Vilniaus universitetas

Matematikos ir informatikos fakultetas

Programų sistemų studijų programa

Optimizavimo metodų antrojo laboratorinio darbo ataskaita

Ataskaitą tikrino: Prof. Dr. Pranas Katauskis

Ataskaitą parengė: Dominykas Daunoravičius

Vilnius

2022

Įvadas

Laboratorinio darbo formulavimas: Suprogramuoti gradientinio nusileidimo, greičiausiojo nusileidimo ir deformuojamo simplekso algoritmus. Apsirašyti tikslo funkciją taip, kad uždavinys būtų formuluojamas be apribojimų, apskaičiuoti tikslo ir gradiento funkcijų reikšmes taškuose $X_0 = (0,0)$, $X_1 = (1,1)$ ir $X_m = (a/10, b/10)$, kur a = 3, b = 9. Vėliau minimizuoti suformuluotą uždavinį, naudojant suprogramuotus algoritmus, pradedant iš taškų X0, X1, Xm bei vizualizuoti tikslo funkciją ir bandymo taškus.

Laboratorinio darbo tikslas: Naudojantis suprogramuotais gradientinio nusileidimo, greičiausiojo nusileidimo ir deformuojamo simplekso algoritmais rasti tikslo bei gradiento funkcijų minimumus bei surasti kokie turėtų būti stačiakampio gretasienio formos dėžės matmenys, kad vienetiniam paviršiaus plotui jos tūris būtų maksimalus.

Darbo eiga

Norint atlikti laboratorinį darbą, pirmiausiai reikėjo susirasti tikslo funkciją, išsireiškiant ją per vieną iš kintamųjų, kurį gavome iš paviršiaus ploto reikalavimo. Tuomet galima buvo susirasti gradiento funkciją, apsiskaičiuoti funkcijų reikšmes duotuose taškuose bei minimizuoti suformuluotą uždavinį pasinaudojant optimizavimo algoritmais.

Laikant kintamaisiais dėžės priekinės ir galinės sienų plotų sumą, šoninių sienų plotų sumą, viršutinės ir apatinės sienų plotų sumą reikėjo aprašyti vienetinio dėžės paviršiaus ploto reikalavimą ir dėžės tūrio pakelto kvadratu funkciją, vėliau iš vienetinio paviršiaus ploto reikalavimo išvesti vieno iš kintamųjų išraišką per kitus.

$V^2 = (abc)^2$, kur a,b,c – dežės kraštinių ilgiai.

$x_1 + x_2 + x_3 = 1$, kur x_1, x_2, x_3 – priešingų dėžės sienų plotų sumos.

$x_3 = 1 - x_1 - x_2$

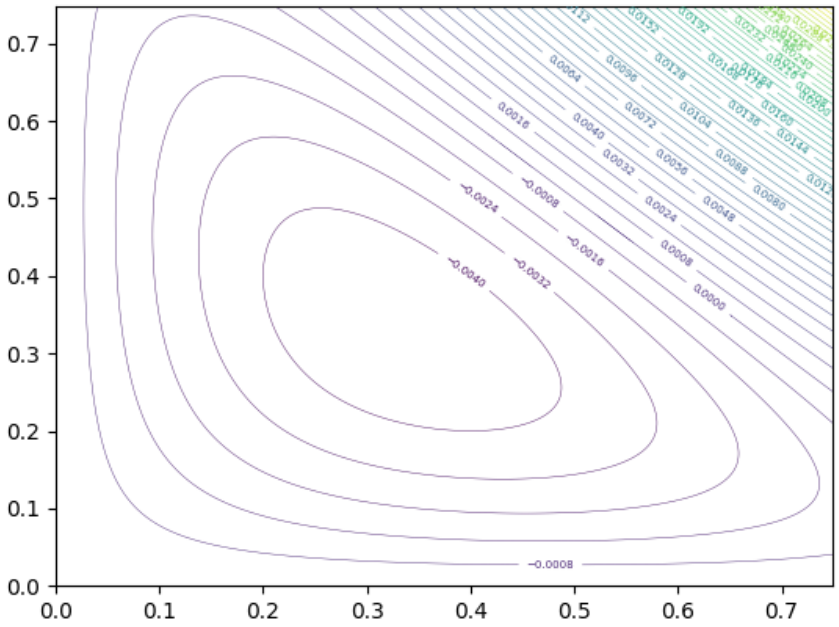
$ab = 0.5x_1$, $bc = 0.5x_2$, $ac = 0.5x_3$

$V^2 = 0.5x_1 * 0.5x_2 * 0.5\, x_3 = 0.125 * x_1 * x_2 * (1 - x_1 - x_2)$

Gauta tikslo funkcija: $f(X) = 0.125 * x_1 * x_2 * (1 - x_1 - x_2)$

Tam, kad gauti galutinio uždavinio atsakymą į klausimą kokia turėtų būti stačiakampio gretasienio formos dėžė, kad vienetiniam paviršiaus plotui jos tūris būtų maksimalus reikia spręsti optimizavimo uždavinį be apribojimų ir susidaryti min f(X). f(X) reikia padauginti iš -1, kad galėtume skaičiuoti funkcijos minimumą. Taigi tikslo funkcija, kurią naudosime uždavinyje yra:

$f(X) = -0.125 * x_1 * x_2 * (1 - x_1 - x_2)$



1 pav. Tikslo funkcijos grafikas

Gauta gradiento funkcija: $\nabla f(X) = (0.125 * x_2 * (2x_1 + x_2 - 1), 0.125 * x_1 * (x_1 + 2x_2 - 1))$

Gavę šia funkciją galime apsiskaičiuoti tikslo ir gradiento funkcijų reikšmes taškuose: $X_0 = (0,0)$, $X_1 = (1,1)$, $X_m = (0.3,0.9)$.

Tikslo ir gradiento funkcijų reikšmės taške (0, 0): $f(X) = 0.0$, $\nabla f(X) = [0.0, 0.0]$
Tikslo ir gradiento funkcijų reikšmės taške (1, 1): $f(X) = 0.125$, $\nabla f(X) = [0.25, 0.25]$
Tikslo ir gradiento funkcijų reikšmės taške (0.3, 0.9): $f(X) = 0.00675$, $\nabla f(X) = [0.05625, 0.04125]$

Pasirinkta programavimo kalba: Python.

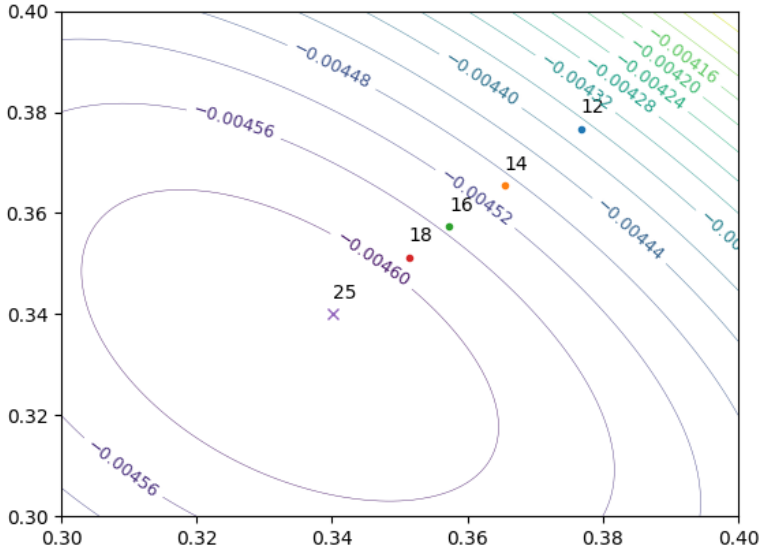
Metodų aprašymas bei analizė

Gradientinio nusileidimo metodas

Gradientinis nusileidimas yra optimizavimo metodas, besiremiantis tuo, kad skaliarinio lauko gradientas visada rodo greičiausio lauko augimo kryptį, o antigradientas mažėjimo kryptį. Gradientinio nusileidimo žingsnis skaičiuojamas pasinaudojant formule: $X_{i+1} = X_i - \gamma\nabla f(X_i)$, vėliau tikriname ar gradientas yra pakankamai mažas ir jei taip, tai skaičiavimus baigiame, jei ne - skaičiuojame kitą žingsnį.

Naudojant gradientinio nusileidimo metodą ir parametrus: gamma = 1, epsilon = 0.001, bei pradinį tašką $X_0(0,0)$ skaičiuojame minimumo tašką. Gavome, kad minimumui gauti mūsų funkcijai prireikė 1 iteracijos ir 2 funkcijų skaičiavimų. Pasirinkus bet kokį kitą gamma ar epsilon gradientinis nusileidimo metodas taip pat randa minimumo tašką per 1 iteraciją ir 2 funkcijų skaičiavimus dėl to, jog $X_0(0,0)$ taške gradientas lygus nuliui.

Toliau pasirinkam pradinį tašką $X_1(1,1)$ ir gamma = 1.



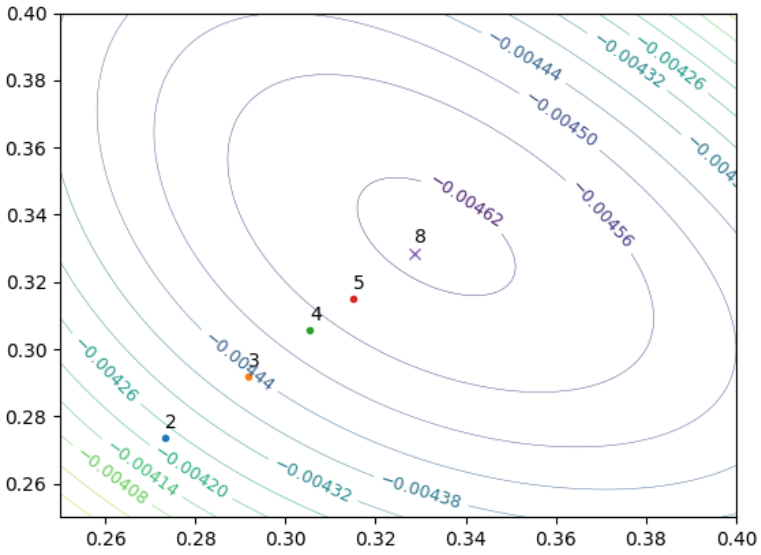
2 pav. Gradientinio nusileidimo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, gamma = 1, epsilon = 0.001

Gavome, kad minimumui gauti mūsų funkcijai prireikė 25 iteracijų ir 50 funkcijų skaičiavimų. Grafike vaizduojamas violetinis „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 12, 14, 16, 18, 25 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

1 lentelė. Gradientinio nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_1(1,1)$, gamma = 1, epsilon = 0.001

Iteracija	x_0	x_1	$F(X_1)$
1	0.75	0.75	0.03515625
2	0.6328125	0.6328125	0.013296246528625488
3	0.5617446899414062	0.5617446899414062	0.00487099377328859
23	0.3422901505128752	0.3422901505128752	-0.004619421918670533
24	0.34114046415018706	0.34114046415018706	-0.004621891754503968
25	0.34014171606373356	0.34014171606373356	-0.004623756471132563

Toliau bandom keisti tik gamma reikšmė į 3.



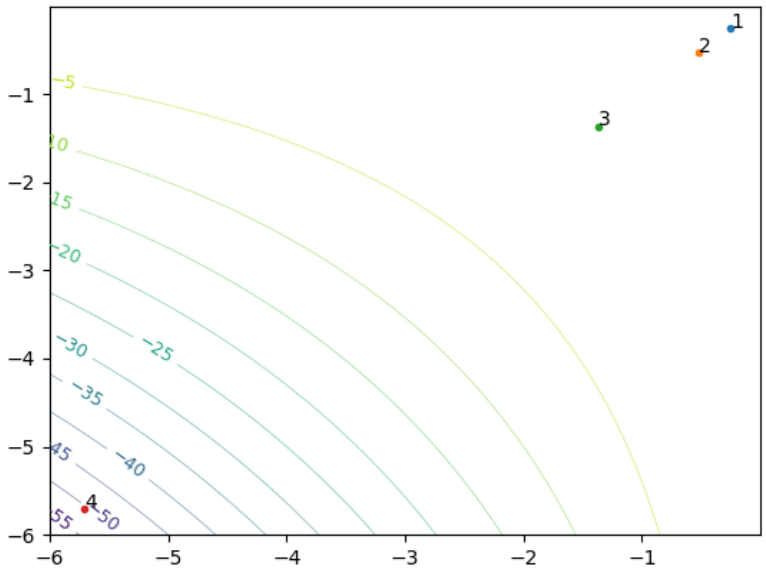
3 pav. Gradientinio nusileidimo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, gamma = 3, epsilon = 0.001

Gavome, kad minimumui gauti mūsų funkcijai prireikė 8 iteracijų ir 16 funkcijų skaičiavimų. Grafike vaizduojamas violetinis „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 2, 3, 4, 5, 8 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

2 lentelė. Gradientinio nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_1(1,1)$, gamma = 3, epsilon = 0.001

Iteracija	x_0	x_1	$F(X_1)$
1	0.25	0.25	-0.00390625
2	0.2734375	0.2734375	-0.004234910011291504
3	0.29186248779296875	0.29186248779296875	-0.004432481462648963
6	0.32153131484142505	0.32153131484142505	-0.004612629643396381
7	0.3258003731803469	0.3258003731803469	-0.004622643308922565 5
8	0.328561394562967	0.328561394562967	-0.004626810370607299

Bandom toliau didint gamma reiksmę į 5.



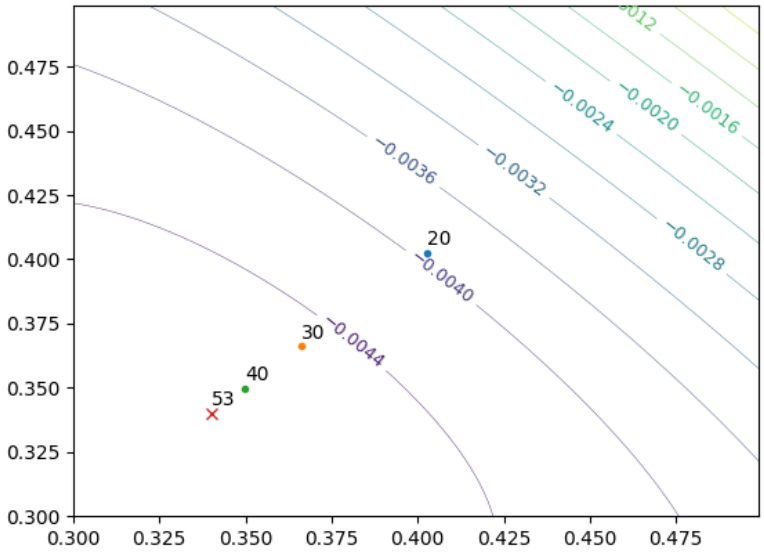
4 pav. Gradientinio nusileidimo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, gamma = 5, epsilon = 0.001

Tačiau toliau didinant gamma reikšmę metodui nebepavyko rasti minimumo iš šio pradinio taško, sunaudojo maksimalų kiekį iteracijų (1000) ir suskaičiavo 2000 funkcijų reikšmių, tai programa turėjo baigti darbą. Grafike pavaizduoti taškai tik po 1, 2, 3, 4 iteracijų. Grafike matosi kaip po pirmo žingsnio X „peršoka“ duobę ir lygus $(-0.25, -0.25)$, ir toliau staigiai artėja link $(-\infty, -\infty)$, nes funkcija toliau mažėja ta kryptimi.

3 lentelė. Gradientinio nusileidimo metodo rezultatai po šešių iteracijų, kai pradinis taškas $X_1(1,1)$, gamma = 5, epsilon = 0.001

Iteracija	x_0	x_1	$F(X_1)$
1	-0.25	-0.25	-0.01171875
2	-0.5234375	-0.5234375	-0.07010209560394287
3	-1.3643112182617188	-1.3643112182617188	-0.8675316378746868
4	-5.707027792690496	-5.707027792690496	-50.54098174517351
5	-70.34298183788785	-70.34298183788785	-87635.16114617664
6	-9392.060646446731	-9392.060646446731	-207131329650.32285
...			

Bandom gamma mažint pakeitus į 0.5.



5 pav. Gradientinio nusileidimo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, gamma = 0.5, epsilon = 0.001

Gavome, kad minimumui gauti mūsų funkcijai prireikė 53 iteracijų ir 106 funkcijų skaičiavimų. Grafike vaizduojamas raudonas „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 20, 30, 40, 53 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

4 lentelė. Gradientinio nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_1(1,1)$, gamma = 0.5, epsilon = 0.001

Iteracija	x_0	x_1	$F(X_1)$
1	0.875	0.875	0.07177734375
2	0.7861328125	0.7861328125	0.04420786281116307
3	0.7193902134895325	0.7193902134895325	0.028384830833357855
51	0.3411909837454413	0.3411909837454413	-0.004621790507800677
52	0.3406883038440598	0.3406883038440598	-0.004622768212858388
53	0.34021847526378685	0.34021847526378685	-0.004623622384357211

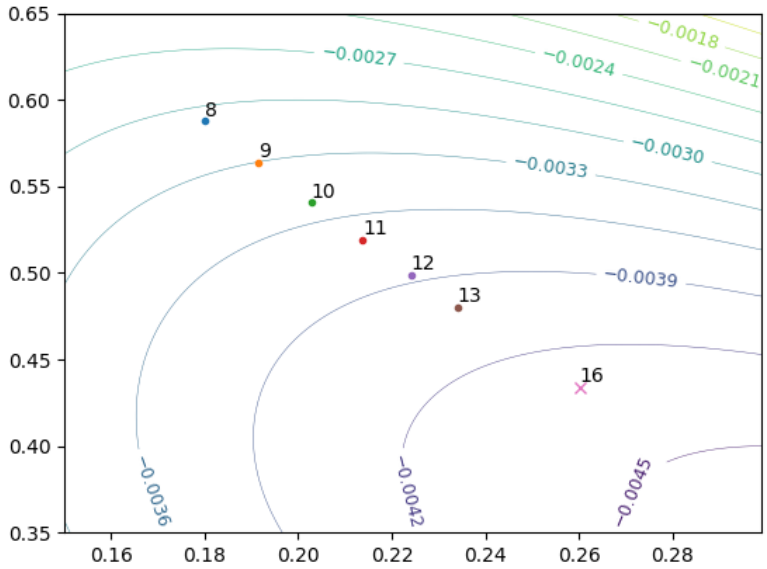
5 lentelė. Gradientinio nusileidimo metodo rezultatų su pradiniu tašku $X_1(1,1)$, epsilon = 0.001 palyginimas

	gamma = 0.5	gamma = 1	gamma = 3	gamma = 5
Iteracijų skaičius	53	25	8	1000
Skaičiuotų funkcijų kiekis	106	50	16	2000
Minimumo X reikšmė	(0.34021847526378685, 0.34021847526378685)	(0.34014171606373356, 0.34014171606373356)	(0.328561394562967, 0.328561394562967)	Nepasiektas
Funkcijos minimumo reikšmė	-0.004623622384357211	-0.004623756471132563	-0.004626810370607299	Nepasiektas
Apytikslis atstumas iki realaus Xmin	(0.00689, 0.00689)	(0.00681, 0.00681)	(0.00477, 0.00477)	-

Reali Xmin reikšmė = (1/3, 1/3).

Gradientinio nusileidimo metodui skaičiuojant tikslo funckijos f(X) = -0.125 * x_1 * x_2 * (1 – x_1 – x_2) minimumo tašką iš pradinio taško X_1(1,1), bandymais atrasta, jog renkantis gamma per mažą, metodui prireikia daug iteracijų ir funkcijų skaičiavimų atrasti minimui, o pasirinkus gamma per daug didelį, metodas neberanda minimumo. Bandimuose optimaliausia gamma reikšmė buvo 3, su ja prireikė tik 8 iteracijų ir 16 funkcijų skaičiavimų, taip pat atstumas iki realaus Xmin mažiausias, tik (0.00477, 0.00477).

Toliau atliekam skaičiavimus su gamma = 3, o pradinį tašką keičiam į X_m(0.3,0.9).



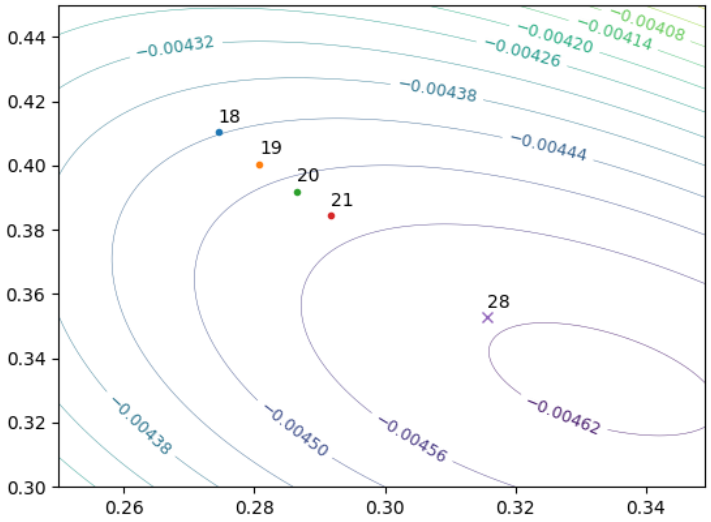
6 pav. Gradientinio nusileidimo grafiko ir taškų vaizdavimas, pradinis taškas X_m(0.3,0.9), gamma = 3, epsilon = 0.001

Gavome, kad minimumui gauti mūsų funkcijai prireikė 16 iteracijų ir 32 funkcijų skaičiavimų. Grafike vaizduojamas rausvas „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 8, 9, 10, 11, 12, 13, 16 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

6 lentelė. Gradientinio nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas X_m(0.3,0.9), gamma = 3, epsilon = 0.001

Iteracija	x_0	x_1	F(X_m)
1	0.13125	0.77625	-0.0011780200195312504
2	0.11997011718750002	0.7425966796875	-0.001530480384372479
3	0.12483312830035687	0.7153711047372007	-0.0017837604553312182
14	0.24330667998275995	0.46321449094417533	-0.004134499659155903
15	0.2520218549204253	0.4477278083074742	-0.004234925640417196
16	0.26011931709946806	0.4337899785817296	-0.004317300454129901

Rastas minimumas yra gerokai toliau realaus minimumo taško (1/3, 1/3), dėl to bandom mažinti epsilon į 0.0001.



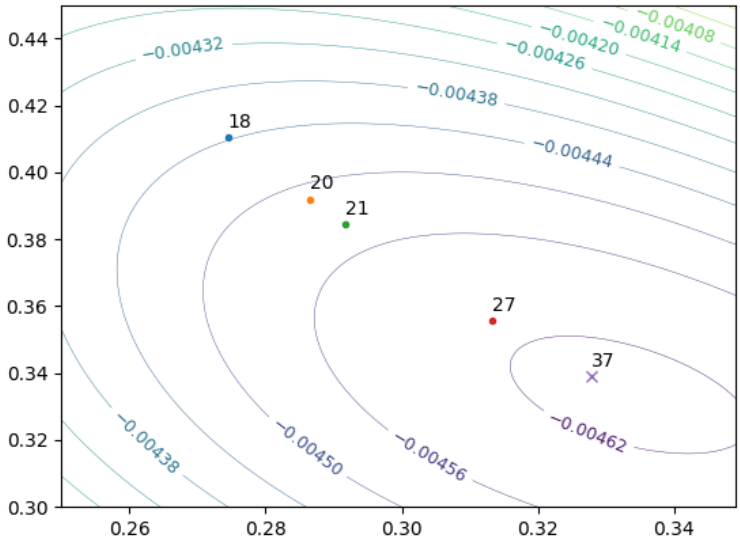
7 pav. Gradientinio nusileidimo grafiko ir taškų vaizdavimas, pradinis taškas $X_m(0.3,0.9)$, gamma = 3, epsilon = 0.0001

Gavome, kad minimumui gauti mūsų funkcijai prireikė 28 iteracijų ir 56 funkcijų skaičiavimų. Grafike vaizduojamas violetinis „x“ žymi, jog tai yra paskutinė iteracija ir rastas taškas. Grafike pavaizduoti taškai tik po 18, 19, 20, 21, 28 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

7 lentelė. Gradientinio nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_m(0.3,0.9)$, gamma = 3, epsilon = 0.0001

Iteracija	x_0	x_1	$F(X_m)$
1	0.13125	0.77625	-0.0011780200195312504
2	0.11997011718750002	0.7425966796875	-0.0015304803843724 79
3	0.12483312830035687	0.7153711047372007	-0.0017837604553312182
26	0.31064284149178367	0.358825799207017	-0.004605403573192116
27	0.3133190339785361	0.35552974975298135	-0.004611042164246589
28	0.31569648571869646	0.35266540279254016	-0.004615375606789164

Bandom toliau mažinti epsilon į 0.00001.



8 pav. Gradientinio nusileidimo grafiko ir taškų vaizdavimas, pradinis taškas $X_m(0.3,0.9)$, gamma = 3, epsilon = 0.00001

Gavome, kad minimumui gauti mūsų funkcijai prireikė 37 iteracijų ir 74 funkcijų skaičiavimų. Grafike vaizduojamas violetinis „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 18, 20, 21, 27, 37 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

8 lentelė. Gradientinio nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_m(0.3,0.9)$, gamma = 3, epsilon = 0.00001

Iteracija	x_0	x_1	$F(X_m)$
1	0.13125	0.77625	-0.0011780200195312504
2	0.11997011718750002	0.7425966796875	-0.001530480384372479
3	0.12483312830035687	0.7153711047372007	-0.0017837604553312182
35	0.32619335015898693	0.3407497857248024	-0.004627421885437685
36	0.3270703780003281	0.33980876827279605	-0.004627939018970874
37	0.3278413797643666	0.33898848624583305	-0.004628335076905019

9 lentelė. Gradientinio nusileidimo metodo rezultatų su pradiniu tašku $X_m(0.3,0.9)$, gamma = 3 palyginimas

	epsilon = 0.001	epsilon = 0.0001	epsilon = 0.00001
Iteracijų skaičius	16	28	37
Skaičiuotų funkcijų kiekis	32	56	74
Minimumo X reikšmė	(0.26011931709946806, 0.4337899785817296)	(0.31569648571869646, 0.35266540279254016)	(0.3278413797643666, 0.33898848624583305)
Funkcijos minimumo reikšmė	-0.004317300454129901	-0.004615375606789164	-0.004628335076905019
Apytikslis atstumas iki realaus Xmin	(0.07321, 0.10046)	(0.01764, 0.01933)	(0.00549, 0.00566)

Gradientinio nusileidimo metodui skaičiuojant tikslo funckijos $f(X) = -0.125 * x_1 * x_2 * (1 - x_1 - x_2)$ minimumo tašką iš pradinio taško $X_m(0.3,0.9)$, bandymais atrasta, jog renkantis per didelį epsilon atstumas iki realaus Xmin yra labai netikslus, sumažinus epsilon net iki 0.00001, rasta, jog atstumas iki realaus Xmin sumažėjo iki (0.00549, 0.00566), tačiau metodui prireikė daug daugiau funkcijų skaičiacimų, net 74, palyginus su 32 skaičiuojant su epsilon = 0.001. Optimaliausias rezultatas pasiektas su epsilon = 0.0001, nes nuo realaus Xmin nutole nedaug, tik per (0.01764, 0.01933), o iteracijų skaičius ir skaičiuotų funkcijų kiekis palyginus su kitais bandymais yra vidutinis.

Greičiausiojo nusileidimo metodas

Greičiausio nusileidimo metodo idėja yra eiti nusileidimo kryptimi iki pirmo lokaliojo minimumo šia kryptimi. Kiekvienam žingsniui apskaičiuoti naudojama formulė: $X_{i+1} = X_i - \arg\min_{\gamma \geq 0} f(X_i - \gamma \cdot \nabla f(X_i))\nabla f(X_i)$, kaip ir gradientinio nusileidimo metode, pasiekus atitinkamą tikslumą mūsų funkcija baigia darbą.

Norint apskaičiuoti šią formulė mums reikėjo gauti gamma, kuriai gauti būtų naudojamas vienas iš trijų 1 laboratoriniame darbe aptartų algoritmų: intervalo dalijimo pusiau, auksinio pjūvio ar Niutono metodas. Pasirinkau auksinio pjūvio metodą.

Naudojant greičiausiojo nusileidimo ir auksinio pjūvio metodą, ir parametrus: epsilon = 0.001, bei pradinį tašką $X_0(0,0)$ skaičiuojame minimumo tašką. Bei auksinio pjūvio metodui nustačiau intervalą [0,5] (toliau intervalą žymėsime raide L).

Gavome, kad minimumui gauti mūsų funkcijai prireikė 1 iteracijos ir 2 funkcijų skaičiavimų. Pasirinkus bet kokį kitą epsilon greičiausiojo nusileidimo metodas taip pat randa minimumo tašką per 1 iteraciją ir 2 funkcijų skaičiavimus dėl to, jog $X_0(0,0)$ taške gradientas lygus nuliui.

Toliau pasirinkam pradinį tašką $X_1(1,1)$.

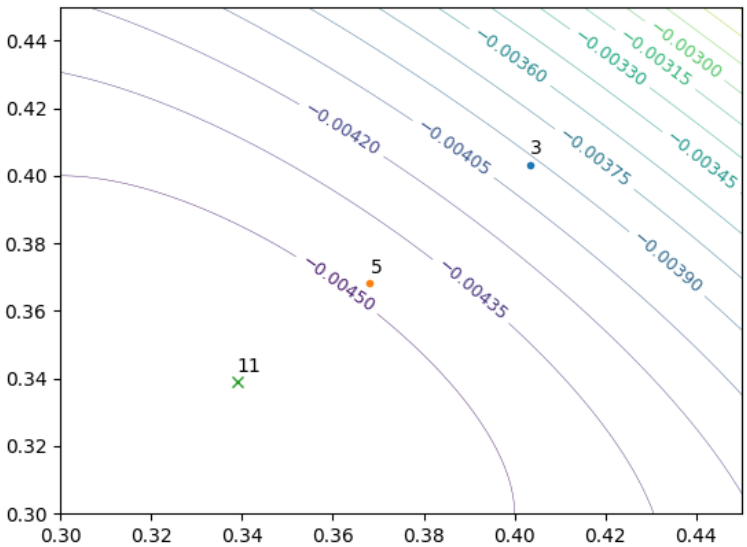
Metodas suskaičiavo minimumą per vieną iteraciją, atlikdamas 24 funkcijų skaičiavimus. Metodui prireikė tik vienos iteracijos, nes taške (1,1) antigradiento kryptis eina tiesiai per funkcijos minimumo tašką.

Iteracija	gamma	x_0	x_1	$F(X_1)$
1	2.6667107379742343	0.3333223155064414	0.3333223155064414	-0.0046296296144559

Gautas gamma yra pakankamai toli nuo L kraštų [0,5], dėl to gammos didint nereikėtų, nes minimumo taškas yra intervale.

Jei pasirinktume L per daug didelį, metodas nerastu minimumo taško. Pavyzdžiui su L = [0,15] pirminis apskaičiuotas gamma = 14.999621240814848 ir metodui žengus žingsnį X „peršoka“ duobę ir lygus $(-2.749905310203712, -2.749905310203712)$, kur funkcijos reikšmė lygi -6.143929101689989. Metodas toliau tolsta į $(-\infty, -\infty)$, nes funkcija toliau mažėja ta kryptimi.

Jei pasirinktume intervalą auksinio pjūvio metodui mažesnį nei gautas gamma, metodui prireiktų daugiau žingsnių rasti minimumą. Pavyzdžiui pasirinkus L = [0,2].



9 pav. Greičiausiojo nusileidimo metodo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, epsilon = 0.001, L = [0,2]

Gavome, kad minimumui gauti mūsų funkcijai prireikė 11 iteracijų ir 222 funkcijų skaičiavimų. Grafike vaizduojamas žalias „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 3, 5, 11 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

10 lentelė. Greičiausiojo nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_1(1,1)$, epsilon = 0.001, L = [0,2]

Iteracija	gamma	x_0	x_1	$F(X_1)$
1	1.9996538594565756	0.5000865351358561	0.5000865351358561	5.410318235441744e-06
2	1.9996538594565756	0.4375540863329645	0.4375540863329645	-0.002988873408043754
3	1.9996538594565756	0.4033583433571806	0.4033583433571806	-0.003930849931344273
9	1.9996538594565756	0.34337935671745745	0.34337935671745745	-0.004616760838736014
10	1.9996538594565756	0.34079260669907435	0.34079260669907435	-0.004622570774828338
11	1.9996538594565756	0.3388863877552185	0.3388863877552185	-0.004625732268882614

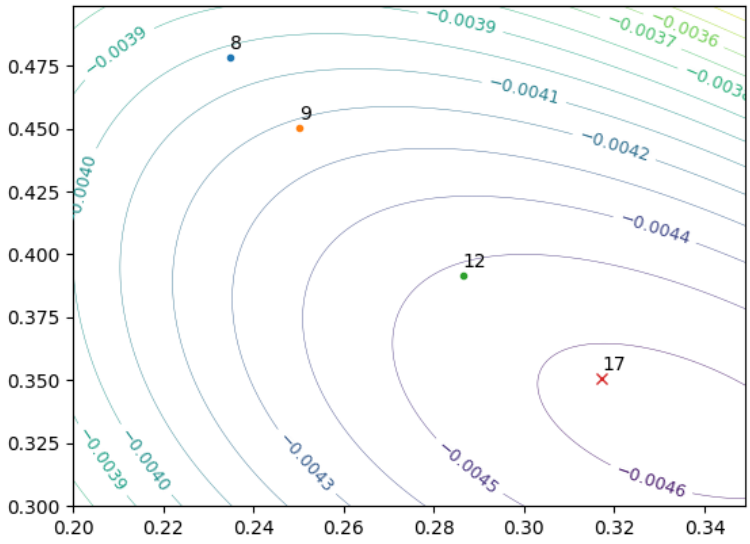
Taip pat matome, jog gamma išlieka arti L krašto, vadinas nėra pasiektas didžiausias gamma, nes pasiekti didesnį trukdo L apribojimai, dėl to reiktų didinti L intervalą.

11 lentelė. Greičiausiojo nusileidimo metodo rezultatų su pradiniu tašku $X_1(1,1)$, epsilon = 0.001 palyginimas

	L = [0,2]	L = [0,5]	L = [0,15]
Iteracijų skaičius	11	1	-
Skaičiuotų funkcijų kiekis	222	24	-
Minimumo X reikšmė	(0.3388863877552185, 0.3388863877552185)	(0.3333223155064414, 0.3333223155064414)	Nepasiekta
Funkcijos minimumo reikšmė	-0.004625732268882614	-0.0046296296144559	Nepasiekta
Apytikslis atstumas iki realaus Xmin	(0.00555,0.00555)	(0.00001, 0.00001)	-

Greičiausiojo nusileidimo metodui skaičiuojant tikslo funckijos $f(X) = -0.125 * x_1 * x_2 * (1 - x_1 - x_2)$ minimumo tašką iš pradinio taško $X_1(1,1)$, bandymais atrasta, jog renkantis per maža L prireiks daugiau iteracijų ir funkcijų skaičiavimų minimumui pasiekti, o pasirinkus L per didelį, funkcija gali peršokti mūsų minimumo tašką ir nukonverguoti į begalybę. Geriausias bandymas pavyko su L = [0,5], kur prireikė tik 1 iteracijos ir 24 funkcijų skaičiavimų bei X reikšmė labai mažai nutolus nuo realaus Xmin, tik per (0.00001, 0.00001).

Toliau bandom skaičiuoti su tašku $X_m(0.3,0.9)$ ir L = [0,5].



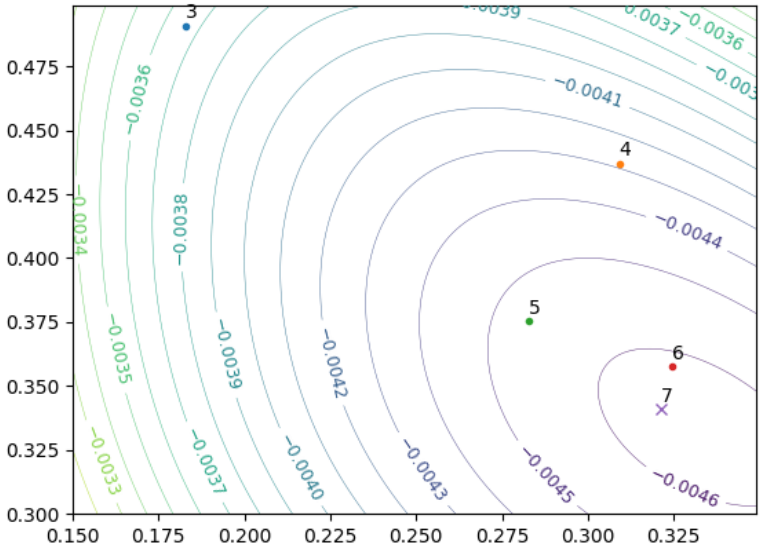
10 pav. Greičiausiojo nusileidimo metodo grafiko ir taškų vaizdavimas, pradinis taškas X_m(0.3,0.9), epsilon = 0.001, L = [0,5]

Gavome, kad minimumui gauti mūsų funkcijai prireikė 17 iteracijų ir 376 funkcijų skaičiavimų. Grafike vaizduojamas raudonas „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 8, 9, 12, 17 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

12 lentelė. Greičiausiojo nusileidimo metodo rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas X_m(0.3,0.9), epsilon = 0.001, L = [0,5]

Iteracija	gamma	x_0	x_1	$F(X_m)$
1	3.618329628068633	0.09646895842113942	0.750743902842169	-0.0013831720806661938
2	4.99966946519324	0.12289254307968413	0.7146936197905608	-0.0017831113993969912
3	4.99966946519324	0.1405448864994095	0.6722770071451359	-0.0022106926606823766
15	4.99966946519324	0.30835423155035646	0.36156876017349115	-0.00460008757802171
16	4.99966946519324	0.31326283044151637	0.35550002703772193	-0.004611025757089809
17	4.99966946519324	0.3172562342011231	0.35074992864901583	-0.004617934739246484

Matome, jog gamma pasiekė L krašta, dėl to toliau bandom su L = [0,15].



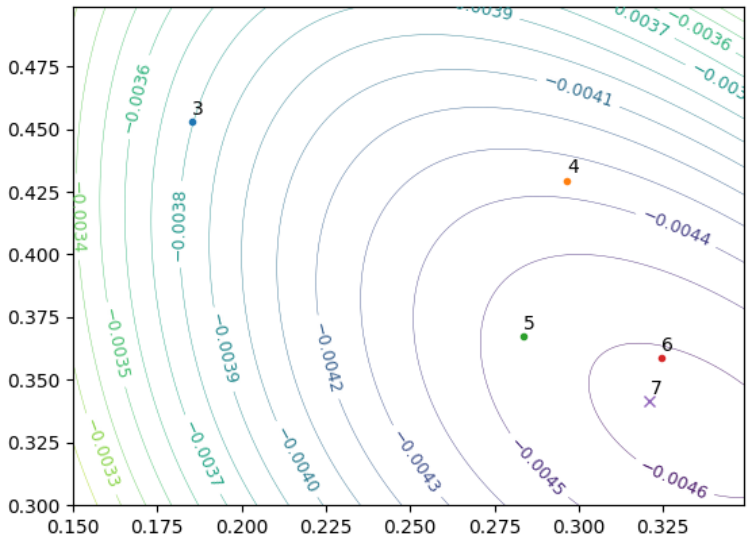
11 pav. Greičiausiojo nusileidimo metodo grafiko ir taškų vaizdavimas, pradinis taškas X_m(0.3,0.9), epsilon = 0.001, L = [0,15]

Gavome, kad minimumui gauti mūsų funkcijai prireikė 7 iteracijų ir 170 funkcijų skaičiavimų. Grafike vaizduojamas violetinis „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 3, 4, 5, 6, 7 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

13 lentelė. Greičiausiojo nusileidimo metodo rezultatai, kai pradinis taškas $X_m(0.3,0.9)$, epsilon = 0.001, L = [0,15]

Iteracija	gamma	x_0	x_1	$F(X_m)$
1	3.6183778524470243	0.09646624579985488	0.7507419135865603	-0.001383172086669951
2	14.999621240814848	0.17575046533863664	0.6425908969434413	-0.002564467022028476
3	14.999621240814848	0.18286879394538208	0.4907027465932018	-0.0036614753772160956
4	14.34361929308912	0.3091735456257898	0.4368413072882446	-0.004287897129336021
5	8.718673545514813	0.282899199331093	0.3752283078038398	-0.004536297041541873
6	14.999621240814848	0.32438899379637043	0.3575356295550527	-0.004611322547762996
7	10.455741096396913	0.3214387194428629	0.3408058272580708	-0.004625062266240865

Matom, jog kartais gamma vis dar siekia L kraštą, dėl to bandom su L = [0,17]



12 pav. Greičiausiojo nusileidimo metodo grafiko ir taškų vaizdavimas, pradinis taškas $X_m(0.3,0.9)$, epsilon = 0.001, L = [0,17]

Gavome, kad minimumui gauti mūsų funkcijai prireikė 7 iteracijų ir 177 funkcijų skaičiavimų. Grafike vaizduojamas violetinis „x“ žymi, jog tai yra paskutinė iteracija ir rastas minimumo taškas. Grafike pavaizduoti taškai tik po 3, 4, 5, 6, 7 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo.

14 lentelė. Greičiausiojo nusileidimo metodo rezultatai, kai pradinis taškas $X_m(0.3,0.9)$, epsilon = 0.001, L = [0,17]

Iteracija	gamma	x_0	x_1	$F(X_m)$
1	3.6184961999888583	0.09645958875062674	0.7507370317504596	-0.001383172091175534
2	16.999734702476694	0.1863443477367251	0.628176521760121	-0.00271395716592784
3	16.999734702476694	0.18518940881193194	0.45287917463267596	-0.003794327306060233
4	11.132686823846116	0.29657580156439833	0.42944130425020355	-0.004361872713246593
5	10.795052922444585	0.2834836590595771	0.36722779152569895	-0.004545251640214711
6	13.647401212211099	0.32470799637117836	0.3585523398838603	-0.0046095444707215
7	10.160117323489358	0.32107948279979326	0.3413094647017359	-0.004624742484039845

15 lentelė. Greičiausiojo nusileidimo metodo rezultatų su pradiniu tašku $X_m(0.3,0.9)$, epsilon = 0.001 palyginimas

	L = [0,5]	L = [0,15]	L = [0,17]
Iteracijų skaičius	17	7	7
Skaičiuotų funkcijų kiekis	376	170	177
Minimumo X reikšmė	(0.3172562342011231, 0.35074992864901583)	(0.3214387194428629, 0.3408058272580708)	(0.32107948279979326, 0.3413094647017359)
Funkcijos minimumo reikšmė	-0.004617934739246484	-0.004625062266240865	-0.004624742484039845
Apytikslis atstumas iki realaus Xmin	(0.01608, 0. 01742)	(0.01190, 0.00747)	(0.01225, 0.00798)

Greičiausiojo nusileidimo metodui skaičiuojant tikslo funckijos $f(X) = -0.125 * x_1 * x_2 * (1 – x_1 – x_2)$ minimumo tašką iš pradinio taško $X_m(0.3,0.9)$, bandymais atrasta, jog teisingai pasirinkus L, pasiekiamas optimaliausias metodo panaudojimas. Jei gamma nesilaiko ant L krašto, tai galima toliau gammos nedidinti, nes kuo didesnis L, tuo daugiau funkcijų skaičiavimų reikia atlikti auksinio pjūvio metodui, jog rastu didžiausią gamma, tai galim pamatyti, jog su L = [0,15] prireikė 170 funkcijų skaičiavimų, o su L = [0.17] – 177. Geriausias bandymas pavyko su L = [0,15], kur prireikė tik 7 iteracijos ir 170 funkcijų skaičiavimų bei X reikšmė mažai nutolus nuo realaus Xmin, tik per (0.01190, 0.00747).

Deformuojamo simplekso metodas

Nelder-Mead deformuojamo simplekso metodas buvo sukurtas dviejų britų statistikų Nelder ir Mead 1965m., jie pasiūlė strategiją su netaisyklingu deformuojamu simpleksu, kuris gali ir plėstis ir trauktis bet kuria kryptimi.

Paprastai tariant simplekso algoritmas veikia, naudojant 6 pagrindinius žingsnius:

1. Surikiavimu
2. Reflektavimu
3. Pratęsimu
4. Sutraukimu (viena forma sumažinimo, kai pasirenkamas blogiausias taškas ir jis mažinamas)
5. Simplekso mažinimas, kai pasiliekam tik geriausią viršūnę
6. Tikslumo tikrinimas

1, 2 ir 6 punktai yra atliekami kiekvienos iteracijos metu, tačiau ar mes rinksimės tašką pratęsti, sutraukti ar sumažinti du blogiausius taškus priklauso nuo 2 punkto.

Tai reiškia, kad jei 2 žingsnyje gavome tašką, kuris geresnis nei du blogiausi, mes jį bandysim praplėsti, jei geresnis tik už blogiausiąjį, mes jį pasirinksim.

Jei 2 žingsnyje gavome tašką, kuris blogesnis už blogiausiąjį, bandysime jį sutraukti, o jei sutraukus ir toliau gausim blogiausią tašką, bandysime esamus simplekso blogiausius taškus sutraukti arčiau geriausiojo, t.y. mažinsime simpleksą.

Simplekso deformavimo koeficientų reikšmės priklauso nuo sprendžiamų uždavinių. ($\gamma > 1$; $0 < \beta < 1$; $-1 < \nu < 0$). Eksperimentai parodė, kad geri rezultatai gaunami prie gamma = 2, beta = 0.5 ir niu = −0.5 bei alpha = 0.5, epsilon 0.001. Dėl to pradines reikšmes tokias ir pasirinksime.

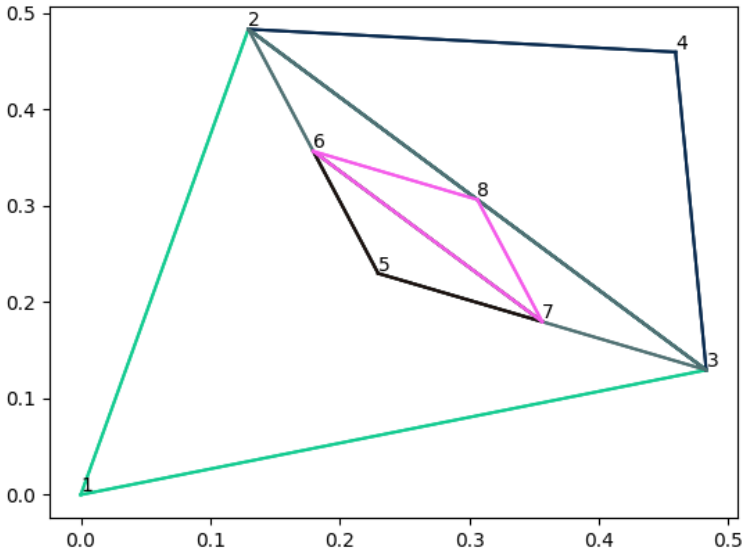
Pradinis deformuojamo simplekso metodo taškas pasirenkamas, o kitus du taškus apskaičiuoju su formule: $X_1 = (x_0 + alpha, x_1)$, $X_2 = (x_0, x_1 + alpha)$, kai $X_{pradinis} = (x_0, x_1)$.

Naudojant deformuojamo simpleso metoda ir pradinį tašką $X_0(0,0)$ skaičiuojame minimumo tašką. Gavome, kad minimumui gauti mūsų funkcijai prireikė 1 iteracijos ir 3 funkcijų skaičiavimų, dėl to, jog visų trijų pirminių tašku [(0,0), (0.5,0), (0,0.5)] funkcijos riekšmės yra lygios nuliui. Pasirinkus bet kokius kitus gamma, beta, niu, alpha ir epsilon parametrus deformuojamo simplekso metodas metodas taip pat randa minimumo tašką per 1 iteraciją ir 3 funkcijų skaičiavimus, dėl to, jog kad ir kokius parametrus pasirinkti, visų pradinių tašku funkcijos reikšmė bus lygi nuliui.

Tačiau kitokie rezultatai gaunami pabandžius pakeisti pirminiu tašku apskaičiavimo formulę į:

$delta_1 = \frac{\sqrt{3}+1}{2\sqrt{2}} * alpha$, $delta_2 = \frac{\sqrt{3}-1}{2\sqrt{2}} * alpha$;

$X_1 = (x_0 + delta_2; x_1 + delta_1)$, $X_2 = (x_0 + delta_1; x_1 + delta_2)$;



13 pav. Deformuojamo simplekso metodo grafiko ir taškų vaizdavimas, pradinis taškas $X_0(0,0)$, gamma = 2, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

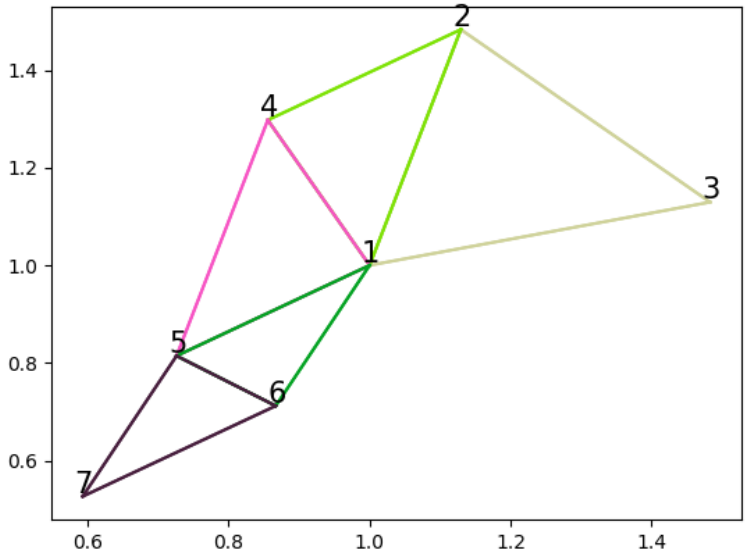
Gavome, kad minimumui gauti mūsų funkcijai prireikė 24 iteracijų ir 44 funkcijų skaičiavimų. Grafike pavaizduoti simpleksai ir jų virūnės po pirmų 5 iteracijų, jog grafike aiškiai

matytųsi artėjimas link minimumo. Grafike pavaizduotų simpleksų viršūnės: (1,2,3), (2,3,4), (2,3,5), (5,6,7), (6,7,8).

16 lentelė. Deformuojamo simplekso metodo geriausiojo taško[1] rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_0(0,0)$, gamma = 2, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(0.12940952255126034, 0.4829629131445341)	-0.0030283403461266044
2	(0.12940952255126034, 0.4829629131445341)	-0.0030283403461266044
3	(0.22963966338592293, 0.22963966338592293)	-0.0035643208440332428
22	(0.33355004943061595, 0.333131889473115)	-0.004629627800989286
23	(0.33355004943061595, 0.333131889473115)	-0.004629627800989286
24	(0.33355004943061595, 0.333131889473115)	-0.004629627800989286

Toliau bandome pakeisti pradinį tašką į $X_1(1,1)$



14 pav. Deformuojamo simplekso metodo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, gamma = 2, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

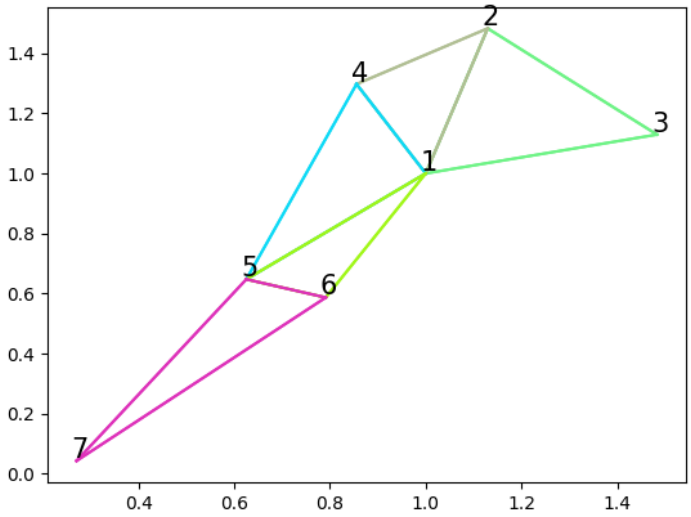
Gavome, kad minimumui gauti mūsų funkcijai prireikė 28 iteracijų ir 53 funkcijų skaičiavimų. Grafike pavaizduoti simpleksai ir jų virūnės po pirmų 5 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo. Grafike pavaizduotų simpleksų viršūnės: (1,2,3), (1,2,4), (1,4,5), (1,5,6), (5,6,7).

17 lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_1(1,1)$, gamma = 2, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(1, 1)	0.125
2	(1, 1)	0.125
3	(0.7261661627899177, 0.8145545104382361)	0.03997966476795941
26	(0.3334744142238716, 0.33442854651658055)	-0.004629572359527189
27	(0.33329853059088677, 0.334389708033079)	-0.004629584618741002
28	(0.33329853059088677, 0.334389708033079)	-0.004629584618741002

[1] Geriausiasis taškas – taškas, kuriame tikslo funkcijos reikšmė mažiausia ir žymimas X_L

Toliau bandom keisti parametrus: gama didinti iki 3, tai reiškia, jog simpleksas bus labiau pratęsiamas.



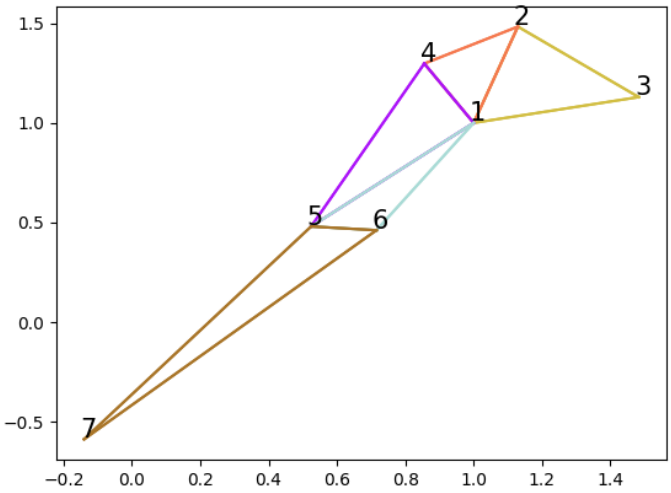
15 pav. Deformuojamo simplekso metodo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, gamma = 3, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

Gavome, kad minimumui gauti mūsų funkcijai prireikė 21 iteracijų ir 46 funkcijų skaičiavimų. Grafike pavaizduoti simpleksai ir jų virūnės po pirmų 5 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo. Grafike pavaizduotų simpleksų viršūnės: (1,2,3), (1,2,4), (1,4,5), (1,5,6), (5,6,7).

18 lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_1(1,1)$, gamma = 3, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(1, 1)	0.125
2	(1, 1)	0.125
3	(0.6253553228495821, 0.6474524097616616)	0.01380706570111063
19	(0.33367648450520926, 0.33294181587409877)	-0.004629623933443517
20	(0.33367648450520926, 0.33294181587409877)	-0.004629623933443517
21	(0.33367648450520926, 0.33294181587409877)	-0.004629623933443517

Toliau bandom didinti gamma į 4.



16 pav. Deformuojamo simplekso metodo grafiko ir taškų vaizdavimas, pradinis taškas $X_1(1,1)$, gamma = 4, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

Gavome, jog metodui nepavyko rasti minimumo ir po 100 iteracijų bei 153 funkcijų skaičiavimų programai teko sustoti. Grafike pavaizduotų simpleksai ir jų viršūnės po pirmų 5 iteracijų: (1,2,3), (1,2,4), (1,4,5), (1,5,6), (5,6,7). Grafike matom, jog metodui pasirinkus tašką 7, peršoko duobę, kurios ieškojom ir toliau artėjo link $(-\infty, -\infty)$.

19 lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų devynių iteracijų, kai pradinis taškas $X_1(1,1)$, gamma = 4, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

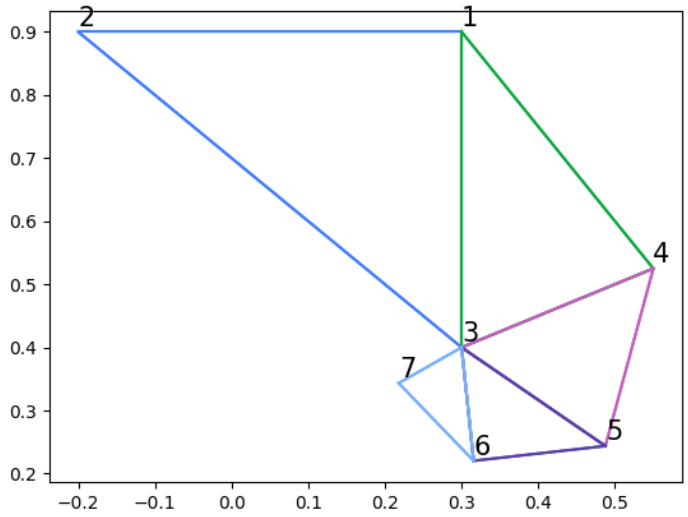
Iteracija	X_L	$F(X_L)$
1	(1, 1)	0.125
2	(1, 1)	0.125
3	(0.5245444829092464, 0.48035030908508713)	0.00015416459704128047
4	(0.5245444829092464, 0.48035030908508713)	0.00015416459704128047
5	(-0.1397524963691118, -0.5872185063387239)	-0.01771553885451159
6	(-0.1397524963691118, -0.5872185063387239)	-0.01771553885451159
7	(-1.3625421151480184, -2.3078831146054237)	-1.8358203568047045
8	(-1.3625421151480184, -2.3078831146054237)	-1.8358203568047045
9	(-3.4024774155526467, -5.311199614329273)	-21.94226964951965
...		

20 lentelė. Deformuojamo simplekso metodo rezultatų palyginimas, kai pradinis taškas $X_1(1,1)$, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

	gamma = 2	gamma = 3	gamma = 4
Iteracijų skaičius	28	21	100
Skaičiuotų funkcijų kiekis	53	46	153
Minimumo X reikšmė	(0.33329853059088677, 0.334389708033079)	(0.33367648450520926, 0.33294181587409877)	Nepasiekta
Funkcijos minimumo reikšmė	-0.004629584618741002	-0.004629623933443517	Nepasiekta
Apytikslis atstumas iki realaus Xmin	(0.000034, 0.000391)	(0.000035, 0.001056)	-

Deformuojamo simplekso metodui skaičiuojant tikslo funckijos $f(X) = -0.125 * x_1 * x_2 * (1 - x_1 - x_2)$ minimumo tašką iš pradinio taško $X_1(1,1)$, bandymais atrasta, jog renkantis per maža gamma prireiks daugiau iteracijų ir funkcijų skaičiavimų minimumui pasiekti, o pasirinkus gamma per didelį, funkcija gali peršokti mūsų minimumo tašką ir nukonverguoti į begalybę. Geriausias bandymas pavyko su gamma = 3, kur prireikė 21 iteracijos ir 46 funkcijų skaičiavimų, tačiau Xmin reikšmė artimesnė realiai gavos su gamm = 2.

Toliau keičiame pradinį tašką į $X_m(0.3,0.9)$, o gamma pasirenkam 3. Dėja funkcija nukonverguoja į (-11,12), bėda yra su pradinių taškų pasirinkimu, nes jie yra arti kitos funkcijos duobės. Dėl to keičiame pradinių taškų pasirinkimo formulę į: $X_1 = (x_0 - \text{alpha}, x_1)$, $X_2 = (x_0, x_1 - \text{alpha})$, kai $X_{pradinis} = (x_0, x_1)$.



17 pav. Deformuojamo simplekso metodo grafiko ir taškų vaizdavimas, pradinis taškas X_m(0.3,0.9), gamma = 3, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.00

Gavome, kad minimumui gauti mūsų funkcijai prireikė 23 iteracijų ir 42 funkcijų skaičiavimų. Grafike pavaizduoti simpleksai ir jų virūnės po pirmų 5 iteracijų, jog grafike aiškiai matytųsi artėjimas link minimumo. Grafike pavaizduotų simpleksų viršūnės: (1,2,3), (1,3,4), (3,4,5), (3,5,6), (3,6,7).

21lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas X_m(0.3,0.9), gamma = 3, beta = 0.5, niu = −0.5, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(0.3, 0.4)	-0.0045000000000000005
2	(0.3, 0.4)	-0.0045000000000000005
3	(0.3, 0.4)	-0.0045000000000000005
21	(0.3333214854472317, 0.33432802104507575)	-0.004629588891116237
22	(0.33414627663732976, 0.3330000526897493)	-0.004629608770268861
23	(0.33373388104228074, 0.33366403686741253)	-0.004629612856445141

Toliau bandom keisti suspaudimo parametrus, pirmą keičiame niu į -0.7, tai reiškia simpleksas bus mažiau sutraukiamas. Brėžinys pirmų 5 iteracijų atitinka 17 pav., nes pirmose 5 iteracijose nebuvo sutraukimo operacijų. Gavome, kad minimumui gauti mūsų funkcijai prireikė 19 iteracijų ir 42 funkcijų skaičiavimų.

22 lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas X_m(0.3,0.9), gamma = 3, beta = 0.5, niu = −0.7, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(0.3, 0.4)	-0.0045000000000000005
2	(0.3, 0.4)	-0.0045000000000000005
3	(0.3, 0.4)	-0.0045000000000000005
17	(0.32802892774343495, 0.3379651062190533)	-0.004628585012376295
18	(0.3280577335506678, 0.33743404466658833)	-0.004628667532701236
19	(0.3280577335506678, 0.33743404466658833)	-0.004628667532701236

Toliau bandom keisti sutraukimo parametrus, keičiame niu į -0.3. Brėžinys pirmų 5 iteracijų atitinka 17 pav., nes pirmose 5 iteracijose nebuvo sutraukimo operacijų. Gavome, kad minimumui gauti mūsų funkcijai prireikė 21 iteracijų ir 34 funkcijų skaičiavimų. Sumažėjo funkcijų skaičiavimų skaičius, nors iteracijų skaičius padidėjo, vadinasi padaugėjo reflektavimo operacijų, nes ji atliekama pirma ir nereikia skaičiuoti papildomų funkcijų atliekant kitas operacijas.

23 lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_m(0.3,0.9)$, gamma = 3, beta = 0.5, niu = −0.3, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(0.3, 0.4)	-0.0045000000000000005
2	(0.3, 0.4)	-0.0045000000000000005
3	(0.3, 0.4)	-0.0045000000000000005
19	(0.3330976520646364, 0.3337094888398422)	-0.004629625115110538
20	(0.3330976520646364, 0.3337094888398422)	-0.004629625115110538
21	(0.3333063512223017, 0.3330390670983905)	-0.004629625660757817

Toliau bandom keisti beta į 0.7 (niu grąžiname į -0.5). Brėžinys pirmų 5 iteracijų atitinka 17 pav., nes pirmose 5 iteracijose nebuvo sutraukimo operacijų. Gavome, kad minimumui gauti mūsų funkcijai prireikė 28 iteracijų ir 61 funkcijų skaičiavimų.

lentelė 24. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_m(0.3,0.9)$, gamma = 3, beta = 0.7, niu = −0.5, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(0.3, 0.4)	-0.0045000000000000005
2	(0.3, 0.4)	-0.0045000000000000005
3	(0.3, 0.4)	-0.0045000000000000005
26	(0.33306193747423773, 0.3345886043292392)	-0.004629575142959248
27	(0.33325736001005657, 0.3343911103343409)	-0.00462958612693821
28	(0.33325736001005657, 0.3343911103343409)	-0.00462958612693821

Toliau bandom keiti kitą sutraukimo parametrą beta į 0.2. Brėžinys pirmų 5 iteracijų atitinka 17 pav., nes pirmose 5 iteracijose nebuvo sutraukimo operacijų. Gavome, kad minimumui gauti mūsų funkcijai prireikė 19 iteracijų ir 33 funkcijų skaičiavimų.

25 lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_m(0.3,0.9)$, gamma = 3, beta = 0.2, niu = −0.5, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(0.3, 0.4)	-0.0045000000000000005
2	(0.3, 0.4)	-0.0045000000000000005
3	(0.3, 0.4)	-0.0045000000000000005
17	(0.33781594187021263, 0.3281282193958758)	-0.004628633584805819
18	(0.33781594187021263, 0.3281282193958758)	-0.004628633584805819
19	(0.33681133389472967, 0.32807743608951556)	-0.004628732195041946

Toliau bandom keisti abu sutraukimo parametrus: beta į 0.2, o niu į 0.7. Brėžinys pirmų 5 iteracijų atitinka 17 pav., nes pirmose 5 iteracijose nebuvo sutraukimo operacijų. Gavome, kad minimumui gauti mūsų funkcijai prireikė 20 iteracijų ir 33 funkcijų skaičiavimų.

26 lentelė. Deformuojamo simplekso metodo geriausiojo taško rezultatai po pirmų ir paskutinių trijų iteracijų, kai pradinis taškas $X_m(0.3,0.9)$, gamma = 3, beta = 0.2, niu = −0.7, alpha = 0.5, epsilon 0.001

Iteracija	X_L	$F(X_L)$
1	(0.3, 0.4)	-0.0045000000000000005
2	(0.3, 0.4)	-0.0045000000000000005
3	(0.3, 0.4)	-0.0045000000000000005
18	(0.3332527399063107, 0.33354197740554814)	-0.004629628246052592
19	(0.3332527399063107, 0.33354197740554814)	-0.004629628246052592
20	(0.3332527399063107, 0.33354197740554814)	-0.004629628246052592

27 lentelė. Deformuojamo simplekso metodo rezultatų palyginimas, kai pradinis taškas $X_m(0.3,0.9)$, gamma = 3, alpha = 0.5, epsilon 0.001

	beta = 0.5, niu = -0.5	beta = 0.5, niu = -0.7	beta = 0.5, niu = -0.3
Iteracijų skaičius	23	19	21
Skaičiuotų funkcijų kiekis	42	42	34
Minimumo X reikšmė	(0.33373388104228074, 0.33366403686741253)	(0.3280577335506678, 0.33743404466658833)	(0.3333063512223017, 0.3330390670983905)
Funkcijos minimumo reikšmė	-0.004629612856445141	-0.004628667532701236	-0.004629625660757817
Apytikslis atstumas iki realaus Xmin	(0.0004, 0.00033)	(0.00528, 0.0041)	(0.00003, 0.0003)
	beta = 0.7, niu = -0.5	beta = 0.2, niu = -0.5	beta = 0.2, niu = -0.7
Iteracijų skaičius	28	19	20
Skaičiuotų funkcijų kiekis	61	33	33
Minimumo X reikšmė	(0.33325736001005657, 0.3343911103343409)	(0.33681133389472967, 0.32807743608951556)	(0.3332527399063107, 0.33354197740554814)
Funkcijos minimumo reikšmė	-0.00462958612693821	-0.004628732195041946	-0.004629628246052592
Apytikslis atstumas iki realaus Xmin	(0.00008, 0.00106)	(0.00348, 0.00526)	(0.00008, 0.00021)

Deformuojamo simplekso metodui skaičiuojant tikslo funckijos $f(X) = -0.125 * x_1 * x_2 * (1 - x_1 - x_2)$ minimumo tašką iš pradinio taško $X_m(0.3,0.9)$, bandymais atrasta, jog mažinant niu sumažėjo iteracijų skaičius, o skaičiuotų funkcijų kiekis nepakito. Niu didinant sumažėjo iteracijų skaičius ir skaičiuotų funkcijų kiekis. Didinant beta itreacijų skaičius bei skaičiuotų funkcijų kiekis

padidėjo, o beta mažinant – sumažėjo. Galima būtų teigti, jog geriausias rezultatas pasiektas, kai beta = 0.2, niu = -0.5, nes pasiektas mažiausias iteracijų skaičius bei skaičiuotų funkcijų kiekis, tačiau Xmin reikšmė ganėtinai toli realaus Xmin, dėl to geriausiu rezultatu reikėtų laikyti kai beta = 0.2, niu = -0.7, nes padidėjo tik iteracijų kiekis per 1, tačiau Xmin daug artimesnis realiam Xmin.

Rezultatų palyginimai

28 lentelė. Tikslo funkcijos geriausių rezultatų palyginimai, kai pradinis taškas $X_0(0,0)$, epsilon = 0.001

	Gradientinio nusileidimo metodas	Greičiausiojo nusileidimo metodas	Deformuojamo simplekso metodas (gamma = 2, beta = 0.5, niu = −0.5, alpha = 0.5)
Iteracijų skaičius	1	1	24
Skaičiuotų funkcijų kiekis	2	2	44
Minimumo X reikšmė	(0,0)	(0,0)	(0.33355004943061595, 0.333131889473115)
Funkcijos minimumo reikšmė	0	0	-0.004629627800989286
Apytikslis atstumas iki realaus Xmin	(0.33333,0.33333)	(0.33333,0.33333)	(0.00022, 0.000201)

Lyginat rezultatus galima pamatyti, jog gradientinio nusileidimo bei greičiausiojo nusileidimo metodai sustoje taške (0,0), nes šie metodai skaičivimuose remiasi gradeintu, o taške (0,0) gradientas lygus 0. Tačiau deformuojamo simplekso metodas remiasi tikslo funkcijos reikšmėm, dėl to jam pavyko per 24 iteracijas ir 44 funkcijų skaičiavimus pasiekti minimumo taška, kuris nedaug nutolęs nuo realaus minimumo, tik per (0.00022, 0.000201).

29 lentelė. Tikslo funkcijos geriausių rezultatų palyginimai, kai pradinis taškas $X_1(1,1)$, epsilon = 0.001

	Gradientinio nusileidimo metodas (gamma = 3)	Greičiausiojo nusileidimo metodas (L = [0,5])	Deformuojamo simplekso metodas (gamma = 3, beta = 0.5, niu = −0.5, alpha = 0.5)
Iteracijų skaičius	8	1	21
Skaičiuotų funkcijų kiekis	16	24	46
Minimumo X reikšmė	(0.328561394562967, 0.328561394562967)	(0.3333223155064414, 0.3333223155064414)	(0.33367648450520926, 0.33294181587409877)
Funkcijos minimumo reikšmė	-0.004626810370607299	-0.0046296296144559	-0.004629623933443517
Apytikslis atstumas iki realaus Xmin	(0.00477, 0.00477)	(0.00001, 0.00001)	(0.000035, 0.001056)

Lyginat rezultatus galima pamatyti, jog gradientinio nusileidimo metodui prireikė mažiausiai funkcijų skaičiavimų, o greičiausiojo nusileidimo metodui prireikė tik 1 iteracijos, tačiau skaičiuotų funkcijų kiekis gerokai didesnis nei gradientinio nusileidimo metodo, taip yra dėl to, jog kiekvienos iteracijos metu, greičiausiojo nusileidimo metodas turi skaičiuoti gamma pasitelkiant auksinio pjūvio metodu. O šiam metodui prireikė tik vienos iteracijos, dėl to, jog antigradiento kryptis taške (1,1) eina tiesiai per tikslo funkcijos minimumo tašką. Blogiausiai pasirodė simplekso metodas, kuriam prireikė net 21 iteracijos ir 46 funkcijų skaičiavimų, tačiau pamodifikuojant metodo parametrus būtų galima potencialiai pasiekti ir geresnių rezultatų.

30 lentelė. Tikslo funkcijos geriausių rezultatų palyginimai, kai pradinis taškas $X_m(0.3,0.9)$, epsilon = 0.001

	Gradientinio nusileidimo metodas (gamma = 3)	Greičiausiojo nusileidimo metodas (L = [0,15])	Deformuojamo simplekso metodas (gamma = 3, beta = 0.2, niu = -0.7, alpha = 0.5)
Iteracijų skaičius	16	7	20
Skaičiuotų funkcijų kiekis	32	170	33
Minimumo X reikšmė	(0.26011931709946806, 0.4337899785817296)	(0.3214387194428629, 0.3408058272580708)	(0.3332527399063107, 0.33354197740554814)
Funkcijos minimumo reikšmė	-0.004317300454129901	-0.004625062266240865	-0.004629628246052592
Apytikslis atstumas iki realaus Xmin	(0.07321, 0.10046)	(0.01190, 0.00747)	(0.00008, 0.00021)

Lyginat rezultatus galima pamatyti, jog greičiausiojo nusileidimo metodui prireikė tik 7 iteracijų, tačiau 170 funkcijų skaičiavimu, kai tuo tarpu gradientiniam nusileidimo ir deformuojamo simplekso metodam prireikė atitinkamai tik 32 ir 33 funkcijų skaičiavimų, tačiau gradientiniam nusileidimo metodui prireikė mažiau iteracijų (16) , nei deformuojamo simplekso (20). Bet deformuojamo simplekso metodo rastas Xmin yra gerokai arčiau realaus minimumo taško nei kitų metodų. Būtų galima teigti, jog geriausiai pasirodė simplekso metodas skaičiuojant šį tašką, nes jam prireikė palyginus nedaug iteracijų ir funkcijų skaičiavimų, bei Xmin yra arčiausiai realaus Xmin.

Išvados

Apibendrinant galima būtų teigti, jog visi metodai ieškant nuo X_1 ir X_m tinkamai atliko savo darbą – surado funkcijos minimumą. Tačiau ieškant nuo X_0 tik simplekso metodui pavyko rasti minimo tašką. Dėl to galima būtų teigti, jog geriausias metodas minimumui atrasti yra simplekso metodas – iš visų bandymo tašku, tinkamai pakoregavus parametrus, galima pasiekti minimumo tašką bei rastas minimumas yra artimas realiam Xmin. Tačiau gradientinio nusileidimo metodui daugeliu atveju pavyksta rasti minimumą prireikus mažiausiai funkcijų skaičiavimų, o greičiausiajam nusileidimui – iteracijų.

Priedas

```
import itertools
import math
from math import sin
from operator import itemgetter
import random

import matplotlib.pyplot as plt
import numpy as np
from sympy import Symbol

fig, ax = plt.subplots()


def f(x1, x2):
    return gradientFunction(str(F), x1, x2)

def plot2d(points, method):
    plot2d(points, [], method)

def plot2d(points, pointsForNumbers, method):
    delta = 0.001
    if method == 'gd' or method == 'sd':
        x = np.arange(0.15, 0.35, delta)
        y = np.arange(0.30, 0.5, delta)
        X, Y = np.meshgrid(x, y)
        Z = -0.125 * X * Y * (1 - X - Y)
        CS = ax.contour(X, Y, Z, 15, linewidths=0.3)
        ax.clabel(CS, inline=True, fontsize=9)
        iterations = [2, 3, 4, 5, 6]
        for i in range(0, len(points), 2):
            if int(i / 2) in iterations:
                if i == (len(points) - 2):
                    ax.plot(points[i], points[i + 1], marker='x')
                else:
                    ax.plot(points[i], points[i + 1], marker='.')
                plt.annotate(int(i / 2) + 1, (points[i], points[i + 1] + 0.003))
    elif method == 'simplex' and len(points) % 3 == 0:
        for i in range(0, min(5*3, len(points)), 3):
            simp = [points[i], points[i + 1], points[i + 2]]
            r = random.random()
            b = random.random()
            g = random.random()
            a = 1
            color = (r, g, b, a)
            for a, b in itertools.product(simp, simp):
                x = np.linspace(a[0], b[0], 100)
                y = np.linspace(a[1], b[1], 100)

                ax.plot(x, y, color=color)
        for i in range(0, min(7, len(pointsForNumbers))):
            plt.annotate(i + 1, (pointsForNumbers[i][0], pointsForNumbers[i][1]
+ 0.009), fontsize=15)
    plt.draw()
    plt.show()


def getGrad(func, args):
    grad = []
    for x in args:
```

```
        grad.append(func.diff(x))
    return grad


def gradientFunction(func, x1, x2):
    return eval(func)


def gradient_descent(func, args, X0, gama, epsilon):
    i = 1
    Xi = X0

    grad = getGrad(func, args)
    points = []
    counter = 0

    max_iterations = 100
    while i < max_iterations:
        gradMod = 0
        Xtemp = list(Xi)
        for j in range(0, len(Xi)):
            gradFunc = gradientFunction(str(grad[j]), Xtemp[0], Xtemp[1])
            Xi[j] = Xi[j] - gama * gradFunc
            counter += 1
            gradMod += gradFunc
        gradMod = abs(gradMod) / len(Xi)
        points.append(Xi[0])
        points.append(Xi[1])
        print("i: ", i, "Xi[0]:", Xi[0], ". Xi[1]:", Xi[1], ". f(Xi) =",
f(Xi[0], Xi[1]))
        if gradMod < epsilon:
            print("i: ", i, "[", Xi[0], ",", Xi[1], "]")
            print("Counter: ", counter)
            print("f(X) = ", f(Xi[0], Xi[1]))
            break
        i += 1
    plot2d(points, 'gd')

def golden_section_method(left, right, xi, grad, func, epsilon):
    f = lambda alpha: fPoint(xi - alpha * grad)

    def fPoint(point):
        return gradientFunction(str(func), point[0], point[1])

    tau = (-1 + math.sqrt(5)) / 2
    length = right - left
    x1 = right - tau * length
    x2 = left + tau * length
    steps = 1

    fx1 = f(x1)
    fx2 = f(x2)
    counter = 2
    while length >= epsilon:
        steps = steps + 1
        if fx2 < fx1:
            left = x1
            length = right - left
            x1 = x2
            fx1 = fx2
            x2 = left + tau * length
```

```
            fx2 = f(x2)
            counter += 1
        else:
            right = x2
            length = right - left
            x2 = x1
            fx2 = fx1
            x1 = right - tau * length
            fx1 = f(x1)
            counter += 1

        min_reiksme = min([fx1, fx2])
        sprendinys = x1
        if min_reiksme == fx2:
            sprendinys = x2
    return sprendinys, counter

def steepest_descent(func, args, Xi, epsilon):
    Xi = np.array(Xi)

    grad = getGrad(func, args)
    points = []
    # points = [Xi[0], Xi[1]]

    counter = 0
    i = 1
    max_iterations = 500
    while i < max_iterations:
        gradValue = np.array(gradientFunction(str(grad), Xi[0], Xi[1]))
        counter += 2
        gradMod = math.sqrt(gradValue[0] * gradValue[0] + gradValue[1] *
gradValue[1])
        if gradMod < epsilon:
            print("i: ", i - 1, "[", Xi[0], ",", Xi[1], "]")
            print("Counter: ", counter)
            print("f(X) = ", f(Xi[0], Xi[1]))
            break

        gama_min, n_count = golden_section_method(0, 17, Xi, gradValue, func,
epsilon)
        counter += n_count

        Xi = Xi - gama_min * gradValue
        points.append(Xi[0])
        points.append(Xi[1])
        print("i: ", i, "gamma", gama_min, "Xi[0]:", Xi[0], ". Xi[1]:", Xi[1],
". f(Xi) =", f(Xi[0], Xi[1]))
        i += 1
    plot2d(points, 'sd')

def getModVector(x):
    return math.sqrt(x[0] * x[0] + x[1] * x[1])
def getPoint(arg, value):
    return {"arg": arg, "value": value}

def generate_simplex_method_points(x0, alpha):
    n = 2
    delta1 = (math.sqrt(n + 1) + n - 1) / (n * math.sqrt(2)) * alpha
    delta2 = (math.sqrt(n + 1) - 1) / (n * math.sqrt(2)) * alpha

    x1 = [x0[0] + delta2, x0[0] + delta1]
```

```
    x2 = [x0[0] + delta1, x0[0] + delta2]
    return x1, x2

def simplex_method(args, X0, epsilon=0.001, alpha=0.5, gama=3, beta=0.2, niu=-
0.7):
    simplex = [getPoint(X0, f(X0[0], X0[1]))]
    max_iterations = 100
    counter = 1
    points = []
    pointsForNumbers = [X0]

    for i in range(0, len(args)):
        argList = list(X0)
        argList[i] -= alpha
        simplex.append(getPoint(argList, f(argList[0], argList[1])))
        counter += 1

    # Geresnis:
    # X1, X2 = generate_simplex_method_points(X0, alpha)
    # simplex.append(getPoint(X1, f(X1[0], X1[1])))
    # counter += 1
    # simplex.append(getPoint(X2, f(X2[0], X2[1])))
    # counter += 1

    pointsForNumbers.append(simplex[-2]['arg'])

    for i in range(0, max_iterations):
        pointsForNumbers.append(simplex[-1]['arg'])
        # 1. Sort
        simplex.sort(key=itemgetter('value'))

        print("i: ", i + 1)
        print("     [0]: ("+str(simplex[0]['arg'][0])+",
"+str(simplex[0]['arg'][1])+")", ". f:", simplex[0]['value'])
        # print("     [1]:", simplex[1]['arg'], ". f:", simplex[1]['value'])
        # print("     [2]:", simplex[2]['arg'], ". f:", simplex[2]['value'])

        # if i < 6:
        points.extend([tuple(sim['arg'] + [sim['value']]) for sim in simplex])

        # 6. Check convergence
        if getModVector(np.array((simplex[0]['arg']) - np.array(simplex[-
1]['arg']))) < epsilon:
            break

        centroid = [0] * len(args)
        for j in range(0, len(args)):
            for k in range(0, len(simplex) - 1):
                centroid[j] += simplex[k]['arg'][j]
            centroid[j] /= (len(simplex) - 1)

        # 2. Reflect
        reflection = [0] * len(args)
        for j in range(0, len(args)):
            reflection[j] = centroid[j] + alpha * (centroid[j] - simplex[-
1]['arg'][j])
        reflection_value = f(reflection[0], reflection[1])
        counter += 1

        # 3. Evaluate or Extend
        if simplex[0]['value'] <= reflection_value < simplex[-2]['value']:
```

```
            simplex[-1] = getPoint(reflection, reflection_value)
            continue
        elif reflection_value < simplex[0]['value']:
            extend = [0] * len(args)
            for j in range(0, len(args)):
                extend[j] = centroid[j] + gama * (reflection[j] - centroid[j])
            extended_value = f(extend[0], extend[1])
            counter += 1
            if extended_value < simplex[0]['value']:
                simplex[-1] = getPoint(extend, extended_value)
            else:
                simplex[-1] = getPoint(reflection, reflection_value)
            continue

        # 4. Contract
        contraction = [0] * len(args)
        for j in range(0, len(args)):
            contraction[j] = centroid[j] + niu * (simplex[-1]['arg'][j] -
centroid[j])
        contraction_value = f(contraction[0], contraction[1])
        counter += 1
        if contraction_value < simplex[-1]['value']:
            simplex[-1] = getPoint(contraction, contraction_value)
            continue

        # 5. Reduce
        for j in range(1, len(simplex)):
            reduce = [0] * len(args)
            for k in range(0, len(args)):
                reduce[k] = simplex[0]['arg'][k] + beta * (simplex[j]['arg'][k]
- simplex[0]['arg'][k])
            reduce_value = f(reduce[0], reduce[1])
            counter += 1
            simplex[j] = getPoint(reduce, reduce_value)
        pointsForNumbers.append(simplex[-2]['arg'])

    plot2d(points, pointsForNumbers, 'simplex')
    print("i: ", i + 1, simplex[0]['arg'])
    print("f(X) = ", f(simplex[0]['arg'][0], simplex[0]['arg'][1]))
    print("Counter: ", counter)


x1 = Symbol('x1')
x2 = Symbol('x2')
F = -0.125 * x1 * x2 * (1 - x1 - x2)
def main():

    # plot2d([], 'a')
    #
    # grad = getGrad(F, [x1, x2])
    # print("Tikslo ir gradiento funkciju reiksmes (0, 0)", f(0, 0),
gradientFunction(str(grad), 0, 0))
    # print("Tikslo ir gradiento funkciju reiksmes (1, 1)", f(1, 1),
gradientFunction(str(grad), 1, 1))
    # print("Tikslo ir gradiento funkciju reiksmes (0.3, 0.9)", f(0.3, 0.9),
gradientFunction(str(grad), 0.3, 0.9))

    # gradient_descent(F, [x1, x2], [0, 0], 0.1, 0.001)
    # gradient_descent(F, [x1, x2], [1, 1], 3.6, 0.001)
    # gradient_descent(F, [x1, x2], [0.3, 0.9], 3.4, 0.0001)
```

```
    # steepest_descent(F, [x1, x2], [0, 0], 0.001)
    # steepest_descent(F, [x1, x2], [1, 1], 0.001)
    # steepest_descent(F, [x1, x2], [0.3, 0.9], 0.001)

    # simplex_method([x1, x2], [0, 0])
    # simplex_method([x1, x2], [1, 1])
    simplex_method([x1, x2], [0.3, 0.9])


if __name__ == "__main__":
    main()
```