VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Optimizavimas be apribojimų

Antro 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

Antro laboratorinio darbo užduotis pateikta lentelėje 1

1lentelė. Antro laboratorinio darbo užduotis

Optimizavimas be apribojimų
1. Suprogramuoti gradientinio nusileidimo, greičiausiojo nusileidimo ir deformuojamo simplekso algoritmus.
2. Laikant kintamaisiais dėžės priekinės ir galinės sienų plotų sumą, šoninių sienų plotų sumą, viršutinės ir apatinės sienų plotų sumą, aprašyti vienetinio dėžės paviršiaus ploto reikalavimą ir dėžės tūrio pakelto kvadratu funkciją.
3. Iš vienetinio paviršiaus ploto reikalavimo išvesti vieno iš kintamojo išraišką per kitus.
4. Aprašyti tikslo funkciją f(X) taip, kad optimizavimo uždavinys būtų formuluojamas be apribojimų: min f(X).
5. Išveskite ir aprašykite tikslo funkcijos gradiento funkciją.
6. Apskaičiuoti tikslo ir gradiento funkcijų reikšmes taškuose X0= (0,0), X1=(1,1) ir Xm= (a/10, b/10), čia a ir b – studento knygelės numerio “xxxxxab” skaitmenys.
7. Minimizuoti suformuluotą uždavinį naudojant suprogramuotus optimizavimo algoritmus pradedant iš taškų X0, X1 ir Xm.
8. Palyginti rezultatus: gauti sprendiniai, rastas funkcijos minimumo įvertis, atliktų žingsnių ir funkcijų skaičiavimų skaičius priklausomai nuo pradinio taško.
9. Vizualizuoti tikslo funkciją ir bandymo taškus.

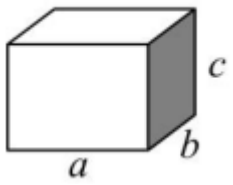
ATASKAITA

1. DARBO EIGA

Užduoties klausimas:

Kokia turėtų būti stačiakampio gretasienio formos dėžė, kad vienetiniam paviršiaus plotui jos tūris būtų maksimalus?

Laikant stačiakampio gretasienio briaunas a, b, c (1 pav.)



1 pav. Stačiakampio gretasienio briaunų žymėjimas

Čia tūris: V=abc. Viso paviršiausiaus plotas yra vienetinis, t.y. 2ab+2ac+2bc=1, arba, pažymėjus kintamaisiais dėžės priekinės ir galinės sienų plotų sumą, šoninių sienų plotų sumą, viršutinės ir apatinės sienų plotų sumą kaip: x_1=2ab, x_2=2ac, x_3=2bc ir įsistačius, turime $x_1x_2x_3$=1. Išreiškus per x_3=1-x_1-x_2. Čia $x_1x_2x_3$=2ab*2ac*2bc=8$a^2b^2c^2$=8V^2. Iš ko seka, kad V^2 = 1/8 * $x_1x_2x_3$ = 1/8 * x_1x_2 * (1-x_1-x_2) = 1/8 * (x_1x_2 - x_1^2 x_2 - $x_1x_2^2$). Tikslo funkcija ir yra stačiakampio gretasienio tūris pakeltas kvadratu.

$$f(X) = -\frac{1}{8}(x_1x_2 - x_1^2x_2 - x_1x_2^2)$$

Dalinė išvestinė pagal x_1 bus lygi $(-\frac{1}{8}(x_2 - 2x_1x_2 - x_2^2))$, dalinė išvestinė pagal x_2 lygi $(-\frac{1}{8}(x_1 - x_1^2 - 2x_1x_2))$. Atitinkamai tikslo funkcijos gradiento funkcija:

$$\nabla f(X) = (-\frac{1}{8}(x_2 - 2x_1x_2 - x_2^2), -\frac{1}{8}(x_1 - x_1^2 - 2x_1x_2))$$

Apskaičiuotos tikslo ir gradiento funkcijos reikšmės buvo įvertintos taškuose: X_0= (0;0), X_1=(1;1) ir X_m= (0,9; 0,9). Minimizuotas suformuluotas uždavinys, naudojant tris optimizavimo metodus: gradientinio nusileidimo, greičiausio nusileidimo ir deformuoto simplekso algoritmus. Gauti rezultatai, funkcijos vizualizacijos bei programos kodas, atliktas naudojant „Python“ programavimo kalbą, aprašyti žemiau (programos kodas pateiktas priede).

1.1. Gradientinio nusileidimo metodas

Gradientinio nusileidimo metodas siekia rasti funkcijos minimumą, žengdamas žingsnius priešinga kryptimi nuo funkcijos gradiento. Kiekviename žingsnyje apskaičiuojamas funkcijos gradientas ir pagal jį parenkama kryptis, kurioje mažinamas funkcijos reikšmės dydis. Taikant nustatytą žingsnio dydį taškas nuolat koreguojamas, kol pasiekiama norima tikslumo riba arba maksimalus iteracijų skaičius. Procesas sustoja, kai gradientas tampa labai mažas (didesnis už nustatytą ribą, epsilon), arba pasiekus maksimalų iteracijų skaičių.

Pvz.: pradinis taškas yra X_1=(1;1): apskaičiuojame gradiento funkciją taške X_1=(1;1). Įrašius į gradiento formulę:

$$\nabla f(1;1) = \left(-\frac{1}{8}(1-\ 2-1);\ -\frac{1}{8}(1-\ 1-2)\right) = (\frac{1}{4};\frac{1}{4})$$

Turint gradientą $(\frac{1}{4};\frac{1}{4})$ ir pasirinkus žingsnio dydį zings=3, atnaujiname tašką:

$$X_2 = X_1 - \text{zings} * \nabla f(X_1) = (1;1) - 3 * (\frac{1}{4};\frac{1}{4}) = (\frac{1}{4};\frac{1}{4}).$$

Procedūra kartojama, kol gradientas taps pakankamai mažas arba pasieksime maksimalų iteracijų skaičių.

1.1.1. Rezultatas

Kai pradinis taškas yra X_0= (0;0), gradiento reikšmė šiame taške lygi nuliui ($\nabla f(0;0) = (0;0)$). Kadangi gradientas yra mažesnis už nustatytą epsilon reikšmę, ciklas yra nutraukiamas ir iteracijos nevyksta. Šiuo atveju gradientas apskaičiuojamas tik vieną kartą, todėl funkcijos kvietimų skaičius yra 1. Rezultatai 2 pav.

```
Pradinis taskas: (0.0, 0.0)
f(x) = -0.0
Rastas minimumo taskas: (0.0, 0.0)
Iteraciju skaicius: 0
Funkcijos kvietimu skaicius: 1
```

2 pav. Taško X_0= (0;0) rezultatai

Taške X_1=(1;1), kaip aprašyta pavyzdyje, ieškomas minimumo taškas. Rezultatai pavaizduoti 3 pav. Kadangi pasirinktas žingsnio dydis yra pakankamai didelis, pradžioje funkcija gali „peršokti“ per minimumą, tačiau vėlesnėse iteracijose grįžtama prie jo ir randamas minimumo taškas (5 pav.).

```
Pradinis taskas: (1.0, 1.0)
f(x) = -0.0046
Rastas minimumo taskas: (0.3333, 0.3333)
Iteraciju skaicius: 23
Funkcijos kvietimu skaicius: 24
```

3 pav. Taško X_1=(1;1) rezultatai

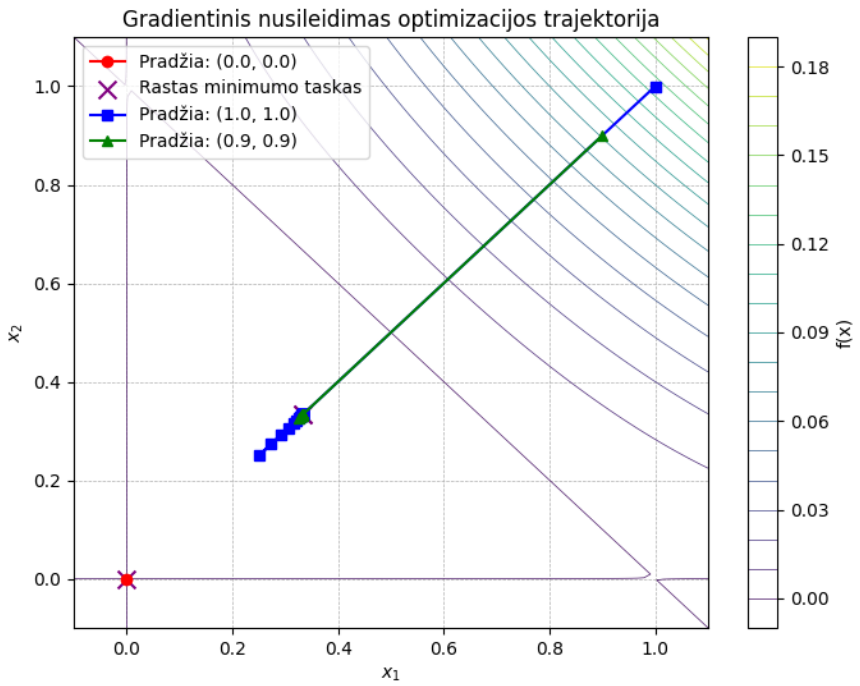
Nuo pradinio taško X_m= (0,9; 0,9) minimumo paieškos rezultatai pavaizduoti 4 pav.

```
Pradinis taskas: (0.9, 0.9)
f(x) = -0.0046
Rastas minimumo taskas: (0.3333, 0.3333)
Iteraciju skaicius: 17
Funkcijos kvietimu skaicius: 18
```

4 pav. Taško X_m= (0,9; 0,9) rezultatai

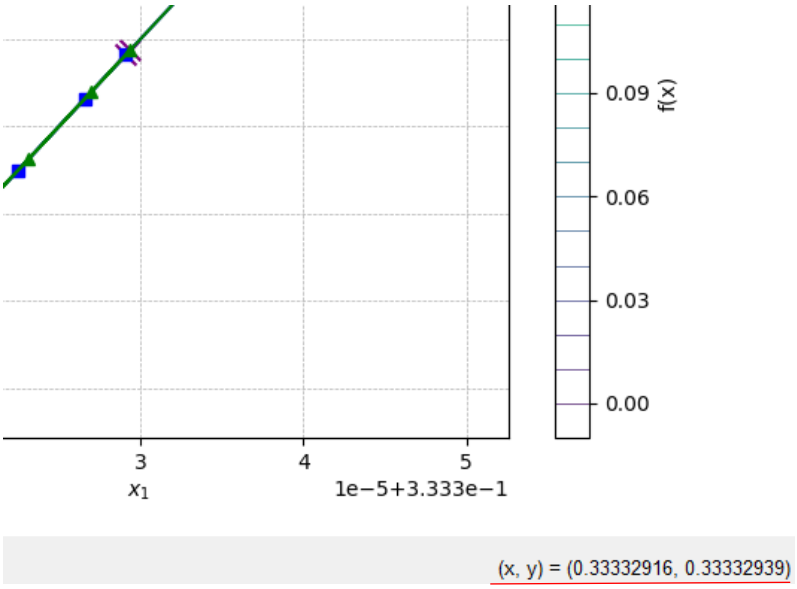
1.1.2. Vizualizacija

Gradientinio nusileidimo metodo trajektorijos nuo pradinių taškų X_0= (0;0) (raudona), X_1=(1;1) (mėlyna), X_m= (0,9; 0,9) (žalia) vaizduojamos 5 pav. Rasti minimumo taškai pažymėti kryžiukais.



5 pav. Gradientinio nusileidimo metodo trajektorijos vizualizacija

Minimumo taškai, pažymėti kryžiuku iš arčiau 6 pav.



6 pav. Taškų X_1=(1;1) ir X_m= (0,9; 0,9) rasti minimumo taškai

1.2. Greičiausio nusileidimo metodas

Greičiausio nusileidimo metodas ieško funkcijos minimumo, kiekviename žingsnyje judėdamas priešinga gradiento krypčiai, tačiau vietoje fiksuoto žingsnio naudoja optimalų žingsnį, parenkamą kiekvienoje iteracijoje. Kiekviename iteracijos žingsnyje apskaičiuojamas funkcijos gradientas, kuris nurodo stačiausią mažėjimo kryptį. Tuomet optimalus žingsnio dydis randamas sprendžiant vienmačio optimizavimo uždavinį – pasirenkant žingsnį, kuris duoda didžiausią funkcijos reikšmės sumažėjimą. Šis žingsnis nustatomas naudojant Niutono metodą. Procesas kartojamas tol, kol gradiento dydis tampa mažesnis už nustatytą ribą arba pasiekiamas maksimalus iteracijų skaičius.

Pvz.: pradinis taškas yra X_1=(1;1): apskaičiuojame gradiento funkciją taške X_1=(1;1). Įrašius į gradiento formulę:

$$\nabla f(1;1) = (\frac{1}{4};\frac{1}{4})$$

Tai nurodo kryptį, kuria funkcijos reikšmė didėja. Norėdami minimizuoti funkciją, judame priešinga kryptimi

Optimalų žingsnio dydį α randame spręsdami pagalbinį optimizavimo uždavinį:

$$\alpha^* = \arg min_\alpha f(X_1 - \alpha\nabla f)$$

Tai reiškia, kad ieškome tokio α, kuris užtikrina didžiausią funkcijos reikšmės mažėjimą.

Funkcija $\varphi(\alpha)= f(X_1 - \alpha\nabla f)$ yra minimizuojama naudojant Niutono metodą:

$$\varphi(\alpha) = f(1-\frac{\alpha}{4};1-\frac{\alpha}{4})$$

Įstatome į tikslinę funkciją:

$$\varphi(\alpha) = -\frac{1}{8}(\left(1-\frac{\alpha}{4}\right)^2\left(1-\frac{\alpha}{4}\right) + (1-\frac{\alpha}{4})(1-\frac{\alpha}{4})^2 - \left(1-\frac{\alpha}{4}\right)\left(1-\frac{\alpha}{4}\right))$$

Randame $\varphi'(\alpha)$ ir sprendžiame lygtį: $\varphi'(\alpha) = 0$. Gauname:

$$\alpha^* = 2{,}66667$$

Kai surandamas optimalus žingsnio dydis α^*, atnaujinamas taškas $X_2 = X_1 - \alpha * \nabla f(X_1)$:

$$X_2 = (1;1) - 2{,}66667 * \left(\frac{1}{4};\frac{1}{4}\right) = (0{,}3333325; 0{,}3333325)$$

Procesas kartojamas tol, kol gradientas tampa pakankamai mažas (funkcija beveik nebesikeičia) arba pasiekiamas maksimalus iteracijų skaičius. Kiekviename žingsnyje iš naujo apskaičiuojamas gradientas, randamas optimalus α^*, ir atnaujinamas taškas.

1.2.1. Rezultatai

Kai pradinis taškas yra X_0= (0;0), gradientas šiame taške yra nulinis, todėl optimizavimo procesas iš karto sustoja, nes nėra kur judėti. Šiuo atveju funkcijos kvietimų skaičius yra 1, nes apskaičiuojamas gradientas ir iteracijos nevyksta. Rezultatai 7 pav.

```
Pradinis taskas: (0.0, 0.0)
f(x) = -0.0
Rastas minimumo taskas: (0.0, 0.0)
Iteraciju skaicius: 0
Funkcijos kvietimu skaicius: 1
```

7 pav. Taško X_0= (0;0) rezultatai

Kai pradinis taškas yra X_1=(1;1), algoritmas dinamiškai parenka žingsnio dydžius ir artėja prie minimumo, užtikrindamas greitesnį konvergavimą nei gradientinio nusileidimo metodas su fiksuotu žingsniu. Rezultatai 8 pav.

```
Pradinis taskas: (1.0, 1.0)
f(x) = -0.0046
Rastas minimumo taskas: (0.3333, 0.3333)
Iteraciju skaicius: 13
Funkcijos kvietimu skaicius: 14
```

8 pav. Taško X_1= (1;1) rezultatai

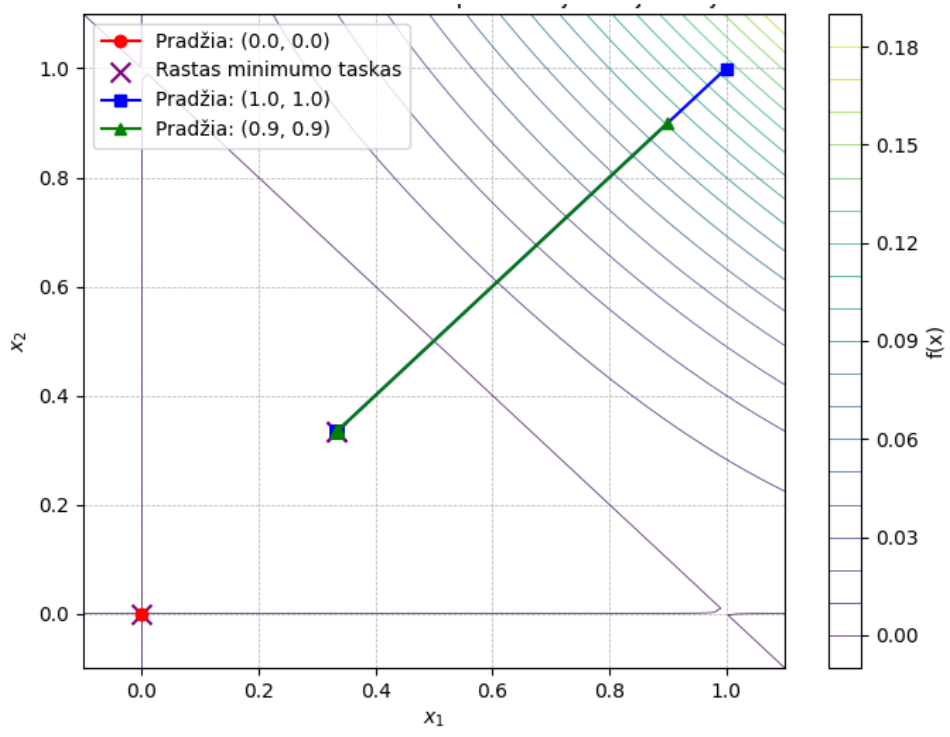
Kai pradinis taškas yra X_m= (0,9; 0,9) greičiausio nusileidimo metodas teoriškai turėtų būti efektyvesnis, tačiau praktikoje jis gali užtrukti ilgiau dėl papildomų skaičiavimų optimaliam žingsniui rasti. Jei pradinis taškas nepalankus, šis metodas gali atlikti daugiau iteracijų ir sukelti daugiau funkcijos kvietimų nei gradientinis nusileidimas. Rezultatai 9 pav.

```
Pradinis taskas: (0.9, 0.9)
f(x) = -0.0046
Rastas minimumo taskas: (0.3333, 0.3333)
Iteraciju skaicius: 46
Funkcijos kvietimu skaicius: 47
```

9 pav. Taško X_m= (0,9; 0,9) rezultatai

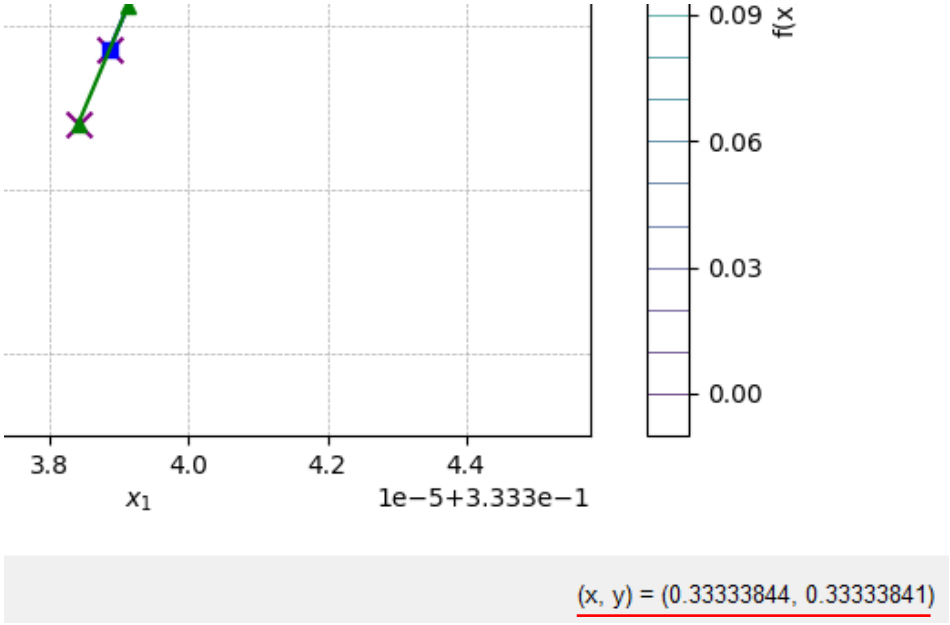
1.2.2. Vizualizacija

Greičiausio nusileidimo metodo trajektorijos nuo skirtingų pradinių taškų X_0, X_1, X_m pavaizduotos grafike (10 pav.).



10 pav. Greičiausio nusileidimo metodo trajektorijos vizualizacija

Minimumo taškai, pažymėti kryžiuku iš arčiau 11 pav.



11 pav. Taškų X_1=(1;1) ir X_m= (0,9; 0,9) rasti minimumo taškai

1.3. Deformuojamo simplekso metodas

Deformuojamo simplekso metodas siekia rasti funkcijos minimumą, iteratyviai atnaujindamas taškų rinkinį (simpleksą) pagal tam tikras transformacijas. Kiekviename žingsnyje surandama blogiausia simplekso viršūnė, o tada ji keičiama pagal įvairias operacijas: atspindį, išplėtimą, suspaudimą ar sumažinimą. Šios operacijos leidžia efektyviai ieškoti minimumo, prisitaikant prie funkcijos formos. Procesas kartojamas tol, kol simplekso viršūnės tampa pakankamai artimos viena kitai arba pasiekiamas maksimalus iteracijų skaičius.

Pvz.: pradinis taškas yra X_1 = (1;1). Sudaromas pradinis simpleksas su trimis viršūnėmis, viena iš jų – X_1, o kitos dvi nustatomos pagal specialias formulės reikšmes.

$$S_1 = (1;1), S_2 = (1+\delta_1; 1+\delta_2),\ S_3 = (1+\delta_2; 1+\delta_1)$$

kur δ_1, δ_2 => $\delta_1 = \frac{\sqrt{n+1}+n-1}{n\sqrt{2}}\alpha$, $\delta_2 = \frac{\sqrt{n+1}-1}{n\sqrt{2}}\alpha$, α – mastelio koeficientas. Įsistatome reikšmes α=0.1, n=2:

$$\delta_1 = \frac{\sqrt{2+1}+2-1}{2\sqrt{2}} * 0.1 \approx 0.072\,, \qquad \delta_2 = \frac{\sqrt{2+1}-1}{2\sqrt{2}} * 0.1 \approx 0.028$$

Iš ko seka: $S_1 = (1;1), S_2 = (1.072; 1.028),\ S_3 = (1.028; 1.072)$

Apskaičiuojamos funkcijos reikšmės taškuose $S_1, S_2,\ S_3$, ir surandamas blogiausias taškas (tas, kur funkcijos reikšmė didžiausia).

$$f(S_1) = 0.125, f(S_2) \approx 0.152\,, f(S_3) \approx 0.152$$

Čia $S_{blogiausias}$=S_3.

Apskaičiuojame centroidą (geriausių dviejų taškų vidurkį):

$$C = \frac{S_{1geriausias} + S_{2geriausias}}{n}$$

$$C = \frac{(1;1) + (1.028; 1.072)}{2} = (1.036; 1.014)$$

Sukuriamas naujas taškas simetriškai blogiausiam taškui (atspindys) $S_{blogiausias}$:

$$R = C + \alpha(C - S_{blogiausias})$$

$$R = (1.036; 1.014) + 0.1 * \big((1.036; 1.014) - (1; 1)\big) = (1.0396; 1.0154)$$

Jei f(R) yra geresnė nei blogiausio taško, jis pakeičiamas.

$$\text{f(R)} = f(1.0396; 1.0154) = 0.1398$$

f(R) < f($S_{blogiausias}$) => $0.1398 < 0.152$ => todėl atspindžio taškas gali pakeisti S_3 ($S_{blogiausias}$). Atnaujintas simpleksas atrodo taip:

$$S_1 = (1; 1), S_2 = (1.072; 1.028),\ S_3 = (1.0396; 1.0154)$$

Procedūra kartojama tol, kol simplekso viršūnių skirtumas tampa pakankamai mažas arba pasiekiamas maksimalus iteracijų skaičius.

Jei f(R) būtų geriausia reikšmė, būtų atliekamas papildomas išplėtimas:

$$E = C + \gamma(R - C)$$

Jei f(E) < f(R), tai $S_{blogiausias}$ keičiamas į E, kitaip į R.

Jei atspindžio taškas nebūtų geresnis, taikytume suspaudimą:

$$S = C + \rho(S_{blogiausias} - C)$$

Jei ir suspaudimas nepadėtų, simpleksas būtų sumažintas aplink geriausią tašką.

Kiekvienoje iteracijoje: Įvertiname funkcijos reikšmes visose simplekso viršūnėse ir nustatome blogiausią tašką. Apskaičiuojame centroidą – vidurkį iš geriausių viršūnių (išskyrus blogiausią). Atliekame atspindžio operaciją, sukurdami naują tašką simetriškai prieš blogiausią tašką pagal centroidą. Jei atspindžio taškas yra geresnis, galime atlikti išplėtimą, judant toliau ta pačia kryptimi. Jei atspindžio taškas nėra pakankamai geras, taikomas suspaudimas arba kontrakcija, mažinant žingsnį ir priartėjant prie minimumo. Jei nė viena iš ankstesnių operacijų neveda prie pagerėjimo, simpleksas sumažinamas aplink geriausią viršūnę. Procedūra kartojama tol, kol simplekso viršūnių skirtumas tampa pakankamai mažas arba pasiekiamas maksimalus iteracijų skaičius.

1.3.1. Rezultatai

Deformuojamo simplekso metodas visais atvejais surado minimumo tašką (0.33, 0.33), tačiau tam prireikė daugiau iteracijų nei gradientinio ar greičiausio nusileidimo metodams. Pradėjus nuo X_0= (0;0), minimumas pasiektas per 48 iteracijas, atliekant 90 funkcijos kvietimų (12 pav.). Pradėjus nuo X_1=(1;1) (13 pav.) ir X_m= (0,9; 0,9) (14 pav.), reikėjo 59 iteracijų su daugiau nei 110 funkcijos kvietimų. Šis metodas užtrunka ilgiau, nes naudoja sudėtingesnes transformacijas (atspindį, išplėtimą, suspaudimą).

```
Pradinis taskas: (0.0, 0.0)
f(x) = -0.0046
Rastas minimumo taskas: (0.3333, 0.3333)
Iteraciju skaicius: 48
Funkcijos kvietimu skaicius: 90
```

12 pav. Taško X_0= (0;0) rezultatai

```
Pradinis taskas: (1.0, 1.0)
f(x) = -0.0046
Rastas minimumo taskas: (0.3333, 0.3333)
Iteraciju skaicius: 59
Funkcijos kvietimu skaicius: 111
```

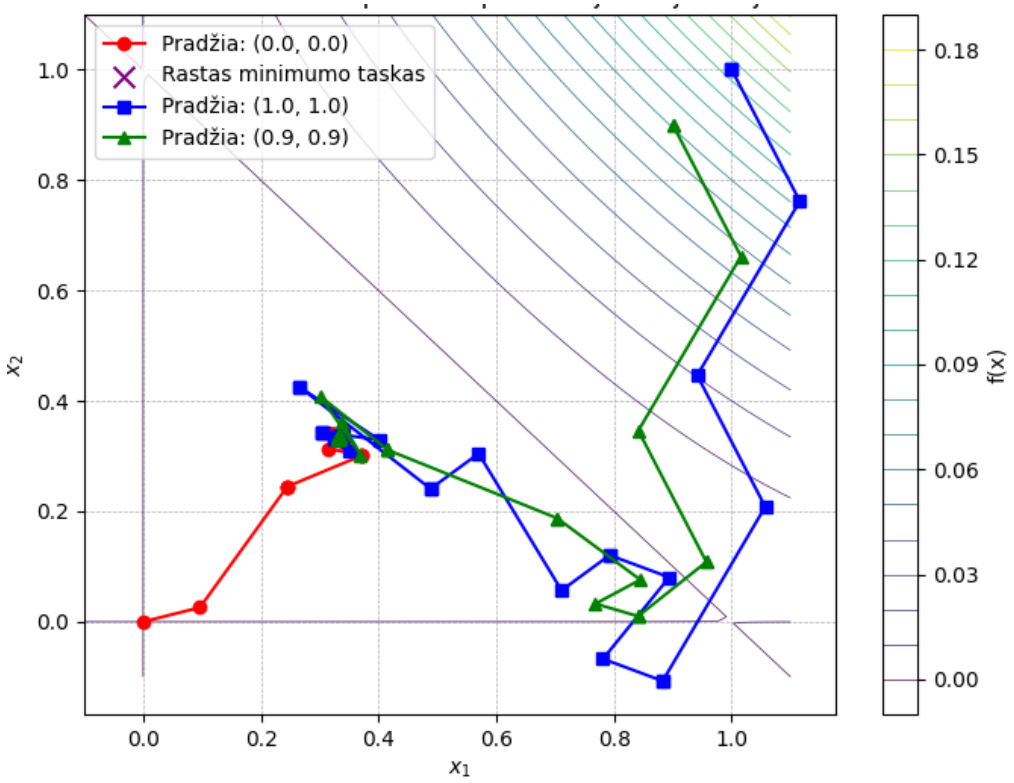
13 pav. Taško X_1= (1;1) rezultatai

```
Pradinis taskas: (0.9, 0.9)
f(x) = -0.0046
Rastas minimumo taskas: (0.3333, 0.3333)
Iteraciju skaicius: 59
Funkcijos kvietimu skaicius: 112
```

14 pav. Taško X_m= (0,9; 0,9) rezultatai

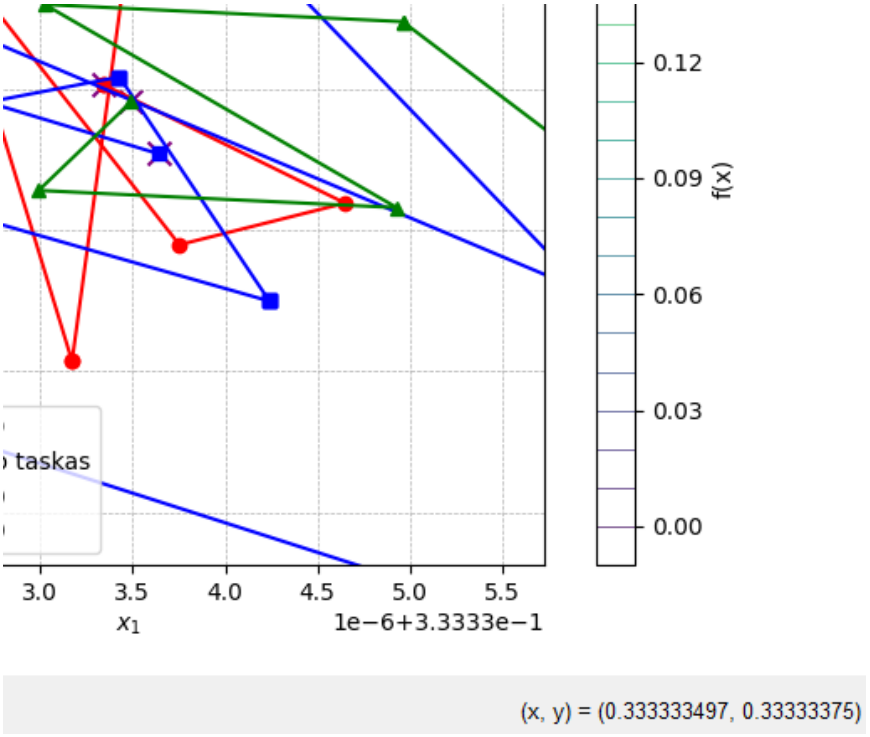
1.3.2. Vizualizacija

Deformuojamo simplekso metodo trajektorijos nuo skirtingų pradinių taškų X_0, X_1, X_m pavaizduotos grafike (15 pav.).



15 pav. Deformuojamo simplekso trajektorijų vizualizacija

Minimumo taškai, pažymėti kryžiuku iš arčiau 16 pav.



16 pav. Taškų X_1=(1;1) ir X_m= (0,9; 0,9) rasti minimumo taškai

2. IŠVADOS

Algoritmų rezultatų palyginimui paruošta 2 lentelė. (0.3333, 0.3333)

2 lentelė. Įgyvendintų metodų rezultatų palyginimas

Metodas	Pradinis taškas	Rasto minimumo taško $f(x)$	Rastas minimumo taškas	Iteracijų skaičius	Funkcijos įvertinimų skaičius
Gradientinis nusileidimas	X_0= (0;0)	-0.0	(0.0, 0.0)	0	1
	X_1=(1;1)	-0.0046	(0.3333, 0.3333)	23	24
	X_m= (0,9; 0,9)	-0.0046	(0.3333, 0.3333)	17	18
Greičiausias nusileidimas	X_0= (0;0)	-0.0	(0.0, 0.0)	0	1
	X_1=(1;1)	-0.0046	(0.3333, 0.3333)	13	14
	X_m= (0,9; 0,9)	-0.0046	(0.3333, 0.3333)	46	47
Deformuojamo simplekso metodas	X_0= (0;0)	-0.0046	(0.3333, 0.3333)	48	90
	X_1=(1;1)	-0.0046	(0.3333, 0.3333)	59	111
	X_m= (0,9; 0,9)	-0.0046	(0.3333, 0.3333)	59	112

Remiantis atliktų eksperimentų rezultatais, galima daryti kelias išvadas apie nagrinėtus optimizavimo metodus. Gradientinis nusileidimas užtikrina pakankamai greitą minimumo radimą, ypač kai pradinis taškas yra arti minimumo, tačiau gali reikalauti daugiau iteracijų, jei pradinis taškas yra toliau. Greičiausias nusileidimas dažniausiai reikalauja mažiau iteracijų nei gradientinis nusileidimas, tačiau kai kuriais atvejais (pvz., Xm = (0.9; 0.9)) šis metodas užtrunka ilgiau dėl papildomų skaičiavimų optimaliam žingsniui parinkti. Deformuojamo simplekso metodas, nenaudodamas gradientinės informacijos, pasiekia tą patį minimumą, tačiau reikalauja daugiau iteracijų ir funkcijos įvertinimų, todėl yra mažiau efektyvus skaičiavimų atžvilgiu. Apibendrinant, gradientiniai metodai yra efektyvesni skaičiavimo prasme, tačiau deformuojamas

simpleksas gali būti tinkamesnis, kai funkcijos gradientas nėra žinomas arba sunkiai apskaičiuojamas.

3. PRIEDAS

Žemiau pateikiamas kodas su anksčiau aprašytais įgyvendintais algoritmais.

2_laboratorinis.py

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

def tikslo_funkcija(X):
    x1, x2 = X
    return - x1 * x2 * (1 - x1 - x2)/8

def gradientas(X):
    x1, x2 = X
    df_dx1 = - 1/8 *(x2 - x2**2 - 2 * x1 * x2)
    df_dx2 = - 1/8*( x1 - x1**2 - 2 * x1 * x2)
    return np.array([df_dx1, df_dx2])


def gradient_nusileidimas(f, grad_f, X0, zings=3.0, epsilon=1e-6, max_iter=1000):
    X = np.array(X0, dtype=float)
    kelias = [X.copy()]
    i = 0
    kvietimai = 0

    while i < max_iter:
        grad = grad_f(X)
        kvietimai += 1
        if np.linalg.norm(grad) <= epsilon:
            break
        X -= zings * grad
        kelias.append(X.copy())
        i += 1

    return X, f(X), i, kvietimai, np.array(kelias)


X0 = np.array([0.0, 0.0])
X1 = np.array([1.0, 1.0])
Xm = np.array([0.9, 0.9])

# Gradientinio nusileidimo rezultatai
results_gd = {}
for X_init in [X0, X1, Xm]:
    X_min, f_min, zin, fk, kelias = gradient_nusileidimas(tikslo_funkcija, gradientas, X_init)
```

```
    results_gd[tuple(X_init)] = (X_min, f_min, zin, fk, kelias)

print("Gradientinis nusileidimas-------------------------------")
for X_init, (X_sol, f_val, iterations, kvietimai, _) in results_gd.items():
    X_init_fmt = tuple(round(float(coord), 4) for coord in X_init)
    X_sol_fmt = tuple(round(float(coord), 4) for coord in X_sol)
    print(f"Pradinis taskas: {X_init_fmt}")
    print(f"f(x) = {round(f_val,4)}")
    print(f"Rastas minimumo taskas: {X_sol_fmt}")
    print(f"Iteraciju skaicius: {iterations}")
    print(f"Funkcijos kvietimu skaicius: {kvietimai}\n")

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
    x_dabar = x0

    while itera < max_iter:
        f_isvest_reiksme = f_isvestine_func(x_dabar)
        f_antra_isv_reiks = f_antra_isv_func(x_dabar)
        itera += 1

        if abs(f_isvest_reiksme) < tiks:
            break
        if f_antra_isv_reiks == 0:
            print("Klaida: antra isvestine lygi 0")
            break

        x_naujas = x_dabar - f_isvest_reiksme / f_antra_isv_reiks

        if abs(x_naujas - x_dabar) < tiks:
            break

        x_dabar = x_naujas

    return x_dabar

def greiciausias_nusileidimas(X0, gradientas, tikslo_funkcija, max_iter=1000, tol=1e-6):
    X = np.array(X0, dtype=float)
    kelias = [X.copy()]
    i = 0
    kvietimai = 0
```

```
    while i < max_iter:
        grad = gradientas(X)
        kvietimai += 1
        norm_grad = np.linalg.norm(grad)
        if norm_grad < tol:
            break

        def phi(opt_zings):
            return tikslo_funkcija(X - opt_zings * grad)

        opt_zings = niutono_met(phi, 1.0)

        X -= opt_zings * grad
        kelias.append(X.copy())
        i += 1

    return X, tikslo_funkcija(X), i, kvietimai, kelias

results_gn = {}
for X_init in [X0, X1, Xm]:
    X_min, f_min, zin, fk, kelias = greiciausias_nusileidimas(X_init, gradientas, tikslo_funkcija)
    results_gn[tuple(X_init)] = (X_min, f_min, zin, fk, kelias)

print("Greiciausias nusileidimas-------------------------------")
for X_init, (X_sol, f_val, iterations, kvietimai, _) in results_gn.items():
    X_init_fmt = tuple(round(float(coord), 4) for coord in X_init)
    X_sol_fmt = tuple(round(float(coord), 4) for coord in X_sol)
    print(f"Pradinis taskas: {X_init_fmt}")
    print(f"f(x) = {round(f_val,4)}")
    print(f"Rastas minimumo taskas: {X_sol_fmt}")
    print(f"Iteraciju skaicius: {iterations}")
    print(f"Funkcijos kvietimu skaicius: {kvietimai}\n")



def deformuojamo_simplekso(X0, max_iter=1000, tol=1e-6):
    alpha, gamma, rho, sigma = 1.0, 3.0, 0.5, 0.5
    n = len(X0)
    laikina_virsune1 = (np.sqrt(n+1) + n-1)/(n*np.sqrt(2)) * 0.1
    laikina_virsune2 = (np.sqrt(n+1) - 1)/(n*np.sqrt(2)) * 0.1

    simpleksas = np.array([
        X0,
        [X0[0] + laikina_virsune1, X0[1] + laikina_virsune2],
        [X0[0] + laikina_virsune2, X0[1] + laikina_virsune1]
    ])
    kelias = [X0.copy()]
    kvietimai = 0
    for _ in range(max_iter):
        reiksmes = np.array([tikslo_funkcija(p) for p in simpleksas])
        indeksai = np.argsort(reiksmes)
```

```
        simpleksas = simpleksas[indeksai]
        centroidas = np.mean(simpleksas[:-1], axis=0)
        atspindys = centroidas + alpha * (centroidas - simpleksas[-1])
        kvietimai += 1
        if tikslo_funkcija(atspindys) < reiksmes[indeksai[0]]:
            ispletimas = centroidas + gamma * (atspindys - centroidas)
            kvietimai += 1
            simpleksas[-1] = ispletimas if tikslo_funkcija(ispletimas) < tikslo_funkcija(atspindys) else
atspindys
        else:
            if tikslo_funkcija(atspindys) < reiksmes[indeksai[1]]:
                simpleksas[-1] = atspindys
            else:
                suspaudimas = centroidas + rho * (simpleksas[-1] - centroidas)
                kvietimai += 1
                simpleksas[-1] = suspaudimas if tikslo_funkcija(suspaudimas) < reiksmes[indeksai[-1]]
else simpleksas[0] + sigma * (simpleksas - simpleksas[0])

        kelias.append(simpleksas[0].copy())
        if np.max(np.linalg.norm(simpleksas - centroidas, axis=1)) < tol:
            break

    return simpleksas[0], tikslo_funkcija(simpleksas[0]), len(kelias), np.array(kelias), kvietimai



    # Deformuoto simplekso metodo rezultatai
results_ds = {}
for X_init in [X0, X1, Xm]:
    X_min, f_min, zin, kelias, kvietimai = deformuojamo_simplekso(X_init)
    results_ds[tuple(X_init)] = (X_min, f_min, zin, kelias, kvietimai)

print("Deformuotas Simpleksas----------------------------------")
for X_init, (X_sol, f_val, iterations, _, kvietimai) in results_ds.items():
    X_init_fmt = tuple(round(float(coord), 4) for coord in X_init)
    X_sol_fmt = tuple(round(float(coord), 4) for coord in X_sol)
    print(f"Pradinis taskas: {X_init_fmt}")
    print(f"f(x) = {round(f_val,4)}")
    print(f"Rastas minimumo taskas: {X_sol_fmt}")
    print(f"Iteraciju skaicius: {iterations}")
    print(f"Funkcijos kvietimu skaicius: {kvietimai}\n")

def plot_optimization(paths, method_name):
    x = np.linspace(-0.1, 1.1, 100)
    y = np.linspace(-0.1, 1.1, 100)
    X, Y = np.meshgrid(x, y)
    Z = tikslo_funkcija([X, Y])

    plt.figure(figsize=(8, 6))
    plt.contour(X, Y, Z, levels=20, cmap='viridis', linewidths=0.5)
    plt.colorbar(label='f(x)')
```

```
    markers = ['o', 's', '^']
    colors = ['r', 'b', 'g']

    for i, (start_point, path) in enumerate(paths.items()):
        path = np.array(path)
        start_point_fmt = tuple(round(float(coord), 4) for coord in start_point)
        plt.plot(path[:, 0], path[:, 1], linestyle='-', marker=markers[i], color=colors[i],
label=f'Pradžia: {start_point_fmt}')
        plt.scatter(path[-1, 0], path[-1, 1], color='purple', marker='x', s=100, label='Rastas minimumo
taskas' if i == 0 else "")

    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title(f'{method_name} optimizacijos trajektorija')
    plt.legend()
    plt.grid(True, linestyle='--', linewidth=0.5)
    plt.show()

# Gradientinio nusileidimo trajektorijos
keliai_gd = {tuple(X_init): kelias for X_init, (_, _, _, _, kelias) in results_gd.items()}
plot_optimization(keliai_gd, 'Gradientinis nusileidimas')

# Greičiausio nusileidimo trajektorijos
keliai_gn = {tuple(X_init): kelias for X_init, (_, _, _, _, kelias) in results_gn.items()}
plot_optimization(keliai_gn, 'Greičiausias nusileidimas')

# Deformuoto Simplekso trajektorijos
keliai_ds = {tuple(X_init): kelias for X_init, (_, _, _, kelias, _) in results_ds.items()}
plot_optimization(keliai_ds, 'Deformuotas Simpleksas')
```