VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Laboratorinis darbas 3

Netiesinis programavimas

Nikita Gainulin

VILNIUS 2024

Turinys

1 Įvadas

Savo ankstesniame laboratoriniame darbe gilinausi į optimizavimą be apribojimų. Kaip jau nustačiau, jo algoritmai be jokių apribojimų nagrinėja visą funkciją ir gali rasti minimumo tašką beveik bet kurioje jos vietoje. Šiame laboratoriniame darbe gilinsiuosi į netiesinį optimizavimą. Vėl susidursime su tam tikrais apribojimais, tačiau šį kartą jie bus pagrįsti tikra, realia problema, kitaip nei vienmačio optimizavimo atveju, kai apribojimai kyla iš pačių algoritmų intervalų pavidalu. Ir kaip ir ankstesnio laboratorinio darbo optimizavimo tipe, toliau veiksime daugiamatėje erdvėje.

2 Nagrinėjama problema

Kaip įprasta, apibrėžkime šiuo metu nagrinėjamą problemą:

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

Pati problema, palyginti su ankstesniu laboratoriniu darbu, nepasikeitė, tačiau optimizavimo, kitaip tariant, sprendimo būdas skirsis. Prieš tęsdami toliau, pažvelkime į išvadą, kurią gavau optimizavęs šį uždavinį be apribojimų: **dėžė turi būti kubas, kurio viena į kitą atgręžtų sienų plotų sumos, kurių yra 3, lygūs $\frac{1}{3}$ paviršiaus vieneto ploto, t.y $2ab = 2bc = 2ac = \frac{1}{3}$, kur a, b, c - atitinkamai ilgis, plotis ir aukštis.**

Nors šis atsakymas ir yra teisingas, ar jis neskambėtų šiek tiek painiai, jei iš tikrųjų bandytume spręsti šią problemą realiame pasaulyje? Ar negalėtume rasti glaustesnio atsakymo, kuriame būtų tiesiai nurodytas konkretus dėžės ilgis, plotis ir aukštis? Netiesinis optimizavimas puikiai tinka tokio tipo problemoms spręsti.

Pirmiausia turėsime iš naujo apibrėžti tikslo funkciją, jei norime gauti daug glaustesnį atsakymą. Tačiau daug transformacijų atlikti nereikės, nes tūrio formulėje ($V = a \cdot b \cdot c$) yra viskas, ko mums reikia:

$$f(X) = -(x_0 \cdot x_1 \cdot x_2) \tag{1}$$

kur x_0 - ilgis a, x_1 - plotis b, x_2 - dėžės aukštis c. Svarbu nepamiršti minuso ženklo, nes, norėdami rasti didžiausią tūrį, turėsime funkciją minimizuoti.

Dabar, kai turime tikslo funkciją, gali kilti klausimas, kodėl pasirinkome būtent netiesinį optimizavimą? Ar negalime rasti dėžės parametrų bet kuriuo kitu optimizavimo tipu? Tiesa, tikrai yra keletas sprendimo būdų, kaip gauti norimus rezultatus, tačiau, kaip minėjau anksčiau, netiesinis optimizavimas puikiai tinka dėl to, kad pačiame uždavinyje jau yra tam tikri apribojimai, kurių turime laikytis. Pažvelkime į juos.

Pirmiausia - lygybinis apribojimas. Pagal užduotį norime rasti didžiausią reikšmę dėžutės ploto vienete, todėl galime sudaryti tokią funkciją $g_i(X)$, kuri užtikrintų, kad mūsų algoritmo pasirinktos reikšmės neišeitų iš ribų:

$$g_i(X) = 2 \cdot (x_0x_1 + x_0x_2 + x_1x_2) - 1 \tag{2}$$

Toliau - nelygybės apribojimas. Kadangi realiame gyvenime dėžutės ilgis, plotis ir aukštis negali būti neigiami, turime tokią nelygybę:

$$x_0, x_1, x_2 \geq 0$$

Pagalvokime apie tai, kaip galime paversti šią nelygybę info funkcija. Pagal bendrąjį standartą žinome, kad funkcija bus tokio pavidalo - $h_i(x) \leq 0$, todėl belieka tik padauginti duotą nelygybę iš

-1, tam, kad gautume nelygybę $-x_0, -x_1, -x_2 \leq 0$, kuri funkcijos pavidalu atrodo taip:

$$h_i(X) = [-x_0, -x_1, -x_2] \tag{3}$$

Čia Python kalba sukūriau savo pasirinktinę klasę, skirtą šios problemos tikslo (1) ir apribojimų (2), (3) funkcijoms:

```
class Optimization:
    counter = 0

    def __init__(self):
        pass

    def f(self, x):
        Optimization.counter += 1
        return -(x[0]*x[1]*x[2])

    def g(self, x):
        return 2*(x[0]*x[1]+x[0]*x[2]+x[1]*x[2])-1

    def h(self, x):
        return [-x[0], -x[1], -x[2]]

    def reset(self):
        Optimization.counter = 0
```

3 Netiesinis optimizavimas ir problemos sprendimo algoritmas

3.1 Netiesinis optimizavimas

Pirmiausia norėčiau dar kartą trumpai paminėti netiesinį optimizavimą. Kaip minėta anksčiau, tai viena iš optimizavimo rūšių, skirta dirbti su uždavinyje pateiktais arba iš jo išvestais apribojimais. Kadangi pačiame termino pavadinime yra žodis „netiesinis“, svarbu nepamiršti, kad tikslo arba bent viena iš apribojimų funkcijų turi būti netiesinė, t. y. būti kokios nors kitos formos nei $f(x) = ax + b$. Mūsų pavyzdyje tiek tikslo funkcija (1), tiek lygybės apribojimo funkcija (2) yra netiesinės dėl kelių nežinomų kintamųjų daugybos, todėl pasirinktas netiesinis optimizavimas.

Dabar, kai jau turime tikslo ir apribojimų funkcijas, yra keletas optimizavimo metodų.

3.2 Baudos metodas

Šiame laboratoriniame darbe optimizavimo uždavinį spręsime taikydami baudos metodą. Pradėsime nuo pradinio taško X_1, kuris yra apribojimais apibrėžtoje įgyvendinamoje srityje arba netoli jos. Naudodami neapriboto optimizavimo metodą (mūsų atveju Nelderio-Medo simplekso metodą), iteratyviai ieškosime tikslo funkcijos lokalaus minimumo. Tačiau tiesiogiai taikant neribotąjį metodą galima gauti sprendinius, kurie pažeidžia apribojimus.

Siekdami išspręsti šią problemą, įvedame baudos funkciją, kuri modifikuoja pradinę tikslo funkciją, pridedant narius, baudžiančius už apribojimų pažeidimus. Šios baudos užtikrina, kad optimizavimo procesas palaipsniui nukreiptų paiešką į galimą sprendinį, kuris tenkina visus apribojimus. Mažėjant

baudos parametrui, metodas konverguoja į minimumo tašką, kuris yra problemos apribojimų apibrėžtose ribose.

Štai kaip atrodo mano Python įgyvendinta kvadratinės baudos funkcija:

```
def penalty(obj: Optimization, x, r):
    bX = 0
    bX += obj.g(x)**2

    for h in obj.h(x):
        bX += max(0, h)**2

    return obj.f(x) + (1/r)*bX
```

Pirma skaičiuojamas lygybės apribojimo pažeidimas (jei toks yra), o vėliau nelygybės apribojimų pažeidimai tikrinami su sąlyga, kad pažeidimas atsiranda tik tada, kai apribojimas nėra tenkinamas. Grąžinama tikslo funkcijos reikšmė, pakoreguota atsižvelgiant į apribojimų pažeidimus, o baudos svoris kontroliuojamas parametru r, taip vadinamu baudos koeficientu.

3.3 Algoritmo kodas

Dabar, kai žinome pagrindinę šio netiesinio optimizavimo metodo darbo eigą, pažvelkime, kaip aš įgyvendinau visą algoritmą:

```
def minPen(obj: Optimization, x0, initR=10, eps=1e-6):
    # initR - baudos koeficientas
    results = {
        "points": [],
        "funcVals": [],
        "niter": [],
        "neval": [],
        "rVals": []
    }

    class PenaltyObjectiveFunction:
        def f(self, x):
            return penalty(obj, x, r)

    penaltyObjFunc = PenaltyObjectiveFunction()
    currX = x0
    r = initR

    while True:
        point, funcVal, iterations = simplex(penaltyObjFunc, currX)

        results["points"].append(point)
        results["funcVals"].append(funcVal)
        results["niter"].append(iterations)
        results["neval"].append(Optimization.counter)
        results["rVals"].append(r)

        if sum([obj.g(point)**2]+[max(0, h)**2 for h in obj.h(point)])<eps:
            break

        if np.linalg.norm(np.array(point)-np.array(currX))<eps:
            break

        r *= 0.25
```

```
        obj.reset()
        currX = point
```

x_0 - pradinis taškas, initR - baudos koeficientas, eps - tikslumo reikšmė ε.

Prieš pradėdamas pagrindinį ciklą, turėjau sukurti papildomą apgaubiančiąją klasę aplink baudų funkciją, nes mano simplekso algoritmo įgyvendinimas iš esmės nepasikeitė nuo to, kaip jis atrodė antrajame laboratoriniame darbe - jis vis dar kviečia tikslo funkciją per $f(X)$ metodą, kuris, nors ir yra baudos funkcijoje, pats savaime nėra pakankamas, kaip matyti iš jos įgyvendinimo.

Pagrindiniame cikle iteracija po iteracijos konverguojame prie mažiausio taško, optimizuodami baudos funkcijos rezultatus simplekso algoritmu ir perpus sumažindami baudos koeficientą. Baudos koeficientas r yra labai svarbus metodo dalis, nes jis reguliuoja, kaip stipriai tikslo funkcija yra „baudžiama“ už apribojimų pažeidimus. Didėjant r reikšmei (arba mažėjant baudos narių svoriui), optimizavimo procesas tampa labiau orientuotas į tikslo funkcijos minimizavimą, tačiau gali ignoruoti apribojimus. Priešingai, mažėjant r, apribojimų pažeidimai yra stipriau baudžiami, o optimizavimo procesas daugiau dėmesio skiria sprendinių paieškai apribojimų apibrėžtoje srityje.

Galiausiai, algoritmas turės kažkaip sustoti, ir už tai atsakingi šie du sąlyginiai teiginiai:

```
    if sum([obj.g(point)**2]+[max(0, h)**2 for h in obj.h(point)])<eps:
        break

    if np.linalg.norm(np.array(point)-np.array(currX))<eps:
        break
```

Pirma sąlyga tikrina, ar sprendinys tenkina visus apribojimus. Lygybiniai apribojimai $g_i(x)$ ir nelygybiniai apribojimai $h_i(x)$ yra vertinami pagal tai, kiek stipriai jie pažeidžiami. Jei visų apribojimų pažeidimų kvadratų suma tampa mažesnė už ε, laikoma, kad sprendinys praktiškai atitinka visus apribojimus.

Antra sąlyga tikrina sprendinių pokytį tarp dabartinės ir ankstesnės iteracijos. Jei dviejų taškų atstumas yra mažesnis už ε, laikoma, kad algoritmas pasiekė minimumo tašką - sprendinys daugiau reikšmingai nesikeičia.

4 Rezultatai ir jų analyzė

Šiame skyriuje stebėsime algoritmo rezultatus ir bandysime rasti bei paaiškinti jų prasmę.

Išnagrinėsime šiuos tris pradinius taškus: (0, 0, 0), (1, 1, 1), (0.1, 0.5, 0.7). Be to, pabandysime atkreipti dėmesį į rezultatų skirtumą, jei padauginsime baudos koeficientą r iš skirtingų reikšmių - 0.5, 0.25 ir 0.75. Pats baudos koeficientas r išliks pastovus - 10.

4.1 Minimumo taškai

Pažvelkime į šias minimumo taškų lenteles:

		Pradiniai taškai x_0		
		(0, 0, 0)	(1, 1, 1)	(0.1, 0.5, 0.7)
$r = \frac{r}{4}$	10	[0.525501, 0.525467, 0.525496]	[0.525469, 0.525508, 0.52551]	[0.525488, 0.52545, 0.525507]
	2.5	[0.435123, 0.435088, 0.435134]	[0.435093, 0.435145, 0.435103]	[0.435128, 0.435095, 0.43515]
	0.625	[0.414829, 0.414786, 0.414815]	[0.41477, 0.414815, 0.414847]	[0.414837, 0.414787, 0.414797]
	0.15625	[0.409867, 0.409881, 0.409887]	[0.409909, 0.409852, 0.409872]	[0.40988, 0.409895, 0.40986]
	0.0390625	[0.408683, 0.408631, 0.408651]	[0.408682, 0.408618, 0.408667]	[0.408632, 0.408647, 0.408686]
	0.009765625	[0.408376, 0.408341, 0.408334]	[0.40835, 0.408373, 0.408326]	[0.408341, 0.408331, 0.408379]

1 lentelė: Minimumo taškai, kai baudos koeficientas r dalijamas iš 4

		Pradiniai taškai x_0		
		(0, 0, 0)	(1, 1, 1)	(0.1, 0.5, 0.7)
$r = \frac{r}{2}$	10	[0.525502, 0.525468, 0.525496]	[0.525469, 0.525508, 0.52551]	[0.525488, 0.52545, 0.525507]
	5	[0.463604, 0.463687, 0.463645]	[0.463669, 0.463595, 0.46365]	[0.463633, 0.463635, 0.46363]
	2.5	[0.435049, 0.435148, 0.435151]	[0.435141, 0.435088, 0.435125]	[0.43513, 0.435102, 0.435142]
	1.25	[0.421432, 0.421497, 0.421495]	[0.421496, 0.421443, 0.421489]	[0.421477, 0.421475, 0.421482]
	0.625	[0.414818, 0.414817, 0.414804]	[0.414806, 0.4148, 0.414833]	[0.414814, 0.414808, 0.414803]
	0.3125	[0.41154, 0.411519, 0.411493]	[0.411542, 0.411497, 0.411514]	[0.411523, 0.411532, 0.411489]
	0.15625	[0.409904, 0.40987, 0.409864]	[0.409848, 0.409922, 0.409863]	[0.409893, 0.40989, 0.409859]
	0.078125	[0.409079, 0.409046, 0.409065]	[0.409098, 0.409054, 0.409038]	[0.409069, 0.409051, 0.409068]
	0.0390625	[0.408647, 0.408697, 0.408624]	[0.408686, 0.408666, 0.408615]	[0.408648, 0.408684, 0.408632]
	0.01953125	[0.408458, 0.408423, 0.408473]	[0.408424, 0.408449, 0.408482]	[0.408443, 0.408445, 0.408465]

2 lentelė: Minimumo taškai, kai baudos koeficientas r dalijamas iš 2

		Pradiniai taškai x_0		
		(0, 0, 0)	(1, 1, 1)	(0.1, 0.5, 0.7)
	10	[0.525502, 0.525468, 0.525496]	[0.525469, 0.525508, 0.52551]	[0.525488, 0.52545, 0.525507]
	7.5	[0.493744, 0.493845, 0.493745]	[0.493763, 0.493773, 0.493777]	[0.493799, 0.493783, 0.493778]
	5.625	[0.471085, 0.470962, 0.471027]	[0.470993, 0.471005, 0.471062]	[0.470953, 0.471043, 0.471037]
	4.21875	[0.454545, 0.45457, 0.454526]	[0.454536, 0.454531, 0.454592]	[0.454562, 0.454603, 0.454488]
	3.1640625	[0.442553, 0.442488, 0.442572]	[0.442537, 0.442514, 0.442535]	[0.442542, 0.442517, 0.442546]
	2.373046875	[0.433669, 0.433738, 0.433728]	[0.433679, 0.433731, 0.433732]	[0.433706, 0.433747, 0.4337]
	1.77978515625	[0.427158, 0.427298, 0.42718]	[0.42721, 0.427229, 0.427178]	[0.427212, 0.427244, 0.427177]
	1.3348388671875	[0.422425, 0.422345, 0.422401]	[0.42238, 0.422394, 0.422386]	[0.422366, 0.422406, 0.422406]
	1.001129150390625	[0.418747, 0.418844, 0.418836]	[0.418841, 0.418767, 0.418818]	[0.418838, 0.418775, 0.418824]
	0.7508468627929688	[0.416129, 0.41613, 0.416183]	[0.416096, 0.41615, 0.416182]	[0.416177, 0.416163, 0.416097]
	0.5631351470947266	[0.414173, 0.414134, 0.41417]	[0.414161, 0.414165, 0.414151]	[0.414135, 0.414156, 0.414173]
$r = r \cdot \frac{3}{4}$	0.4223513603210449	[0.412644, 0.412699, 0.412673]	[0.412679, 0.412658, 0.412684]	[0.412694, 0.412648, 0.412671]
	0.3167635202407837	[0.411521, 0.411609, 0.411556]	[0.411587, 0.411562, 0.411533]	[0.411598, 0.411513, 0.411572]
	0.23757264018058777	[0.41074, 0.41071, 0.410739]	[0.410759, 0.410726, 0.410711]	[0.410742, 0.410689, 0.41076]
	0.17817948013544083	[0.410044, 0.410074, 0.410207]	[0.410073, 0.410114, 0.41014]	[0.410101, 0.41014, 0.410087]
	0.13363461010158062	[0.409627, 0.409645, 0.409651]	[0.409655, 0.40962, 0.409655]	[0.409659, 0.409585, 0.409682]
	0.10022595757618546	[0.409261, 0.409304, 0.409316]	[0.409293, 0.409293, 0.409291]	[0.409287, 0.409326, 0.409269]
	0.0751694681821391	[0.409018, 0.409048, 0.409027]	[0.409051, 0.409076, 0.408969]	[0.409061, 0.409003, 0.409036]
	0.056377101136604324	[0.408805, 0.40884, 0.408862]	[0.408778, 0.40882, 0.40891]	[0.408822, 0.408812, 0.408873]
	0.04228282585245324	[0.408652, 0.408671, 0.408743]	[0.408674, 0.408656, 0.408737]	[0.408687, 0.408653, 0.408728]
	0.03171211938933993	[0.408509, 0.408594, 0.408636]	[0.408595, 0.408586, 0.408554]	[0.408502, 0.408592, 0.40864]
	0.02378408954200495	[0.408459, 0.408529, 0.408501]	[0.408511, 0.408521, 0.408456]	[0.408481, 0.408493, 0.408512]
	0.017838067156503712	[0.408399, 0.408469, 0.408434]	[0.408434, 0.408479, 0.408389]	[0.408454, 0.408443, 0.408405]

3 lentelė: Minimumo taškai, kai baudos koeficientas r dauginamas iš $\frac{3}{4}$

4.2 Funkcijos reikšmės

4.3 Greitis

4.4 Efektyvumas

5 Išvada

6 Priedas