Skaitiniai funkcijų minimizavimo metodai

A.Domarkas, VU,

2015 m. balandžio mėn.

Čia yra naudojama atvirojo kodo kompiuterinės algebros sistema Maxima 5.31.2

1 Vieno kintamojo funkcijos minimizavimas

1.1 Atkarpos dalijimo pusiau metodas

Apibrėžimas. Vieno kintamojo funkcija f(x) vadinama unimodaliąja intervale [a, b], jeigu egzistuoja toks taškas x* iš [a, b], kad intervale [a, x*] ta funkcija mažėja, o intervale [x*, b] didėja.

Unimodalumo intervalus lengva nustatyti nusibrėžiant funkcijos grafiką. Pateiktuose algoritmuose reikia, kad minimizuojama funkcija būtų unimodali.

Atkarpos dalijimo pusiau algoritmas([1], 226 p.):
1. Atkarpą [a, b] taškais x1=a, x2, x3, x4, x5=b padalijame į keturias lygias dalis.
2. Randame mažiausią reikšmę m iš penkių funkcijos reikšmių
f(x1), f(x2), f(x3), f(x4), f(x5).
3. Jei m=f(x1), tai intervalą [a, b] pakeičiame intervalu [x1, x2],
jei m=f(x2), tai intervalą [a, b] pakeičiame intervalu [x1, x3],
jei m=f(x3), tai intervalą [a, b] pakeičiame intervalu [x2, x4],
jei m=f(x4), tai intervalą [a, b] pakeičiame intervalu [x3, x5],
jei m=f(x5), tai intervalą [a, b] pakeičiame intervalu [x4, x5].
4. Veiksmus 1.-3. kartojame tol, kol bus b-a<Delta.
5. Minimumo tašku laikome atkarpos [a, b] vidurio tašką (a+b)/2.

```
(%i1) minimize_hs(f,x,s,Delta):=block([a,b,m,k,g],
      [a,b]:[s[1],s[2]],
      define(g(x),f),
      for k while abs(b-a)>Delta
      do
      (
      [x1,x2,x3,x4,x5]:[a,a+(b-a)/4,a+(b-a)/2,a+3/4*(b-a),b],
      m:lmin([g(x1),g(x2),g(x3),g(x4),g(x5)]),
      if m=g(x(1)) then [a,b]:[x1,x2]
      elseif m=g(x2) then [a,b]:[x1,x3]
      elseif m=g(x3) then [a,b]:[x2,x4]
      elseif m=g(x4) then [a,b]:[x3,x5]
      else [a,b]:[x4,x5]
      ),
      float((a+b)/2),
      [x=%%, f_min=g(%%)]
      )$
```

1 pavyzdys. Atkarpos dalijimo pusiau metodu rasime funkcijos f(x) = x^2 - 2x minimumą intervale [0, 5]. Laikysime, kad tikslumas Delta=1/1000.

```
(%i2) minimize_hs(x^2-2*x,x,[0,5],10^-3);
(%o2) [x=1.00006103515625, f_min=-0.9999999962747097]
```

Aišku, tikslusis sprendinys yra x = 1, f_min=f(1) = -1.

2 pavyzdys. Atkarpos dalijimo pusiau metodu rasime funkcijos f(x) = x^4-3x+1 minimumą intervale [0, 2]. Laikysime, kad tikslumas Delta=10^(-10).

```
(%i3) minimize_hs(x^4-3*x+1,x,[0,2],10^-10);
(%o3) [x=0.9085602964041755, f_min=-1.044260666936157]
```

Kitas sprendimo būdas yra ieškant funcijos stacionariuosius taškus:

(%i4) f:x^4-3*x+1$

(%i5) f1:diff(f,x);

(%o5) $4\,x^3-3$

(%i6) solve(f1,x);

(%o6) $[x=\frac{3^{5/6}\,\%i-3^{1/3}}{2\,4^{1/3}},x=-\frac{3^{5/6}\,\%i+3^{1/3}}{2\,4^{1/3}},x=\frac{3^{1/3}}{4^{1/3}}]$

(%i7) ats:subst(%[3],[x,f_min=f]);

(%o7) $[\frac{3^{1/3}}{4^{1/3}},f\_min=-\frac{3^{4/3}}{4^{1/3}}+\frac{3^{4/3}}{4^{4/3}}+1]$

(%i8) float(%), numer;

(%o8) [0.9085602964160698, f_min=-1.044260666936157]

Atkarpos dalijimo metodas yra bendresnis už šį metodą, nes tinka funkcijoms, kurios yra nediferncijuojamos arba neturi analizinės išraiškos.

3 pavyzdys. Atkarpos dalijimo pusiau metodu rasime funkcijos f(x) = sqrt(x)*cos(x)*erf(x) loaliuosius minimumus intervale [0, 20]. Laikysime, kad tikslumas Delta=1/100.

(%i9) f:sqrt(x)*cos(x)*erf(x);

(%o9) $\sqrt{x}\,\cos(x)\,\mathrm{erf}(x)$

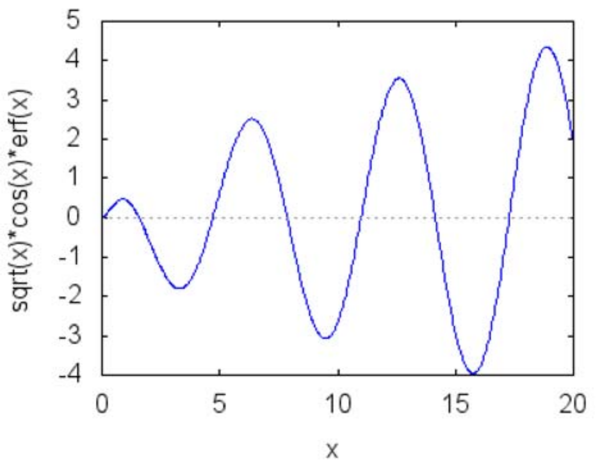
(%i10) Delta:1/1000;

(%o10) $\frac{1}{1000}$

Grafiškai rasime funkcijos unimodalumo intervalus.

(%i11) wxplot2d([f], [x,0,20])$

(%t11)



Iš grafiko matome, kad intervale [0, 20] funkcija turi tris lokaliuosius minimumus, kurie yra intervaluose: [2, 5], [8, 10], [15, 18].

(%i12) minimize_hs(f,x,[2,5],Delta);

(%o12) [x=3.2923583984375, f_min=-1.79389705191039]

(%i13) minimize_hs(f,x,[8,10],Delta);

(%o13) [x=9.4775390625, f_min=-3.074277246458224]

(%i14) minimize_hs(f,x,[15,18],Delta);

(%o14) [x=15.73974609375, f_min=-3.965331256446146]

1.2 Auksinio pjūvio metodas

Auksinio pjūvio algoritmas([1], 226 p.; [2], 215 p.):
1. Apskaičiuojame funkcijos f(x) reikšmes taškuose
x1=r*a+1-r)*b ir x2=(1-r)*a+r*b, čia r = (sqrt(5)-1)/2~~0.618
2. Jei f(x1)<f(x2) , tai intervalą [a, b] pakeičiame intervalu [a, x2],
jei f(x1)>=f(x2), tai intervalą [a, b] pakeičiame intervalu [x1, b].
3. Apskaičiuojame naujas x1 ir x2 reikšmes(pagal tas pačias formules kaip 1. žingsnyje).
4. Veiksmus 1.-3. kartojame tol, kol bus b-a<Delta.
5. Minimumo tašku laikome atkarpos [a, b] vidurio tašką (a+b)/2.

```
(%i15) r:(sqrt(5)-1)/2;
```
(%o15) $\frac{\sqrt{5}-1}{2}$

```
(%i16) float(%);
(%o16) 0.6180339887498949
```

```
(%i17) algebraic:true;
(%o17) true
```

```
(%i18) r/(1-r);
```
(%o18) $\frac{\sqrt{5}-1}{2\left(1-\frac{\sqrt{5}-1}{2}\right)}$

```
(%i19) ratsimp(%);
```
(%o19) $\frac{\sqrt{5}+1}{2}$

Atkarpos [a, b] taškai gali būti išreikšti pavidalu (1-r)*a+r*b.
Kai r=0, tai gauname a;
kai r=1, tai gauname b;
kai r=1/2, tai gauname atkarpos vidurį (a+b)/2;
kai r=(sqrt(5)-1)/2 tai atkarpa pasidalina į dalis, kurių ilgių santykis yra "auksinė proporcija".

Patikrinsime paskutinį teiginį:

```
(%i20) ((1-r)*a+r*b-a)/(b-((1-r)*a+r*b)),radcan;
```
(%o20) $\frac{\sqrt{5}+1}{2}$

```
(%i21) float(%), numer;
(%o21) 1.618033988749895
```

Skaičius (sqrt(5)+1)/2 yra vadinamas "auksine proporcija" arba "aukso pjūviu".
Gamtoje ir mūsų gyvenime jis daug kur pasitaiko(žr. internete).

Maxima sistemoje šis skaičius yra viena iš konstantų ir žymima %phi.
Svarbesnės už %phi yra tik %pi ir %e.

```
(%i22) %phi;
```
(%o22) φ

```
(%i23) float(%);
(%o23) 1.618033988749895
```

Yra teisinga tapatybė: 1/%phi = r. Čia r=(sqrt(5)-1)/2.
Patikrinimas.

```
(%i24) 1/%phi;
```
(%o24) $\frac{1}{\varphi}$

(%i25) subst(%phi=(sqrt(5)+1)/2,%);

(%o25) $\frac{2}{\sqrt{5}+1}$

(%i26) ratsimp(%);

(%o26) $\frac{\sqrt{5}-1}{2}$

Gavome skaičių r. Tapatybė patikrinta.

Todėl skaičius %phi =(sqrt(5)+1)/2 yra auksinė proporcija, kai didesnę dalį dalijame iš mažesnės,
o skaičius r = (sqrt(5)-1)/2 yra auksinės proporcijos atvirkštinis dydis, kuris gaunamas
mažesnę dalį dalijant iš didesnės.

```
(%i27) minimize_gs(f,x,s,Delta):=block([a,b,r,x1,x2,fx1,fx2,xs],
       [a,b]:[s[1],s[2]],
       define(g(x),f),
       r:float((sqrt(5)-1)/2),
       x1:r*a+(1-r)*b,
       x2:(1-r)*a+r*b,
       for i while b-a>Delta do
       (
       if g(x1)<g(x2) then
       b:x2 else a:x1,
       x1:r*a+(1-r)*b,
       x2:(1-r)*a+r*b
       ),
       xs:float((a+b)/2),
       [x=xs, f_min=g(xs)]
       )$
```

1 pavyzdys. Auksinio pjūvio metodu rasime funkcijos f(x) = x^2 - 2x
minimumą intervale [0, 5]. Laikysime, kad tikslumas Delta=1/1000.

(%i28) minimize_gs(x^2-2*x,x,[0,5],10^-3);

(%o28) $[x=0.9998317521661272, f\_min=-0.9999999716926664]$

Aišku, tikslusis sprendinys yra [x = 1, f_min=f(1) = -1.

2 pavyzdys. Auksinio pjūvio metodu rasime funkcijos f(x) = x^4-3x+1
minimumą intervale [0, 2]. Laikysime, kad tikslumas Delta=10^(-10).

(%i29) minimize_gs(x^4-3*x+1,x,[0,2],10^-10);

(%o29) $[x=0.908560301481708, f\_min=-1.044260666936157]$

3 pavyzdys. Auksinio pjūvio metodu rasime funkcijos f(x) = sqrt(x)*cos(x)*erf(x)
loaliuosius minimumus intervale [0, 20]. Laikysime, kad tikslumas Delta=1/100.

(%i30) f:sqrt(x)*cos(x)*erf(x);

(%o30) $\sqrt{x}\cos\left(x\right)\mathrm{erf}\left(x\right)$

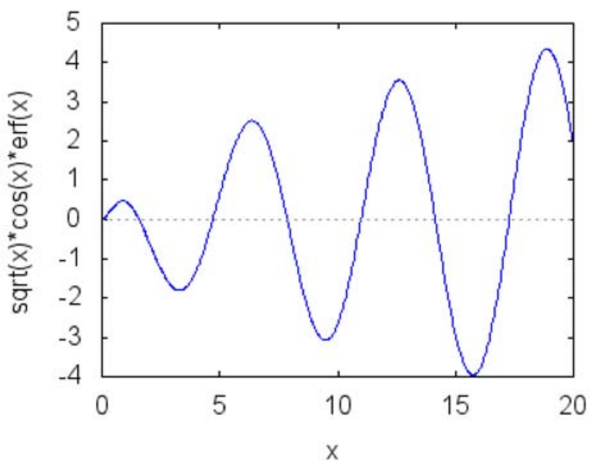
(%i31) Delta:1/1000;

(%o31) $\frac{1}{1000}$

Grafiškai rasime funkcijos unimodalumo intervalus.

```
(%i32) wxplot2d([f], [x,0,20])$
```

(%t32)



Iš grafiko matome, kad intervale [0, 20] funkcija turi tris lokaliuosius minimumus,
kurie yra intervaluose: [2, 5], [8, 10], [15, 18].

```
(%i33) minimize_gs(f,x,[2,5],Delta);
```

(%o33) $[x = 3.292150010912388, f\_min = -1.793897021325034]$

```
(%i34) minimize_gs(f,x,[8,10],Delta);
```

(%o34) $[x = 9.477721020904367, f\_min = -3.074277165043013]$

```
(%i35) minimize_gs(f,x,[15,18],Delta);
```

(%o35) $[x = 15.73961407606254, f\_min = -3.96533123582578]$

Kitas sprendimo būdas yra ieškant funcijos stacionariuosius taškus:

```
(%i36) f1:diff(f,x);
```

(%o36) $-\sqrt{x}\,\mathrm{erf}(x)\sin(x) + \frac{\cos(x)\,\mathrm{erf}(x)}{2\sqrt{x}} + \frac{2\sqrt{x}\,\%e^{-x^2}\cos(x)}{\sqrt{\pi}}$

```
(%i37) find_root(f1, x, 2, 5);
```

(%o37) 3.292330717436803

```
(%i38) ats1:subst(x=%,[x,f_min=f]);
```

(%o38) $[3.292330717436803, f\_min = -1.793897052645326]$

Čia išnagrinėjome tik kelis pagrindinius netiesinių optimizavimo uždavinių sprendimo metodus.
Skaitytojui siūlau užprogamuoti dar kitus metodus. Galima taip pat modifikuoti jau pasiūlytus.
Išvedamų skaitmenų skaičių galima kontroliuoti su Maxima sistemos parametru fpprintprec,
priskiriant jam reikšmes nuo 2 iki 16. Pagal nutylėjimą yra išvedama 16 skaitmenų.

2 Gradientų metodas

Iteraciniai sprendimo metodai remiasi rekurentiniu sąryšiu

x^(k+1) = x^k + alpha_k*p^k, k = 0, 1, ...   (1)

Jame vektorius p^k rodo kryptį, kuria einama iš taško x^k į tašką x^(k+1).
Skaliarinis dydis alpha_k yra vadinamas žingsnio ilgiu.
Iteraciniai metodai skiriasi vieni nuo kitų pagal tai, kaip pasirenkamas
krypties vektorius p^k ir apskaičiuojamas žingsnio ilgis alpha_k.

Funkcijos f(x) gradientas grad(f(x)) apibūdina funkcijos greičiausio augimo kryptį taške x.
Priešinga kryptis -grad(f(x)) nurodo funkcijos greičiausio mažėjimo kryptį.

Gradientų metode rekurentinėje formulėje (1) pasirenkame p^k = -grad(f(x^k)).
Žingsnio ilgis alpha_k yra parenkamas taip, kad minimizuotų vieno kintamojo r funkciją f(x^k - r*grad(f(x^k))) . Jos minimumą randame ieškodami stacionariųjų taškų, arba naudojant skaitinius metodus. Gradientų iteracinį procesą galima užrašyti taip:

x^(k+1) = x^k - alpha_k*grad(f(x^k)), k = 0, 1, ...   (2)

1 pavyzdys ([2], 226 p.) Gradientų metodu rasime funkcijos f(x,y) = x^2+2y^2 + exp(x+y) minimumą.

```
(%i39) f:x^2+2*y^2+exp(x+y);
```
(%o39) $\%e^{y+x}+2\,y^2+x^2$

```
(%i40) fpprintprec:16;
(%o40) 16
```

```
(%i41) zingsnis(f,x0):=block([v1,r],
       v1:x0-r*at([diff(f,x),diff(f,y)],[x=x0[1],y=x0[2]]),
       float(ev(f,[x=%%[1],y=%%[2]])),
       find_root(diff(%%,r), r, 0, 0.5),
       float(ev(v1,r=%%))
       )$
```

```
(%i42)  v:[1,1]$ for k thru 30 do
       (v:zingsnis(f,v),
       if k>=20 then print([k, ev(f,[x=v[1],y=v[2]]),v]))$
[20,0.772268227723419,[-0.3127668071295656,-0.1563834035647138]]
[21,0.772268227723419,[-0.3127668071298583,-0.1563834035650687]]
[22,0.7722682277234189,[-0.3127668071299668,-0.1563834035649793]]
[23,0.772268227723419,[-0.3127668071299842,-0.1563834035650004]]
[24,0.7722682277234189,[-0.3127668071299907,-0.1563834035649951]]
[25,0.7722682277234189,[-0.3127668071299917,-0.1563834035649963]]
[26,0.7722682277234189,[-0.3127668071299921,-0.156383403564996]]
[27,0.772268227723419,[-0.3127668071299922,-0.1563834035649961]]
[28,0.772268227723419,[-0.3127668071299922,-0.1563834035649961]]
[29,0.772268227723419,[-0.3127668071299922,-0.1563834035649961]]
[30,0.772268227723419,[-0.3127668071299922,-0.1563834035649961]]
```

Matome, kad nuo k = 24 iteracijų reikšmės nesikeičia.

Atsakymas: f_min = f(-0.31276680712999,-0.156383403565) = 0.77226822772342.

2 būdas:

```
(%i44) load(mnewton)$
```

```
(%i45) mnewton([diff(f,x),diff(f,y)],[x,y],[0,0]);
(%o45) [[x=-0.3127668071299921,y=-0.1563834035649961]]
```

```
(%i46) ev(f,%);
(%o46) 0.7722682277234189
```

3 būdas:

```
(%i26) load(fmin_cobyla)$
```

```
(%i48) fmin_cobyla(f, [x, y], [0,0], constraints = []);
(%o48) [[x=-0.3127657288111336,y=-0.1563837502679593],0.7722682277249895,59,0]
```

Komanda "fmin_cobyla" gali rasti funkcijų eksteremumus su apribojimais. Matome, kad gaunamas šiek tiek mažesnis tikslumas.

3 Niutono metodas

Niutono metodas gaunamas, kai preito skyrelio (1) formulėje imame alpha_k = 1,
p^k = - grad(f(x^k)*Hf(x^k)^(-1).
Čia Hf(x^k)^(-1) yra Hesės matricos atvirkštinė matrica, apskaičiuota taške x^k.
Tada gaunme iteracinį procesą:

x^(k+1) = x^k - grad(f(x^k)*Hf(x^k)^(-1), k = 0, 1, ... (3)

Modifikuotas Niutono metodas yra gaunamas, jei alpha_k yra apskaičiuojamas
koiu nors pasirinktu būdu.

2 pavyzdys ([2], 226 p.) Niutono metodu rasime funkcijos f(x,y) = x^2+2y^2 + exp(x+y) minimumą.

```
(%i49) f:x^2+2*y^2+exp(x+y);
```

(%o49) $\%e^{y+x}+2\,y^2+x^2$

```
(%i50) H:hessian(f,[x,y]);
```

(%o50) $\begin{bmatrix} \%e^{y+x}+2 & \%e^{y+x} \\ \%e^{y+x} & \%e^{y+x}+4 \end{bmatrix}$

```
(%i51) g:[diff(f,x),diff(f,y)];
```

(%o51) $[\%e^{y+x}+2\,x, \%e^{y+x}+4\,y]$

```
(%i52) v:[0,0]; for k thru 5 do
       (v:float(v-at(list_matrix_entries(g.invert(H)),[x=v[1],y=v[2]])),
       print([k, ev(f,[x=v[1],y=v[2]]),v]))$
(%o52) [0,0]
[1,0.7738880371228923,[-0.2857142857142857,-0.1428571428571428]]
[2,0.7722682973444378,[-0.3125890670815802,-0.1562945335407901]]
[3,0.772268227723419,[-0.3127667995630959,-0.1563833997815479]]
[4,0.7722682277234189,[-0.3127668071299921,-0.1563834035649961]]
[5,0.772268227723419,[-0.3127668071299922,-0.1563834035649961]]
```

Matome, kad nuo ketvirtosios iteracijų reikšmės nesikeičia. Todėl Niutono metodas konverguoja greičiau negu
gradientų metodas.

3 pavyzdys ([2], 244 p.) Niutono metodu rasime funkcijos f(x,y) = (x-1)^4+(2y-x)^2 minimumą.

```
(%i54) f:(x-1)^4+(2*y-x)^2;
```

(%o54) $(2\,y-x)^2+(x-1)^4$

```
(%i55) H:hessian(f,[x,y]);
```

(%o55) $\begin{bmatrix} 12\,(x-1)^2+2 & -4 \\ -4 & 8 \end{bmatrix}$

```
(%i56) g:[diff(f,x),diff(f,y)];
```

(%o56) $[4\,(x-1)^3-2\,(2\,y-x), 4\,(2\,y-x)]$

```
(%i57) v:[-1,2.5]; for k thru 45 do
       (v:float(v-at(list_matrix_entries(g.invert(H)),[x=v[1],y=v[2]])),
       if k>=40 then print([k, ev(f,[x=v[1],y=v[2]]),v]))$
(%o57) [-1,2.5]
```

$[40, 1.070243163576028\ 10^{-27}, [0.999999819128302, 0.499999909564151]]$
$[41, 2.114060655671518\ 10^{-28}, [0.9999998794188668, 0.4999999397094334]]$
$[42, 4.178284073113492\ 10^{-29}, [0.999999919601214, 0.499999959800607]]$
$[43, 8.284888060430139\ 10^{-30}, [0.999999946349761, 0.4999999731748805]]$
$[44, 1.645874876256895\ 10^{-30}, [0.9999999641821755, 0.4999999820910878]]$
$[45, 3.313520406578505\ 10^{-31}, [0.9999999760076708, 0.4999999880038354]]$

2 būdas:

```
(%i59) load(mnewton)$
```

```
(%i60) mnewton([diff(f,x),diff(f,y)],[x,y],[0,0]);
```

(%o60) $[[x=0.9999999818662911, y=0.4999999909331456]]$

```
(%i61) ev(f,%);
```

(%o61) $1.081300878523633\ 10^{-31}$

3 būdas:

```
(%i62) load(nopt)$
```

```
(%i63) minimize_nopt(f, []);
```

(%o63) $[0, [x=1, y=\frac{1}{2}]]$

Atsakymas: f_min = f(1, 1/2) = 0.

4 COBYLA ir nopt programos

COBYLA yra pavadinimo "Constrained Optimization BY Linear Approximations" sutrumpinimas.
Ji skaitiniais metodais sprendžia netiesinius optimizacijos uždavinius su apribojimais.
Plačiau apie COBYLA žr. Maxima dokumentacijoje.

1 pavyzdys. Raskite funkcijos f = (x1-1)^2+(x2-2.5)^2 minimumą, kai
-x1+2*x2-2<=0, x1+2*x2-6<=0, x1-2*x2-2<=0, x1>=0, x2>=0.

```
(%i64) kill(all)$
```

```
(%i1) load(fmin_cobyla)$
```

```
(%i2) f:(x1-1)^2+(x2-2.5)^2;
```

(%o2) $(x2-2.5)^2+(x1-1)^2$

```
(%i3) apr:[-x1+2*x2-2<=0,x1+2*x2-6<=0,x1-2*x2-2<=0,x1>=0,x2>=0];
```

(%o3) $[2\ x2-x1-2<=0, 2\ x2+x1-6<=0, -2\ x2+x1-2<=0, x1>=0, x2>=0]$

```
(%i4) load(fmin_cobyla);
```

(%o4)
C:/Program Files (x86)/Maxima-sbcl-5.35.1.2/share/maxima/5.35.1.2/share/cobyla/fmin_cobyla.mac

```
(%i5) fmin_cobyla(f,[x1,x2],[1,1],constraints=apr);
```

(%o5) $[[x1=1.400000557888516, x2=1.700000278944258], 0.8000000000003893, 48, 0]$

Tiksliai išspręsti galima su programa nopt(autorius -- A.Domarkas)

```
(%i6) load(nopt);
```

(%o6) *C:/Users/Aleksas/maxima/nopt.mac*

```
(%i7) minimize_nopt(f,apr);
```

(%o7) $[0.8000000000000002, [x1=\frac{7}{5}, x2=\frac{17}{10}]]$

2 pavyzdys. Raskite funkcijos
f = 3*x1^2+2*x1*x2+x1*x3+2.5*x2^2+2*x2*x3+2*x3^2-8*x1-3*x2-3*x3
minimumą, kai x1+x3=3, x2+x3=0.

Čia turime kvadratinio programavimo uždavinį. Panašiai galima išspręsti
daugelį kitų netiesinio programavimo uždavinių.

```
(%i8) kill(all)$
```

```
(%i1) load(fmin_cobyla)$
```

```
(%i2) f:3*x1^2+2*x1*x2+x1*x3+2.5*x2^2+2*x2*x3+2*x3^2-8*x1-3*x2-3*x3$
```

```
(%i3) apr:[x1+x3=3, x2+x3=0];
(%o3) [x3+x1=3,x3+x2=0]
```

```
(%i4) fmin_cobyla(f,[x1,x2,x3],[1,1,1],constraints=apr);
(%o4) [[x1=2.000000566054223,x2=-0.9999994339457774,x3=0.9999994339457774],-
3.499999999997917,81,0]
```

Tiksliai išspręsti galima su programa nopt(autorius -- A.Domarkas)

```
(%i5) load(nopt);
(%o5) C:/Users/Aleksas/maxima/nopt.mac
```

```
(%i6) minimize_nopt(f,apr);
(%o6) [-3.5,[x1=2,x2=-1,x3=1]]
```

3 pavyzdys. Raskite funkcijos f = 2*x1+x1^2/2-6*x2-x1*x2+x2^2 minimumą
ir maksimumą, kai 3*x1+x2<=25, -x1+2*x2<=10, x1+2*x2<=15, x1>=0, x2>=0.

Čia turime kvadratinio programavimo uždavinį. Panašiai galima išspręsti
daugelį kitų netiesinio programavimo uždavinių.

```
(%i7) load(fmin_cobyla)$
```

```
(%i8) f:2*x1+x1^2/2-6*x2-x1*x2+x2^2;
```

(%o8) $x2^2 - x1\,x2 - 6\,x2 + \frac{x1^2}{2} + 2\,x1$

```
(%i9) apr:[3*x1+x2<=25, -x1+2*x2<=10, x1+2*x2<=15, x1>=0, x2>=0];
(%o9) [x2+3 x1<=25,2 x2-x1<=10,2 x2+x1<=15,x1>=0,x2>=0]
```

```
(%i10) fmin_cobyla(f,[x1,x2],[1,1],constraints=apr);
(%o10) [[x1=1.999999356058475,x2=4.000000556730018],-9.999999999999122,71,0]
```

Maksimumo radimui minimizuojame -f(x):

```
(%i11) fmin_cobyla(-f,[x1,x2],[1,1],constraints=apr);
(%o11) [[x1=8.333333333333332,x2=0.0],-51.38888888888888,32,0]
```

Tiksliai išspręsti galima su programa nopt(autorius -- A.Domarkas)

```
(%i12) load(nopt);
(%o12) C:/Users/Aleksas/maxima/nopt.mac
```

```
(%i13) minimize_nopt(f,apr);
(%o13) [-10,[x1=2,x2=4]]
```

```
(%i14) maximize_nopt(f,apr);
```

(%o14) $\left[\frac{925}{18}, \left[x1=\frac{25}{3}, x2=0\right]\right]$

```
(%i15) float(%), numer;
(%o15) [51.38888888888889,[x1=8.333333333333334,x2=0.0]]
```

4 pavyzdys. Raskite funkcijos f = x^2+2*y^2+3*z^2 minimumą
ir maksimumą, kai z^2+y^2+x^2=1, 3*z+2*y+x=0.

Čia turime kvadratinio programavimo uždavinį. Panašiai galima išspręsti
daugelį kitų netiesinio programavimo uždavinių.

```
(%i16) load(fmin_cobyla)$
```

(%i17) f:x^2+2*y^2+3*z^2;

(%o17) $3\,z^2+2\,y^2+x^2$

(%i18) apr:[x^2+y^2+z^2=1,x+2*y+3*z=0];

(%o18) $[z^2+y^2+x^2=1,3\,z+2\,y+x=0]$

(%i19) fmin_cobyla(f,[x,y,z],[1,1,1],constraints=apr);

(%o19) $[[x=0.958569047545116,y=-0.232586703458568,z=-0.16446521354266],$
$1.108194187559897,82,0]$

Maksimumo radimui minimizuojame -f(x):

(%i20) fmin_cobyla(-f,[x,y,z],[1,1,1],constraints=apr);

(%o20) $[[x=0.09857569990448392,y=0.8125197977661545,z=-0.5745384318122644],-$
$2.320377241023452,88,0]$

Tiksliai išspręsti galima su programa nopt(autorius -- A.Domarkas)

(%i21) load(nopt);

(%o21) $C:/Users/Aleksas/maxima/nopt.mac$

(%i22) minimize_nopt(f,apr),radcan,factor;

(%o22) $$[-\frac{3\left(\sqrt{2}-4\right)}{7},[x=\frac{\sqrt{9\sqrt{2}+13}}{2\sqrt{7}},y=-\frac{\left(3\sqrt{2}-4\right)\sqrt{9\sqrt{2}+13}}{2\sqrt{7}},z=\frac{\sqrt{9\sqrt{2}+13}\left(2^{3/2}-3\right)}{2\sqrt{7}}],[x=-\frac{\sqrt{9\sqrt{2}+13}}{2\sqrt{7}},y=\frac{\left(3\sqrt{2}-4\right)\sqrt{9\sqrt{2}+13}}{2\sqrt{7}},z=-\frac{\sqrt{9\sqrt{2}+13}\left(2^{3/2}-3\right)}{2\sqrt{7}}]]$$

(%i23) float(%), numer;

(%o23) $[1.108194187554388,[x=0.9585689121467528,y=-0.2325878194944743,z=-$
$0.1644644243859345],[x=-0.9585689121467528,y=0.2325878194944743,z=0.1644644243859345]]$

Sprendinių prastinimui pritaikomos komandos radcan ir factor.

(%i24) spr:maximize_nopt(f,apr),radcan,factor;

(%o24) $$[\frac{3\left(\sqrt{2}+4\right)}{7},[x=\frac{\sqrt{13-9\sqrt{2}}}{2\sqrt{7}},y=\frac{\sqrt{13-9\sqrt{2}}\left(3\sqrt{2}+4\right)}{2\sqrt{7}},z=-\frac{\sqrt{13-9\sqrt{2}}\left(2^{3/2}+3\right)}{2\sqrt{7}}],[x=-\frac{\sqrt{13-9\sqrt{2}}}{2\sqrt{7}},y=-\frac{\sqrt{13-9\sqrt{2}}\left(3\sqrt{2}+4\right)}{2\sqrt{7}},z=\frac{\sqrt{13-9\sqrt{2}}\left(2^{3/2}+3\right)}{2\sqrt{7}}]]$$

(%i25) float(%), numer;

(%o25) $[2.320377241017041,[x=0.09857519585179322,y=0.8125199200687432,z=-$
$0.5745383453297598],[x=-0.09857519585179322,y=-0.8125199200687432,z=0.5745383453297598$
$]]$

Matome, kad programa nopt randa du tikslius sprendinius.

Literatūra:
[1] V.Čiočys, R.Jasilionis, Matematinis programavimas, Vilnius, Mokslas, 1990.
[2] R.Čiegis, V.Būda, Skaičiuojamoji matematika, Vilnius, TEV, 1997.