RK45')

    return sol.t, sol.y[0], sol.y[1]

def compare_methods(deltat):
```

```
    times_euler, speeds1_euler, speeds2_euler = calculate(deltat)
    times_solve_ivp, speeds1_solve_ivp, speeds2_solve_ivp =
calculate_with_solve_ivp(deltat)

    plt.plot(times_euler, speeds1_euler, label="Mano objektas 1")
    plt.plot(times_euler, speeds2_euler, label="Mano objektas 2")
    plt.plot(times_solve_ivp, speeds1_solve_ivp, label="solve_ivp objektas 1")
    plt.plot(times_solve_ivp, speeds2_solve_ivp, label="solve_ivp objektas 2")

    plt.xlabel("Laikas (s)")
    plt.ylabel("Greitis (m/s)")
    plt.legend()
    plt.show()
zingsnioKeitimas(1, 2, 0.1) #4 salyga
```

IV Rungės ir Kutos metodas:

```
def system(t, y):
    v1, v2 = y
    if t <= ts:
        dv1dt = (-g * (m1 + m2) - np.sign(v1) * ks * v1 ** 2) / (m1 + m2)
        dv2dt = (-g * (m1 + m2) - np.sign(v2) * ks * v2 ** 2) / (m1 + m2)
    else:
        dv1dt = (-g * m1 - np.sign(v1) * k1 * v1 ** 2) / m1
        dv2dt = (-g * m2 - np.sign(v2) * k2 * v2 ** 2) / m2
    return [dv1dt, dv2dt]

def calculate_with_solve_ivp(dz):
    y0 = [v0, v0]  # pradiniai greičiai
    t_span = (0, tmax)
    t_eval = np.arange(0, tmax, dz)  # Šis sąrašas yra jūsų norimas žingsnių
dydis

    sol = solve_ivp(system, t_span, y0, t_eval=t_eval, method='RK45')

    return sol.t, sol.y[0], sol.y[1]

def compare_methods(deltat):
    times_euler, speeds1_euler, speeds2_euler = calculate(deltat)
    times_solve_ivp, speeds1_solve_ivp, speeds2_solve_ivp =
calculate_with_solve_ivp(deltat)

    plt.plot(times_euler, speeds1_euler, label="Mano objektas 1")
    plt.plot(times_euler, speeds2_euler, label="Mano objektas 2")
    plt.plot(times_solve_ivp, speeds1_solve_ivp, label="solve_ivp objektas 1")
```

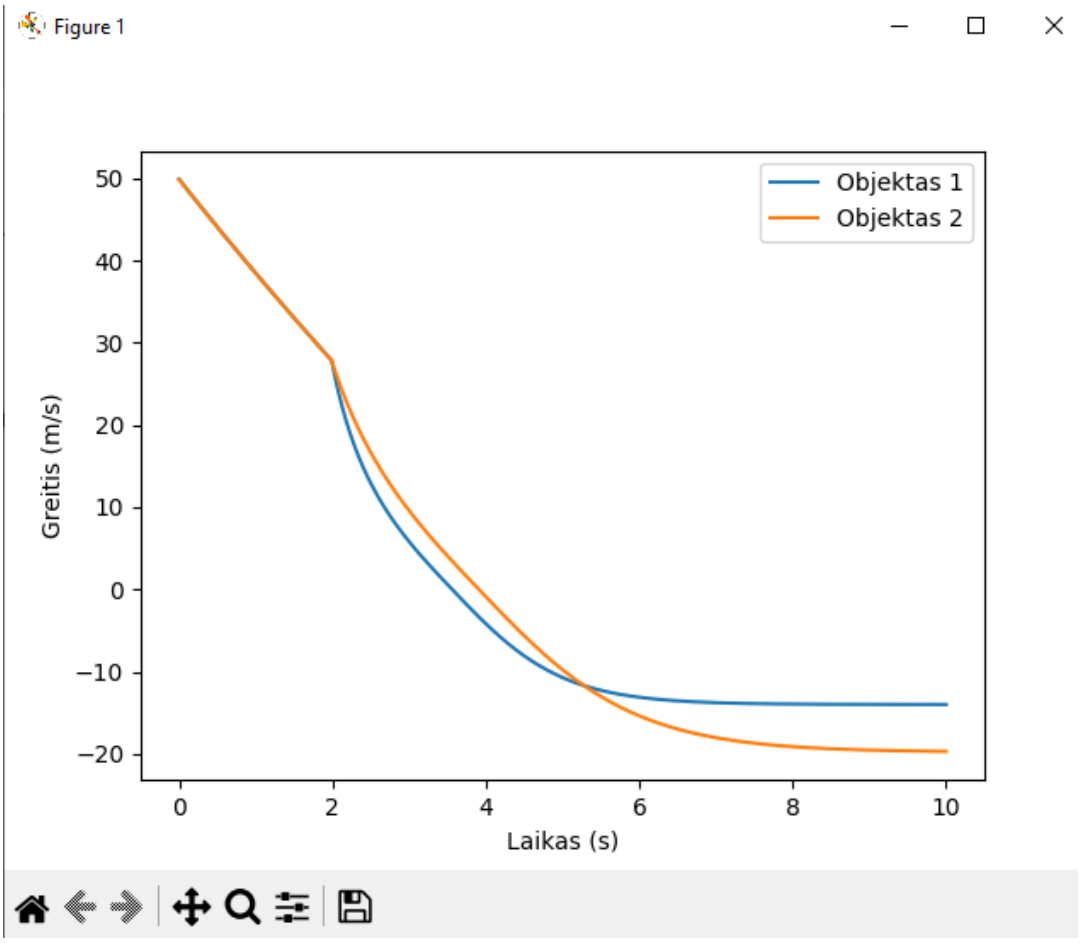
```
    plt.plot(times_solve_ivp, speeds2_solve_ivp, label="solve_ivp objektas 2")

    plt.xlabel("Laikas (s)")
    plt.ylabel("Greitis (m/s)")
    plt.legend()
    plt.show()
```

```
zingsnioKeitimas(0.05, 0.1, 0.01) #3 salyga
```

3. Gauti rezultatai

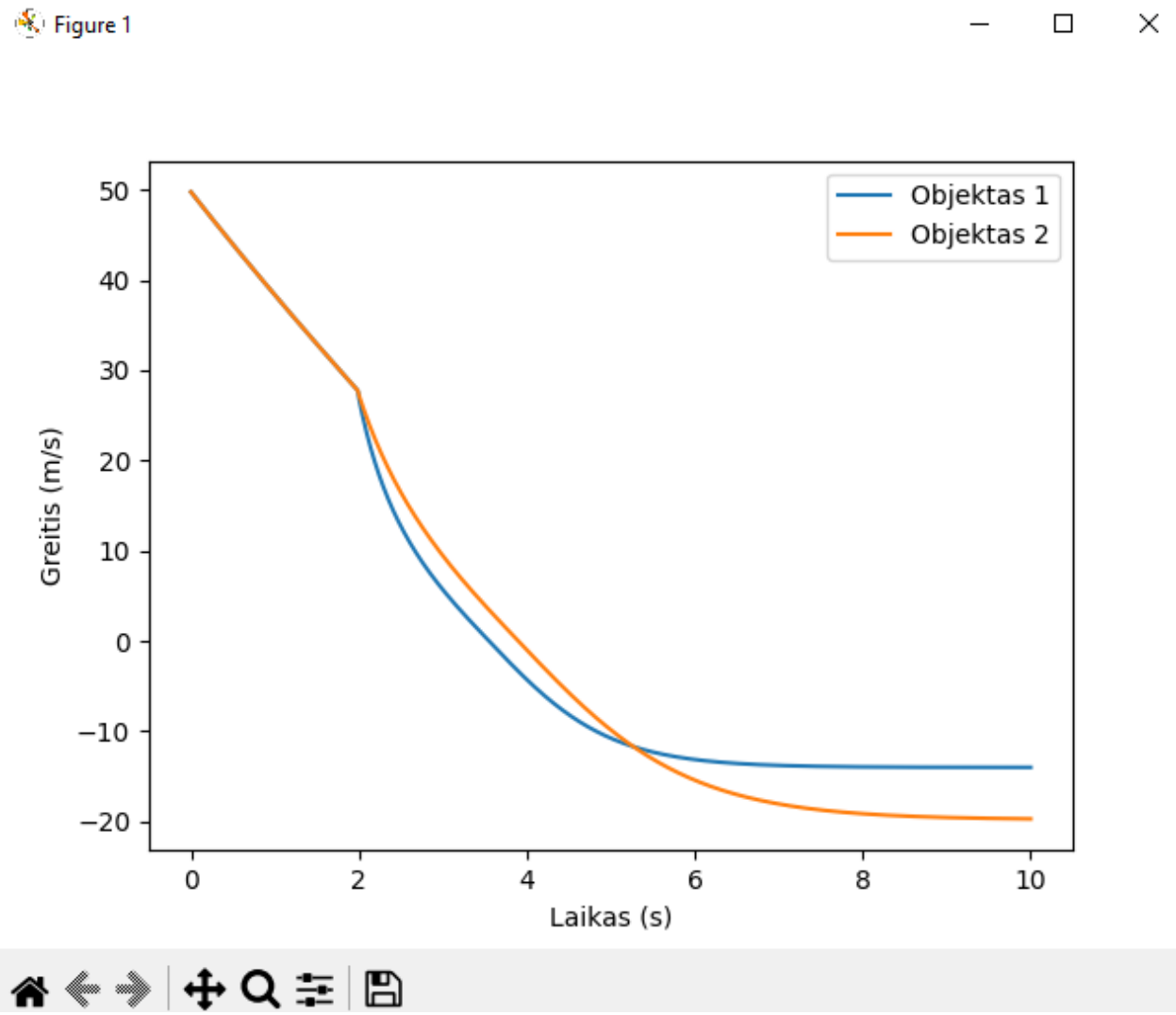
3.1. Gautų diferencialinių lygčių grafikai ir laikai naudojant abu metodus

Eulerio metodas:




```
Pirmas objektas pasieks auksciausia taska po: 3.569999999999968 sec.
Antras objektas pasieks auksciausia taska po: 3.9099999999999606 sec.
```

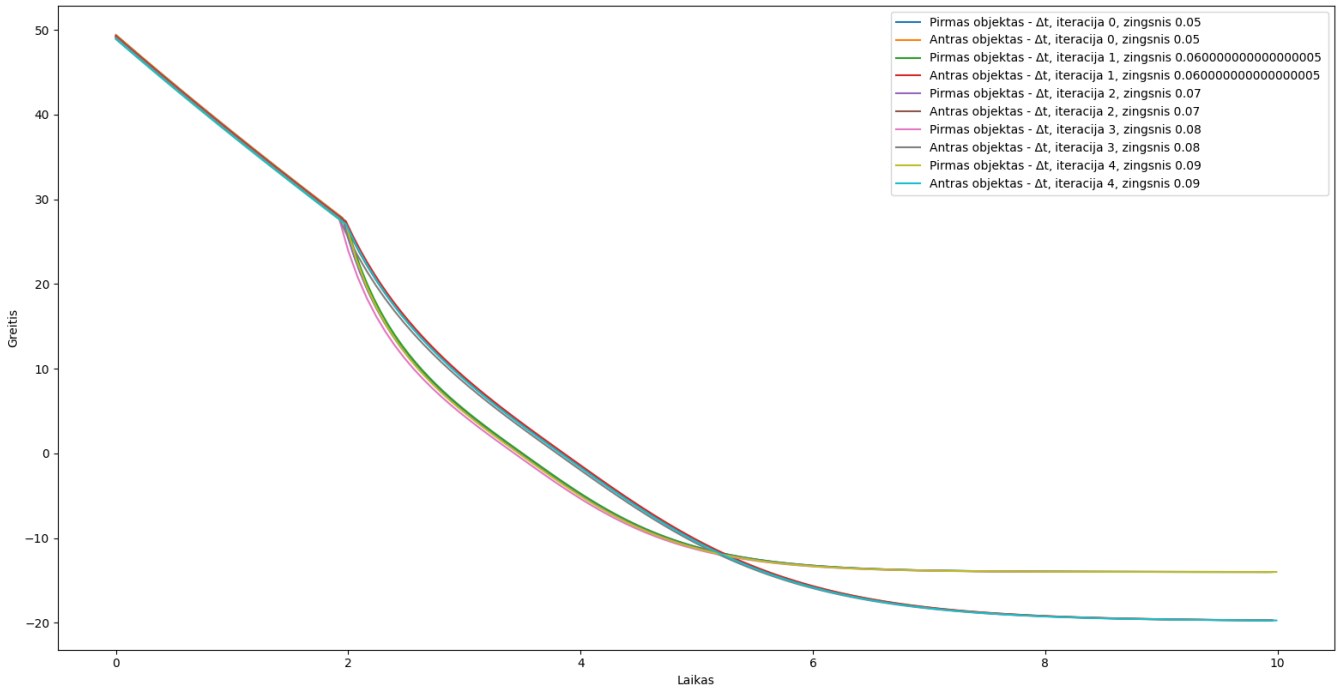
IV Rungės ir Kutos metodas:



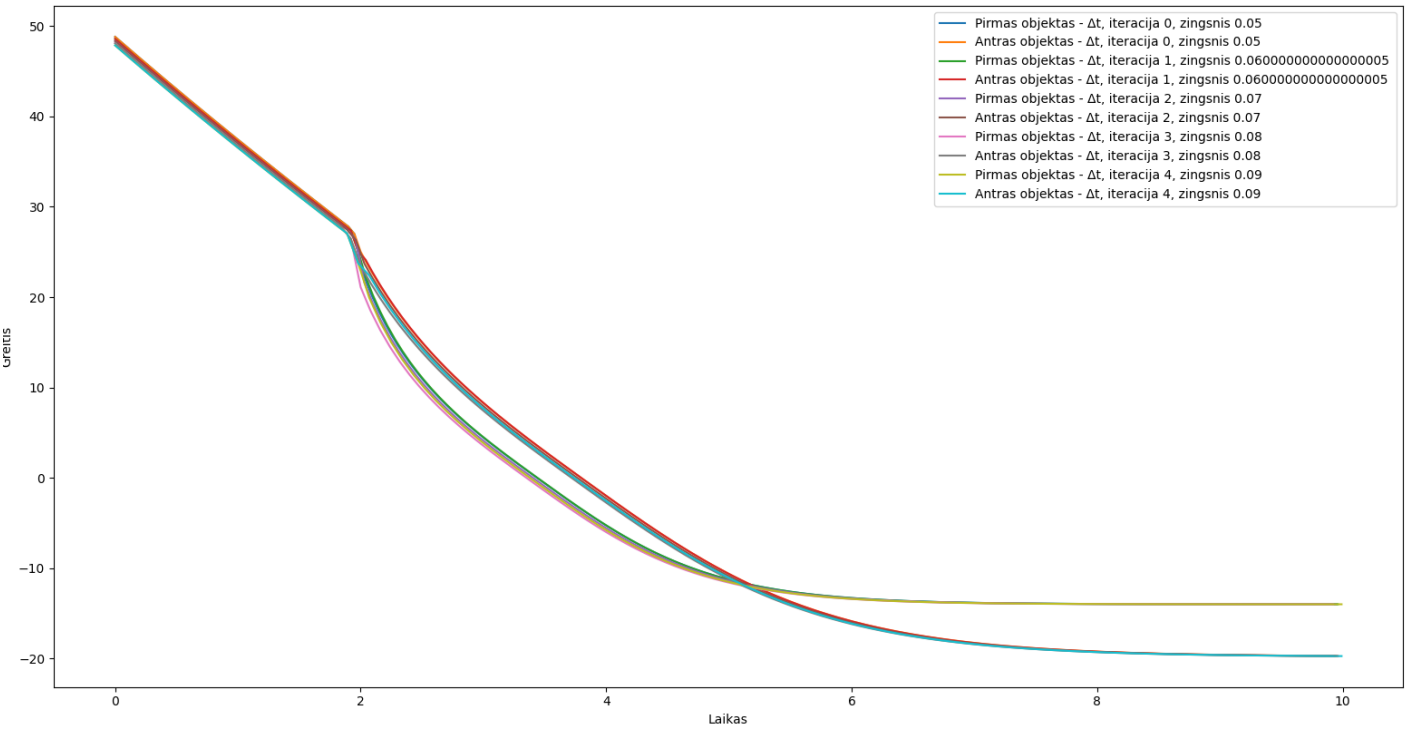

```
Pirmas objektas pasieks auksciausia taska po: 3.559999999999968 sec.
Antras objektas pasieks auksciausia taska po: 3.899999999999961 sec.
```

3.2. Tikslaus sprendinio gavimo grafikai naudojant abu metodus

Eulerio metodas:



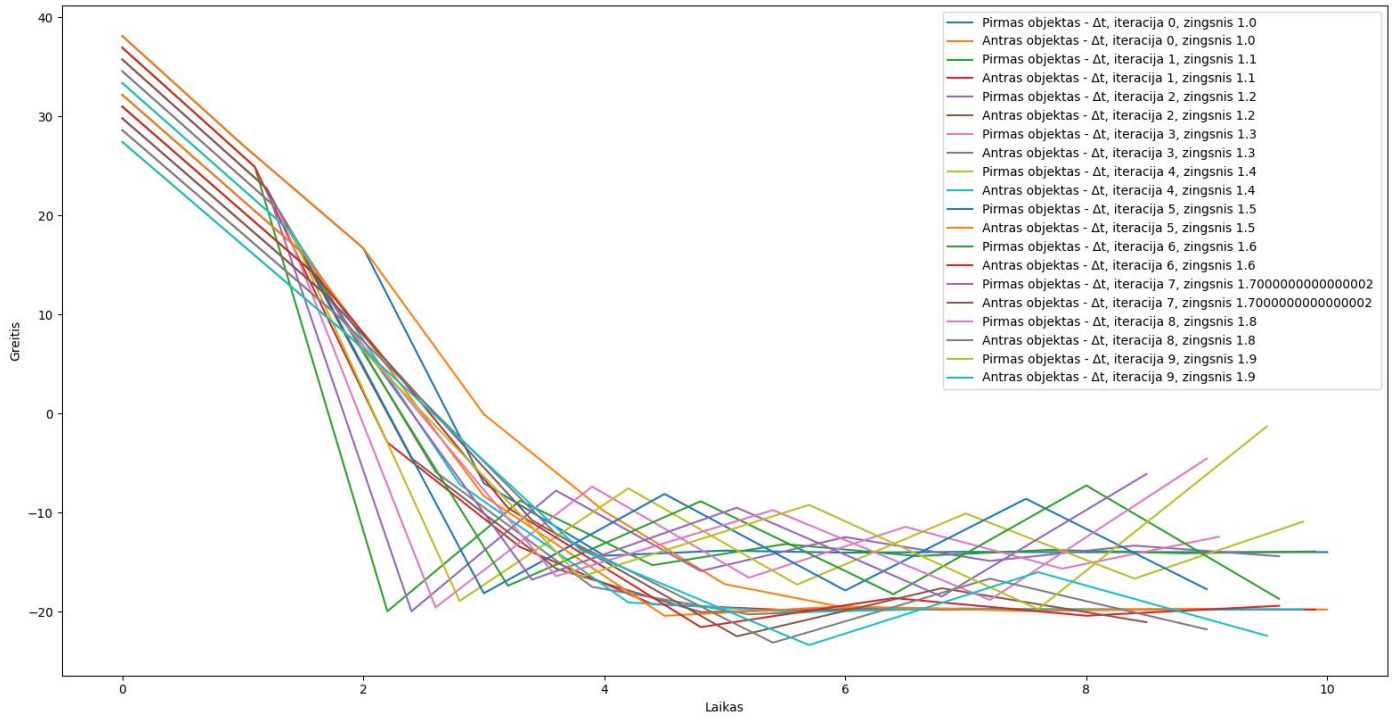
IV Rungės ir Kutos metodas:



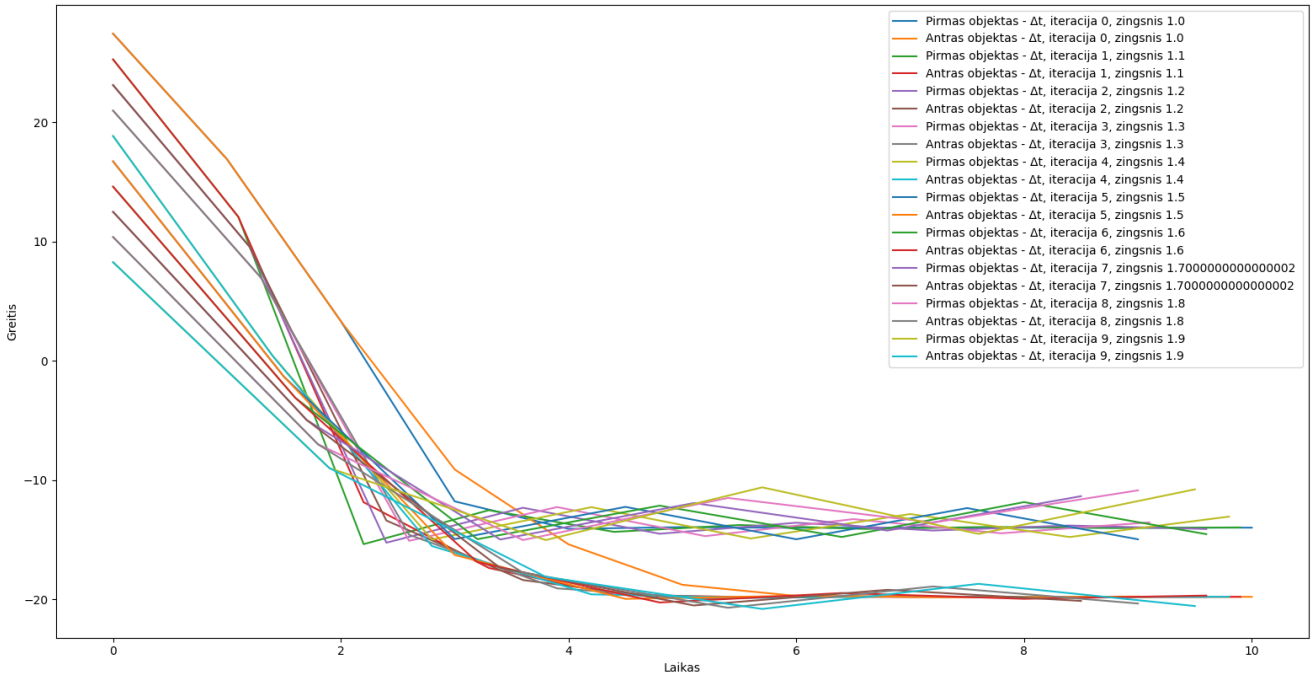
Metodų grafikai gavosi labai panašūs, todėl pasakyti, kuris metodas gauna tikslesnį įvertį sunku.

3.3. Didžiausio žingsnio radimo grafikai naudojant abu metodus

Eulerio metodas:



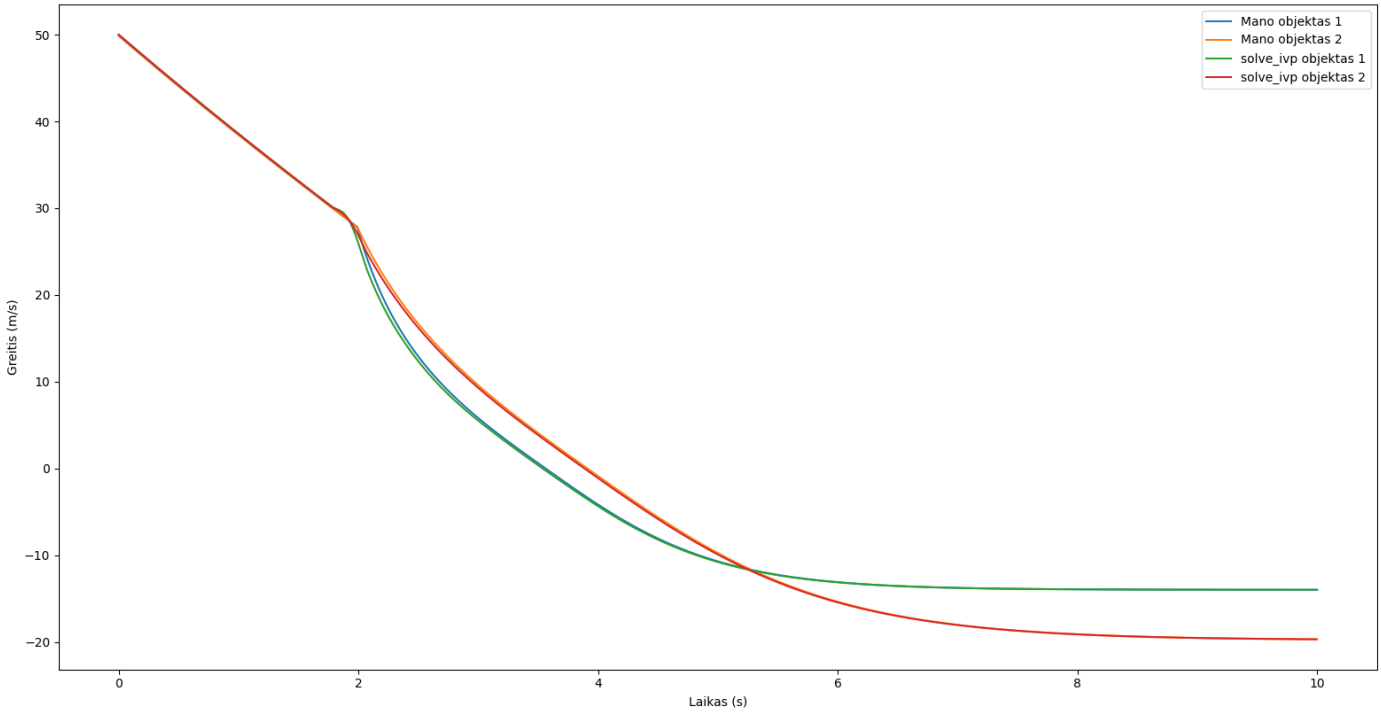
IV Rungės ir Kutos metodas:



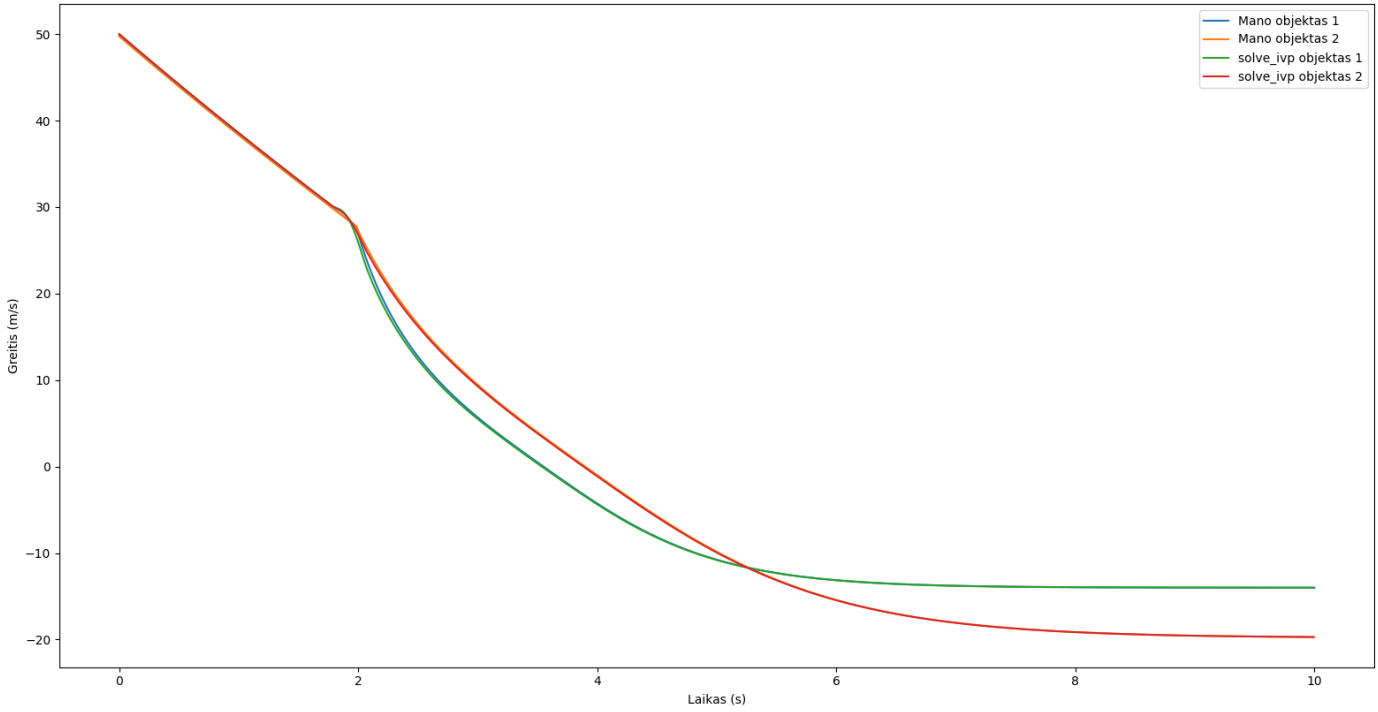
Galima pastebėti akyvaizdų skirtumą tarp metodų: IV Rungės ir Kutos metodas gauna stabilius sprendinius, kurie tarpusavyja yra pakankamai glaudūs. Vis dėlto, abejų metodų didžiausi žingsniai gali būti iki skaičiaus: 2, nes poto gaunasi perpildymo situacija, funkcijų reikšmės tampa labai didelės.

3.4. Sprendinio patikrinimo gauti grafikai

Eulerio metodas:



IV Rungės ir Kutos metodas:



Pasitelkus Python scipy.integrate biblioteką galima pastebėti, kad abejų metodų grafikuose braižyti sprendiniai yra labai panašūs į išorinių sprendimo išteklių gautus sprendinius. Šiek tiek tikslesnis Python bibliotekos įrankui gavosi IV Rungės ir Kutos metodas.