Изображение выглядит как металлоизделия, коллекция картинок

Автоматически созданное описание

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

**(ДВФУ)**

Институт математики и компьютерных технологий

Департамент математического и

компьютерного моделирования

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Методы оптимизации»

Выполнил студент

гр. Б9119-02.03.01сцт

Петров С.Д.

«27» марта 2022 г.

Проверил

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«13» июня 2022 г.

г. Владивосток

2022

**Постановка задачи**

Дана матричная игра с матрицей 𝐴 размерности 6х8. Необходимо вычислить верхнюю и нижнюю цены игры, а также найти равновесное решение в смешанных стратегиях.

**Реализация вычисления верхней и нижней цен игры**

Для реализации воспользуемся языком программирования Python.

**def** **findTopPrice** (A, n, m):

maxs = [ ]

**for** i **in** range(m):

max = A[**0**, i]

**for** j **in** range(**1**, n):

**if** (A[j, i] > max):

max = A[j, i]

maxs.append (max )

**return** min ( maxs )

**def** **findBottomPrice**(A, n , m):

mins = [ ]

**for** i **in** range(n):

min = A[i, **0**]

**for** j **in** range(**1**, m):

**if** (A[i, j] < min):

min = A[i, j]

mins.append(min)

**return** max (mins)

**Реализация алгоритма для нахождения решения**

Для реализации воспользуемся языком программирования Python и библиотекой Numpy для удобной работы с матрицами.

**import** **numpy** **as** **np**

**import** **copy**

n = **8**

m = **6**

**def** **set\_matrix**():

**return** [

np.random.randint(**1**, **10**, (n, m)).astype('float'),

np.random.randint(**1**, **10**, n).astype('float'),

np.random.randint(**1**, **10**, m).astype('float')

]

**def** **find\_lead\_str**(A, b, leadCol):

leadStrVal = np.inf

**for** i **in** range(**0**, n):

**if** (**not**((b[i] > **0**) **and** (A[i, leadCol] < **0**))):

leadStrNew = b[i] / A[i, leadCol]

**if** (leadStrNew < leadStrVal):

leadStr = i

leadStrVal = leadStrNew

**return** leadStr

**def** **calculate**(A, b ,c):

A = copy.deepcopy(A)

b = copy.deepcopy(b)

c = copy.deepcopy(c)

prevLeads = []

c = -c

cFree = **0**

colInd = [i **for** i **in** range(**0**, m)]

prevLeads.append(copy.deepcopy(colInd))

strInd = [i **for** i **in** range(m, m+n)]

#пока в нижней строке есть отрицательные числа

**while** min(c) < **0**:

oldColInd = copy.deepcopy(colInd)

oldStrInd = copy.deepcopy(strInd)

changeC = copy.deepcopy(c)

check = True

**while** check:

check = False

#находим индекс минимального с (выбираем разрешающий столбец)

leadCol = np.argmax(np.abs(changeC))

leadStr = find\_lead\_str(A, b, leadCol)

#пересечение разрешающих строки и столбца

leadVal = A[leadStr, leadCol]

#изменение базиса

strInd = np.insert(strInd, **0**, colInd[leadCol])

colInd[leadCol] = strInd[leadStr + **1**]

strInd = np.delete(strInd, leadStr + **1**)

#обнуляем с из разрешающего столбца

**for** i **in** range(len(prevLeads)):

prevLeads[i].sort()

sortColInd = copy.deepcopy(colInd)

sortColInd.sort()

**if** prevLeads[i] == sortColInd:

changeC[leadCol] = **0**

colInd = copy.deepcopy(oldColInd)

strInd = copy.deepcopy(oldStrInd)

check = True

**break**

prevLeads.append(copy.deepcopy(colInd))

#копируем разрешающую строку

leadStrVals = A[leadStr]

leadB = b[leadStr]

A = np.delete(A, leadStr, **0**)

b = np.delete(b, leadStr)

helpVals = copy.deepcopy(-A[:, leadCol])

A = np.reshape(np.insert(A, **0**, [**0** **for** i **in** range(**0**, m)]), (n, m))

b = np.insert(b, **0**, **0**)

**for** i **in** range(**0**, m):

**if** i != leadCol:

A[**0**, i] = leadStrVals[i]/leadVal

A[**0**, leadCol] = **1**/leadVal

b[**0**] = leadB/leadVal

#пересчет матрицы

**for** i **in** range(**1**, n):

**for** j **in** range(**0**, m):

oldVal = A[i, j]

A[i, j] = A[**0**, j]\*helpVals[i-**1**]

**if** (oldColInd[j] == colInd[j]):

A[i, j] += oldVal

b[i] = b[**0**]\*helpVals[i-**1**] + b[i]

leadC = c[leadCol]

#пересчет с

**for** i **in** range(**0**, m):

oldVal = c[i]

c[i] = A[**0**, i] \* leadC \*(-**1**)

**if** (oldColInd[i] == colInd[i]):

c[i] += oldVal

cFree = b[**0**] \* leadC\*(-**1**) + cFree

**return** [

A,

b,

c,

strInd,

colInd

]

**def** **print\_result**(n, new\_par, old\_par, Ind, offset = **0**, letter = "x"):

**for** i **in** range(**0**, n):

**if** i **in** Ind:

**print**(letter + "\_" + str(i) + " = " + str(new\_par[list(Ind).index(i)]) + " ", end='')

**else**:

**print**(letter + "\_" + str(i) + " = " + str(**0**) + " ", end='')

**print**()

res = **0**

check = False

**for** i **in** range(offset, offset + n):

**if** i **in** Ind:

**if** check:

**print**(" + ", end='')

**print**(str(old\_par[i - offset]) +" \* " + str(new\_par[list(Ind).index(i)]), end='')

res += old\_par[i - offset]\*new\_par[list(Ind).index(i)]

check = True

**print**(" = " + str(res))

A, b, c = set\_matrix()

**print**("A =")

**print**(A)

**print**("b = " + str(b))

**print**("c = " + str(c))

newA, newb, newc, strInd, colInd = calculate(A, b ,c)

**print**()

**print**("Решение прямой задачи: ")

print\_result(m, newb, c, strInd)

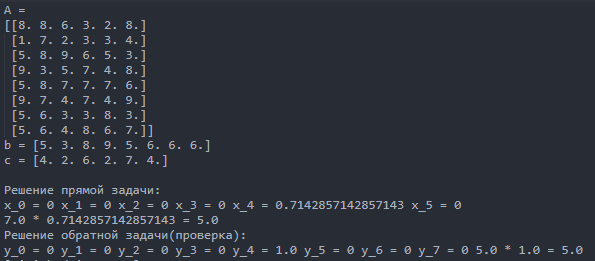
**print**()

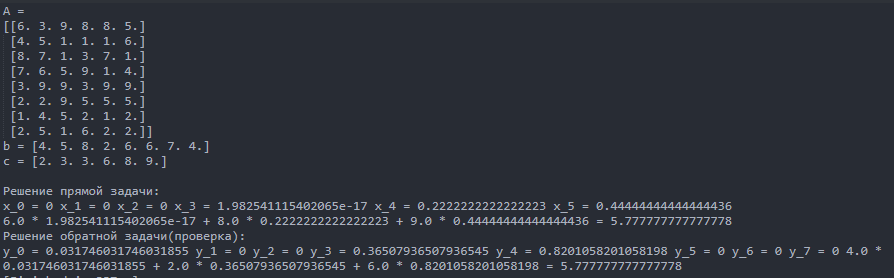
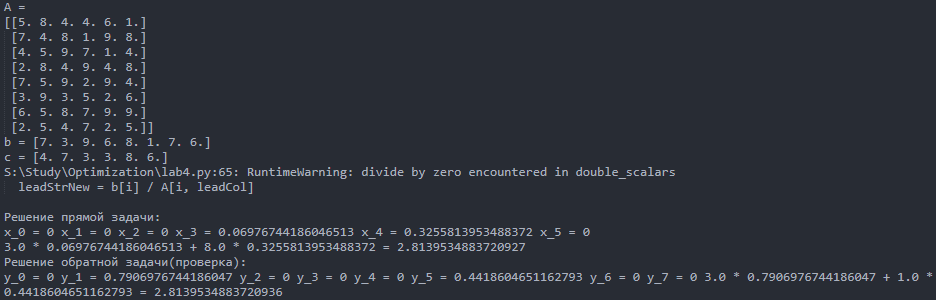
**print**("Решение обратной задачи(проверка): ")

print\_result(n, newc, b, colInd, offset = m, letter = "y")

**print**()

**Тесты**

****

**** ****

**Заключение**

В ходе данной лабораторной работы был реализован алгоритм нахождения верхней и нижней цен игры, а также разработана программа, позволяющая находить решения прямой и двойственных задач линейной оптимизации симплекс-методом.