VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Optimizavimo metodai

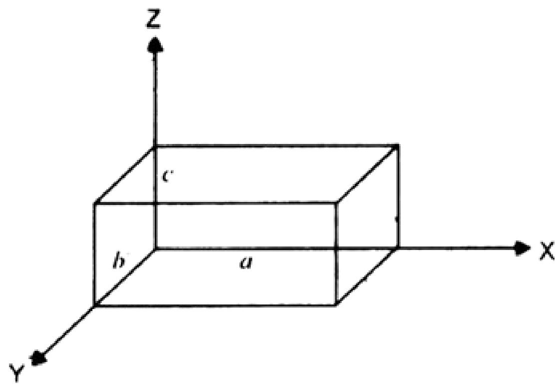
Laboratorinis darbas nr. 2

Optimizavimas be apribojimų

Tomas Giedraitis
VU MIF Informatika
3 kursas 3 grupė

Vilnius
2019

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



Pažymėkime kraštinių ilgius *a*, *b*, *c*, $a>0, b>0, c>0$

Reikalavimas vienetinio dėžės paviršiaus plotui:

$$S_{pav}=2ab+2ab+2bc=1 \quad ,$$

kur $ab>0, ac>0, bc>0$

Dėžės priekinės ir galinės sienų plotų sumą, šoninių sienų plotų sumą, viršutinės ir apatinės sienų plotų sumą pakeiskime kintamaisiais:

$$x_1=2ab; x_2=2ac; x_3=2bc$$

Apribojimai:

$$x_1>0, x_2>0, x_3>0$$

$$x_1+x_2+x_3=1$$

Išsireiškiame apribojimą kintamajam x_3 per kintamuosius x_1, x_2 ir vienetinį paviršiaus plotą:

$$x_3=1-x_1+x_2$$

Tai padės suvesti uždavinį į optimizavimą be apribojimų.

Dėžės tūris, ir tūris, pakeltas kvadratu:

$$V=abc$$
$$V^2=a^2b^2c^2$$

Turime, kad:

$$x_1x_2x_3=2ab\,2ac\,2bc=8a^2b^2c^2=8V^2$$

Taigi dėžės tūrio, pakelto kvadratu, funkcija pagal kintamuosius x_1, x_2, x_3 :

$f(X)=\frac{1}{8}x_1 x_2 x_3$, kur X yra kintamųjų vektorius, $X=[x_1, x_2, x_3]$

Sumažiname kintamųjų skaičių, pasinaudodami apribojimų x_3 :

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

Ieškosime dežės parametrų esant maksimaliam tūriui: max(V), arba min(-V). Vietoje V įsistatydami V^2 , kadangi tai nekeis argumentų reikšmių. Daugiklis $\frac{1}{8}$ taip pat nekeis argumentų reikšmių, todėl galime jį ignoruoti. Gauname tikslo funkciją, kurią minimizuosime, sprendžiant optimizavimo uždavinį be apribojimų:

$$f(X)=-x_1 x_2+x_1^2 x_2+x_1 x_2^2$$

Tikslo funkcijos gradiento funkcijos reikšmė šiuo atveju bus vektorius, sudarytas iš tikslo funkcijos dalinių išvestinių:

$$\nabla f(X)=[\frac{\partial f(X)}{\partial x_2}, \frac{\partial f(X)}{\partial x_2}]=[-x_2+2x_1 x_2+x_2^2, -x_1+x_1^2+2x_1 x_2]$$

Šiuos duomenis (tikslo funkciją, jos gradiento funkciją) naudosime spręsdami optimizavimo uždavinį naudojant suprogramuotus optimizavimo algoritmus:

GD – Gradientinis nusileidimas (Gradient Descent)
SD – Greičiausias nusileidimas (Steepest Descent)
SIM – Simpleksas (deformuojamas)

Ir pradedant iš taškų:

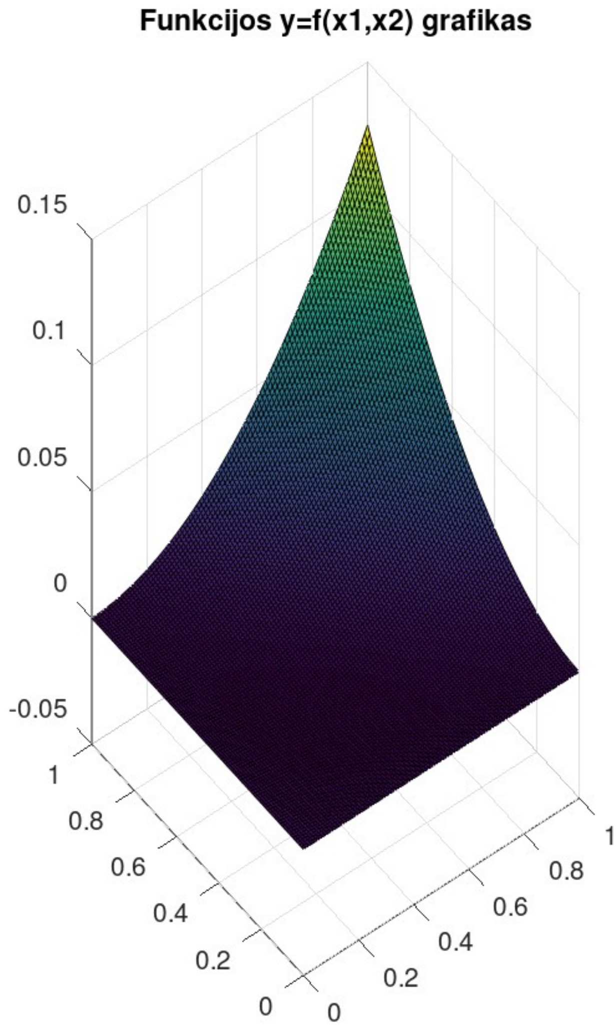
$X_0=[0,0]$,

$X_1=[1,1]$,

$X_m=[0.4, 0.7]$.

Daugiklio $\frac{1}{8}$ neignoruosime, programuojant algoritmus, tačiau didelio skirtumo tai nedarys, keisis tik tikslo funkcijos minimumo reikšmė.

Funkciją $f(X)$ vietomis žymėsime $f(x_1, x_2)$.



GD algoritmas

[1] Pradinis artinys: $X_0=[0,0]$

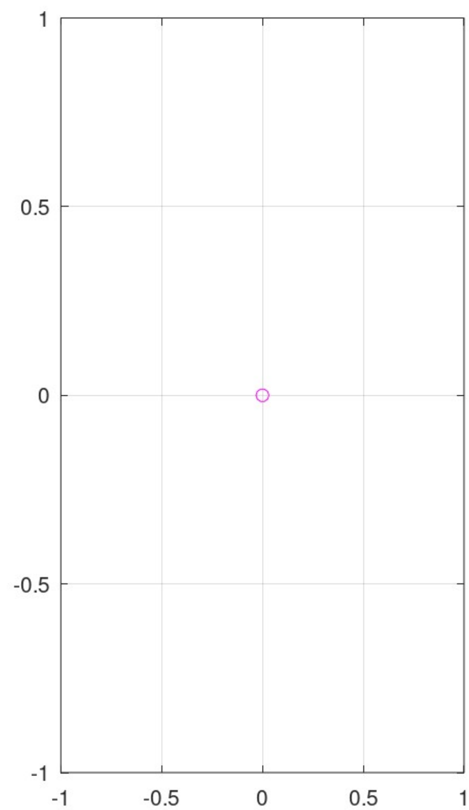
Bet kokį *gamma* pasirinkus intervale $[0;2]$, gradiento funkcijos reikšmė vis tiek visada būna $\nabla f(X)=[0,0]$, ir iš pradinio artinio nepajudama. Taigi visais *gamma* atvejais gaunamas toks rezultatas (lentelėje ir programos išvestyje):

Algoritmas	Parametras *gamma*	*k* (iteracijų kiekis)	Funkcijos iškvietimai	Funkcijos iškvietimų kiekis	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
GD	(bet koks nuo 0 iki 2)	1	*k*	1	0.000000	0.000000	0.000000

Programos išvestis (iteracijų rezultatai):

x1	x2	f(x1,x2)	k	(funkc. kviet. sk)
0.000000	0.000000	0.000000	1	1

Bandymo grafikas (artiniai):



[2] Pradinis artinys: $X_1 = [1, 1]$

Buvo keičiamas parametras *gamma* siekiant efektyvesnio uždavinio sprendimo (mažesnio iteracijų skaičiaus). Geriausias rezultatas buvo pasiektas su *gamma* = 0.33, iteracijų skaičius = 14. *Gamma* reikšmės buvo parenkamos iš šios reikšmių aibės: {0.10, 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00}. Tuomet gaunamas intervalas iš dviejų geriausių rezultatų, ir jame ieškoma naujos *gamma* reikšmės, su kuria iteracijų skaičius būtų dar mažesnis. Šiuo atveju intervalas buvo nuo 0.25 iki 0.50, bet, kadangi iš pradinio artinio nebuvo pajudama jau esant *gamma* = 0.33 ir daugiau, buvo ieškoma gamma reikšmių tarp 0.25 ir 0.33, ir gauta geriausia gamma reikšmė *gamma* = 0.33.

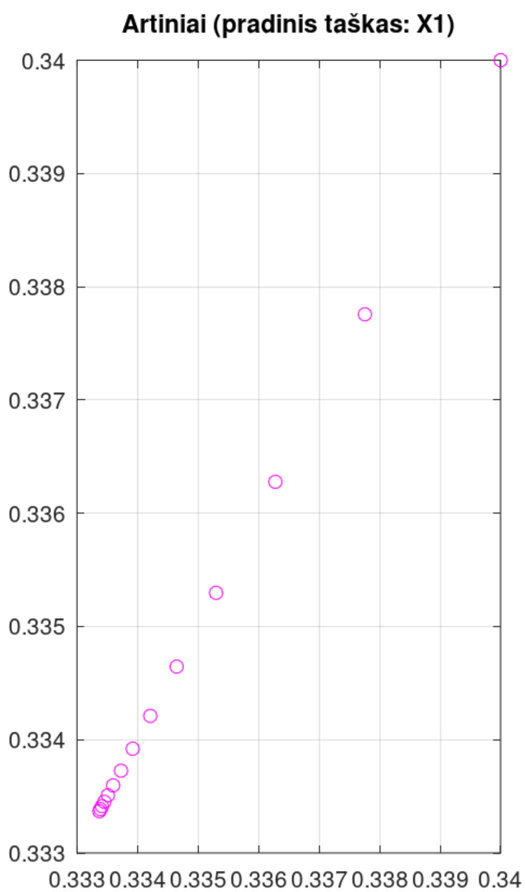
Algoritmas	Parametras *gamma*	*k* (iteracijų kiekis)	Funkcijos iškvietimai	Funkcijos iškvietimų kiekis	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
GD	0.10	77	*k*	77	-0.004630	0.333369	0.333369
	0.25	28		28			
	0.30	21		21			
	0.33	14		14			
	0.50	1		1	0.000000	0.000000	0.000000
	0.75	*inf*		*inf*	*nepasiekta*	*nepasiekta*	*Nepasiekta*
	1.00						
	1.25						
	1.50						
	1.75						
	2.00						

Programos išvestis (iteracijų rezultatai):

Šiuo atveju gauti rezultatai su *gamma* = 0.33

x1	x2	f(x1,x2)	k (funkc. kviet. sk)	
0.340000	0.340000	-0.004624	1	1
0.337756	0.337756	-0.004627	2	2
0.336277	0.336277	-0.004629	3	3
0.335297	0.335297	-0.004629	4	4
0.334645	0.334645	-0.004629	5	5
0.334211	0.334211	-0.004630	6	6
0.333920	0.333920	-0.004630	7	7
0.333726	0.333726	-0.004630	8	8
0.333596	0.333596	-0.004630	9	9
0.333510	0.333510	-0.004630	10	10
0.333451	0.333451	-0.004630	11	11
0.333412	0.333412	-0.004630	12	12
0.333386	0.333386	-0.004630	13	13
0.333369	0.333369	-0.004630	14	14

Bandymo grafikas (artiniai):
gamma = 0.33



[3] Pradinis artinys: $X_m = [0.4\,, 0.7]$

Tinkamiausio parametro *gamma buvo* ieškoma naudojant tokį patį metodą, kaip ir pradinio artinio X_1 atveju.

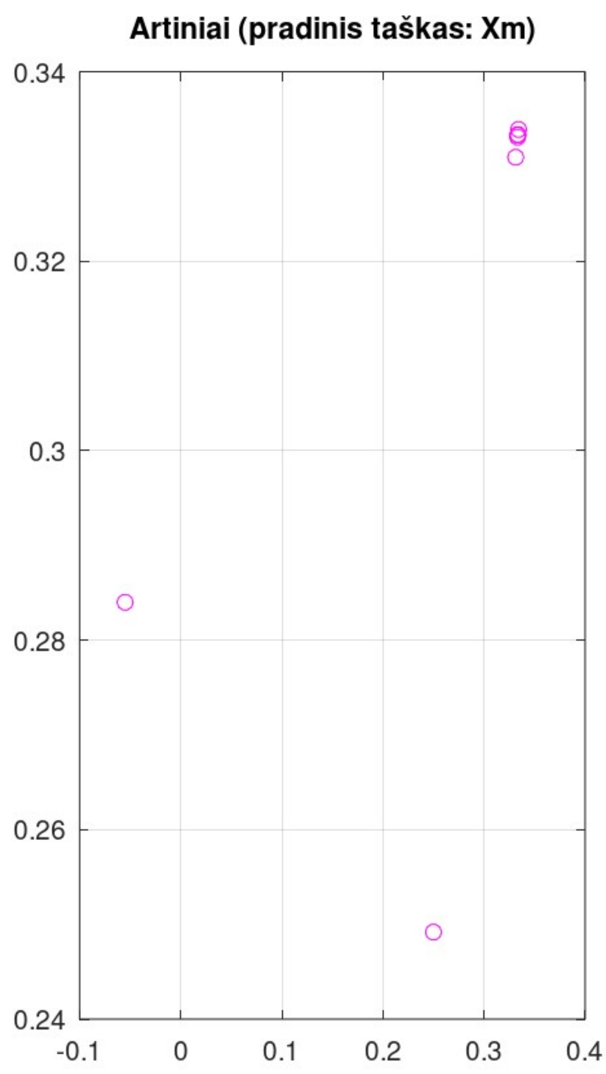
Algoritmas	Parametras *gamma*	*k* (iteracijų kiekis)	Funkcijos iškvietimai	Funkcijos iškvietimų kiekis	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
GD	0.10	*inf*	*k*	*inf*	*nepasiekta*	*nepasiekta*	*nepasiekta*
	0.25	82		82	-0.004630	0.333330	0.333302
	0.50	40		40			
	0.75	26		26			
	1.00	18		18			
	1.25	11		11			
	1.29	7		7			
	1.30	7		7			
	1.50	13		13			
	1.75	*inf*		*inf*	*nepasiekta*	*nepasiekta*	*nepasiekta*
	2.00						

Programos išvestis (iteracijų rezultatai):

Šiuo atveju gauti rezultatai su *gamma* = 1.30

x1	x2	f(x1,x2)	k (funkc. kviet. sk)	
-0.055000	0.284000	0.001505	1	1
0.249959	0.249180	-0.003900	2	2
0.331235	0.330963	-0.004629	3	3
0.334061	0.333908	-0.004630	4	4
0.333180	0.333093	-0.004630	5	5
0.333417	0.333368	-0.004630	6	6
0.333330	0.333302	-0.004630	7	7

Bandymo grafikas (artiniai):
gamma = 1.30



GD kodas:

```
function GradientDescent

f = @(x1, x2) (1 / 8) * ((x1 .^ 2) .* x2 + x1 .* (x2 .^ 2) - x1 .* x2);
% ieskome min(f(x1, x2))

gradf = @(X) [2 * X(1) * X(2) + X(2) .^ 2 - X(2), X(1) ^ 2 + 2 * X(1) * X(2) - X(1)];

% pradiniai artiniai
X_0 = [0, 0];
X_1 = [1, 1];
X_m = [4 / 10, 7 / 10];

% pasirenkamas pradinis artinys
X0 = X_m;

% Funkcijos grafiko y=f(x1,x2) braizymas
subplot(1, 2, 1);
[x1, x2] = meshgrid(0:0.01:1, 0:0.01:1);
y = f(x1, x2);
surf(x1, x2, y);
title(['Funkcijos y=f(x1,x2) grafikas']);

% pasirenkamas iteracijos parametras (kokio ilgio zingsnis atliekamas gradiento kryptimi)
% gamma reziai: (0;2/L), kur L - Lipsico konstanta.
% Testuosime 0 < gamma < 2
gamma = 0.25; %

epsilon = 10 ^ (- 4); % tikslumas

k = 1; % iteraciju skaitliukas
kmax = 100; % maksimalus iteraciju skaitliukas

% Metodo realizavimas

disp(['x1       x2       f(x1,x2)  k (funkc. kviet. sk)']);

format long;

norma = Inf;

while norma >= epsilon

    grad = gradf(X0);
    X0 = X0 - gamma .* grad; % naujas artinys
    norma = norm(grad);

     fprintf('%f %f %f %d %d\n', X0, f(X0(1), X0(2)), k, k);

    subplot(1, 2, 2);
    title(['Artiniai (pradinis taškas: X1)']);
    plot(X0(1), X0(2), 'mo');
```

```
    hold on;

    if k == kmax
        format short;
        disp(['Pasiektas maksimalus iteraciju skaicius k=', num2str(kmax)]);
        break
    endif
    k = k + 1;

endwhile
grid on;
hold off;
```

SD algoritmas

Gamma parametrui surasti atliekant vieno kintamojo funkcijos minimizavimo uždavinį buvo naudojamas Auksinio Pjūvio algoritmas. Taip pat, prireikdavo papildomai pakoreguoti Auksinio Pjūvio algoritmo intervalo viršutinį rėžį tam, kad algoritmas išduotų atsakymą, o ne dirbtų be galo.

[1] Pradinis artinys: $X_0=[0,0]$

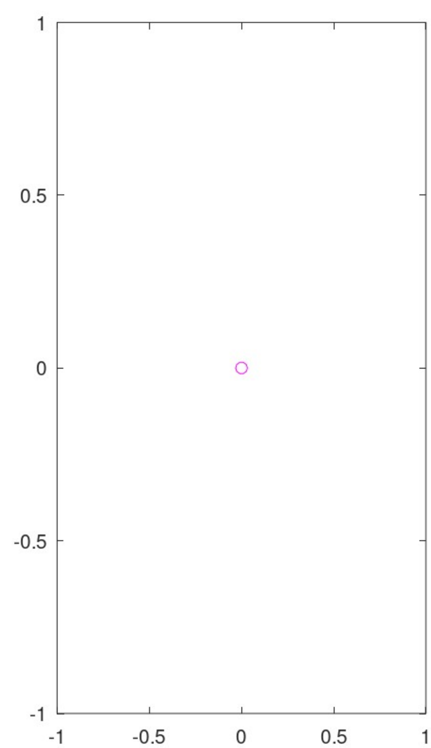
Kaip ir GD algoritmo atveju, iš šio pradinio artinio nepajudama. Keičiant viršutinį intervalo rėžį, rezultatas nesikeičia taip pat. Gaunamas identiškas rezultatas (lentelėje ir programos išvestyje):

Algoritmas	*k* (iteracijų kiekis)	Funkcijos iškvietimai	Funkcijos iškvietimų kiekis	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
SD	1	*Negalime spręsti dar*	27	0.000000	0.000000	0.000000

Programos išvestis (iteracijų rezultatai):

x1	x2	f(x1,x2)	k	(funkc. kviet. sk)
0.000000	0.000000	0.000000	1	27

Bandymo grafikas (artiniai):



[2] Pradinis artinys: $X_1 = [1, 1]$

Su artiniu X_1 galima buvo tikėtis perspektyvesnio rezultato, tačiau algoritmas nepasibaigė. Gauta tokia išvestis:

x1 x2 f(x1,x2) k (funkc. kviet. sk)
-8.999904 -8.999904 -192.368924 1 27
-1268.961228 -1268.961228 -511041483.812080 2 54
...
...
-Inf -Inf -Inf 9 243
-Inf -Inf -Inf 10 270
-Inf -Inf -Inf 11 297
-Inf -Inf -Inf 12 324
-Inf -Inf -Inf 13 351
-Inf -Inf -Inf 14 378
...
...
-Inf -Inf -Inf 95 2565
-Inf -Inf -Inf 96 2592
-Inf -Inf -Inf 97 2619
-Inf -Inf -Inf 98 2646
-Inf -Inf -Inf 99 2673
-Inf -Inf -Inf 100 2700
Pasiektas maksimalus iteraciju skaicius k=100

Algoritmo rezultatas pasikeitė, sumažinus viršutinį intervalo rėžį Auksinio Pjūvio algoritme:

l = 0;
r = 5;

pakeitus į

l = 0;
r = 0.5;

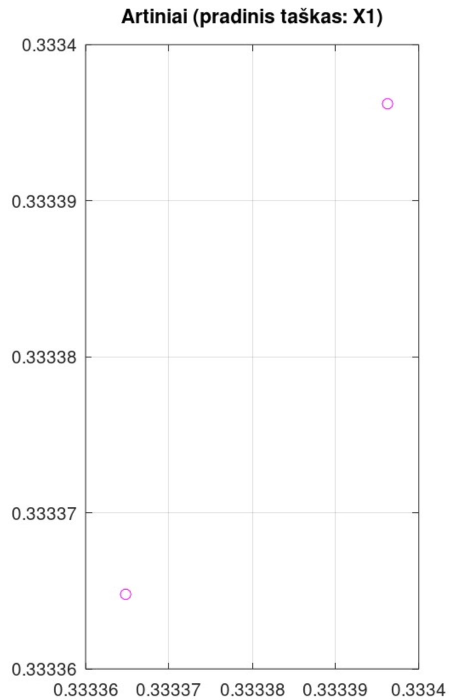
Tuomet su pradiniu artiniu X_1 algoritmas išduoda atsakymą:

Algoritmas	*k* (iteracijų kiekis)	Funkcijos iškvietimai	Funkcijos iškvietimų kiekis	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
SD	2	*~25k*	44	-0.004630	0.333365	0.333365

Programos išvestis (iteracijų rezultatai):

x1 x2 f(x1,x2) k (funkc. kviet. sk)
0.333396 0.333396 -0.004630 1 22
0.333365 0.333365 -0.004630 2 44

Bandymo grafikas (artiniai):



[3] Pradinis artinys: $X_m = [0.4, 0.7]$

Su artiniu X_m taip pat reikėjo pakoreguoti viršutinijį intervalo rėžį AP algoritme:

l = 0;
r = 5;

pakeičiama į

l = 0;
r = 3;

Tokio viršutinio rėžio užteko, kad algoritmas baigtų darbą ir išduotų atsakymą. Verta pastebėti, kad toliau mažinant viršutinį rėžį, iteracijų skaičius didėja, link minimumo judama mažesniais žingsniais.

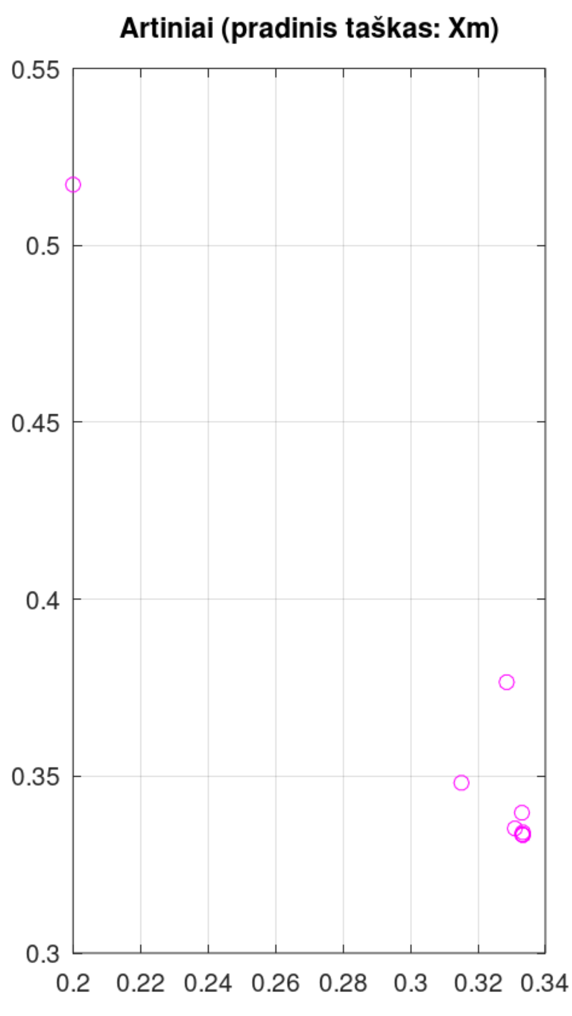
Tuomet su pradiniu artiniu X_m algoritmas išduoda atsakymą:

Algoritmas	*k* (iteracijų kiekis)	Funkcijos iškvietimai	Funkcijos iškvietimų kiekis	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
SD	9	*~25k*	234	-0.004630	0.333299	0.333360

Programos išvestis (iteracijų rezultatai):

x1	x2	f(x1,x2)	k (funkc. kviet. sk)	
0.200011	0.517153	-0.003657	1	26
0.328508	0.376557	-0.004561	2	52
0.315094	0.348107	-0.004618	3	78
0.333024	0.339652	-0.004628	4	104
0.330936	0.335225	-0.004629	5	130
0.333303	0.334109	-0.004630	6	156
0.333045	0.333560	-0.004630	7	182
0.333330	0.333426	-0.004630	8	208
0.333299	0.333360	-0.004630	9	234

Bandymo grafikas (artiniai):



SD kodas:

```
function SteepestDescent

f=@(X) (1 / 8) * (X(1)^2.*X(2)+X(1)*X(2)^2-X(1)*X(2));
% ieskome min(f(X))

gradf = @(X) [2 * X(1) * X(2) + X(2) .^ 2 - X(2), X(1) ^ 2 + 2 * X(1) * X(2) - X(1)];

% pradiniai artiniai
X_0 = [0, 0];
X_1 = [1, 1];
X_m = [4 / 10, 7 / 10];

% pasirenkamas pradinis artinys
X0 = X_m;

epsilon = 10 ^ (- 4); % tikslumas

i=0; % funkcijos kvietimu skaicius
k=1; %iteraciju skaitliukas
kmax=100; % maksimalus iteraciju skaitliukas

% Metodo realizavimas

disp(['x1       x2       f(x1,x2)  k (funkc. kviet. sk)']);

format short;
norma = Inf;

while norma>=epsilon
    grad=gradf(X0);
    res=GoldenSection(f,X0,grad);
    gamma=res(1);
    i=i+res(2)+1;
    X0 = X0 - gamma .* grad; % naujas artinys
    norma = norm(grad);

    fprintf('%f %f %f %d %d\n', X0, f(X0), k, i);

    subplot(1, 2, 2);
    grid on;
    title(['Artiniai (pradinis taškas: Xm)']);
    plot(X0(1), X0(2), 'mo');
    hold on;

    if k==kmax
        disp(['Pasiektas maksimalus iteraciju skaicius k=', num2str(kmax)]);
        break
    end
    k=k+1;
end
end
```

Auksinio Pjūvio (naudojamas SD) kodas:

```
function res=GoldenSection(f, X0, grad)

f1=@(x) f(X0-x*grad);

% Auksinio pjuvio metodu randamas funkcijos f(x) minimumas intervale [l,r].
l=0;   % apatinis intervalo rezis
r=3;  % virsutinis intervalo rezis

epsilon=10^(-4); %tikslumas

k=1; %iteraciju skaitliukas
kmax=100; % maksimalus iteraciju skaitliukas

%Metodo realizavimas
gR = (sqrt(5) - 1) / 2;

L = r-l; % intervalo ilgis
x1 = r-gR*L;
y1 = f1(x1);
x2 = l + gR*L;
y2 = f1(x2);
format short;

while L>= epsilon
    if y2 < y1
     l = x1;
     L = r - l;
     x1 = x2;
     y1=y2;
     x2 = l + gR*L;
     y2 = f1(x2);
   else
     r = x2;
     L = r - l;
     x2 = x1;
     y2=y1;
     x1 = r - gR*L;
     y1 = f1(x1);
   end

   if k==kmax
     format short
     disp(['Pasiektas maksimalus iteraciju skaicius k=', num2str(kmax)]);
     break
   end

   k=k+1;
   L=r-l;
end
res=[x1, k+2];
end
```

SIM algoritmas

Parametrų reikšmės pasinaudojant teorija:

alpha = 1/2;
teta = 1;
gamma = 2;
beta = 0.5;
eta = - 0.5;

Su kiekvienu artiniu buvo bandoma pagerinti iteracijų ir funkcijos kvietimo skaičių, keičiant parametrus eksperimentiškai, pradiniais parametrais laikant teorinius duomenis. Pradedama keisti *alpha,* kol gaunamas geresnis rezultatas. Po to keičiama *teta*, ir taip judama parametrų sąrašu žemyn, iki kol pakoreguojami visi parametrai, gaunant mažiausią iteracijų skaičių. Šis pasirinktas eksperimentavimo variantas turi trūkumų, nes, pvz., pasirinkus *alpha,* ieškome *teta* reikšmės, tačiau, ją radus, *alpha* jau nebekeičiamas. Tokiu būdu ignoruojama daugiau įmanomų variantų.

Lentelėse pavaizduoti pakeisti parametrai, ir gauti iteracijų ir funkcijos kvietimo skaičiai tuo metu (pakeitus naują parametrą, parametrai esantys lentelėje virš jo tuo metu turi naująsias reikšmes, o žemiau jo – teorines reikšmes).

[1] Pradinis artinys: $X_0 = [0, 0]$

Algoritmas	Pakeistas parametras	Nauja parametro reikšmė	Iteracijų skaičius	Funkcijos iškvietimų skaičius
SIM	*parametrai iš teorijos*	*nepakeista*	*54*	100
	alpha	0.7	*52*	100
	teta	0.2	*52*	100
	gamma	1.9	*51*	100
	beta	0.5	*52*	100
	eta	-0.6	*50*	100

[2] Pradinis artinys: $X_1 = [1, 1]$

Algoritmas	Pakeistas parametras	Nauja parametro reikšmė	Iteracijų skaičius	Funkcijos iškvietimų skaičius
SIM	*parametrai iš teorijos*	*nepakeista*	*59*	112
	alpha	0.2	*55*	100
	teta	0.1	*53*	100
	gamma	2.0	*53*	100
	beta	0.5	*53*	100
	eta	-0.5	*53*	100

[3] Pradinis artinys: $X_m = [0.4\,,0.7]$

Algoritmas	Pakeistas parametras	Nauja parametro reikšmė	Iteracijų skaičius	Funkcijos iškvietimų skaičius
SIM	*parametrai iš teorijos*	*nepakeista*	*55*	100
	alpha	0.5	*55*	100
	teta	0.2	*54*	100
	gamma	2.0	*54*	100
	beta	0.5	*54*	100
	eta	-0.5	*54*	100

Atlikę parametrų pakeitimus, matome, kad nors ir galima pagerinti iteracijų ir funkcijos kvietimų skaičių (labiausiai pasikeičia pradinio artinio X_1 atveju), tačiau apskritai skirtumas nėra žymus, ir teoriniai parametrai yra tinkami ir pilnai pakankami bendru atveju.

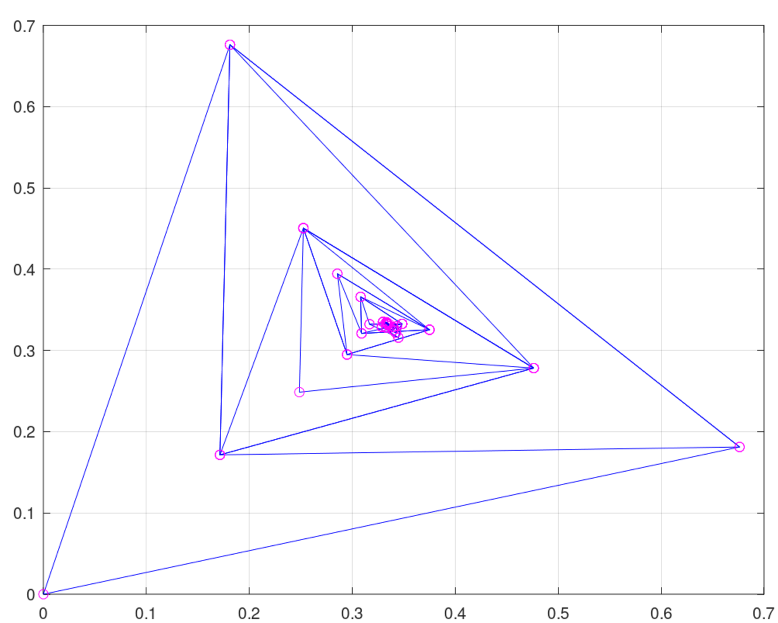
Minimumo rezultatai su skirtingais artiniais:

Algoritmas	Artinys	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
SIM	X_0	-0.004630	0.333331	0.333333
	X_1	-0.004630	0.333322	0.333356
	X_m	-0.004630	0.333315	0.333363

Šie rezultatai gauti su mažiausiu iteracijų ir funkcijos kvietimo skaičiumi kiekvienam artiniui, pagal prieš tai buvusias tris lenteles.

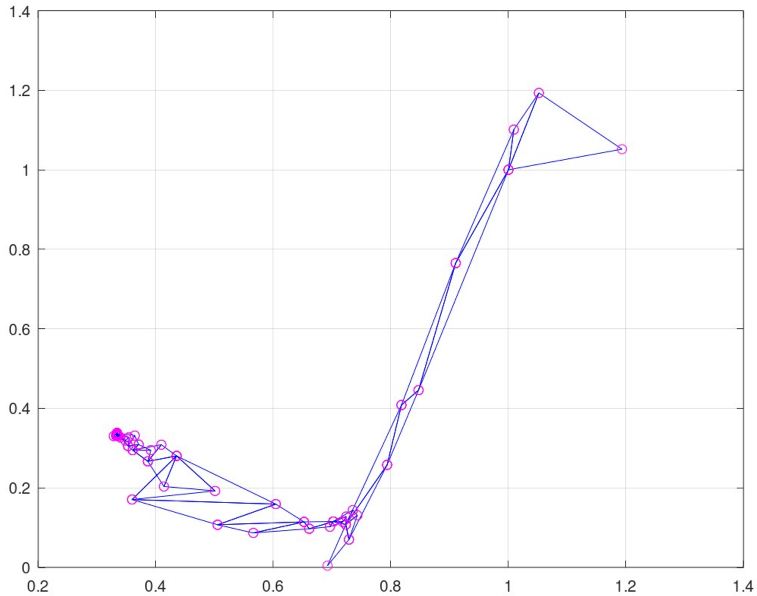
Atitinkamai galime atvaizduoti SIM algoritmo veikimą ir grafikuose kiekvienam pradiniam artiniui:

Bandymo grafikas (artinys X_0):

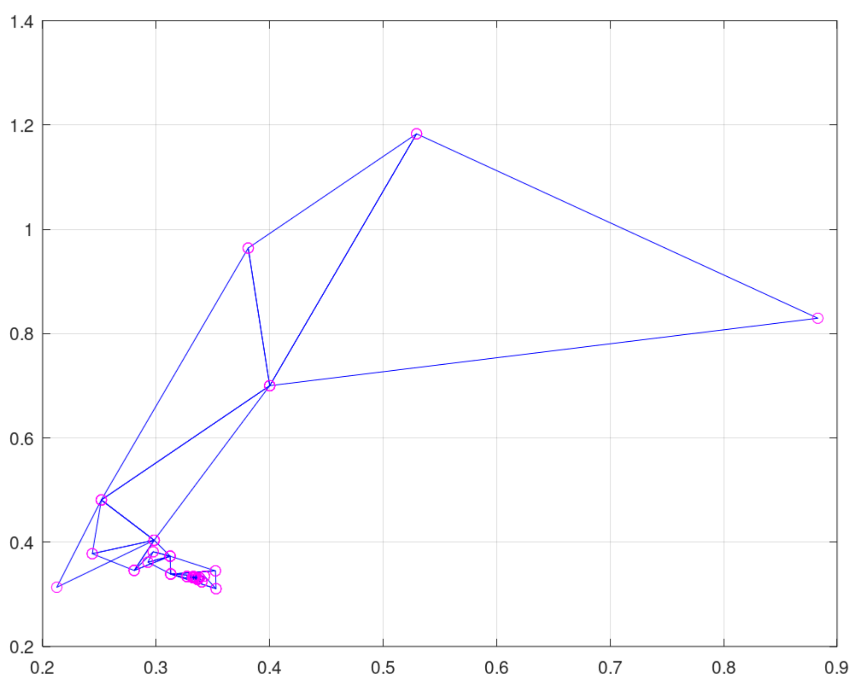


Bandymo

grafikas (artinys X_1):



Bandymo grafikas (artinys X_m):



Programos išvestis (iteracijų rezultatai):

šiuo atveju pateikiama išvestis, naudojant artinį X_m :

```
  x1         x2         f(x1,x2)   k  (f kv. sk.)
-----------------------------------------------
0.381053   0.963896   0.015837 1   4
0.251644   0.480933   -0.004046 2   6
0.298206   0.403752   -0.004486 3   9
0.212387   0.313514   -0.003946 4   11
0.243656   0.377928   -0.004356 5   13
0.280575   0.345793   -0.004531 6   15
0.312258   0.373195   -0.004582 7   17
0.297311   0.381623   -0.004554 8   19
Funkcijos reiksmes simplekso virsunese panasios (tikslumu epsilon=0.0001)
0.292680   0.361601   -0.004574 9   20
0.312784   0.338948   -0.004615 10   22
0.352203   0.345012   -0.004599 11   23
0.352729   0.310765   -0.004611 12   24
0.342479   0.334934   -0.004625 13   26
0.340180   0.323853   -0.004627 14   28
0.327057   0.334171   -0.004628 15   30
0.338049   0.331973   -0.004629 16   32
0.336366   0.328463   -0.004629 17   34
0.332132   0.332195   -0.004629 18   36
0.335728   0.330273   -0.004629 19   38
0.335990   0.331604   -0.004629 20   39
0.332393   0.333525   -0.004630 21   41
0.334126   0.332232   -0.004630 22   43
0.333824   0.333220   -0.004630 23   45
0.332600   0.333943   -0.004630 24   46
0.332802   0.333553   -0.004630 25   48
0.332956   0.333665   -0.004630 26   50
0.333096   0.333498   -0.004630 27   52
0.333425   0.333401   -0.004630 28   54
0.333108   0.333557   -0.004630 29   56
0.333184   0.333503   -0.004630 30   57
0.333200   0.333475   -0.004630 31   59
0.333249   0.333470   -0.004630 32   61
0.333325   0.333437   -0.004630 33   63
0.333244   0.333464   -0.004630 34   64
0.333355   0.333410   -0.004630 35   66
Simpleksas tapo mazas (krastiniu ilgiai mazesni uz epsilon=0.0001)
0.333292   0.333444   -0.004630 36   68
0.333324   0.333432   -0.004630 37   69
0.333316   0.333432   -0.004630 38   71
0.333330   0.333427   -0.004630 39   72
0.333329   0.333425   -0.004630 40   74
0.333348   0.333414   -0.004630 41   76
0.333340   0.333419   -0.004630 42   78
0.333339   0.333419   -0.004630 43   80
0.333350   0.333411   -0.004630 44   82
0.333337   0.333419   -0.004630 45   84
```

0.333353	0.333407	-0.004630	46	86
0.333334	0.333416	-0.004630	47	88
0.333356	0.333397	-0.004630	48	90
0.333330	0.333404	-0.004630	49	92
0.333362	0.333370	-0.004630	50	94
0.333325	0.333368	-0.004630	51	96
0.333370	0.333299	-0.004630	52	97
0.333341	0.333316	-0.004630	53	99
0.333315	0.333363	-0.004630	54	100

Pasiektas maksimalus funkciju kvietimu skaicius i=100

Patenkinamos sustojimo salygos. Skaiciavimai baigiami, nes:

1) simpleksas tapo mazas (krastiniu ilgiai mazesni uz epsilon=0.0001)
2) funkcijos reiksmes simplekso virsunes panasios (tikslumu epsilon=0.0001)
3) pasiektas maksimalus funkciju kvietimu skaicius=100)

SIM kodas:

```
function Simplex

f = @(X) (1 / 8) * ((X(1) .^ 2) .* X(2) + X(1) .* (X(2) .^ 2) - X(1) .* X(2));

% pradiniai artiniai
X_0 = [0, 0];
X_1 = [1, 1];
X_m = [4 / 10, 7 / 10];

% pasirenkamas pradinis artinys
X0 = X_m;

% pasirenkami parametrai
alpha = 0.5; % reguliuoja pradinio simplekso krastines ilgi
teta = 0.2; % reguliuoja tieses lygti, breziamos per vidurio taska, ieskant naujos virsunes
% simplekso deformavimo koeficientai
gamma = 2.0; % reguliuoja simplekso ispletima, gamma > 1
beta = 0.5; % reguliuoja simplekso suspaudima, 0 < beta < 1
eta = - 0.5; % reguliuoja simplekso suspaudima, -1 < eta < 0

epsilon = 10 ^ (- 4); % tikslumas

% Pradinio simplekso sudarymas
n = 2; % keliu kintamuju funkcija yra minimizuojama
delta1 = alpha * (sqrt(n + 1) + n - 1) / (n * sqrt(2));
delta2 = alpha * (sqrt(n + 1) - 1) / (n * sqrt(2));

% kitos simplekso virsunes (apskaiciuojame pagal teorine medziaga)
X1 = [X0(1, 1) + delta2, X0(1, 2) + delta1];
X2 = [X0(1, 1) + delta1, X0(1, 2) + delta2];

% funkcijos reiksmes simplekso virsunese
y0 = f(X0);
y1 = f(X1);
y2 = f(X2);

% simplekso virsuniu masyvas
X = [X0; X1; X2];

% funkcijos reiksmiu simplekso virsunese masyvas
y = [y0, y1, y2];

% Pradinio simplekso braizymas:
deltax = [X0(1), X0(1), X1(1); X1(1), X2(1), X2(1)];
deltay = [X0(2), X0(2), X1(2); X1(2), X2(2), X2(2)];
plot(deltax, deltay, 'b');
grid on;
hold on;
plot(X(:, 1), X(:, 2), 'mo'); % atvaizduoja bandymo taskus rutuliukais
hold on;
```

```
k = 1; % iteraciju skaitliukas (pradinio simplekso sudarymas = 1 iteracija)
i = 3; % funkcijos kvietimu skaiciaus skaitliukas
kmax = 100; % maksimalus iteraciju skaicius
imax = 100; % maksimalus funkcijos kvietimu skaicius

format short;

% Metodo realizavimas
disp(['   x1   x2    f(x1,x2)    k    (f kv. sk.)']);
disp('----------------------------------------------');

goal = false;
while ~ goal
   % Randami Xh, Xg, Xl ir funkcijos reiksmes siuose taskuose yh, yg, yl
   [~, nr] = sort(y); % y0, y1, y2 reiksmes isdestomos didejimo tvarka; nr rodys ju numerius masyve y

   yl = y(nr(1)); % maziausia y reiksme
   Xl = X(nr(1), :);

   yh = y(nr(n + 1)); % didziausia y reiksme
   Xh = X(nr(n + 1), :);

   yg = y(nr(n)); % antra pagal dydi y reiksme
   Xg = X(nr(n), :);

   % Viduriu tasko Xc ir naujo artinio Xnew apskaiciavimas
   Xc = (Xg + Xl) / 2;

   Xnew = Xh + (1 + teta) * (Xc - Xh);
   ynew = f(Xnew);
   i = i + 1;

   % Jei bent viena neigiama koordinate, keiciame krypti
   if Xnew(1) <= 0 || Xnew(2) <= 0
      %disp('Neigiama artinio koordinate! Keiciame krypti.');
      teta = - 1 / 2;
      Xnew = Xh + (1 + teta) * (Xc - Xh);
      ynew = f(Xnew);
      i = i + 1;
   endif

   % Naujo simplekso sudarymas
   if (yl < ynew) && (ynew < yg)
      teta = 1;
   elseif ynew < yl
      teta = gamma;
      Z = Xh + (1 + teta) * (Xc - Xh);
      yz = f(Z);
      i = i + 1;
      if yz < ynew
         ynew = yz;
         Xnew = Z;
      endif
```

```
elseif ynew > yh
    teta = eta;
    Z = Xh + (1 + teta) * (Xc - Xh);
    Xnew = Z;
    ynew = f(Z);
    i = i + 1;
elseif (yg < ynew) && (ynew < yh)
    teta = beta;
    Z = Xh + (1 + teta) * (Xc - Xh);
    Xnew = Z;
    ynew = f(Z);
    i = i + 1;
endif

if Xnew(1) <= 0 || Xnew(2) <= 0
    disp('Neigiama artinio koordinate! Keiciame krypti.');
    teta = - 1 / 2;
    Xnew = Xh + (1 + teta) * (Xc - Xh);
    ynew = f(Xnew);
    i = i + 1;
endif

 fprintf('%f    %f   %f %d   %d', Xnew, ynew, k, i);

count = 0;

if max([norm(Xl - Xg), norm(Xl - Xh), norm(Xg - Xh)]) < epsilon
    disp(' ')
    disp(['Simpleksas tapo mazas (krastiniu ilgiai mazesni uz epsilon=', num2str(epsilon), ')']);
    count = count + 1;
endif

if max([abs(yl - yg), abs(yl - yh), abs(yg - yh)]) < epsilon
    % used for pretty output
    if ~count
      disp(' ')
    endif
    disp(['Funkcijos reiksmes simplekso virsunese panasios (tikslumu epsilon=', num2str(epsilon), ')']);
    count = count + 1;
endif

if i >= imax
    count = count + 1;
    disp(['Pasiektas maksimalus funkciju kvietimu skaicius i=', num2str(imax)]);
    if count == 3
        disp(' ');
        disp('Patenkinamos sustojimo salygos. Skaiciavimai baigiami, nes:');
        disp(['1) simpleksas tapo mazas (krastiniu ilgiai mazesni uz epsilon=', num2str(epsilon), ')']);
        disp(['2) funkcijos reiksmes simplekso virsunes panasios (tikslumu epsilon=', num2str(epsilon), ')']);
        disp(['3) pasiektas maksimalus funkciju kvietimu skaicius=', num2str(imax), ')']);
        goal = true;
    endif
endif
```

```
    if k == kmax
        format short;
        disp(['Pasiektas maksimalus iteraciju skaicius k=', num2str(kmax)]);
        break
    endif

    k = k + 1;
    % naujas artinys
    X = [Xl; Xg; Xnew];
    % funckijos reiksmes naujame artinyje
    y = [yl, yg, ynew];


    % Simplekso braizymas:
    deltax = [Xl(1), Xl(1), Xg(1); Xg(1), Xnew(1), Xnew(1)];
    deltay = [Xl(2), Xl(2), Xg(2); Xg(2), Xnew(2), Xnew(2)];
    plot(deltax, deltay, 'b');
    hold on;
    plot(X(:, 1), X(:, 2), 'mo');
    hold on;

    % used for pretty output
    if ~ count
        disp(' ');
    endif

endwhile
```

Bendras algoritmų palyginimas

Algoritmas	Pradinis artinys	*k* (iteracijų kiekis)	Funkcijos iškvietimai	Funkcijos iškvietimų kiekis	Funkcijos minimumas, y_m	Argumentas x_1	Argumentas x_2
GD	X_0	1	*k*	1	*nepasiektas*	*nerastas*	*nerastas*
	X_1	14	*k*	14	-0.004630	0.333369	0.333369
	X_m	7	*k*	7	-0.004630	0.333330	0.333302
SD	X_0	1	-	27	*nepasiektas*	*nerastas*	*nerastas*
	X_1	2	~25*k*	44	-0.004630	0.333365	0.333365
	X_m	9	~25*k*	234	-0.004630	0.333299	0.333360
SIM	X_0	50	~2*k*	100	-0.004630	0.333331	0.333333
	X_1	53	~2*k*	100	-0.004630	0.333322	0.333356
	X_m	54	~2*k*	100	-0.004630	0.333315	0.333363

Bandymų išvados:

[1] Matome, kad su pradiniu artiniu X_0 GD ir SD algoritmai atsakymo neišdavė. Taip nutiko jau dėl minėtos problemos, kad gradiento funkcija yra nulinė, ir nėra pajudama naujo artinio link. Šios problemos nėra vykdant SIM algoritmą, tad šiuo atžvilgiu jis turi privalumą.

[2] Su pradiniais artiniais X_1 ir X_m visi algoritmai išduoda atsakymą. SD vykdo mažiausiai iteracijų, tačiau kiekvienos iteracijos metu funkcijos yra kviečiamos daugiau nei 20 kartų. Taigi, nors SD juda greičiau minimumo link nei GD, tačiau papildomai resursų yra sunaudojama *gamma* parametro paieškai kiekvieną iteraciją, kuomet kviečiamas Auksinio Pjūvio algoritmas. GD šiuo atveju kiekvieną iteraciją funkciją kviečia tik vieną kartą. SIM algoritmas atlieka kur kas daugiau iteracijų nei GD ir SD, o funkcija kviečiama apytikriai du kartus kas iteraciją, taigi daugiau nei GD. Nepaisant to, jis išduoda atsakymą su visais bandymo metu naudotais artiniais.

[3] Verta paminėti ir pradinių parametrų pasirinkimo klausimą kiekvienam algoritmui. GD atveju, algoritmo žingsnių skaičius priklauso nuo pradinio *gamma* parametro, ir skirtumas tarp nedidelio *gamma* pokyčio gali būti akivaizdus, iteracijų skaičiaus atžvilgiu.

SD atveju, nors *gamma* yra apskaičiuojamas kas iteraciją, tam, kad algoritmas nedirbtų be galo, yra svarbu tinkamai parinkti viršutinįjį intervalo rėžį Auksinio Pjūvio algoritme kiekvienam skirtingam pradiniam artiniui, o tai akivaizdžiai prideda sudėtingumo šiam metodui, nes parametras ne tik nustato algoritmo spartą, bet ir jo veikimo korektiškumą.

SIM algoritme, nors eksperimento metu ir buvo bandoma gauti mažiau iteracijų keičiant visus parametrus, tačiau skirtumas buvo nežymus, ir teoriniai parametrai kiekvienam artiniui buvo pakankami. Taigi pradinių parametrų pasirinkimo klausimu SIM algoritmas turi akivaizdų pranašumą.