VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Vienmatis Optimizavimas

Pirmo laboratorinio darbo ataskaita

Atliko: Paulina Podgaiska

VU el. p.: paulina.podgaiska@mif.stud.vu.lt

Vertino: Vyr. M. Darbuot., Dr. (HP), Julius Žilinskas

Vilnius
2025

UŽDUOTIS

Pirmo laboratorinio darbo užduotis pateikta lentelėje 1

1lentelė. Pirmo laboratorinio darbo užduotis

Vienmatis optimizavimas
1. Suprogramuoti vienmačio optimizavimo intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodo algoritmus.
2. Aprašyti tikslo funkciją $f(x) = (x^2-a)^2/b-1$, čia a ir b – studento knygelės numerio skaitmenys. Jei b yra 0, susumuoti visus kortelės numerio skaitmenis, skaičiuoti rezultato skaitmenų sumą ir taip daryti tol, kol bus gautas vienženklis skaičius, jį ir imti kaip b.
3. Minimizuoti šią funkciją intervalo dalijimo pusiau ir auksinio pjūvio metodais intervale [0,10] iki tikslumo 10^{-4} bei Niutono metodu nuo $x_0 = 5$ kol žingsnio ilgis didesnis už 10^{-4}.
4. Palyginti rezultatus: gauti sprendiniai, rastas funkcijos minimumo įvertis, atliktų žingsnių ir funkcijų skaičiavimų skaičius.
5. Vizualizuoti tikslo funkciją ir bandymo taškus.
6. Aprašyti darbo eigą ir rezultatus ataskaitoje, pateikti išvadas, prieduose pateikti programų kodus.

ATASKAITA

1. DARBO EIGA

Optimizuojama kvadratinė funkcija:

$$f(x) = \frac{(x^2 + a)^2}{b - 1}$$

Čia a = 9 ir b = 9 buvo nustatyti pagal LSP kortelės numerį.

Optimizuojant tikslo funkciją „Python“ programavimo kalba buvo įgyvendinti intervalo dalijimo pusiau, auksinio pjūvio ir Niutono metodai (programos kodas pateiktas priede).

1.1. Intervalo dalijimo pusiau metodas

Šiam metodui pateiktas intervalas buvo [0,10]. Algoritmas dalina intervalą pusiau tol, kol pasiekiamas pakankamas tikslumas. Veikimo principas: pradinis taškas, kuris dalina intervalą pusiau: x = 5. Suskaidytos puselės dalinamos vėl per pusę, gaunami taškai: x = 2,5 ir x = 7,5 (žr. 4 pav.). Apskaičiuojama funkcijos reikšmė šiuose taškuose. Pasirenkamas naujas intervalas, kuriame funkcijos reikšmė yra mažesnė. Procesas kartojamas tol, kol intervalas tampa mažesnis nei 10^{-4}.

1.1.1. Rezultatas

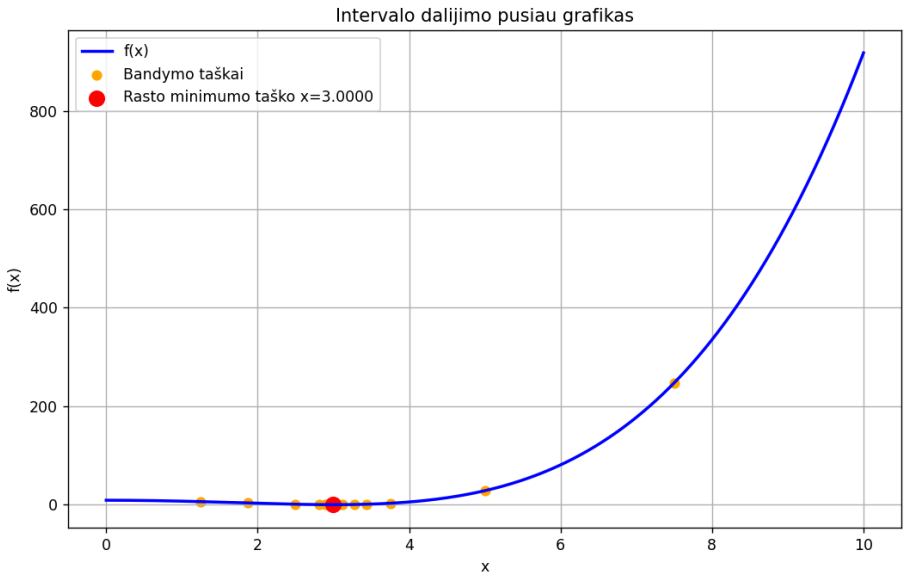
Intervalo dalijimo pusiau metodui prireikė 17 iteracijų, nes jis netiksliai nustato minimumo kryptį, kiekviename žingsnyje tik padalindamas intervalą be griežtesnio optimizavimo. Funkcijos iškvietimo skaičius – 35, nes kiekviename iteracijos žingsnyje reikia įvertinti funkcijos reikšmę dviejuose naujuose taškuose, siekianat nustatyti mažesnę reikšmę turintį intervalą. Rezultatai pateikti 1 pav.

```
[Running] python -u "c:\Users\Paulina\Documen
Intervalo dalijimo pusiau metodas:
Rastas minimumas: x = 3.0000, f(x) = -1.0000
Iteraciju skaicius: 17
Funkciju ivertinimu skaicius: 35
```

1 pav. Intervalo dalijimo pusiau rezultatai

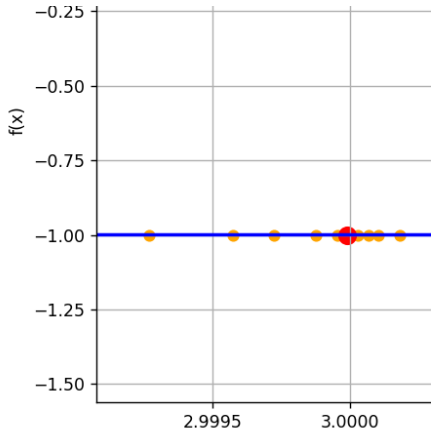
1.1.2. Vizualizacija

2 pav. pateikiamas optimizuojamos funkcijos grafikas su bandymo taškais, pagal intervalo dalijimo pusiau metodą.



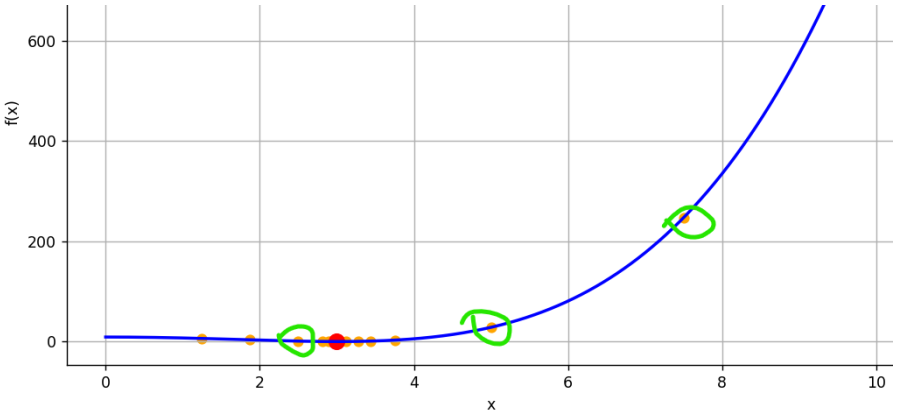
2 pav. Intervalo dalijimo pusiau grafikas

3 pav. minimumo taškas iš arčiau su matomomis x ir y reikšmėmis.



3 pav. Detalesnis intervalo dalijimo pusiau grafikas

4 pav. pateikiamas pavyzdinis algoritmo veikimo grafikas, kuriame apibraukti pirmieji bandymo taškai.



4 pav. Intervalo dalijimo pusiau grafikas su pavyzdiniais taškais.

1.2. Auksinio pjūvio metodas

Auksinio pjūvio metodas naudoja proporciją (~0.618) intervalo siaurinimui, kas optimizuoja minimumo taško radimo greitį, nes sumažėja funkcijos iškvietimų skaičius. Veikimo principas:

1. Pradinė reikšmė: [0,10].
2. Pasirenkami du taškai pagal auksinio pjūvio taisyklę: x = 3,82 ir x = 6,18 (žr 8 pav.).
3. Apskaičiuojama funkcijos reikšmė tuose taškuose.
4. Pasirenkamas naujas intervalas pagal mažiausios reikšmės kryptį.
5. Procesas kartojamas tol, kol intervalas tampa mažesnis nei 10^{-4}.

1.2.1. Rezultatai

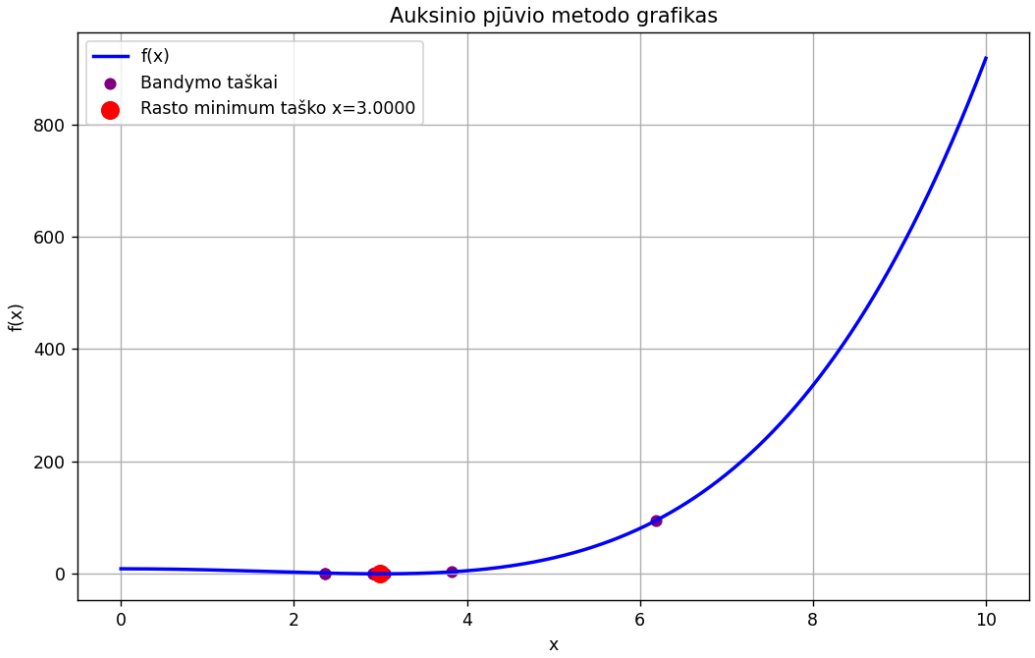
Šis metodas susiaurina intervalą efektyviau nei pusiau dalijimo metodas, nes naudoja optimalią dalijimo proporciją, tačiau reikia daugiau iteracijų, kadangi jis palaipsniui mažina intervalą. Auksinio pjūvio metodui prireikė 24 iteracijų, o funkcijų įvertinimų skaičius buvo 26. Rezultatai pateikti 5 pav.

```
Auksinio pjuvio metodas:
Rastas minimumas: x = 3.0000, f(x) = -1.0000
Iteraciju skaicius: 24
Funkciju ivertinimu skaicius: 26
```

5 pav. Auksinio pjūvio metodo rezultatai

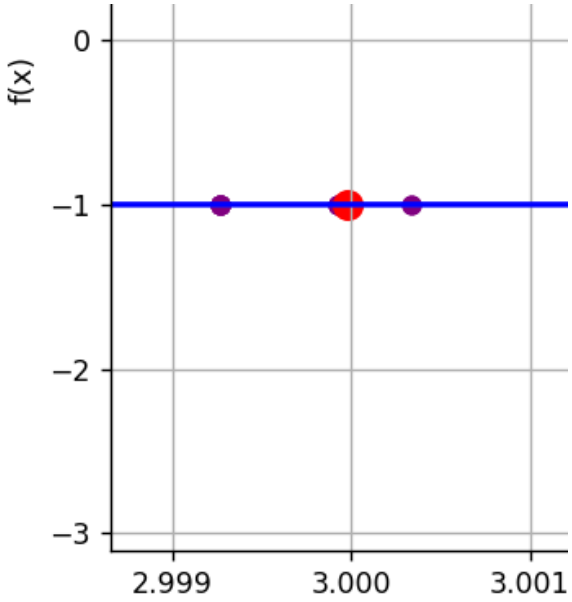
1.2.2. Vizualizacija

6 pav. pateikiamas optimizuojamos funkcijos grafikas su bandymo taškais, pagal auksinio pjūvio metodą.



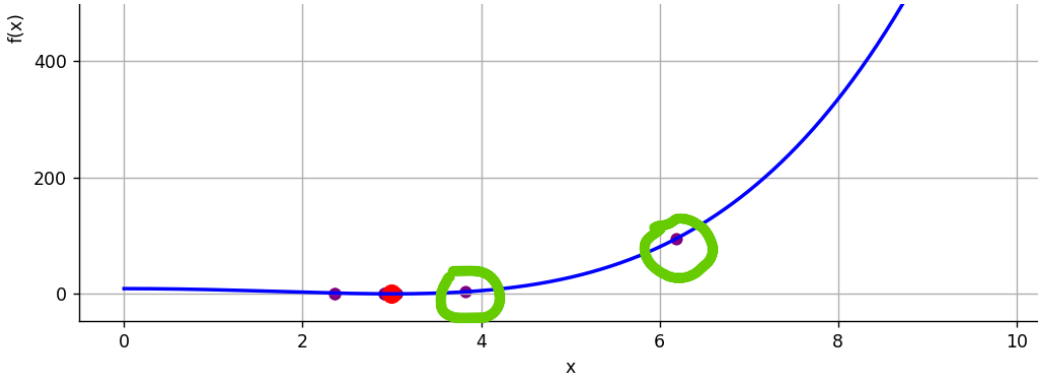
6 pav. Auksinio pjūvio metodo grafikas

7 pav. minimumo taškas iš arčiau su matomomis x ir y reikšmėmis.



7 pav. Detalesnis auksinio pjūvio grafikas

8 pav. pateikiamas pavyzdinis algoritmo veikimo grafikas, kuriame apibraukti pirmieji bandymo taškai.



8 pav. Pavyzdinis auksinio pjūvio grafikas

1.3. Niutono metodas

Niutono metodas naudoja funkcijos pirmąją ir antrąją išvestines, leidžiančias greičiau priartėti prie minimumo. Veikimo principas:

1. Pradinė reikšmė: $x_0 = 5$.
2. Skaičiuojamos pirmoji $f'(x)$ ir antroji $f''(x)$ funkcijos išvestinės.
3. Apskaičiuojamas naujas taškas pagal formulę: $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$.
4. Procesas kartojamas tol, kol žingsnio ilgis tampa mažesnis nei 10^{-4}.

1.3.1. Rezultatai

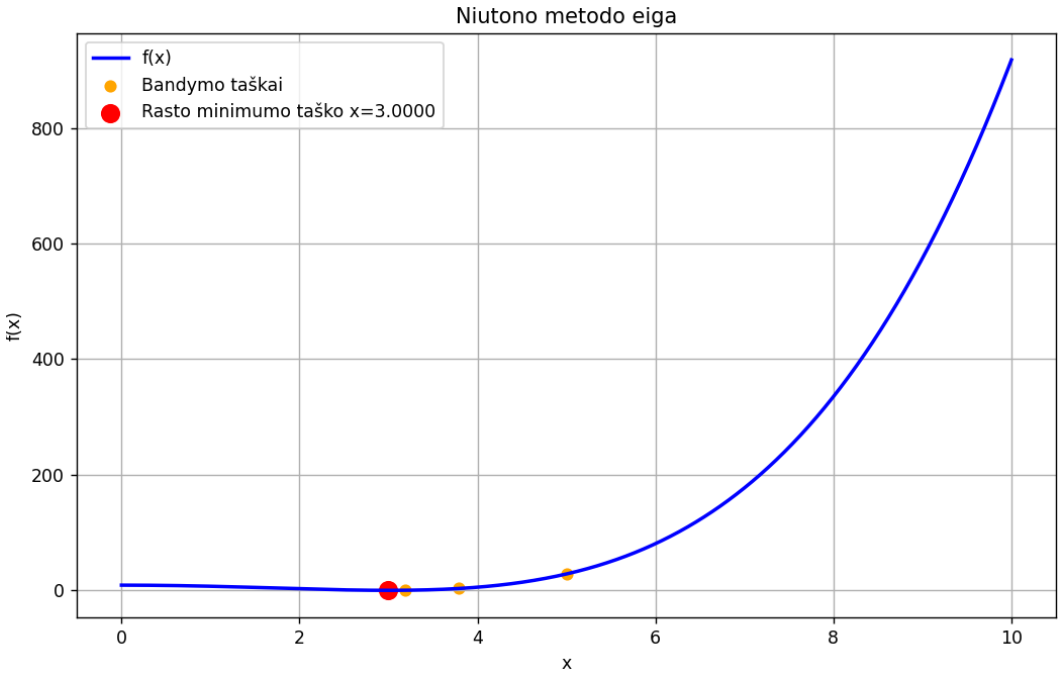
Niutono metodui prireikė 6 iteracijų, o funkcijų įvertinimų skaičius buvo 12. Niutono metodas konverguoja greičiausiai, nes naudoja antrąją išvestinę, kuri padeda tiksliau apskaičiuoti minimumo kryptį ir sparčiau susiaurinti paieškos intervalą. Rezultatai pateikti 9 pav.

```
Niutono Metodas:
Rasto minimumo tasko x = 3.0000
Rasto minimumo tasko f(x) = -1.0000
Iteraciju skaicius: 6
Funkciju ivertinimu skaicius: 12
```

9. Niutono metodo rezultatai

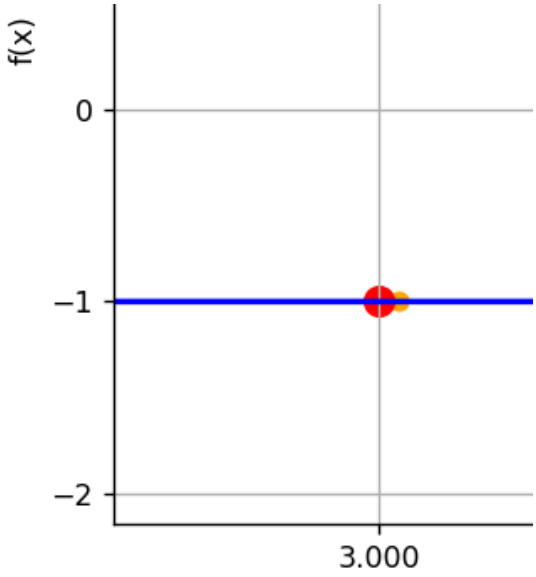
1.3.2. Vizualizacija

10 pav. pateikiamas optimizuojamos funkcijos grafikas su bandymo taškais, pagal Niutono metodą.



10 pav. Niutono metodo grafikas

11 pav. minimumo taškas iš arčiau su matomomis x ir y reikšmėmis.



11 pav. Detalesnis Niutono metodo grafikas

2. IŠVADOS

Struktūrizuotam algoritmų palyginimui galima pasinaudoti 2 lentele.

2 lentelė. Įgyvendintų metodų rezultatų palyginimas

Metodas	Rasto minimumo taško $f(x)$	Rasto minimumo taško x	Iteracijų skaičius	Funkcijos įvertinimų skaičius
Intervalo dalijimo pusiau	-1,0000	3,0000	17	35
Auksinio pjūvio	-1,0000	3,0000	24	26
Niutono	-1,0000	3,0000	6	12

Intervalo dalijimo pusiau metodas – garantuoja minimumo radimą, tačiau yra lėtas, nes kiekviename žingsnyje tik dalina intervalą be papildomos informacijos apie funkcijos savybes.

Auksinio pjūvio metodas – optimizuoja paieškos greitį naudodamas auksinį pjūvį, tačiau reikalauja daugiau iteracijų.

Niutono metodas – efektyviausias metodas, nes naudojant funkcijos išvestines greičiau priartėjama prie minimumo.

3. PRIEDAS

Žemiau pateikiamas kodas su anksčiau aprašytais įgyvendintais algoritmais.

1_laboratorinis.py

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import *

def tikslofunkcija(x, a=9, b=9):
    return ((x**2 - a)**2) / b - 1

def intervalo_dal_pusiau(f, l, r, tiksl=1e-4):
    iter = 0
    fun_ivertinimu = 0

    xm = (l + r) / 2
    L = r - l
    f_xm = f(xm)
    fun_ivertinimu += 1
    while L > tiksl:
        x1 = l + L / 4
        x2 = r - L / 4
        f_x1, f_x2 = f(x1), f(x2)
```

```
        fun_ivertinimu += 2

        if f_x1 < f_xm:
            r = xm
            xm = x1
            f_xm = f_x1
        elif f_x2 < f_xm:
            l = xm
            xm = x2
            f_xm = f_x2
        else:
            l = x1
            r = x2

        L = r - l
        iter += 1

    x_min = xm
    f_min = f_xm

    return x_min, f_min, iter, fun_ivertinimu

# intervalas [0,10]
int_apatinis, int_virsutinis = 0, 10

# x_min, f_min, iter, fun_ivertinimu = intervalo_dal_pusiau(tikslofunkcija, int_apatinis, int_virsutinis)
# dalijimo pusiau  grafikas
x_reiks = np.linspace(int_apatinis, int_virsutinis, 1000)
y_reiks = tikslofunkcija(x_reiks)

x_test_pusiau = []
x_min, f_min, iter, fun_ivertinimu = intervalo_dal_pusiau(
    lambda x: x_test_pusiau.append(x) or tikslofunkcija(x), int_apatinis, int_virsutinis
)

plt.figure(figsize=(10, 6))
plt.plot(x_reiks, y_reiks, label='f(x)', color='blue', linewidth=2)
plt.scatter(x_test_pusiau, [tikslofunkcija(x) for x in x_test_pusiau], color='orange', label='Bandymo taškai')
plt.scatter([x_min], [f_min], color='red', label=f'Rasto minimumo taško x={x_min:.4f}', s=100)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Intervalo dalijimo pusiau grafikas')
plt.legend()
plt.grid()
```

```
plt.show()

print("Intervalo dalijimo pusiau metodas:")
print(f"Rasto minimumo tasko x = {x_min:.4f}")
print(f"Rasko minimumo tasko f(x) = {f_min:.4f}")
print(f"Iteraciju skaicius: {iter}")
print(f"Funkciju ivertinimu skaicius: {fun_ivertinimu}")


# niutono metodas

def niutono_met(f, x0, tiks=1e-4, max_iter=1000):
    x = symbols('x')
    f_sym = f(x)
    f_isvestine = Derivative(f_sym, x).doit()
    f_antra_isv = Derivative(f_isvestine, x).doit()

    f_func = lambdify(x, f_sym, 'numpy')
    f_isvestine_func = lambdify(x, f_isvestine, 'numpy')
    f_antra_isv_func = lambdify(x, f_antra_isv, 'numpy')

    itera = 0
    funkc_iv = 0
    x_dabar = x0
    bandymo_taskai = [x_dabar]

    while itera < max_iter:
        f_isvest_reiksme = f_isvestine_func(x_dabar)
        f_antra_isv_reiks = f_antra_isv_func(x_dabar)
        itera += 1
        funkc_iv += 2
        if abs(f_isvest_reiksme) < tiks:
            break

        if f_antra_isv_reiks == 0:
            print("Klaida")
            break

        x_naujas = x_dabar - f_isvest_reiksme / f_antra_isv_reiks
        bandymo_taskai.append(x_naujas)

        if abs(x_naujas - x_dabar) < tiks:
            break
```

```
        x_dabar = x_naujas

    return x_dabar, f_func(x_dabar), itera, funkc_iv, bandymo_taskai

# niutono grafikas
x_test_niuton = []
f_num = lambda x: x_test_niuton.append(x) or tikslofunkcija(x)

# m
x0=5
x_min_n, f_min_n, iter_n, iv_n, x_test_niuton = niutono_met(tikslofunkcija, x0)

plt.figure(figsize=(10, 6))
plt.plot(x_reiks, y_reiks, label='f(x)', color='blue', linewidth=2)
plt.scatter(x_test_niuton, [tikslofunkcija(x) for x in x_test_niuton], color='orange', label='Bandymo taškai')
plt.scatter([x_min_n], [f_min_n], color='red', label=f'Rasto minimumo taško x={x_min_n:.4f}', s=100)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Niutono metodo eiga')
plt.legend()
plt.grid()
plt.show()

print("Niutono Metodas:")
print(f"Rasto minimumo tasko x = {x_min_n:.4f}")
print(f"Rasto minimumo tasko f(x) = {f_min_n:.4f}")
print(f"Iteraciju skaicius: {iter_n}")
print(f"Funkciju ivertinimu skaicius: {iv_n}")

# auksinio pjuvio met
def auksinis_pjuvis(f, l, r, tiksl=1e-4):
    tau = (np.sqrt(5) - 1) / 2
    iter = 0
    fun_ivertinimu = 0

    x1 = r - tau * (r - l)
    x2 = l + tau * (r - l)
    f_x1, f_x2 = f(x1), f(x2)
    fun_ivertinimu += 2

    x_test_auksinis = [x1, x2]

    while (r - l) > tiksl:
        if f_x1 < f_x2:
```

```
                r = x2
                x2 = x1
                f_x2 = f_x1
                x1 = r - tau * (r - l)
                f_x1 = f(x1)
            else:
                l = x1
                x1 = x2
                f_x1 = f_x2
                x2 = l + tau * (r - l)
                f_x2 = f(x2)

            fun_ivertinimu += 1
            iter += 1
            x_test_auksinis.append(x1 if f_x1 < f_x2 else x2)

        x_min = (l + r) / 2
        f_min = f(x_min)

        return x_min, f_min, iter, fun_ivertinimu, x_test_auksinis

    x_test_auksinis = []
    x_min_a, f_min_a, iter_a, fun_ivertinimu_a, x_test_auksinis = auksinis_pjuvis(
        lambda x: tikslofunkcija(x), int_apatinis, int_virsutinis
    )

    # auksinio pjuvio grafikas
    plt.figure(figsize=(10, 6))
    plt.plot(x_reiks, y_reiks, label='f(x)', color='blue', linewidth=2)
    plt.scatter(x_test_auksinis, [tikslofunkcija(x) for x in x_test_auksinis], color='purple', label='Bandymo
taškai')
    plt.scatter([x_min_a], [f_min_a], color='red', label=f'Rasto minimum taško x={x_min_a:.4f}', s=100)
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.title('Auksinio pjūvio metodo grafikas')
    plt.legend()
    plt.grid()
    plt.show()

    print("Auksinio pjuvio metodas:")
    print(f"Rasto minimumo tasko x = {x_min_a:.4f}")
    print(f"Rasto minimumo tasko f(x) = {f_min_a:.4f}")
    print(f"Iteraciju skaicius: {iter_a}")
    print(f"Funkciju ivertinimu skaicius: {fun_ivertinimu_a}")
```