# KAUNO TECHOLOGIJOS UNIVERSITETAS

# INFORMATIKOS FAKULTETAS

# KOMPIUTERIŲ KATEDRA

**Skaitiniai metodai ir algoritmai (P170B115)**

Laboratorinis darbas nr. 1

Varianto nr. 9

Atliko:

IFF 8/3 gr. studentas

Dovydas Zamas

Priėmė:

doc. Čalnerytė Dalia

# 1. Turinys

# 1. Uždavinys

## 1.1. Netiesinių lygčių sprendimas

**1.** Išspręskite netiesines lygtis (1 ir 2 lentelės):

a) daugianaris $f(x)$=0;

b) transcendentinė funkcija $g(x)$=0.

1. (tik lygčiai su daugianariu $f(x)$) Nustatykite daugianario $f(x)$ šaknų intervalą, taikydami „grubų“ ir tikslesnį įverčius. Grafiškai pavaizduokite apskaičiuotų šaknų intervalo galus.

2. Daugianarį $f(x)$ grafiškai pavaizduokite nustatytame šaknų intervale. Grafiko ašis pakeiskite taip, kad būtų aiškiai matomos daugianario šaknys. Funkciją $g(x)$ grafiškai pavaizduokite užduotyje nurodytame intervale.

3. Naudodami skenavimo algoritmą su nekintančiu skenavimo žingsniu atskirkite šaknų intervalus. Daugianariui skenavimo intervalas parenkamas pagal įverčių reikšmes, funkcija skenuojama užduotyje nurodytame intervale. Šaknies atskyrimo intervalai naudojami kaip pradiniai intervalai (artiniai) šaknų tikslinimui.

4. Skenavimo metodu atskirtas daugianario ir funkcijos šaknis tikslinkite užduotyje nurodytais

metodais. Užrašykite skaičiavimų pabaigos sąlygas. Skaičiavimų rezultatus pateikite lentelėje, kurioje nurodykite šaknies tikslinimui naudojamą metodą, pradinį artinį ar intervalą, gautą sprendinį (šaknį), tikslumą, iteracijų skaičių. Palyginkite, kuris metodas randa sprendinį su mažesniu iteracijų skaičiumi.

5. Gautas šaknų reikšmes patikrinkite naudodami išorinius išteklius (pvz., MATLAB funkcijas roots arba fzero, tinklapį wolframalpha.com ir t.t.).

## 1.2. Žodinio uždavinio sprendimas

Pagal pateiktą uždavinio sąlygą (3 lentelė) sudarykite netiesinę lygtį ir pasirinktu skaitiniu metodu iš 1 lentelės ją išspręskite. Ataskaitoje pateikite pradinius metodo parametrus (metodo žingsnį, pradinį artinį, izoliacijos intervalą ir pan.), iteracijų pabaigos sąlygą, tikslumą, gautą lygties sprendinį ir sudarytos funkcijos reikšmę, argumentus, kodėl pasirinkote šį metodą. Pateikite grafinį lygties sprendimą.

**lentelė 1. Netiesinių lygčių sprendimas. Metodai.**

| Metodo Nr. | Metodo pavadinimas |
|---|---|
| 1 | Stygų |
| 2 | Paprastųjų iteracijų |
| 3 | Niutono (liestinių) |
| 4 | Kvazi-Niutono (kirstinių) |
| 5 | Skenavimo su mažėjančiu žingsniu |

**lentelė 2. Netiesinių lygčių sprendimas. Funkcijos ir metodai.**

| | | | |
|---|---|---|---|
| 9 | $0.48x^5 + 1.71x^4 - 0.67x^3 - 4.86x^2 - 1.33x + 1.50$ | $e^{-x}\sin(x^2) + 0{,}001; 5 \leq x \leq 10$ | 2, 3, 5 |

**lentelė 3. Netiesinių lygčių sprendimas. Uždavinių sąlygos.**

**Uždavinys variantams 6-10**

Krentančio parašiutininko greitis užrašomas dėsniu $v(t) = \frac{mg}{c}\left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$, čia $g = 9{,}8\ \mathrm{m/s^2}$, parašiutininko masė $m$. Koks pasipriešinimo koeficientas $c$ veikia parašiutininką, jei žinoma, kad po $t_1$ laisvojo kritimo, jo greitis lygus $v_1$?

| Varianto Nr. | $m$, kg | $t_1$, s | $v_1$, m/s |
|---|---|---|---|
| 6 | 90 | 3,5 | 30 |
| 7 | 80 | 4 | 36 |
| 8 | 60 | 3 | 25 |
| 9 | 70 | 3 | 27 |
| 10 | 60 | 4 | 30 |

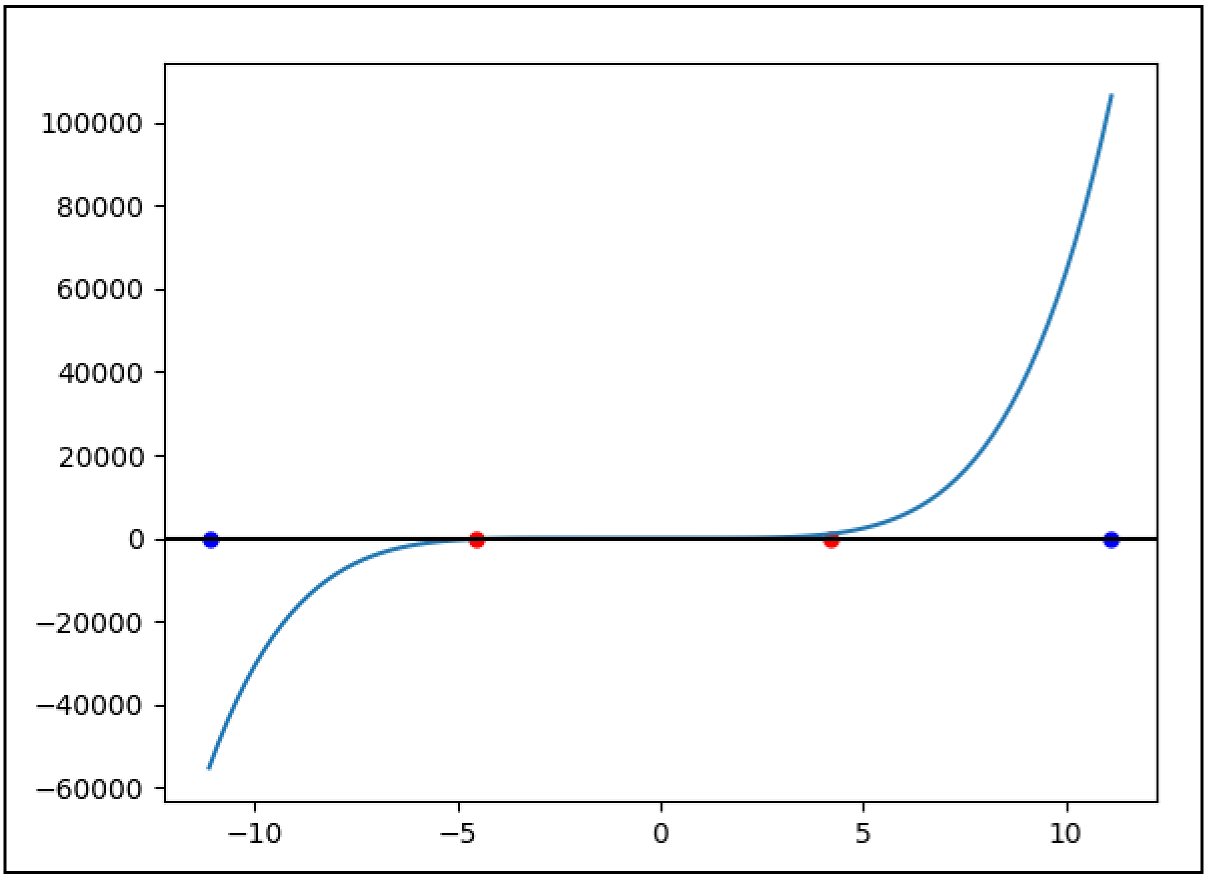
# 2. Pirmosios užduoties sprendimas

## 2.1. Daugianario šaknų intervalų nustatymas

### 2.1.1. Programos kodo dalis

```
# Randama didžiausia reikšmė moduliu masyve nevertinant koeficiento prie
aukščiausio laipsnio
def maxAbs(Coef):
    return abs(max((Coef[1:]), key=abs))
# Randamas grubus rėžis
def getRValue():
    return 1 + maxAbs(CONST_COEF) / CONST_COEF[0]
```

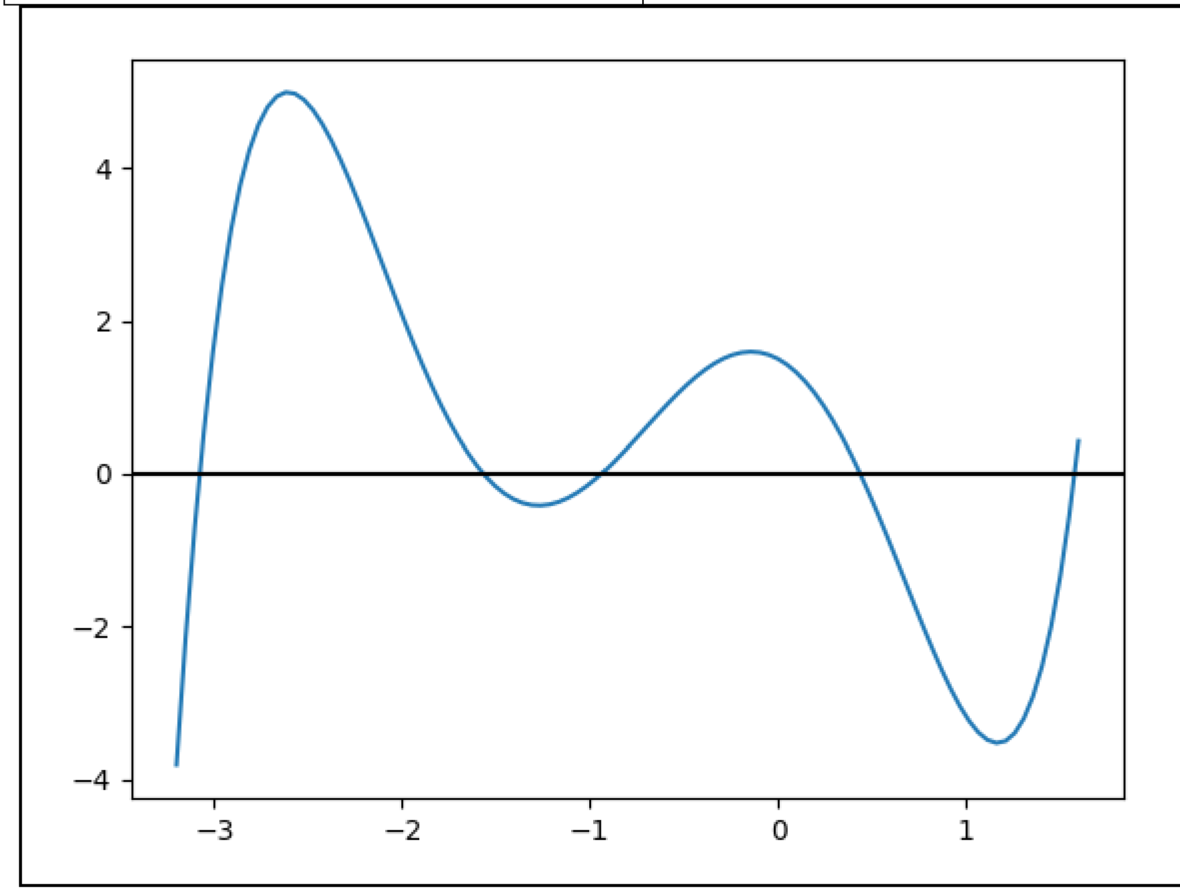
Mėlynai pavaizduota grubūs šaknų interval įverčiai, tikslesni – raudonai (1 pav.)



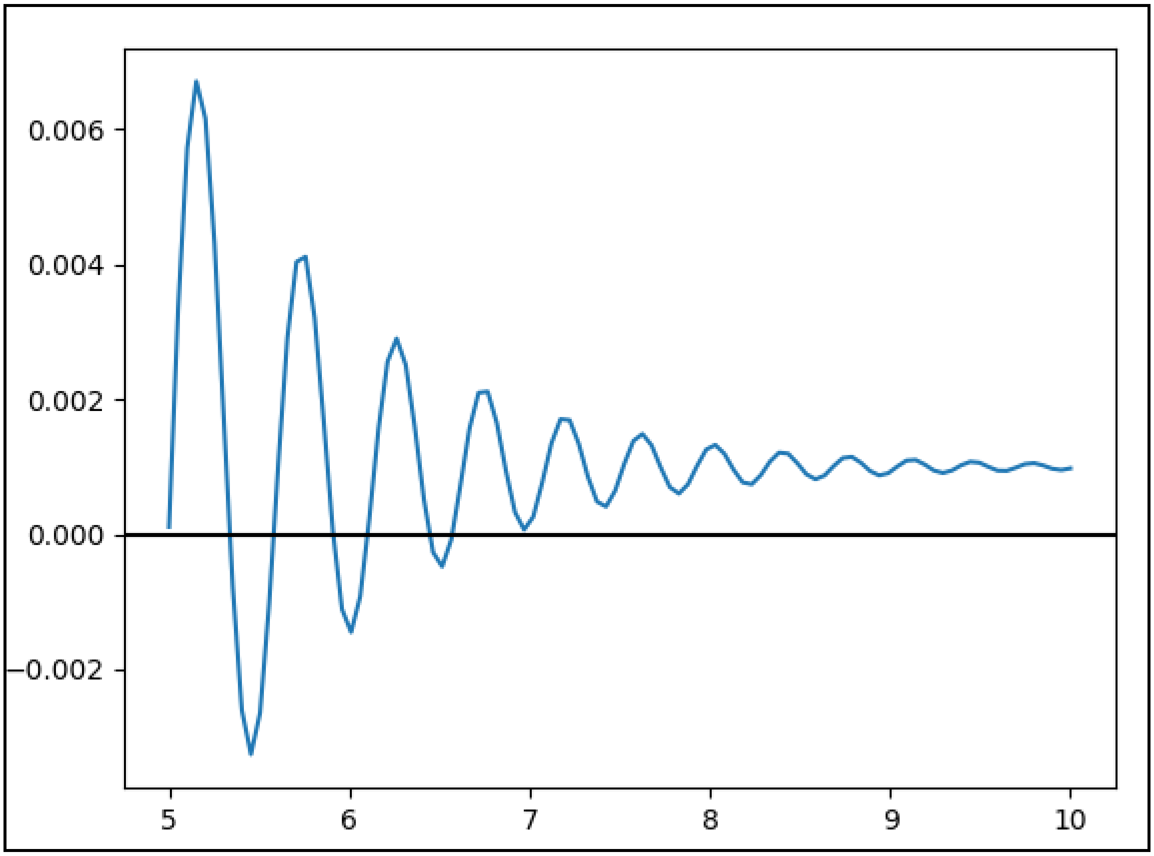

**pav. 1 "Grubus" ir "tikslesnis" šaknų intervalo įverčiai**

**lentelė 4. "Grubus" ir "tikslesnis" šaknų intervalo įverčių reikšmės**

| | |
|---|---|
| Grubus lygties f(x) šaknų intervalo įvertis | [-11.125; 11.125] |
| Tikslesnis lygties f(x) šaknų intervalo įvertis | [-4.5625; 4.1819] |




**pav. 2 Funkcijos f(x) grafikas**




**pav. 3 Funkcijos g(x) grafikas**

## 2.2. Funkcijų šaknų intervalų skaičiavimas, skenavimo su nekintančiu žingsniu algoritmu

### 2.2.1. Programos kodo dalis

```
# Skenavimo metodas su fiksuotu žingsniu artiniams tikslinti
def scanFixedStep(f, xFrom, xTo, step):
    rezArray = []
    while True:
        if np.sign(f(xFrom)) == np.sign(f(xFrom + step)):
            xFrom += step
        else:
            rezArray.append(xFrom)
            rezArray.append(xFrom + CONST_STEP)
            xFrom += CONST_STEP
        if xFrom > xTo:
            break
        else:
            continue
    return rezArray
```

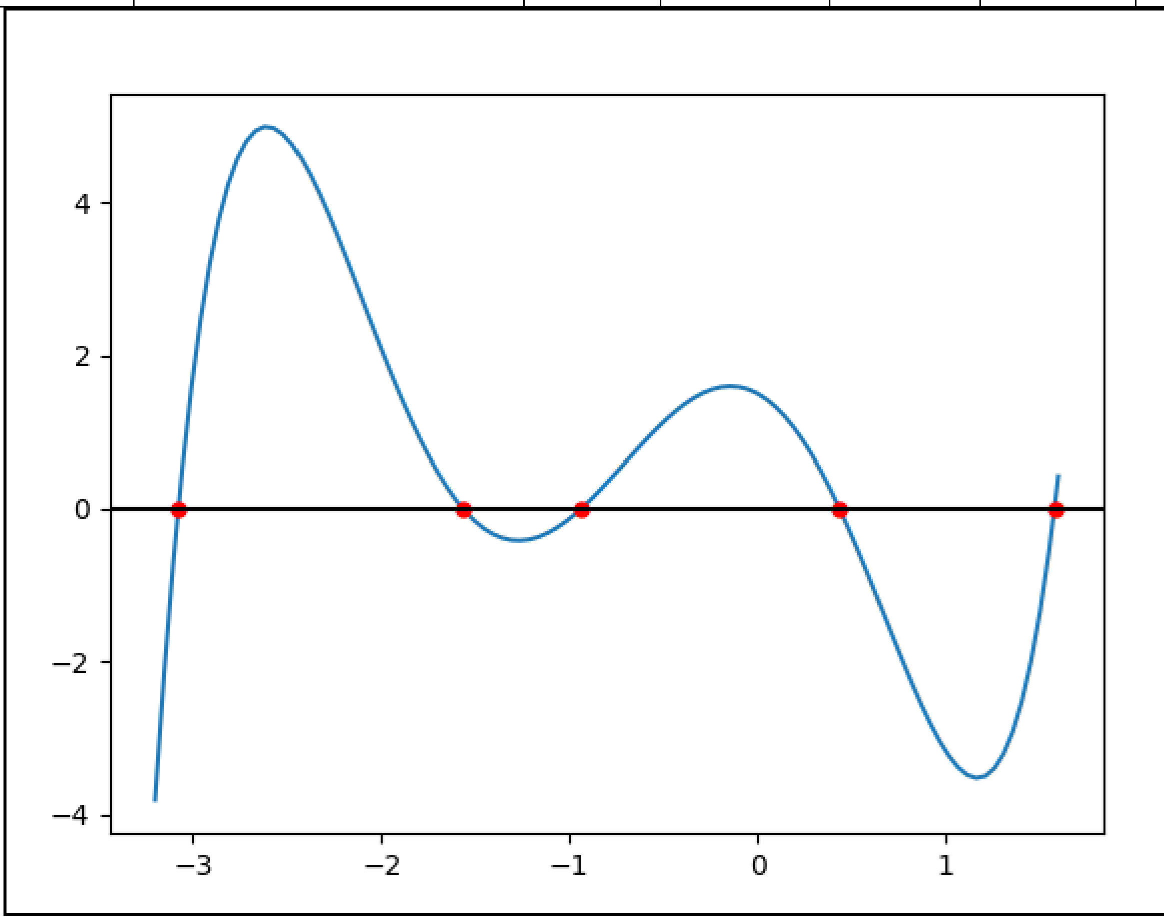
### 2.2.2. Daugianario ir funkcijos šaknų intervalai

| | Intervalo nr. | Intervalas |
|---|---|---|
| Daugianaris f(x) | 1 | [-3,11 ; -3,06] |
| | 2 | [-1,61 ; -1,56] |
| | 3 | [-0,96 ; -0,91] |
| | 4 | [0,44 ; 0,49] |
| | 5 | [1,54 ; 1,59] |
| Funkcija g(x) | 1 | [5,3 ; 5,35] |
| | 2 | [5,55 ; 5,60] |
| | 3 | [5,9 ; 5,95] |
| | 4 | [6,10 ; 6,15] |
| | 5 | [6,4 ; 6,45] |
| | 6 | [6,55 ; 6,60] |

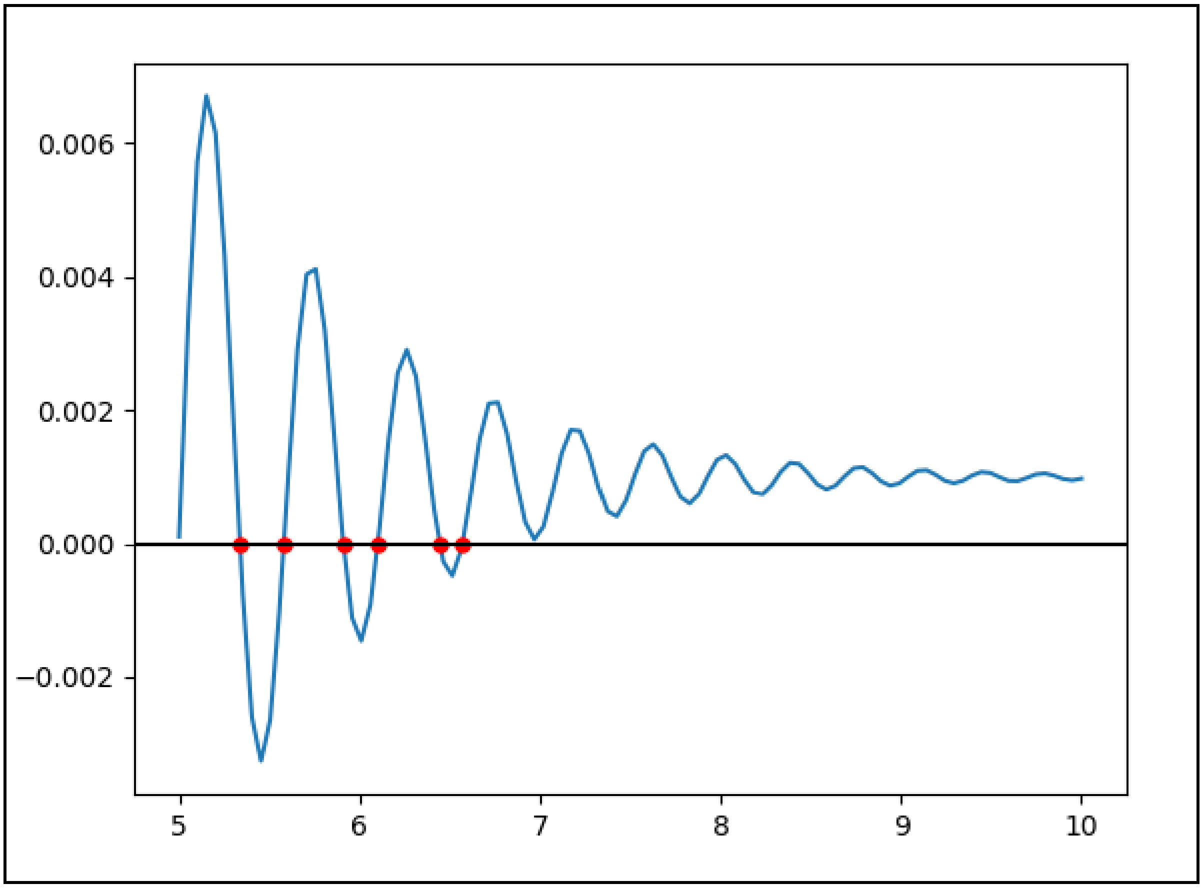
### 2.2.3. Rezultatai, užduotyje nurodytų metodų pritaikymas funkcijoms

| | Metodas | Pradinis artinys | Gautas sprendinys | Funkcijos reikšmė | Tikslumas | Iteracijų skaičius |
|---|---|---|---|---|---|---|
| Daugianaris f(x) | Paprastųjų iteracijų | -3,11 | -0,93 | 0 | 1e-5 | 31 |
| | | -1,61 | -0,93 | 0 | 1e-5 | 29 |
| | | -0,96 | -0,93 | 0 | 1e-5 | 13 |
| | | 0,44 | -0,93 | 0 | 1e-5 | 29 |
| | | 1,54 | -0,93 | 0 | 1e-5 | 25 |
| | Niutono (liestinių) | -3,11 | -3,08 | 0 | 1e-5 | 4 |
| | | -1,61 | -1,57 | 0 | 1e-5 | 4 |
| | | -0,96 | -0,94 | 0 | 1e-5 | 3 |
| | | 0,44 | 0,44 | 0 | 1e-5 | 2 |
| | | 1,54 | 1,58 | 0 | 1e-5 | 4 |
| | Skenavimo su mažėjančiu žingsniu | -3,11 | -3,08 | 0 | 1e-5 | 42 |
| | | -1,61 | -1,57 | 0 | 1e-5 | 28 |
| | | -0,96 | -0,94 | 0 | 1e-5 | 21 |
| | | 0,44 | 0,44 | 0 | 1e-5 | 7 |
| | | 1,54 | 1,58 | 0 | 1e-5 | 27 |
| Funkcija g(x) | Paprastųjų iteracijų | 5,3 | 5,23 | 0,01 | 1e-5 | 100 |
| | | 5,55 | 5,57 | 0 | 1e-5 | 100 |
| | | 5,9 | 5,89 | 0 | 1e-5 | 100 |
| | | 6,1 | 6,10 | 0 | 1e-5 | 16 |
| | | 6,4 | 6,38 | 0 | 1e-5 | 100 |
| | | 6,55 | 6,55 | 0 | 1e-5 | 100 |
| | Niutono (liestinių) | 5,3 | 5,3 | 0 | 1e-5 | 3 |
| | | 5,55 | 5,55 | 0 | 1e-5 | 3 |
| | | 5,9 | 5,9 | 0 | 1e-5 | 2 |
| | | 6,1 | 6,1 | 0 | 1e-5 | 2 |
| | | 6,4 | 6,4 | 0 | 1e-5 | 3 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | 6,55 | 6,55 | 0 | 1e-5 | 3 |
| | Skenavimo su mažėjančiu žingsniu | 5,3 | 5,3 | 0 | 1e-5 | 12 |
| | | 5,55 | 5,55 | 0 | 1e-5 | 14 |
| | | 5,9 | 5,9 | 0 | 1e-5 | 7 |
| | | 6,1 | 6,1 | 0 | 1e-5 | 4 |
| | | 6,4 | 6,4 | 0 | 1e-5 | 9 |
| | | 6,55 | 6,55 | 0 | 1e-5 | 5 |




**pav. 4 Daugianario f(x) grafikas su šaknimis**




**pav. 5 Funkcijos g(x) grafikas su šaknimis**

Daugianario f(x) šaknys pagal python: Roots: [-3.07650233 1.57967813 -1.56573 -0.93784643 0.43790063]

# 3. Antrosios užduoties sprendimas

## 3.1. Pradiniai metodo parametrai

```
CONST_M = 70   # Masė
CONST_G = 9.8   # Laisvo kritimo pagreitis
CONST_T = 3   # Laikas
CONST_V = 27   # Greitis
```

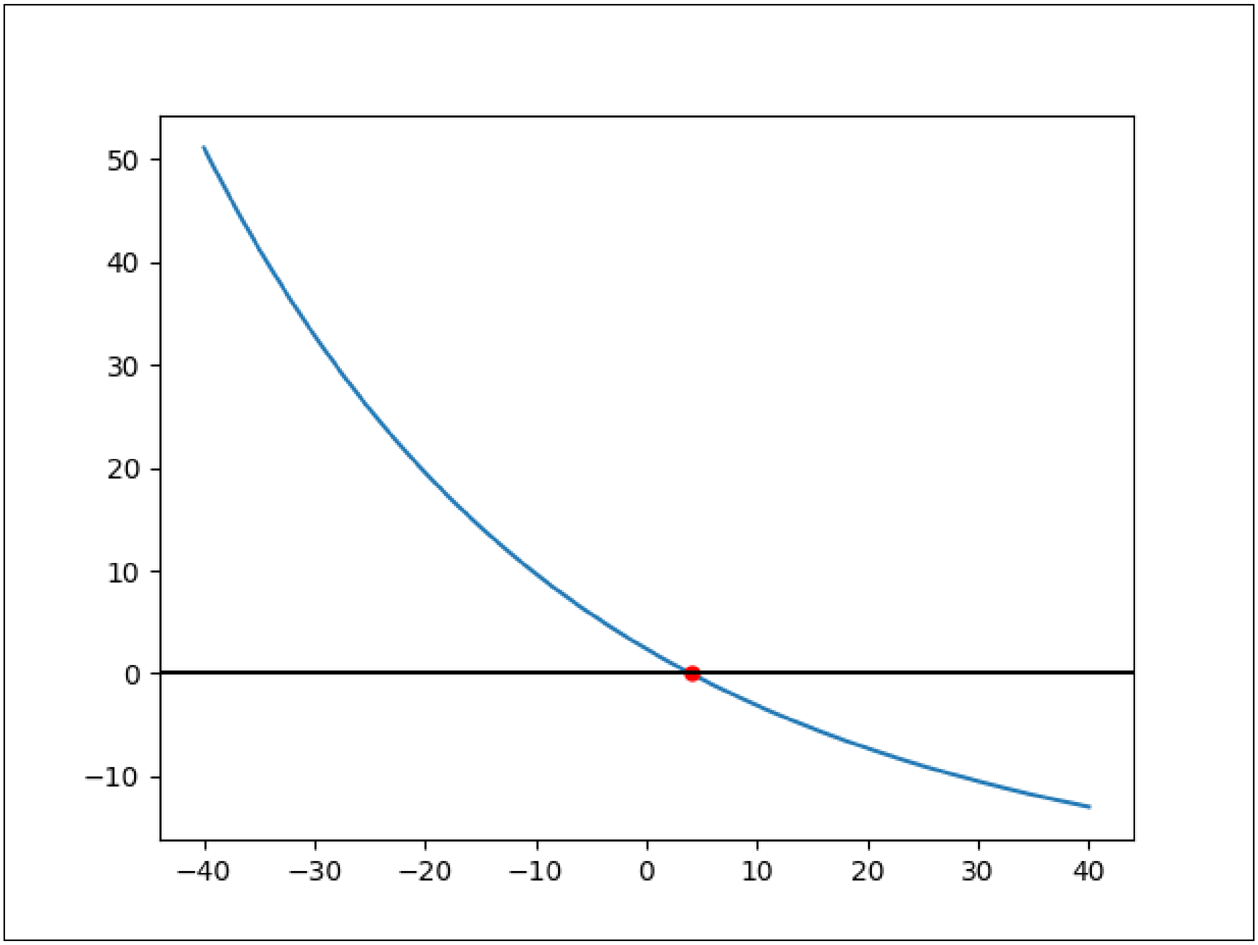
## 3.2. Pasirinktas metodas

Antrajai užduočiai buvo pasirinktas skenavimo su mažėjančiu žingsniu metodas, kadangi lengva buvo suprasti veikimo principus.

## 3.3. Rezultatai

| | Metodas | Iteracijų skaičius | Pradinis artinys | Šaknis | Funkcijos reikšmė šaknyje | Tikslumas |
|---|---|---|---|---|---|---|
| F(c) | Skenavimo su mažėjančiu žingsniu | 16 | ~4 | 4,032 | 0,00 | 1e-05 |

## 3.4. Grafinis lygties atvaizdavimas




**pav. 6 Funkcijos f(c) grafinis atvaizdavimas**

## 3.5. Programos kodas

```
# Netiesinė lygtis žodiniam uždaviniui
def fc(c):
    return ((CONST_M * (CONST_G / c)) * (1 - np.e ** (-3*c/70)))-27

# Skenavimo metodas su fiksuotu žingsniu artiniams tikslinti
def scanFixedStep(f, xFrom, xTo, step):
    rezArray = []
    while True:
        if np.sign(f(xFrom)) == np.sign(f(xFrom + step)):
            xFrom += step
        else:
            rezArray.append(xFrom)
            rezArray.append(xFrom + CONST_STEP)
            xFrom += CONST_STEP
        if xFrom > xTo:
            break
        else:
```

```python
            continue
    return rezArray

# Skenavimo metodas su mažėjančiu žingsniu
def ScanWithSmallerStep(func, xFrom, e):
    h = 0.1
    x = xFrom
    idx = 0
    while np.abs(func(x)) > e and idx < 100:
        idx += 1
        if np.sign(func(x + h)) == np.sign(func(x)):
            x += h
        else:
            h *= 0.1
    print("Iteracijų skaičius: ", str(idx), "   Pradinis artinys: ",
str(xFrom), "   Šaknis:", str(x))
    return x
```

Metodo "ChooseMethod() kodo dalis"

```python
elif function == "fc":
    rangefc = scanFixedStep(fc, 1, 20, CONST_STEP)
    for i in range(len(rangefc)):
        if i % 2 == 0 and i + 1 <= len(rangefc):
            ans.append(ScanWithSmallerStep(fc, rangefc[i], CONST_PRECISION))
    if len(ans) > 0:
        PlotRes(-40, 40, ans, function, R=None)
    else:
        print("funkcijos šaknų nerasta nurodytame intervale")
        PlotRes(-100, 100, ans=[], func="fc", R=None)
    return ans
```

# 4. Išvados

Visi metodai buvo sėkmingai realizuoti, tačiau paprastųjų iteracijų metodui nebuvo rasta optimali alpha reikšmė su kuria rastų visas daugianario f(x) šaknis. Paprastųjų iteracijų metodas yra paprastas, tačiau reikia daug iteracijų, kol yra randama šaknis, taip pat gali būti sunku rasti alpha reikšmę, kuri tiktų visoms šaknims esančioms nurodytame intervale. Niutono (liestinių) metodas yra sudėtingesnis, nes reikalauja funkcijos išvestinės, tačiau šaknis randa su keliomis iteracijomis. Pvz.: Paprastųjų iteracijų metodas, funkcijai g(x), pirmąją šaknį rado netiksliai su 100 iteracijų, o Niutono (liestinių) metodas rado su 3 iteracijomis 1e-5 tikslumu. Skenavimo su mažėjančiu žingsniu metodas yra paprastas tačiau reikia nemažai iteracijų funkcijos sprendiniui rasti. Pirmosios užduoties skaičiavimo pabaigos sąlygos buvo, kai šaknis randama 1e-5 tikslumu arba 100 iteracijų.

## 5. Visas programos kodas

```
import numpy as np
import matplotlib.pyplot as plt


CONST_COEF = np.array([0.48, 1.71, -0.67, -4.86, -1.33, 1.50])  # Daugianarės
funkcijos koeficientai
reversedCoef = CONST_COEF[::-1]  # Daugianarės atvirkštinė funkcija
# Nurodomos konstantos
# ----------------------------------------------------------------------------
-------------------------
CONST_STEP = 0.05  # Step
CONST_PRECISION = 1e-5  # Tikslumas
CONST_ALPHA = -5  # Konstanta paprastųjų iteracijų metodui
CONST_M = 70  # Masė
CONST_G = 9.8  # Laisvo kritimo pagreitis
CONST_T = 3  # Laikas
CONST_V = 27  # Greitis
# ----------------------------------------------------------------------------
-------------------------


# Daugianarė funkcija
def fx(x):
    return 0.48 * x ** 5 + 1.71 * x ** 4 - 0.67 * x ** 3 - 4.86 * x ** 2 -
1.33 * x + 1.50


# Daugianarės funkcijos išvestinė
def Dfx(x):
    return 2.4 * x ** 4 + 6.84 * x ** 3 - 2.01 * x ** 2 - 9.72 * x - 1.33


# Funkcija g(x)
def gx(x):
    return (np.e ** (-x)) * np.sin(x ** 2) + 0.001


# Funkcijos g(x) išvestinė
def Dgx(x):
    return -np.e ** -x * (np.sin(x ** 2) - 2 * x * np.cos(x ** 2))


# Netiesinė lygtis
def fc(c):
    return ((CONST_M * CONST_G) / c) * (1 - np.e ** (-(c / CONST_M) *
CONST_T) - CONST_V)


# Funkcijos f(x) šaknys pagal python
def pythonRoots():
    print('Saknys pagal python funkciją Roots:', np.roots(np.array([0.48,
1.71, -0.67, -4.86, -1.33, 1.50])))
```

```
# Randama didžiausia reikšmė moduliu masyve nevertinant koeficiento prie
aukščiausio laipsnio
def maxAbs(Coef):
    return abs(max((Coef[1:]), key=abs))


# Randamas grubus intervalas
def roughCuts(maxMean, divCoef):
    return 1 + maxMean / divCoef


# Išrenkamos neigiamos reikšmės iš masyvo
def getNegValues(Coef):
    return np.where(Coef < 0, Coef, 0)


def getBValue(Coef):
    return maxAbs(getNegValues(Coef))


def getMaxIndexOfNegValue(reversedFunc):
    tempArr = getNegValues(reversedFunc[:])
    rez = 0
    for i in range(1, len(tempArr)):
        if tempArr[i] != 0:
            rez = i
    return rez

# Randamas grubus rėžis
def getRValue():
    return 1 + maxAbs(CONST_COEF) / CONST_COEF[0]


# Randamas tikslesnio įverčio viršutinis rėžis
def getRPos():
    tempReversedCoef = reversedCoef[:]
    tempCoef = CONST_COEF[:]
    k = 5 - getMaxIndexOfNegValue(tempReversedCoef)
    return 1 + (getBValue(tempCoef) / tempCoef[0]) ** (1 / k)


# Randamas tikslesnio įverčio apatinis rėžis
def getRNeg():
    tempReversedCoef = reversedCoef[:]
    for i in range(len(tempReversedCoef)):
        if i % 2 != 0:
            tempReversedCoef[i] *= -1
        tempReversedCoef[i] *= -1
    tempCoef = tempReversedCoef[::-1]
    k = 5 - getMaxIndexOfNegValue(tempReversedCoef)
    return 1 + (getBValue(tempCoef) / tempCoef[0]) ** (1 / k)


# Skenavimo metodas su fiksuotu žingsniu artiniams tikslinti
def scanFixedStep(f, xFrom, xTo, step):
    rezArray = []
    while True:
```

```
        if np.sign(f(xFrom)) == np.sign(f(xFrom + step)):
            xFrom += step
        else:
            rezArray.append(xFrom)
            rezArray.append(xFrom + CONST_STEP)
            xFrom += CONST_STEP
        if xFrom > xTo:
            break
        else:
            continue
    return rezArray


# Paprastųjų iteracijų metodas
def simpleIterationMethod(f, xFrom):
    idx = 0
    x = xFrom
    precision = 1
    while precision > CONST_PRECISION and idx < 100:
        idx += 1
        x_next = (f(x) / CONST_ALPHA) + x
        precision = abs(x - x_next)
        x = x_next
    print("Iteracijų skaičius: ", str(idx), "   Pradinis artinys: ",
str(xFrom), "   Šaknis:", str(x))
    return x


# Niutono(liestinių) metodas
def newtonsMethod(f, Df, xFrom, e):
    xn = xFrom
    fxn = f(xn)
    idx = 0
    while np.abs(fxn) > e and idx < 100:
        fxn = f(xn)
        Dfxn = Df(xn)
        if Dfxn == 0:
            return None
        xn = xn - fxn / Dfxn
        idx += 1
    print("Iteracijų skaičius: ", str(idx), "   Pradinis artinys: ",
str(xFrom), "   Šaknis:", str(xn))
    return xn


# Skenavimo metodas su mažėjančiu žingsniu
def ScanWithSmallerStep(func, xFrom, e):
    h = 0.1
    x = xFrom
    idx = 0
    while np.abs(func(x)) > e and idx < 100:
        idx += 1
        if np.sign(func(x + h)) == np.sign(func(x)):
            x += h
        else:
            h *= 0.1
    print("Iteracijų skaičius: ", str(idx), "   Pradinis artinys: ",
```

```
str(xFrom), "   Šaknis:", str(x))
    return x


# Pagal pasirinktą funkciją ir metodą atliekami skaičiavimai
def ChooseMethod(function, method):
    ans = []
    if function == "fx":
        R = getRValue()
        Rpos = min(R, getRPos())
        RNeg = -min(R, getRNeg())
        rangefx = scanFixedStep(fx, RNeg, Rpos, CONST_STEP)
        print("R = ", R, "\nRNeg = ", RNeg, "\nRPos = ", Rpos)
        if method == "SimpleIteration":
            for i in range(len(rangefx)):
                if i % 2 == 0 and i + 1 <= len(rangefx):
                    ans.append(simpleIterationMethod(fx, rangefx[i]))
            pythonRoots()
            PlotRes(RNeg, Rpos, ans, function, R)
            return ans
        elif method == "Newtons":
            for i in range(len(rangefx)):
                if i % 2 == 0 and i + 1 <= len(rangefx):
                    ans.append(newtonsMethod(fx, Dfx, rangefx[i],
CONST_PRECISION))
            pythonRoots()
            PlotRes(RNeg, Rpos, ans, function, R)
            return ans
        elif method == "Scan":
            for i in range(len(rangefx)):
                if i % 2 == 0 and i + 1 <= len(rangefx):
                    ans.append(ScanWithSmallerStep(fx, rangefx[i],
CONST_PRECISION))
            pythonRoots()
            PlotRes(RNeg, Rpos, ans, function, R)
            print(rangefx)
        return ans
    elif function == "gx":
        rangegx = scanFixedStep(gx, 5, 10, CONST_STEP)
        if method == "SimpleIteration":
            for i in range(len(rangegx)):
                if i % 2 == 0 and i + 1 <= len(rangegx):
                    ans.append(simpleIterationMethod(gx, rangegx[i]))
            PlotRes(5, 10, ans, function, R=None)
            return ans
        elif method == "Newtons":
            for i in range(len(rangegx)):
                if i % 2 == 0 and i + 1 <= len(rangegx):
                    ans.append(newtonsMethod(gx, Dgx, rangegx[i],
CONST_PRECISION))
            PlotRes(5, 10, ans, function, R=None)
            return ans
        elif method == "Scan":
            for i in range(len(rangegx)):
                if i % 2 == 0 and i + 1 <= len(rangegx):
                    ans.append(ScanWithSmallerStep(gx, rangegx[i],
CONST_PRECISION))
```

```
            PlotRes(5, 10, ans, function, R=None)
            print(rangegx)
            return ans
    elif function == "fc":
        rangegx = scanFixedStep(fc, 5, 10, CONST_STEP)
        for i in range(len(rangegx)):
            if i % 2 == 0 and i + 1 <= len(rangegx):
                ans.append(ScanWithSmallerStep(gx, rangegx[i],
CONST_PRECISION))
        if len(ans) > 0:
            PlotRes(5, 10, ans, function, R=None)
        else:
            print("funkcijos šaknų nerasta nurodytame intervale")
            PlotRes(-100, 100, ans=[], func="fc", R=None)
        return ans


# Grafiko ir funkcijos šaknų atvaizdavimas
def PlotRes(xFrom, xTo, ans, func, R):
    x = np.linspace(xFrom, xTo, 100)
    y = 0
    if func == "fx":
        y = fx(x)
    elif func == "gx":
        y = gx(x)
    elif func == "fc":
        y = fc(x)
    plt.plot(x, y)
    if len(ans) > 0:
        for i in range(len(ans)):
            plt.plot(ans[i], 0, markersize=5, color='red', marker='o')
            if func == "fx":
                print("Funkcijos fx reikšmė: ", str(format(fx(ans[i]),
".2f")), " šaknyje: ",
                      str(format(ans[i], ".2f")))
            elif func == "gx":
                print("Funkcijos gx reikšmė: ", str(format(gx(ans[i]),
".2f")), " šaknyje: ",
                      str(format(ans[i], ".2f")))
            elif func == "fc":
                print("Funkcijos ff reikšmė: ", str(format(fc(ans[i]),
".2f")), " šaknyje: ",
                      str(format(ans[i], ".2f")))
    plt.axhline(color='black')
    plt.show()


# Programos vykdymas
def Execute():
    print("Pasirinkite funkciją: fx, gx, fc")
    function = str(input())
    if function != "fc":
        print("Pasirinkite metodą: SimpleIteration, Newtons, Scan")
        method = str(input())
        ChooseMethod(function, method)
    else:
        ChooseMethod("fc", method=None)
```

```
    print("Tikslumas: ", str(CONST_PRECISION))

Execute()
```

# 6. Paveikslėlių sąrašas



# 7. Lentelių sąrašas