

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 3](#_Toc153395151)

[2. Teorinė dalis 3](#_Toc153395152)

[2.1. Sudaryta lygtis 4](#_Toc153395153)

[2.2. Diferencialinės lygtys Eulerio ir IV Rungės ir Kutos metodais 4](#_Toc153395154)

[2.3. Tikslaus sprendinio gavimas 7](#_Toc153395155)

[2.4. Didžiausio žingsnio radimas 9](#_Toc153395156)

[2.5. Sprendinio patikrinimas 9](#_Toc153395157)

[3. Gauti rezultatai 11](#_Toc153395158)

[3.1. Gautų diferencialinių lygčių grafikai ir laikai naudojant abu metodus 11](#_Toc153395159)

[3.2. Tikslaus sprendinio gavimo grafikai naudojant abu metodus 12](#_Toc153395160)

[3.3. Didžiausio žingsnio radimo grafikai naudojant abu metodus 14](#_Toc153395161)

[3.4. Sprendinio patikrinimo gauti grafikai 15](#_Toc153395162)

# 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.

A table with numbers and a black text

Description automatically generated

# Teorinė dalis

## Sudaryta lygtis

Jeigu t <= ts, tai

Jeigu t > ts, tai

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.

## 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)

## 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

## Didžiausio žingsnio radimas

Eulerio metodas:

zingsnioKeitimas(1, 2, 0.1) #4 salyga

IV Rungės ir Kutos metodas:

zingsnioKeitimas(1, 2, 0.1) #4 salyga

## 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

# Gauti rezultatai

## Gautų diferencialinių lygčių grafikai ir laikai naudojant abu metodus

Eulerio metodas:

A graph with a line

Description automatically generated



IV Rungės ir Kutos metodas:

A screen shot of a graph

Description automatically generated



## Tikslaus sprendinio gavimo grafikai naudojant abu metodus

Eulerio metodas:

A graph of a number of colored lines

Description automatically generated with medium confidence

IV Rungės ir Kutos metodas:

A graph of a line

Description automatically generated with medium confidence

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

## Didžiausio žingsnio radimo grafikai naudojant abu metodus

Eulerio metodas:

A graph of colored lines

Description automatically generated

IV Rungės ir Kutos metodas:

A graph of different colored lines

Description automatically generated

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.

## Sprendinio patikrinimo gauti grafikai

Eulerio metodas:

A graph of a line

Description automatically generated with medium confidence

IV Rungės ir Kutos metodas:

A line graph with red green and blue lines

Description automatically generated

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.