

Vilniaus universitetas

Matematikos ir informatikos fakultetas

Programų sistemų studijų programa

Optimizavimo metodų ketvirtojo laboratorinio darbo ataskaita

Ataskaitą tikrino: Prof. Dr. Pranas Katauskis

Ataskaitą parengė: Dominykas Daunoravičius

Vilnius

2022

# Įvadas

**Laboratorinio darbo formulavimas:** Suprogramuoti simplekso algoritmą tiesinio programavimo uždaviniams spręsti ir išspręsti duotą tiesinio programavimo uždavinį bei naudojantis tuo pačiu algoritmu išspręsti individualų uždavinį, kur apribojimų dešinės pusės konstantos keičiasi į a, b ir c – studento knygelės numerio „1\*1\*abc“ skaitmenis.

**Laboratorinio darbo tikslas:** Suprogramuoti simplekso algoritmą tiesinio programavimo uždaviniams išspręsti bei parašyti ataskaitą apie tai, kaip buvo vykdomas laboratorinis darbas. Tai atlikus rasti optimalius sprendinius, minimalias funkcijų reikšmes ir bazes

# **Darbo** eiga

Laboratoriniame darbe pateiktas tiesinio programavimo uždavinys:

min 2*x*1 − 3*x*2 − 5*x*4

Suprogramavus simplekso algoritmą jį reikia išspręsti du kartus – pirmą kartą, jį reikia spręsti taip, kaip pateikta sąlygoje, antrajį – pakeičiant dešinės pusės apribojimų konstantas, kur a = 0, b = 3, c = 9. Vadinasi antruoju atveju uždavinio formuluotė bus tokia:

min 2*x*1 − 3*x*2 − 5*x*4

**Simplekso algoritmas** – tai tiesinio programavimo uždaviniams spręsti skirtas algoritmas, kuris šiame laboratoriniame darbe realizuojamas uždavinį užrašant matriciniu pavidalu ir sprendžiamas pasirenkant nebazinį stulpelį, kuris yra įkeliamas į bazę. Uždavinys yra sprendžiamas tol, kol 1-oje eilutėje nebelieka neigiamų kintamųjų, tuomet galime teigti, jog radome optimalų uždavinio sprendinį.

Pagrindiniai simplekso algoritmo žingsniai:

1. Pasirenkamas mažiausias neigiamas stulpelis iš pirmos eilutės.
2. Pasirenkama eilutė, kuri sudaro mažiausią santykį su gautu atsakymu.
3. Atliekami veiksmai su eilutėmis tam, kad gauti bazinį stulpelį.
4. Veiksmai kartojami tol, kol nebelieka neigiamų reikšmių pirmoje eilutėje.

## Pagrindinio uždavinio sprendimo eiga.

Iš uždavinio formuluotės susidarome:

* A= iš kairės apribojimų pusės.
* B= iš dešinės apribojimų pusės.
* C= iš min funkcijos konstantų prie x.
* X=T

Uždavinys matriciniu pavidalu kanonine forma, kuri gaunama iš uždavinio formuluotės:

Jog padaryti uždavinį standartine formą, reikia įsivesti 3 naujus kintamuosius, po vieną kiekvienam apribojimui, kurie apribojimų nelygybes leis paversti lygybėmis, pavadinsime juos s1, s2, s3. Uždavinys standartine forma atrodys taip:

min 2*x*1 − 3*x*2 − 5*x*4

Matriciniu pavidalu standartine forma:

, kur:

* A`= iš kairės apribojimų pusės.
* B nesikeičia. B=.
* C`= iš min funkcijos konstantų prie x ir s.
* X`=T

Tada uždavinys užrašomas lentelės pavidalu.

1 lentelė. Pagrindinio uždavinio pradinė lentelė.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | x4 | s1 | s2 | s3 |
| 0 | 2 | -3 | 0 | -5 | 0 | 0 | 0 |
| 8 | -1 | 1 | -1 | -1 | 1 | 0 | 0 |
| 10 | 2 | 4 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Bazės determinantas turi būti nelygus 0 ir bazę turi sudaryti tiek elementų, kiek yra apribojimų. Pirminės bazės elementais pasirenkami s1, s2 ir s3.

Atliekama pirmoji iteracija, kurios metu pasirinksime nebazinį stulpelį. Iš pat pradžių matome, kad 1 matricos eilutėje (0 eilutė – pavadinimai) turime dvi neigiamas reikšmes, pagal simplekso algoritmą, imsime tą reikšmę, kuri yra mažiausia, mūsų atveju tai bus -5 (stulpelis x4). Pasirinkus stulpelį kitas žingsnis gauti mažiausią neneigiamą santykį tam, kad galėtume pasirinkti reikiamą eilutę. 2 eilutė netinka, kadangi santykis yra neigiamas, o 3 eilutė netinka, kadangi 10 negalima dalinti iš 0, vadinasi pasirenkame 4 eilutę. Tai padarius, mums reikės šį stulpelį paversti baziniu, o kad tai padaryti reikės atlikti veiksmus su kiekviena eilute, tam, kad 1, 2 ir 3 eilučių x3 stulpelio reikšmės būtų 0. 4-oje eilutėje nereikia atlikti jokių veiksmų, kadangi pasirinkto stulpelio reikšmė jau yra 1, 3-oje eilutėje taip pat nereikia atlikti jokių veiksmų, kadangi x3 stulpelyje esanti reikšmė yra 0, 2-oje eilutėje reikia x3 stulpelio paversti 0, kad tai padaryti visai eilutei bus atliekamas toks veiksmas: R[[1]](#footnote-1)2 -> R2 + R4, o 1-oje eilutėje bus atliekamas toks veiksmas R1 -> R1 + 5\*R4.

Bazės elementas s3 (nes skaičiuojant buvo pasirinkta R4) keičiamas elementu x4. Bazės elementai dabar yra s1, s2 ir x4.

Atlikus visus veiksmus, po pirmos iteracijos gausime tokią matricą:

2 lentelė. Pagrindinio uždavinio lentelė po pirmos iteracijos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | x4 | s1 | s2 | s3 |
| 15 | 2 | -3 | 5 | 0 | 0 | 0 | 5 |
| 11 | -1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 10 | 2 | 4 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Toliau reikės kartoti tuos pačius veiksmus: pasirinkti mažiausią neigiamą stulpelį (x2 šiuo atveju), pasirinkti eilutę su mažiausiu neneigiamu santykiu (3 eilutė šiuo atveju, kadangi 10/4 yra mažiausias neneigiamas santykis) ir atlikti veiksmus su eilutėmis, kad gauti bazinį stulpelį. Veiksmai: R3 -> R3 / 4, R2 -> R2 – R3, R1 -> R1 + 3\*R3, 4-ai eilutei nereikia atlikti jokių veiksmų, kadangi x2 stulpelyje esanti reikšmė jau yra 0.

Bazės elementas s2 (nes skaičiuojant buvo pasirinkta R3) keičiamas elementu x2. Bazės elementai dabar yra s1, x2 ir x4.

Atlikus visus reikiamus veiksmus gauname tokią matricą:

3 lentelė. Pagrindinio uždavinio lentelė po antros iteracijos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | x4 | s1 | s2 | s3 |
| 22.5 | 3.5 | 0 | 5 | 0 | 0 | 0.75 | 5 |
| 8.5 | -1.5 | 0 | 0 | 0 | 1 | -0.25 | 1 |
| 2.5 | 0.5 | 1 | 0 | 0 | 0 | 0.25 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Trečios iteracijos metu vėl ieškom stulpelio su pirmoje eilutėje mažiausia neigiama reikšme. Kadangi 1-oje eilutėje nebeliko neigiamų reikšmių, galime teigti, jog radome optimalų sprendinį.

Iš uždavinio sprendimo gautieji bazės elementai s1, x2 ir x4, o tai yra bazė: {2, 4, 5}.

Galutinis sprendinys yra X`= (0, 2.5, 0, 3, 8.5, 0, 0), arba užrašius tik x reikšmes X = (0, 2.5, 0, 3).

Minimali funkcijos reikšmė: f(0, 2.5, 0, 3) = -22.5

## Uždavinio su pakeistomis konstantomis sprendimo eiga.

Iš uždavinio formuluotės susidarome:

* A= iš kairės apribojimų pusės.
* B= iš dešinės apribojimų pusės.
* C= iš min funkcijos konstantų prie x.
* X=T

Uždavinys matriciniu pavidalu kanonine forma, kuri gaunama iš uždavinio formuluotės:

Įsivedame 3 naujus kintamuosius: s1, s2, s3. Uždavinys standartine forma atrodys taip:

min 2*x*1 − 3*x*2 − 5*x*4

Matriciniu pavidalu standartine forma:

, kur:

* A`= iš kairės apribojimų pusės.
* B nesikeičia. B=.
* C`= iš min funkcijos konstantų prie x ir s.
* X`=T

Kadangi detaliai paaiškinau kaip buvo sprendžiamas pirmasis uždavinys, šio uždavinio eigoje paminėsiu tik pagrindinius veiksmus, kurie atliekami kiekvienos iteracijos metu.

Sudaroma lentelė.

4 lentelė. Individualaus uždavinio pradinė lentelė.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | x4 | s1 | s2 | s3 |
| 0 | 2 | -3 | 0 | -5 | 0 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | 1 | 0 | 0 |
| 3 | 2 | 4 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Pirminės bazės elementais pasirenkami s1, s2 ir s3.

Šioje iteracijoje pasirenkame x4 stulpelį, kadangi jo reikšmė mažiausia ir pasirenkame 4-ąją eilutę, kadangi santykis mažiausias. Atliekami veiksmai yra analogiški, kaip ir pagrindiniame uždavinyje: R1 -> R1 + 5 \* R4, R2 -> R2 + R4, o R3 ir R4 jokių veiksmų atlikti nereikia.

Bazės elementas s3 keičiamas elementu x4. Bazės elementai dabar yra s1, s2 ir x4.

Po pirmos iteracijos gauname tokią matricą:

5 lentelė. Individualaus uždavinio lentelė po pirmos iteracijos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | x4 | s1 | s2 | s3 |
| 45 | 2 | -3 | 5 | 0 | 0 | 0 | 5 |
| 9 | -1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 3 | 2 | 4 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Toliau renkamės x2 stulpelį ir 3 eilutę, kadangi gautas santykis yra mažiausias. Atliekame tokius veiksmus: R3 -> R3 / 4, R2 -> R2 – R3, R1 -> R1 + 3 \* R3, R4 veiksmai nėra atliekami, kadangi joje jau turime reikiamą skaičių.

Bazės elementas s2 keičiamas elementu x2. Bazės elementai dabar yra s1, x2 ir x4.

Atlikus šiuos veiksmus, gauname tokią matricą:

6 lentelė. Individualaus uždavinio lentelė po antros iteracijos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | x4 | s1 | s2 | s3 |
| 47.25 | 3.5 | 0 | 5 | 0 | 0 | 0.75 | 5 |
| 8.25 | -1.5 | 0 | 0 | 0 | 1 | -0.25 | 1 |
| 0.75 | 0.5 | 1 | 0 | 0 | 0 | 0.25 | 0 |
| 9 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Trečios iteracijos metu vėl ieškom stulpelio su pirmoje eilutėje mažiausia neigiama reikšme. Kadangi 1-oje eilutėje nebeliko neigiamų reikšmių, galime teigti, jog radome optimalų sprendinį.

Iš uždavinio sprendimo gautieji bazės elementai s1, x2 ir x4, o tai yra bazė: {2, 4, 5}.

Galutinis sprendinys yra X`= (0, 0.75, 0, 9, 8.25, 0, 0), arba užrašius tik x reikšmes X = (0, 0.75, 0, 9)

Minimali funkcijos reikšmė: f(0, 0.75, 0, 9) = -47.25

# Rezultatų palyginimai

7 lentelė. Uždavinių rezultatų palyginimai.

|  |  |  |  |
| --- | --- | --- | --- |
| Uždavinys | Minimali funkcijos reikšmė | Optimalus sprendinys | Bazė |
| Pagrindinis | -22.5 | (0, 2.5, 0, 3) | {2, 4, 5} |
| Individualus | -47.25 | (0, 0.75, 0, 9) | {2, 4, 5} |

Gavus atsakymus, matome, kad sprendžiant tą patį uždavinį, tačiau pakeitus apribojimų dešinės pusės konstantas, gavome, jog gautos minimalios tikslo funkcijos reikšmės ir optimalūs sprendiniai skiriasi, tačiau bazės sutampa. Abu uždaviniai buvo išspręsti per 3 iteracijas.

# Išvados

Apibendrinant galima teigti, suprogramuotu simplekso algoritmu buvo galima išspręsti pagrindinį ir individualų uždavinius, buvo gautos minimalios funkcijų reikšmės, optimalūs sprendiniai ir bazės. Atlikus uždavinių rezultatų palyginimą buvo galima pastebėti, kad sprendžiamų uždavinių bazės sutapo, tačiau nesutapo minimalios tikslo funkcijų reikšmės ir optimalūs sprendiniai

# Priedas

import numpy as np  
  
  
class LinearModel:  
  
 def \_\_init\_\_(self, A=np.empty([0, 0]), b=np.empty([0, 0]), c=np.empty([0, 0]), minmax="MIN"):  
 self.A = A  
 self.b = b  
 self.c = c  
 self.x = [float(0)] \* len(c)  
 self.minmax = minmax  
 self.printIter = True  
 self.optimalValue = None  
 self.transform = False  
  
 def addA(self, A):  
 self.A = A  
  
 def addB(self, b):  
 self.b = b  
  
 def addC(self, c):  
 self.c = c  
 self.transform = False  
  
 def setObj(self, minmax):  
 if minmax == "MIN" or minmax == "MAX":  
 self.minmax = minmax  
 else:  
 print("Invalid objective.")  
 self.transform = False  
  
 def setPrintIter(self, printIter):  
 self.printIter = printIter  
  
 def printSoln(self):  
 print("Optimal solution: ")  
 print(self.x)  
 print("Base:")  
 print("[", end="")  
 for i in range(0, len(self.x)):  
 if self.x[i] != 0 and i + 1 != len(self.x):  
 print(i + 1, end=", ")  
 elif self.x[i] != 0 and i != len(self.x):  
 print(i + 1, end="")  
 print("]")  
 print("Optimal value: ")  
 print(self.optimalValue)  
  
 def printTableau(self, tableau):  
  
 print("\t\t", end="")  
 for j in range(0, len(self.c)):  
 print("x\_" + str(j), "|", end="\t")  
 for j in range(0, (len(tableau[0]) - len(self.c) - 2)):  
 print("s\_" + str(j), "|", end="\t")  
  
 print()  
 for j in range(0, len(tableau)):  
 for i in range(1, len(tableau[0])):  
 if not np.isnan(tableau[j, i]):  
 if i == 0:  
 print(int(tableau[j, i]), end="|\t")  
 else:  
 print(round(tableau[j, i], 2), "|", end="\t")  
 else:  
 print(end="\t")  
 print()  
  
 def getTableau(self):  
 # construct starting tableau  
  
 if self.minmax == "MIN" and self.transform is False:  
 self.c[0:len(self.c)] = -1 \* self.c[0:len(self.c)]  
 self.transform = True  
  
 numVar = len(self.c)  
 numSlack = len(self.A)  
  
 t1 = np.hstack(([None], [0], self.c, [0] \* numSlack))  
  
 basis = np.array([0] \* numSlack)  
  
 for i in range(0, len(basis)):  
 basis[i] = numVar + i  
  
 A = self.A  
  
 if not ((numSlack + numVar) == len(self.A[0])):  
 B = np.identity(numSlack)  
 A = np.hstack((self.A, B))  
  
 t2 = np.hstack((np.transpose([basis]), np.transpose([self.b]), A))  
  
 tableau = np.array(np.vstack((t1, t2)), dtype='float')  
  
 return tableau  
  
 def simplexOptimization(self):  
  
 if not self.transform:  
 for i in range(len(self.c)):  
 self.c[i] = -1 \* self.c[i]  
  
 tableau = self.getTableau()  
  
 if self.printIter:  
 print("Starting Tableau:")  
 self.printTableau(tableau)  
  
 # assume initial basis is not optimal  
 optimal = False  
  
 # keep track of iterations for display  
 iter = 1  
  
 while iter != 50:  
  
 if self.printIter:  
 print("----------------------------------")  
 print("Iteration :", iter)  
 self.printTableau(tableau)  
  
 for cost in tableau[0, 2:]:  
 if cost < 0:  
 optimal = False  
 break  
 optimal = True  
  
 # if all directions result in decreased profit or increased cost  
 if optimal:  
 break  
  
 # nth variable enters basis, account for tableau indexing  
 n = tableau[0, 2:].tolist().index(np.amin(tableau[0, 2:])) + 2  
  
 # minimum ratio test, rth variable leaves basis  
 minimum = 99999  
 r = -1  
  
 for i in range(1, len(tableau)):  
 if tableau[i, n] > 0:  
 val = tableau[i, 1] / tableau[i, n]  
 if val < minimum:  
 minimum = val  
 r = i  
  
 pivot = tableau[r, n]  
  
 print("Pivot Column:", n)  
 print("Pivot Row:", r)  
 print("Pivot Element: ", pivot)  
  
 # perform row operations  
 # divide the pivot row with the pivot element  
 tableau[r, 1:] = tableau[r, 1:] / pivot  
  
 # pivot other rows  
 for i in range(0, len(tableau)):  
 if i != r:  
 mult = tableau[i, n] / tableau[r, n]  
 tableau[i, 1:] = tableau[i, 1:] - mult \* tableau[r, 1:]  
  
 # new basic variable  
 tableau[r, 0] = n - 2  
  
 iter += 1  
  
 if self.printIter:  
 print("----------------------------------")  
 print("Final Tableau reached in", iter, "iterations")  
 self.printTableau(tableau)  
 else:  
 print("Solved")  
  
 self.x = np.array([0] \* len(self.c), dtype=float)  
 # save coefficients  
 for key in range(1, (len(tableau))):  
 if tableau[key, 0] < len(self.c):  
 self.x[int(tableau[key, 0])] = tableau[key, 1]  
  
 self.optimalValue = -1 \* tableau[0, 1]  
  
  
def main():  
 model1 = LinearModel()  
  
 # Primary restrictions  
 A = np.array([[-1, 1, -1, -1],  
 [2, 4, 0, 0],  
 [0, 0, 1, 1]])  
 b = np.array([8, 10, 3])  
 c = np.array([2, -3, 0, -5])  
  
 # Restrictions a,b,c, that are from 1\*1\*abc  
 A = np.array([[-1, 1, -1, -1],  
 [2, 4, 0, 0],  
 [0, 0, 1, 1]])  
 b = np.array([0, 3, 9])  
 c = np.array([2, -3, 0, -5])  
  
 model1.addA(A)  
 model1.addB(b)  
 model1.addC(c)  
 model1.setObj("MIN")  
 model1.setPrintIter(True)  
  
 print("A =\n", A, "\n")  
 print("b =\n", b, "\n")  
 print("c =\n", c, "\n\n")  
 model1.simplexOptimization()  
 print("\n")  
 model1.printSoln()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. R – eilutė. [↑](#footnote-ref-1)