Optimizavimo metodai

Užduoties „Netiesinis programavimas“ ataskaita

Autorius: Vilius Minkevičius, Programų Sistemos 4 kursas 2 grupė

Darbo eiga

1. Apribojimų, tikslo funkcijos bei baudos funkcijos formulavimas,
2. Tyrinėjimas, kokį poveikį tikslo funkcijos reikšmėms turi baudos daugiklis,
3. Deformuojamo simplekso metodo pritaikymas tikslo funkcijai,
4. Baudos funkcijos minimizavimas,
5. Rezultatai.

Užduoties formulavimas

1. Tikslo funkcija: **f(X) = - xyz**, kur x, y ir z yra dėžės kraštinės.
2. Lygybinis apribojimas vienetiniui paviršiaus plotui: **g(X) = 0.5 – xy – yz – xz**
3. Nelygybinis apribojimas teigiamui atstumui: **h(X) = (|x| + |y| + |z|) – (x + y + z)**
   Funkcijos reikšmė lygi nuliui, jei visos kraštinės teigiamos, lygi teigiamam skaičiui, jei bent viena kraštinė neigiama.
4. Baudos funkcija: **b(X) =** $\mathbf{g(X)^2 + \max(0, h(X))^2}$
5. Optimizuojama baudos funkcija: **B(X) =** $\mathbf{f(X) + \frac{1}{r} b(X)}$, kur $\frac{1}{r}$ yra baudos daugiklis.

Funkcijų reikšmės įvairiuose taškuose

Taškas	f(X)	g(X)	h(X)
X0 (0, 0, 0)	0	0.5	0
X1 (1, 1, 1)	-1	-2.5	0
Xm (0.2, 0.3, 0.9)	-0.054	-0.01	0
(-0.5, 0.5, 0.2)	0.05	0.75	1
X*(0.408, 0.408, 0.408)	-0.068042	0	0

Lentelė 1 - funkcijų reikšmių lentelė. Paskutinis taškas - skaičiavimų metu rastas minimumo taškas.

Baudos daugiklio poveikis tikslo funkcijos reikšmėms

Pastebėjau, kad baudos daugiklio poveikis priklauso nuo taško atstumo iki minimumo ir nuo to, ar taškas yra leistinoje srityje. Jei taškas yra leistinoje srityje, tai baudos funkcija yra arti arba lygi nuliui ir funkcijos reikšmę nulemia pradinė tikslo funkcija f(X). Kitais atvejais, kuo baudos daugiklis didesnis, tuo tikslo funkcijos reikšmė didesnė.

\ Daugiklis X \	0.1	1	10	100
X0 (0, 0, 0)	0.025	0.25	2.5	25
X1 (1, 1, 1)	-0.375	5.25	61.5	624
Xm (0.2, 0.3, 0.9)	-0.05399	-0.0539	-0.05300	-0.04400
X*(0.408, 0.408, 0.408)	-0.06792	-0.067	-0.06791	-0.06788

Lentelė 2 - funkcijų reikšmių lentelė pagal baudos daugiklį ir tašką. Apatinėje eilutėje galima pastebėti mažiausią poveikį tikslo funkcijai. Kitų eilučių taškai yra ne leistinoje srityje.

Apie skaičiavimą

Optimizavimo uždavinių serijai spręsti naudojau antrame laboratoriniame darbe aprašytą deformuojamo simplekso metodą. Algoritmą teko pritaikyti tikslo funkcijai su 3 argumentais, anksčiau ji naudota optimizuoti tikslo funkcijoms su 2 argumentams (galima perrašyti algoritmą, kad užtektų nurodyti argumentų skaičių ir nereikėtų kaskart keisti programos teksto).

Kiekvienam sekos uždaviniui priskyriau baudos daugiklį pagal nykstamą geometrinę progresiją, pavyzdžiui: 2, 1, 0.5, 0.25, 0.125 ir t.t. Iš sekos imama **r** reikšmė, o daugiklis gaunamas per $\frac{1}{r}$. Paskutinis sekos narys nurodomas pagal mažiausią reikšmę, kokią gali pasiekti r.

Kad skaičiavimai trumpiau užtruktų, padariau, kad simplekso kraštinės ilgis priklausytų nuo baudos daugiklio. Kuo mažesnis daugiklis, tuo mažesnė kraštinė. Sprendimą pagrindžiu tuo, kad sekoje gretimų optimizavimo sprendinių minimumai yra ganėtinai arti vienas kito, tad galima optimizuoti mažesniu žingsniu.

Deformuojamo simplekso algoritmą pakeičiau, kad siekiamas tikslumas priklausytų ne tik nuo iš anksto apibrėžtos mažiausio kraštinės ilgio, bet ir nuo baudos daugiklio. Mažesnis baudos daugiklis – siekiama didesnio tikslumo.

Rezultatai

1. Skaičiavimams pasirinkti šie taškai X_0 (0, 0, 0), X_1 (1, 1, 1), ir X_m (0.2, 0.3, 0.9).
2. Rastas minimumo taškas X* (0.40825, 0.40825, 0.40825), kur f(X) = -0.068042.
3. Skaičiavimai užtruko nuo 190 iteracijų iki 202. Trukmė nežymiai mažesnė pradedant skaičiavimus arčiau minimumo taško.
4. Galima pastebėti, kad uždavinių sekos narių minimumo taškai nežymiai priklauso nuo skaičiavimui pasirinkto pradinio taško. Tikslioms taško koordinatėms didžiausią poveikį turi baudos daugiklis. Skirtumus, tarp rastų sprendinių pagal pradinį tašką, galima paaiškinti paklaidos buvimu.

Nuo pradinio taško (0, 0, 0)

Uždavinio nr.	Uždavinio minimumo taškas	Min. reikšmė	Baudos daugiklis (1 / r)	Def. simplekso iteracijų skaičius
1	(0.499271, 0.502034, 0.497430)	-0.093748	0.5	54
2	(0.446068, 0.425039, 0.431475)	-0.074665	1.666667	39
3	(0.415112, 0.411154, 0.421472)	-0.069945	5.555556	34
4	(0.410666, 0.404019, 0.416749)	-0.068598	18.51852	21
5	(0.409949, 0.402181, 0.414656)	-0.068202	61.7284	20
6	(0.409620, 0.401474, 0.414291)	-0.068083	205.7613	17
7	(0.409503, 0.401315, 0.414159)	-0.068048	685.8711	17

Iš viso 202 iteracijos.

Nuo pradinio taško (1, 1, 1)

Uždavinio nr.	**Uždavinio minimumo taškas**	**Min. reikšmė**	**Baudos daugiklis (1 / r)**	**Def. simplekso iteracijų skaičius**
1	(0.499869, 0.505394, 0.496555)	-0.093744	0.5	42
2	(0.443471, 0.428199, 0.429887)	-0.074675	1.666667	34
3	(0.417352, 0.416930, 0.412177)	-0.069946	5.555556	43
4	(0.414330, 0.412189, 0.404968)	-0.068602	18.51852	23
5	(0.412981, 0.410803, 0.403027)	-0.068205	61.7284	23
6	(0.412576, 0.410494, 0.402304)	-0.068086	205.7613	14
7	(0.412441, 0.410397, 0.402122)	-0.068050	685.8711	14

Iš viso 193 iteracijos.

Nuo pradinio taško (0.2, 0.3, 0.9)

Uždavinio nr.	**Uždavinio minimumo taškas**	**Min. reikšme**	**Paklaida**	**Baudos daugiklis (1 / r)**	**Def. simplekso iteracijų skaičius**
1	(0.496271, 0.498328, 0.503855)	-0.093745	0.025703	0.5	42
2	(0.430887, 0.443214, 0.428714)	-0.074676	0.006634	1.666667	27
3	(0.419778, 0.409914, 0.417436)	-0.069945	0.001903	5.555556	31
4	(0.415073, 0.403490, 0.412926)	-0.068599	0.000557	18.51852	19
5	(0.410287, 0.407287, 0.409228)	-0.068210	0.000168	61.7284	36
6	(0.409887, 0.406529, 0.408948)	-0.068091	4.9E-05	205.7613	19
7	(0.409778, 0.406306, 0.408830)	-0.068056	1.4E-05	685.8711	16

Iš viso 190 iteracijų. Paklaida lygi uždavinio sprendinio ir rasto minimumo (-0.068042) skirtumo moduliui.

Išvados

1. Priklausomybė tarp optimizavimo uždavinių sprendinių tikslumo ir baudos daugiklio yra artima tiesinei. Padidinus daugiklį 10 kartų, paklaida sumažėja panašų kiekį kartų.
2. Naudojant skirtingus pradinius taškus gauti mažai besiskiriantys sprendiniai. Tai parodo, kad sprendinys labiausiai priklauso nuo baudos daugiklio.
3. Skaičiavimai konvergavo visais nagrinėtais atvejais. Šis metodas yra patikimesnis nei ankstesniame darbe nagrinėti optimizavimo be apribojimų metodai.

Priedas – kodo fragmentai

```
# Tikslo funkcija
f = @(X) - X(1) * X(2) * X(3);
# Lygybinis apribojimas pavirsiaus plotui
g = @(X) 0.5 - X(1)*X(2) - X(2)*X(3) - X(1)*X(3);
# Nelygybinis apribojimas krastiniu teigiamumui
h = @(X) abs(X(1)) + abs(X(2)) + abs(X(3)) - X(1) - X(2) - X(3);

# Baudos funkcija:
b = @(X) g(X)^2 + max([0, h(X)])^2;
# Apribojimus apimanti tikslo funkcija:
B = @(X, r) f(X) + (1 / r) * b(X);
```

Pav. 1 - programos tekste apibrėžtos funkcijos: tikslo, apribojimų ir baudos.

```
while r > min_r
  # Is naujo nustatomi deformuojojo simplekso parametrai:
  Xi = X0;
  Iteraciju_kiekis = 0;
  while alfa > tikslumas * r^0.8 #&& Iteraciju_kiekis < max_iter
    ## Simplekso sudarymas:
    # d1 ir d2 - atstumo koeficientai, naudojami apskaiciuojant virsuniu koorc
    d1 = (((n+1)^0.5 + n - 1) / (n * 2^0.5)) * alfa;
    d2 = (((n+1)^0.5 - 1) / (n * 2^0.5)) * alfa;
    # Likusiu simplekso tasku koordinaciu apskaiciavimas:
    X1 = [X0(1) + d2, X0(2) + d1, X0(3) + d1];
    X2 = [X0(1) + d1, X0(2) + d2, X0(3) + d1];
    X3 = [X0(1) + d1, X0(2) + d1, X0(3) + d2];
```

Pav. 2 - algoritmo pagrindinio ciklo pradžia. Vidinis while ciklas - deformuojamo simplekso metodas. Optimizavimo uždaviniai vykdomi tol, kol r yra pakankamai didelis (arba baudos daugiklis pakankamai mažas). Simplekso metodas vykdomas tol, kol kraštinės ilgis didesnis nei siekiamas tikslumas.

```
  printf("%d;(%f, %f, %f);%f;%f;%d \n", uzd_nr, X_pradinis(1), X_pradinis(2),
  X_pradinis(3), simpleksas(1, 1), r, Iteraciju_kiekis);
  r = r * r_daugiklis;
  alfa = alfa_pradinis * min(r^1.5, 1); # Padeda užtikrinti kad antras ir toli
  sprendiniai = [sprendiniai; X_pradinis];
endwhile
```

Pav. 3 - veiksmai optimizavimo uždavinio pabaigoje. Baudos daugiklis keičiamas, nustatomas naujas (mažesnis) pradinis simplekso kraštinės ilgis ir išsaugomas rastas sprendinys. Spausdinimo funkcija pavaizduoja optimizavimo uždavinio informaciją tokiu formatu, kurį patogu paversti lentele.

Priedas – visas kodas

```
# Tikslo funkcija
f = @(X) - X(1) * X(2) * X(3);
# Lygybinis apribojimas pavirsiaus plotui
g = @(X) 0.5 - X(1)*X(2) - X(2)*X(3) - X(1)*X(3);
# Nelygybinis apribojimas krastiniu teigiamumui
h = @(X) abs(X(1)) + abs(X(2)) + abs(X(3)) - X(1) - X(2) - X(3);

# Baudos funkcija:
b = @(X) g(X)^2 + max([0, h(X)])^2;
# Apribojimus apimanti tikslo funkcija:
B = @(X, r) f(X) + (1 / r) * b(X);

# Uzdaviniu sekos parametrai - baudos daugiklis:
r = 2;
r_daugiklis = 0.3; # Baudos daugiklio daugiklis, 0 < r_daugiklis < 1
min_r = 0.001;

# Pradiniai taskai - pasirenkame viena:
X_0 = [0 0 0];
X_1 = [1 1 1];
X_m = [0.2 0.3 0.9];
# Pasirinktas pradinis taskas:
X0 = X_m

# Optimizavimas deformuojamuoju simpleksu:
tikslumas = 0.001; # Maksimalus imanomas tikslumas. Realus tikslumas priklauso :
alfa_pradinis = 0.3;
o = 1; # Simplekso deformacijos koeficientas.
ispletimo_koef = 2; # Ispletimo koeficientas.
niu_suspaudimas = -0.5; # Suspaudimo koeficientas.
beta_suspaudimas = 0.5;
n = 3; # Simpleksas yra dvimateje erdveje.
K = 2; # Simplekso mazinimo daugiklis. 2 -> sumazeja simplekso krastine perpus.

# Pagalbiniai kintamieji
taskai = [X_pradinis];
sprendiniai = [];

Iteraciju_kiekis = 0;
Iteraciju_kiekis2 = 0;
uzd_nr = 0;
max_iter = 100;
```

```
alfa = alfa_pradinis;
printf("Uždavinio nr; Minimumo taškas; Min. reikšme; Baudos daliklis; Iteraciju
while r > min_r
  # Is naujo nustatomi deformuojojo simplekso parametrai:
  Xi = X0;
  Iteraciju_kiekis = 0;
  while alfa > tikslumas * r^0.8 #&& Iteraciju_kiekis < max_iter
    ## Simplekso sudarymas:
    # d1 ir d2 - atstumo koeficientai, naudojami apskaiciuojant virsuniu koordi
    d1 = (((n+1)^0.5 + n - 1) / (n * 2^0.5)) * alfa;
    d2 = (((n+1)^0.5 - 1) / (n * 2^0.5)) * alfa;
    # Likusiu simplekso tasku koordinaciu apskaiciavimas:
    X1 = [X0(1) + d2, X0(2) + d1, X0(3) + d1];
    X2 = [X0(1) + d1, X0(2) + d2, X0(3) + d1];
    X3 = [X0(1) + d1, X0(2) + d1, X0(3) + d2];
    FX0 = B(X0, r);
    FX1 = B(X1, r);
    FX2 = B(X2, r);
    FX3 = B(X3, r);
    taskai = [taskai; X1; X2; X3];
    simpleksas = [FX0, X0; FX1, X1; FX2, X2; FX3, X3];

    zingsnis_sekmingas = true;
    while zingsnis_sekmingas == true #&& Iteraciju_kiekis < max_iter
      simpleksas = sortrows(simpleksas);
      # Isrikiavus simpleksas(1, :) - geriausias taskas, simpleksas(2, :) - ant
      # Centro apskaiciavimas
      Xc = (1 / n) * (simpleksas(1, 2:4) + simpleksas(2, 2:4) + simpleksas(3, 2
      Xnaujas = simpleksas(4, 2:4) + 2 * (Xc - simpleksas(4, 2:4)); # Blogiausi
      FXnaujas = B(Xnaujas, r);

      # Nustatomas deformacijos koeficientas
      if FXnaujas > simpleksas(1, 1) && FXnaujas < simpleksas(2, 1) # Naujas be
        O = 1;
      elseif FXnaujas < simpleksas(1, 1) # Naujas taskas itin geras
        O = ispletimo_koef;
      elseif FXnaujas > simpleksas(4, 1) # Naujas taskas yra prasciausias
        O = niu_suspaudimas;
        zingsnis_sekmingas = false;
      elseif FXnaujas > simpleksas(2, 1) && FXnaujas < simpleksas(4, 1) # Nauja
        O = beta_suspaudimas;
      endif
```

```
      if zingsnis_sekmingas == true
        # Pridedame nauja taska vietoje prasciausio bei atliekame deformacija.
        Xnaujas = simpleksas(4, 2:4) + (1 + O) * (Xc - simpleksas(4, 2:4));
        FXnaujas = B(Xnaujas, r);
        simpleksas(4, :) = [FXnaujas, Xnaujas];
        taskai = [taskai; Xnaujas];
      else # Zingsnis buvo nesekmingas. Sudaromas naujas, mažesnis simpleksas.
        alfa = alfa / K;
        X0 = simpleksas(1, 2:4);
      endif
      Iteraciju_kiekis += 1;
    endwhile
  endwhile
  uzd_nr += 1;
  Iteraciju_kiekis2 += Iteraciju_kiekis;
  simpleksas = sortrows(simpleksas);
  X_pradinis = simpleksas(1, 2:4);
  #printf("Uzd nr %d, spr X (%f, %f, %f), min %f, r %f, iter %d \n",
  # uzd_nr, X_pradinis(1), X_pradinis(2), X_pradinis(3), simpleksas(1, 1),
  # r, Iteraciju_kiekis);
  printf("%d;(%f, %f, %f);%f;%f;%d \n", uzd_nr, X_pradinis(1), X_pradinis(2),
  X_pradinis(3), simpleksas(1, 1), r, Iteraciju_kiekis);
  r = r * r_daugiklis;
  alfa = alfa_pradinis * min(r^1.5, 1); # Padeda užtikrinti kad antras ir toli
  sprendiniai = [sprendiniai; X_pradinis];
endwhile

## Rezultatai:
minimumo_taskas = simpleksas(1, 2:4)
Turis = simpleksas(1, 1)
Iteraciju_kiekis2
```