VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Laboratorinis darbas 4

Tiesinis programavimas

Nikita Gainulin

VILNIUS 2024

Turinys

1 Įvadas

Savo ankstesniame laboratoriniame darbe gilinausi į netiesinį programavimą. Kaip nustačiau ankstesniame laboratoriniame darbe, netiesinio optimizavimo metodų tikslas - optimizuoti uždavinį, kurio tikslo funkcija arba vienas iš apribojimų yra netiesinio tipo. Šiame laboratoriniame darbe gilinsiuosi į tiesinį optimizavimą, kur visos modelio funkcijos bus tiesinės.

2 Nagrinėjamas uždavinys

Šiame laboratoriniame darbe apskaičiuosime mažiausią tikslo funkcijos reikšmę, optimalų sprendinį ir bazę šioms dviem sistemoms:

$$\begin{aligned} \min 2x_1 - 3x_2 - 5x_4 \\ -x_1 + x_2 - x_3 - x_4 \leq 8 \\ 2x_1 + 4x_2 \leq 10 \\ x_3 + x_4 \leq 3 \\ x_i \geq 0 \end{aligned} \tag{1}$$

$$\begin{aligned} \min 2x_1 - 3x_2 - 5x_4 \\ -x_1 + x_2 - x_3 - x_4 \leq 1 \\ 2x_1 + 4x_2 \leq 5 \\ x_3 + x_4 \leq 7 \\ x_i \geq 0 \end{aligned} \tag{2}$$

3 Tiesinis optimizavimas ir uždavinio sprendimo algoritmas

3.1 Tiesinis optimizavimas

Kaip jau buvo matyti iš pateikto uždavinio, tiesinio optimizavimo metodas puikiai tinka jam spręsti ir reikiamiems atsakymams pateikti. Visi apribojimai ir pati tikslo funkcija yra tiesiniai. Kalbant apie minėtą tiesinį optimizavimo metodą, šiame laboratoriniame darbe naudosiu **simplekso metodą**.

3.2 Simplekso metodas

Paprastai, išgirdę sąvoką „simplekso metodas“, galime pagalvoti apie mums jau pažįstamą Nelderio-Medo metodą, kuris veikia su mažų matmenų figūromis. Tačiau, kaip jau žinome, šis metodas netinka nei tiesiniams, nei netiesiniams optimizavimo uţdaviniams spręsti, nes jis neturi galimybės savarankiškai tvarkytis su apribojimais. Netiesiniam optimizavimui jis yra svarbi kito, baudų metodo, dalis. Tačiau tiesinio optimizavimo atveju simplekso metodas taikomas visiškai kitaip ir niekuo nepanašus į tradicinį Nelderio-Medo metodą.

Simplekso metodą pademonstruosiu optimizuodamas pirmąją sistemą (1). Pirmasis simplekso metodo žingsnis yra paversti mūsų sistemos apribojimus iš nelygybių į lygybes įvedant vadinamuosius laisvuosius kintamuosius. Tokių kintamųjų kiekį lemia tai, kiek nelygybių yra sistemoje (išskyrus teigiamų reikšmių apribojimą). Mūsų atveju jų yra 3, todėl įvedame šiuos laisvuosius kintamuosius:

s_1, s_2 ir s_3. Kitas žingsnis - padauginti mūsų tikslo funkciją iš -1, nes pagal numatytuosius nustatymus simplekso lentelė dirba su maksimizavimu, todėl turime pakeisti indeksus, kad išlaikytume minimizavimą. Taigi, mūsų naujoji sistema su lygybėmis atrodo taip:

$$\begin{aligned} \max -2x_1 + 3x_2 + 5x_4 \\ -x_1 + x_2 - x_3 - x_4 + s_1 = 8 \\ 2x_1 + 4x_2 + s_2 = 10 \\ x_3 + x_4 + s_3 = 3 \\ x_i, s_i \geq 0 \end{aligned}$$

Po pirmiau minėtų transformacijų naująją sistemą reikia perrašyti į standartinę matricinę formą $AX = B$, kur A - apribojimų indeksų matrica, X - nežinomų ir laisvųjų kintamųjų matrica, B - dešiniosios pusės reikšmių matrica. Štai kaip atrodo mūsų naujosios sistemos standartinė forma:

$$\underbrace{\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & s_1 & s_2 & s_3 \\ -1 & 1 & -1 & -1 & 1 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}}_{\text{A}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}}_{\text{X}} = \underbrace{\begin{pmatrix} 8 \\ 10 \\ 3 \end{pmatrix}}_{\text{B}}$$

Dabar galime sukurti pradinę simplekso lentelę:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1	8	-1	1	-1	-1	1	0	0
0	s_2	10	2	4	0	0	0	1	0
0	s_3	3	0	0	1	1	0	0	1
$Z_j - C_j$			2	-3	0	-5	0	0	0

1 lentelė: Pradinė simplekso lentelė, kur C_j - maksimizavimo tikslo funkcijos indeksai, B - bazinės reikšmės, C_b - bazinių reikšmių indeksai maksimizavimo tikslo funkcijoje, X_b - atitinka standartinės formos B matricai

Kaip pastebėjote, turime nepažįstamą eilutę $Z_j - C_j$. Šioje eilutėje nustatomas pagrindinis stulpelis ir ji bus naudinga vėliau, kai pradėsime skaičiuoti optimalią vertę. Kol kas pakanka atkreipti dėmesį į šią formulę, pagal kurią apskaičiuojama $Z_j - C_j$:

$$Z_j - C_j \equiv C_b X_j - C_j \tag{3}$$

Kadangi iš pradžių C_b yra 0, visa apatinė $Z_j - C_j$ eilutė yra lygi $-C_j$. Toliau šioje eilutėje mums reikia rasti mažiausią neigiamą reikšmę, šiuo atveju -5. Tai yra pagrindinis stulpelis x_4. Dabar mums reikia pasirinkti pagrindinę eilutę. Tai atliekama visas X_b reikšmes dalijant iš pagrindinės eilutės reikšmių atitinkamai. Gauname štai tokias reikšmes:

$$\frac{8}{-1} = -8$$

$$\frac{10}{0} \notin \mathbb{R}$$

$$\frac{3}{1} = 3$$

Iš apskaičiuotų santykinių dydžių turime pasirinkti tą, kuris duoda mažiausią teigiamą reikšmę, kuri mūsų atveju yra 3. Taigi, kai mes gavome pagrindinius stulpelį ir eilutę, galime pradėti konstruoti sekančią simplekso lentelę:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
	s_1								
	s_2								
	s_3								
$Z_j - C_j$									

Tikslo funkcijos indeksai C_j eilutėje nesikeis, todėl galime drąsiai juos perrašyti. Toliau, kadangi anksčiau nustatėme, kad mūsų pagrindinis stulpelis yra x_4 kintamasis, o pagrindinė eilutė yra trečia nuo viršaus (ta, kurios bazinis kintamasis buvo s_3), galime daryti prielaidą, kad x_4 bus bazinis kintamasis, kurį nurodysime atsakyme, todėl galime jį perrašyti į bazinių kintamųjų stulpelį vietoj s_3:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
	s_1								
	s_2								
	x_4								
$Z_j - C_j$									

Po to indeksų stulpelį C_b užpildome tikslo funkcijos C_j indeksais. Kadangi neseniai pakeitėme trečiąjį kintamąjį į x_4, jo indeksas taip pat pasikeis į 5:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1								
0	s_2								
5	x_4								
$Z_j - C_j$									

Toliau pažvelkime į pirmąją lentelę (1), tiksliau - į raktinę reikšmę, esančią raktinės eilutės ir raktinio stulpelio sankirtoje. Mūsų atveju tai būtų 1. Paprastai, jei reikšmė būtų kokia nors kita, tuomet turėtume atlikti papildomus šios rakto eilutės reikšmių skaičiavimus, kad rakto reikšmė būtų 1, tačiau mums pasisekė, todėl jokių papildomų skaičiavimų atlikti nereikia. Svarbu pažymėti, kad skaičiavimai atliekami visai rakto eilutei, įskaitant X_b reikšmes, o ne tik rakto reikšmei. Toliau visas kitas rakto stulpelio reikšmes, išskyrus rakto reikšmę, turime paversti 0, padaugindami rakto reikšmę (ir visą rakto eilutę) iš reikiamo skaičiaus ir pridėdami visą eilutę prie kitos. Pirmoje lentelėje (1) vienintelė kita eilutė, kurią reikia pakoreguoti, yra pirmoji eilutė su -1 rakto stulpelyje. Vėlgi, kadangi ji yra -1, mums nereikia papildomai dauginti rakto eilutės iš nieko ir galime tiesiog pridėti ją prie tos pirmosios eilutės, kad rakto stulpelio reikšmė būtų 0. Štai kaip atrodo mūsų naujesnė simpleksinė lentelė:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1	11	-1	1	0	0	1	0	1
0	s_2	10	2	4	0	0	0	1	0
5	x_4	3	0	0	1	1	0	0	1
$Z_j - C_j$									

Dabar turime apskaičiuoti $Z_j - C_j$ eilutę. Primenu, kad (3) formulė rodo, kaip tai galime padaryti. Taigi šios iteracijos skaičiavimai atrodo taip:

$$Z_1 - C_1 = C_b X_1 - C_1 = (0 \cdot -1 + 0 \cdot 2 + 5 \cdot 0) - (-2) = -(-2) = 2$$

$$Z_2 - C_2 = C_b X_2 - C_2 = (0 \cdot 1 + 0 \cdot 4 + 5 \cdot 0) - 3 = -3$$

$$Z_3 - C_3 = C_b X_3 - C_3 = (0 \cdot 0 + 0 \cdot 0 + 5 \cdot 1) - 0 = 5 - 0 = 5$$

$$Z_4 - C_4 = C_b X_4 - C_4 = (0 \cdot 0 + 0 \cdot 0 + 5 \cdot 1) - 5 = 5 - 5 = 0$$

$$Z_5 - C_5 = C_b X_5 - C_5 = (0 \cdot 1 + 0 \cdot 0 + 5 \cdot 0) - 0 = 0$$

$$Z_6 - C_6 = C_b X_6 - C_6 = (0 \cdot 0 + 0 \cdot 1 + 5 \cdot 0) - 0 = 0$$

$$Z_7 - C_7 = C_b X_7 - C_7 = (0 \cdot 1 + 0 \cdot 0 + 5 \cdot 1) - 0 = 5$$

Kad ateityje nereikėtų beprasmiškai kartoti, praleisiu skaičiavimus, kai C_b reikšmė yra 0, nes tai nieko nekeičia. Dabar, kai apskaičiavome visą $Z_j - C_j$ eilutę, galime užbaigti pirmosios iteracijos lentelę:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1	11	-1	1	0	0	1	0	1
0	s_2	10	2	4	0	0	0	1	0
5	x_4	3	0	0	1	1	0	0	1
$Z_j - C_j$			2	-3	5	0	0	0	5

2 lentelė: Simplekso lentelė pirmos iteracijos pabaigoje

Tai žymi pirmosios iteracijos pabaigą ir antrosios pradžią. Vėlgi mažiausias skaičius $Z_j - C_j$ eilutėje yra -3, todėl x_2 yra pagrindinis stulpelis. Dabar pažvelkime į santykius:

$$\frac{11}{1} = 11$$

$$\frac{10}{4} = 2.5$$

$$\frac{3}{0} \notin \mathbb{R}$$

Kaip matome, antroji, s_2 eilutė yra mūsų naujoji raktinė eilutė. Dabar, kai turime ir raktinę eilutę, ir raktinį stulpelį, pradėkime atnaujinti simplekso lentelę:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1								
3	x_2								
5	x_4								
$Z_j - C_j$									

Kadangi x_2 tapo raktiniu stulpeliu, galime atnaujinti B bazinių kintamųjų vektorių, kad į jį įtrauktume šį kintamąjį, raktinėje eilutėje pakeisdami s_2. Tuomet nauja tos eilutės C_b reikšmė bus 3. Dabar grįžkime į simplekso lentelę antrosios iteracijos pradžioje. Kaip matome, šį kartą rakto reikšmė yra 4, vadinasi, turime padauginti visą eilutę iš $\frac{1}{4}$, kad tolesniems skaičiavimams rakto reikšmė būtų lygi 1. Taigi naujoji rakto eilutė atrodo taip:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1								
3	x_2	2.5	0.5	1	0	0	0	0.25	0
5	x_4								
$Z_j - C_j$									

Trečioji, x_4 eilutė lieka nepakitusi, nes raktinio stulpelio reikšmė jau yra 0:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1								
3	x_2	2.5	0.5	1	0	0	0	0.25	0
5	x_4	3	0	0	1	1	0	0	1
$Z_j - C_j$									

Kalbant apie pirmąją, s_1 eilutę, jos reikšmė pradinėje antrosios iteracijos lentelėje yra 1, todėl mums tereikia padauginti mūsų rakto eilutę iš -1 ir pridėti ją prie pirmosios eilutės. Gautos reikšmės yra tokios:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1	8.5	-1.5	0	0	0	1	-0.25	1
3	x_2	2.5	0.5	1	0	0	0	0.25	0
5	x_4	3	0	0	1	1	0	0	1
$Z_j - C_j$									

Norint užbaigti antrąją iteraciją, belieka tik apskaičiuoti eilutę $Z_j - C_j$:

$$Z_1 - C_1 = C_b X_1 - C_1 = (3 \cdot 0.5) - (-2) = 1.5 - (-2) = 3.5$$

$$Z_2 - C_2 = C_b X_2 - C_2 = (3 \cdot 1) - 3 = 0$$

$$Z_3 - C_3 = C_b X_3 - C_3 = (5 \cdot 1) - 0 = 5 - 0 = 5$$

$$Z_4 - C_4 = C_b X_4 - C_4 = (5 \cdot 1) - 5 = 5 - 5 = 0$$

$$Z_5 - C_5 = C_b X_5 - C_5 = 0 - 0 = 0$$

$$Z_6 - C_6 = C_b X_6 - C_6 = (3 \cdot 0.25) - 0 = 0.75$$

$$Z_7 - C_7 = C_b X_7 - C_7 = (5 \cdot 1) - 0 = 5$$

Taigi, antros iteracijos pabaigoje gauname štai tokią simplekso lentelę:

		C_j	-2	3	0	5	0	0	0
C_b	B	X_b	x_1	x_2	x_3	x_4	s_1	s_2	s_3
0	s_1	8.5	-1.5	0	0	0	1	-0.25	1
3	x_2	2.5	0.5	1	0	0	0	0.25	0
5	x_4	3	0	0	1	1	0	0	1
$Z_j - C_j$			3.5	0	5	0	0	0.75	5

3 lentelė: Simplekso lentelė pirmos iteracijos pabaigoje

Trečiosios iteracijos pradžioje jau matome, kad $Z_j - C_j$ eilutėje nėra neigiamų reikšmių, vadinasi, jau radome optimalų sprendinį ir toliau tęsti nereikia. Atsakymus nesunkiai galime pateikti iš lentelės: bazinių kintamųjų vektorių B sudaro šie kintamieji - [s_1, x_2, x_4]. Optimalų sprendinį galima rasti

priderinus X_b reikšmes prie bazinės matricos B ir jos yra tokios: $x_1 = 0$, $x_2 = 2.5$, $x_3 = 0$, $x_4 = 3$. Taip pat galime apskaičiuoti minimalią tikslo funkcijos reikšmę pagal šią formulę:

$$-(C_b \cdot X_b) = -(0 \cdot 8.5 + 3 \cdot 2.5 + 5 \cdot 3) = -22.5$$

Taip galima panaudoti simplekso metodą tiesiniam optimizavimo uždaviniui spręsti rankiniu būdu, tačiau vienas iš šio laboratorinio darbo tikslų buvo išspręsti jį taikant kodinį šio metodo įgyvendinimą, todėl pasidalinsiu savo įgyvendinimu Python kalba.

3.3 Algoritmo kodas

Štai kaip atrodo mano algoritmas:

```
from imports import np

class SimplexSolver:
    def __init__(self, c, A, b):
        self.numVars = len(c)
        self.numConstr = len(b)
        self.c = c
        self.A = A
        self.b = b
        self.table = self.createInitTable()
        self.base = list(range(self.numVars, self.numVars + self.numConstr))

    def createInitTable(self):
        numSlack = self.numConstr
        totalVars = self.numVars + numSlack

        table = np.zeros((self.numConstr + 1, totalVars + 1))

        table[:-1, :self.numVars] = self.A
        table[:-1, self.numVars:totalVars] = np.eye(numSlack)
        table[:-1, -1] = self.b

        table[-1, :self.numVars] = self.c

        return table

    def solve(self):
        self.solveOptimal()
        return self.getSolv()

    def solveOptimal(self):
        maxIter = 1000
        iter = 0

        while iter<maxIter:
            print(self.table)
            pivotCol = self.getEnterVar()
            if pivotCol is None:
                break

            pivotRow = self.getDepartVar(pivotCol)
            if pivotRow is None:
                raise Exception("[DEBUG] Pivot row error.")

            self.base[pivotRow] = pivotCol
```

```
            self.pivot(pivotRow, pivotCol)
            iter += 1

        if iter == maxIter:
            raise Exception("[DEBUG] Maximum iterations reached.")

        return -self.table[-1, -1]

    def getEnterVar(self):
        objectRow = self.table[-1, :-1]
        minVal = np.min(objectRow)

        if minVal>=-1e-6:
            return None

        return np.argmin(objectRow)

    def getDepartVar(self, pivotCol):
        ratios = []
        rhsCol = self.table[:-1, -1]
        pivotColVal = self.table[:-1, pivotCol]

        for i in range(len(rhsCol)):
            if pivotColVal[i]<=1e-6:
                ratios.append(float('inf'))
            else:
                ratio = rhsCol[i]/pivotColVal[i]
                if ratio<0:
                    ratios.append(float('inf'))
                else:
                    ratios.append(ratio)

        if min(ratios) == float('inf'):
            return None

        return np.argmin(ratios)

    def pivot(self, pivotRow, pivotCol):
        pivot = self.table[pivotRow, pivotCol]
        self.table[pivotRow] = self.table[pivotRow]/pivot

        for i in range(len(self.table)):
            if i!=pivotRow:
                factor = self.table[i, pivotCol]
                self.table[i] = self.table[i]-factor*self.table[pivotRow]

    def getSolv(self):
        solution = np.zeros(self.numVars)

        for i, var in enumerate(self.base):
            if var<self.numVars:
                solution[var] = max(0, self.table[i, -1])

        objVal = np.dot(self.c, solution)

        return{
            'x': solution,
            'objective': objVal,
            'table': self.table
        }
```

4 Rezultatai

Pirmiausia apžvelkime iteracijas, kurias algoritmas atliko sprendžiant pirmąją sistemą (1):

-1	1	-1	-1	1	0	0	8
2	4	0	0	0	1	0	10
0	0	1	1	0	0	1	3
2	-3	0	-5	0	0	0	0

4 lentelė: Pirmos sistemos (1) simplekso lentelė pirmosios iteracijos pradžioje

-1	1	0	0	1	0	1	11
2	4	0	0	0	1	0	10
0	0	1	1	0	0	1	3
2	-3	5	0	0	0	5	15

5 lentelė: Pirmos sistemos (1) simplekso lentelė antrosios iteracijos pradžioje

-1.5	0	0	0	1	-0.25	1	8.5
0.5	1	0	0	0	0.25	0	2.5
0	0	1	1	0	0	1	3
3.5	0	5	0	0	0.75	5	22.5

6 lentelė: Pirmos sistemos (1) simplekso lentelė trečiosios iteracijos pradžioje

Pirmas pastebimas skirtumas nuo rankinio metodo yra tas, kad pati lentelė atrodo glaustesnė, nėra apibrėžimų, ką reiškia kiekviena reikšmė, nes viskas jau yra iš anksto nustatyta ir turi prasmę, jei tik pažvelgsime į patį kodą. Kitas dalykas yra tai, kad algoritmas eigoje apskaičiuoja „minimalią“ tikslo funkcijos reikšmę ir ją išsaugo apatiniame dešiniajame lentelės langelyje. Tačiau, kaip jau paaiškinau anksčiau, šią reikšmę vis tiek reikia padauginti iš -1, kad ji būtų tikroji minimali reikšmė, nes simplekso lentelėje pagal metodo apibrėžimą dirbame su maksimizavimo operacija. Štai galutiniai atsakymai, kuriuos grąžina algoritmas:

Baziniai kintamieji	[1 3 4]
Optimalus sprendinys x =	[0 2.5 0 3]
Minimali tikslo funkcijos reikšmė =	-22.5

7 lentelė: Pirmos sistemos (1) galutiniai rezultatai

Čia gali tekti šiek tiek paaiškinti rezultatus. Pirmiausia pakalbėkime apie bazinius kintamuosius. Kaip jau žinome, iš viso turime 7 nežinomuosius kintamuosius, įskaitant laisvuosius: $x_1, x_2, x_3, x_4, s_1, s_2, s_3$. Jei juos laikytume masyve, o tai ir darome algoritme, kiekvieno kintamojo padėtis tame masyve prasidėtų nuo 0 ir baigtųsi skaičiumi 6, nes taip apskritai yra struktūrizuojami masyvai. Bazinių kintamųjų masyve kiekviena reikšmė reiškia kintamojo poziciją, todėl 1 reikšmė būtų x_2, 3 - x_4, o 4 - s_1, o tai tiksliai atitinka atsakymą, kurį gavau bandydamas išspręsti sistemą ranka. Ta pati logika taikoma ir kitai galutinio atsakymo daliai - optimaliam sprendiniui. Kiekviena reikšmė masyve atitinkamai reiškia kiekvieno nežinomo kintamojo reikšmę, išskyrus laisvuosius kintamuosius. Taip gauname tokį atsakymą: $x_1 = 0$, $x_2 = 2,5$, $x_3 = 0$, $x_4 = 3$, kuris vėlgi yra lygiai toks pat sprendinys, kokį gavau aš. Galiausiai, minimali tikslo funkcijos vertė taip pat sutampa su mano. Vėlgi, tai tik vienos sistemos pavyzdys, tačiau atrodo, kad mano parašytas algoritmas gali puikiai išspręsti bet kokius panašaus pobūdžio tiesinio optimizavimo uždavinius. Kalbant apie antrąją sistemą (2), štai simpleksų lentelės kiekvienai algoritmo iteracijai:

-1	1	-1	-1	1	0	0	1
2	4	0	0	0	1	0	5
0	0	1	1	0	0	1	7
2	-3	0	-5	0	0	0	0

8 lentelė: Antros sistemos (2) simplekso lentelė pirmosios iteracijos pradžioje

-1	1	0	0	1	0	0	8
2	4	0	0	0	1	0	5
0	0	1	1	0	0	1	7
2	-3	5	0	0	0	5	35

9 lentelė: Antros sistemos (2) simplekso lentelė antrosios iteracijos pradžioje

-1.5	0	0	0	1	-0.25	1	6.75
0.5	1	0	0	0	0.25	0	1.25
0	0	1	1	0	0	1	7
3.5	0	5	0	0	0.75	5	38.75

10 lentelė: Antros sistemos (2) simplekso lentelė trečiosios iteracijos pradžioje

Baziniai kintamieji	[1 3 4]
Optimalus sprendinys x =	[0 1.25 0 7]
Minimali tikslo funkcijos reikšmė =	-38.75

11 lentelė: Antros sistemos (2) galutiniai rezultatai

Dešiniosios pusės reikšmės	[1, 5, 7]	[8, 10, 3]
Baziniai kintamieji	[1 3 4]	[1 3 4]
Optimalus sprendinys x =	[0 1.25 0 7]	[0. 2.5 0. 3.]
Minimali tikslo funkcijos reikšmė =	-38.75	-22.5

12 lentelė: Pirmos (1) ir antros (2) sistemų galutiniai rezultatai vienoje lentelėje

5 Išvada

Apibendrinant, pagrindiniai šio laboratorinio darbo tikslai:

- išnagrinėti ir paaiškinti vieną iš tiesinių optimizavimo uždavinių sprendimo metodų - simplekso metodą;
- parašyti simplekso metodo įgyvendinimą Python kalba;
- palyginti turimo ir Python simplekso metodo įgyvendinimo atsakymus - minimalią tikslo funkcijos reikšmę, optimalų sprendinį ir bazinius kintamuosius;

Remdamasis pateiktais rezultatais galiu drąsiai teigti, kad mano turimas simplekso metodo įgyvendinimas Python gali patogiai spręsti panašaus pobūdžio tiesinio optimizavimo uždavinius.

6 Priedas

imports.py failas:

```
from datetime import datetime
import numpy as np
```

simplex.py failas:

```
from imports import np

class SimplexSolver:
    def __init__(self, c, A, b):
        self.numVars = len(c)
        self.numConstr = len(b)
        self.c = c
        self.A = A
        self.b = b
        self.table = self.createInitTable()
        self.base = list(range(self.numVars, self.numVars + self.numConstr))

    def createInitTable(self):
        numSlack = self.numConstr
        totalVars = self.numVars + numSlack

        table = np.zeros((self.numConstr + 1, totalVars + 1))

        table[:-1, :self.numVars] = self.A
        table[:-1, self.numVars:totalVars] = np.eye(numSlack)
        table[:-1, -1] = self.b

        table[-1, :self.numVars] = self.c

        return table

    def solve(self):
        self.solveOptimal()
        return self.getSolv()

    def solveOptimal(self):
        maxIter = 1000
        iter = 0
```

```
        while iter<maxIter:
            print(self.table)
            pivotCol = self.getEnterVar()
            if pivotCol is None:
                break

            pivotRow = self.getDepartVar(pivotCol)
            if pivotRow is None:
                raise Exception("[DEBUG] Pivot row error.")

            self.base[pivotRow] = pivotCol

            self.pivot(pivotRow, pivotCol)
            iter += 1

        if iter == maxIter:
            raise Exception("[DEBUG] Maximum iterations reached.")

        return -self.table[-1, -1]

    def getEnterVar(self):
        objectRow = self.table[-1, :-1]
        minVal = np.min(objectRow)

        if minVal>=-1e-6:
            return None

        return np.argmin(objectRow)

    def getDepartVar(self, pivotCol):
        ratios = []
        rhsCol = self.table[:-1, -1]
        pivotColVal = self.table[:-1, pivotCol]

        for i in range(len(rhsCol)):
            if pivotColVal[i]<=1e-6:
                ratios.append(float('inf'))
            else:
                ratio = rhsCol[i]/pivotColVal[i]
                if ratio<0:
                    ratios.append(float('inf'))
                else:
                    ratios.append(ratio)

        if min(ratios) == float('inf'):
            return None

        return np.argmin(ratios)

    def pivot(self, pivotRow, pivotCol):
        pivot = self.table[pivotRow, pivotCol]
        self.table[pivotRow] = self.table[pivotRow]/pivot

        for i in range(len(self.table)):
            if i!=pivotRow:
                factor = self.table[i, pivotCol]
                self.table[i] = self.table[i]-factor*self.table[pivotRow]

    def getSolv(self):
        solution = np.zeros(self.numVars)
```

```
        for i, var in enumerate(self.base):
            if var<self.numVars:
                solution[var] = max(0, self.table[i, -1])

        objVal = np.dot(self.c, solution)

        return{
            'x': solution,
            'objective': objVal,
            'table': self.table
        }
```

main.py failas:

```
from imports import datetime, np
from simplex import SimplexSolver
print(f"{datetime.now()}\n")

c = np.array([2, -3, 0, -5])
A = np.array([
    [-1, 1, -1, -1],
    [2, 4, 0, 0],
    [0, 0, 1, 1]
])
b = np.array([8, 10, 3])

print("(b = [8, 10, 3]):")
solver = SimplexSolver(c, A, b)
solution = solver.solve()
print("Simplex Table:")
print(solution['table'])
print(f"Optimal solution x = {solution['x']}")
print(f"Objective value = {solution['objective']}\n")
print("\n(b = [1, 5, 7]):")
b = np.array([1, 5, 7])
solver2 = SimplexSolver(c, A, b)
solution2 = solver2.solve()
print("Simplex Table:")
print(solution2['table'])
print(f"Optimal solution x = {solution2['x']}")
print(f"Objective value = {solution2['objective']}\n")
```