Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

**Курсовая работа по дисциплине  
 Алгоритмы и вычислительные методы оптимизации**

**Симплекс-метод**

Вариант 12

Выполнил:

студент гр.ИП-916 /Меньщиков Д.А./

ФИО студента

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Проверил

/Новожилов Д.И./

ФИО преподавателя

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2022 г.

**Содержание**

1. Задание 3
2. Теоритические сведения 4
3. Выполнение задания 6
   1. Переход к канонической форме записи задачи ЛП 6
   2. Вывод программы 6
   3. Графическое решение 10
   4. Решение двойственной задачи к исходной 12
4. Список используемых источников 13
5. Код программы 14

**Задание**

1. Перейти к канонической форме записи задачи линейного программирования.

2. Написать программу, решающую задачу линейного программирования в канонической форме (с выводом всех промежуточных таблиц) двойственным симплекс-методом.

Программа правильно работать для различных случаев решений задачи: единственное решение, бесконечно много решений, функция не ограничена.

3. Решить исходную задачу графически и отметить на чертеже точки,

соответствующие симплексным таблицам, полученным при выполнении

программы из п.2.

4. Составить двойственную задачу к исходной и найти ее решение на основании теоремы равновесия.

**Теоритические сведения**

Пусть имеется пара двойственных задач в симметричной форме:

Канонические формы:

Матрица системы ограничений первой задачи:

Составим симплексную таблицу для первой задачи

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Б.п | 1 |  | … |  |  |  | … |  |
|  |  |  | … |  | 1 | 0 | … | 0 |
|  |  |  | … |  | 0 | 1 | … | 0 |
| … | … | … | … | … | … | … | … | … |
|  |  |  | … |  | 0 | 0 | … | 1 |
| Z | 0 |  | … |  | 0 | 0 | … | 0 |

Можно заметить, что построение самой двойственной задачи не обязательно, в столбцах таблицы записана исходная задача, а в строках – двойственная. При этом оценками решения исходной задачи являются коэффициенты Z-строки, а оценками решения двойственной задачи – коэффициенты столбца свободных членов.

При решении симплекс-методом все коэффициенты в столбце свободных членов должны были быть неотрицательны, а в Z-строке допускались отрицательные коэффициенты, которые в процессе решения преобразовывались в неотрицательные. Поскольку в двойственной задаче столбец свободных членов и Z-строка меняются местами, то можно допустить, что в Z-строке все коэффициенты неотрицательны, а в столбце свободных членов могут быть отрицательные коэффициенты. Тогда при выполнении симплексных преобразований необходимо преобразовывать коэффициенты столбца свободных членов в неотрицательные.

Симплексная таблица, в которой все коэффициенты Z-строки неотрицательны, а в столбце свободных членов имеются отрицательные, называется двойственно допустимой, а соответствующее решение – псевдопланом.

Алгоритм двойственного симплекс-метода:

1. В столбце свободных членов выбирают среди отрицательных минимальный. Это определяет разрешающую строку.

2. Для отрицательных элементов разрешающей строки находим симплексные отношения: отношения элементов Z-строки к отрицательным элементам разрешающей строки, взятые по модулю.

3. Выбираем минимальное симплексное отношение, соответствующий столбец – разрешающий.

4. Выполняют шаг симплексных преобразований таблицы.

5. Если в столбце свободных членов нет отрицательных, то решение оптимально, иначе переход на п.1

**Выполнение задания**

**Переход к канонической форме записи задачи ЛП**

Переход к канонической форме записи задачи линейного программирования:

Каноническая форма записи:

**Вывод программы**

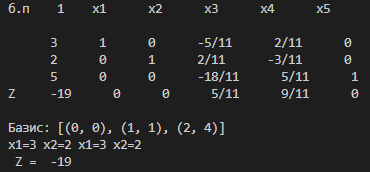


Рис 1. Вывод программы

Примеры работы программы для различных случаев:

1. Решений нет

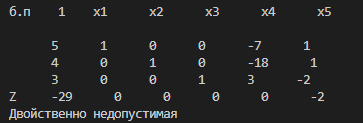


Рис 2. Вывод программы при отсутствии решений

Когда в программе, по алгоритму, создался базис, программа смотрит на Z-строку, проверяет есть ли в ней отрицательные элементы. Когда такой элемент есть программа возвращает ответ “Двойственно недопустимая”

1. Единственное решение

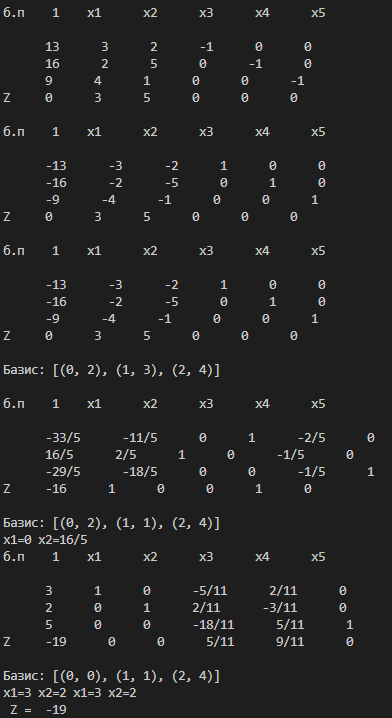


Рис 3. Вывод программы при единственном решении

1. Бесконечно много решений

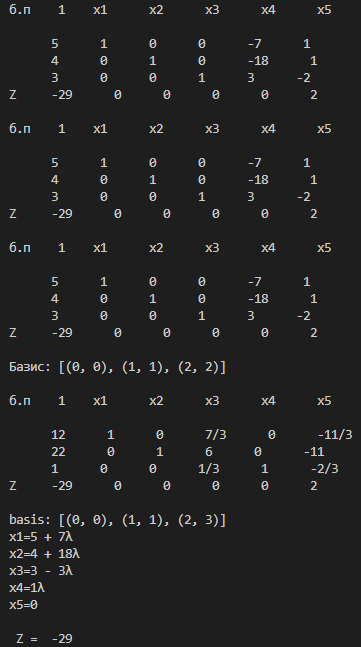


Рис 4. Вывод программы при бесконечном множестве решений

После того как программа нашла одно решение, происходит проверка на то, существует ли множество решений. Алгоритм проверяет строку Z, и если находит то что элемент строки не в базисе, но имеет 0, то алгоритм выполняет еще одну итерацию, и находит множество решений.

**Графическое решение**

(1)

|  |  |  |
| --- | --- | --- |
| X1 | 1 | 3 |
| X2 | 5 | 2 |

(2)

|  |  |  |
| --- | --- | --- |
| X1 | 3 | -2 |
| X2 | 2 | 4 |

(3)

|  |  |  |
| --- | --- | --- |
| X1 | 1 | 2 |
| X2 | 5 | 1 |



Рис 5. Симплекс таблица

|  |  |  |  |
| --- | --- | --- | --- |
| № | X | Значение функции | Точка на графике |
| 1 | [0;0;0;0;0] | 0 | A |
| 2 | [0;16/5;-33/5;0;-29/5] | -16 | B |
| 3 | [3;2;0;0;5] | -19 | C |

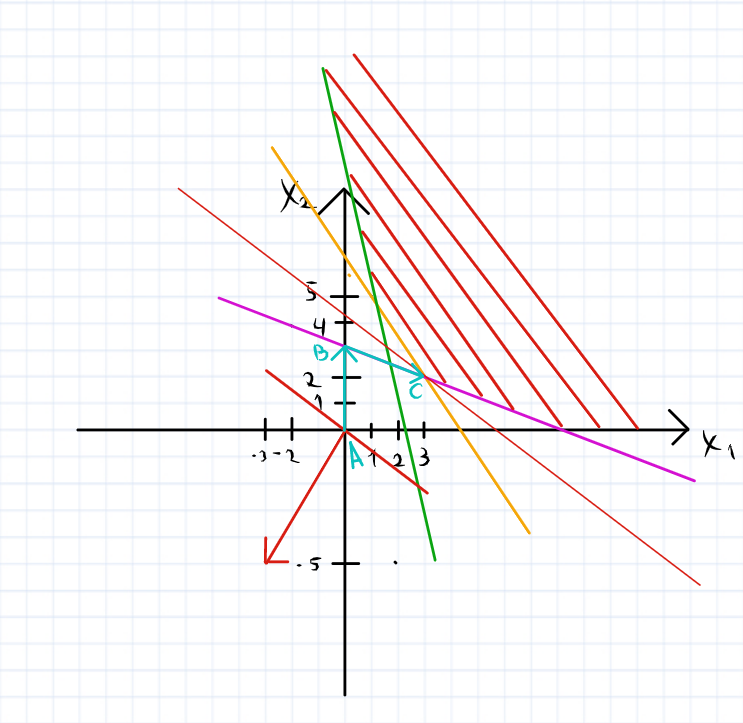


Рис 6. Чертеж графического решения

**Решение двойственной задачи к исходной**

Двойственная задача к исходной:

Теорема равновесия:

**Список используемых источников**

1. Википедия: свободная библиотека: [Электронный ресурс]. URL: <https://en.wikipedia.org/wiki/Simplex_algorithm> (дата обращения: 2.05.2022)
2. Трушков, А. С. Исследование операций. Том 1. Линейное программирование: учебник для вузов / А. С. Трушков. — Санкт-Петербург : Лань, 2022. — 292 с. — ISBN 978-5-8114-8282-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/187580> (дата обращения: 2.05.2022).
3. Stack Overflow: [Электронный ресурс]. URL: https://ru.stackoverflow.com. (Дата обращения: 25.04.2022)

**Код программы**

from my\_fraction import Fraction

basis = []

checkfunction = []

def makeSolutionX(bs, coeff, solutions, matrix):

solution = []

for i in range(0, len(matrix[0])):

solution.append(Fraction(0,1))

for item in bs:

solution[item[1]] = coeff[0][item[0]]

solutions.append(solution)

def findGreatestSolution(solution):

min = Fraction(999)

index = -1

for i in range(0, len(solution[0])):

if solution[0][i] < Fraction(0, 1):

if solution[0][i] < min:

min = solution[0][i]

index = i

return index

def findSolve(solution, matrix, function):

index = findGreatestSolution(solution)

so = Fraction(-999)

i = -1

if(index != -1):

for k in range(0, len(matrix[index])):

if(matrix[index][k] < Fraction(0, 1)):

if function[0][k] / matrix[index][k] > so:

so = function[0][k] / matrix[index][k]

i = k

return index, i

def findNegative(function):

max = Fraction(0)

j = -1

for i in range(0, len(function[0]) - 1):

if Fraction(0, 1) > function[0][i]:

if function[0][i] < max:

max = function[0][i]

j = i

return j

def findPrevious(i):

for item in basis:

if item[0] == i:

return item

def findPreviousByJ(j):

for item in basis:

if item[1] == j:

return item

def rowInBasis(row):

for item in basis:

if item[0] == row:

return True

return False

def columnInBasis(column):

for item in basis:

if item[1] == column:

return True

return False

def createBasis(matrix, solution, function):

for i in range(0, len(matrix)):

if rowInBasis(i):

continue

for j in range(0, len(matrix[i])):

if columnInBasis(j):

continue

basis.append((i, j))

transformationJordan(

basis[len(basis)-1], matrix, solution, function)

break

def transformationJordan(element, matrix, solution, function):

i = element[0]

j = element[1]

elementMatrix = matrix[i][j]

for k in range(0, len(matrix[0])):

matrix[i][k] /= elementMatrix

solution[0][i] /= elementMatrix

for c in range(0, len(matrix)):

if c != i:

newElementMatrix = matrix[c][j]

for k in range(0, len(matrix[0])):

matrix[c][k] -= matrix[i][k]\*newElementMatrix

solution[0][c] -= solution[0][i]\*newElementMatrix

newElementMatrix = function[0][j]

for c in range(0, len(function[0]) - 1):

function[0][c] -= matrix[i][c] \* newElementMatrix

function[0][len(function[0]) - 1] -= solution[0][i]\*newElementMatrix

def isBasis(matrix, column, row):

for i in range(0, len(matrix)):

if i == row:

continue

else:

if matrix[i][column] != Fraction(0, 1):

return False

return True

def findAndCreateBasisVar(matrix, function, solution):

for i in range(0, len(matrix)):

for j in range(0, len(matrix[i])):

if(matrix[i][j] == Fraction(1, 1) or matrix[i][j] == Fraction(-1, 1)):

if isBasis(matrix, j, i) and function[0][j] == Fraction(0, 1):

basis.append((i, j))

if(matrix[i][j] == -1):

for k in range(0, len(matrix[i])):

matrix[i][k] \*= Fraction(-1)

solution[0][i] \*= Fraction(-1)

def printFunction(function):

print("Z", end=" ")

print(function[0][len(function[0]) - 1], end="")

for i in range(0, len(function[0])-1):

print(" ", function[0][i], end="")

def printMatrix(matrix, solution, function):

print("")

print("б.п", end=" ")

print("1", end=" ")

for i in range(0, len(matrix[0])):

print("x" + str(i + 1), end=" ")

print("")

for i in range(0, len(matrix)):

print("\n", end=" ")

print(solution[0][i], end=" ")

for j in range(0, len(matrix[i])):

print(matrix[i][j], end=" ")

print("")

printFunction(function)

print("")

def findSimplexRatio(matrix, function, j):

so = Fraction(999)

i = -1

for k in range(0, len(matrix)):

if matrix[k][j] > Fraction(0,1):

if function[0][k]/matrix[k][j] < so:

so = function[0][k]/matrix[k][j]

i = k

return i

def printSol(solutions, checkfunction):

for i in range(0,len(solutions[0])):

#if checkfunction[i]:

print(f"x{i+1}={solutions[len(solutions)-1][i]}", end=" ")

#print("")

def inBasis(j, bs):

for item in bs:

if item[1] == j:

return True

return False

def checkMultipleSolution(function, bs):

j = -1

for i in range(0,len(function[0])-1):

if function[0][i] == Fraction(0,1) and not(inBasis(i, bs)):

j = i

return j

def printMulSolution(solutions, chFunction):

fsl = []

for i in range(0, len(solutions[0])):

#(1-λ)X1 + λX2 =X1 - λX1 + λX2 = X1 + λ(-X1+X2)

fsl.append(solutions[len(solutions)-2][i]\*Fraction(-1) + solutions[len(solutions) -1][i])

for i in range(0, len(fsl)):

if solutions[len(solutions)-2][i] == Fraction(0):

if fsl[i]==Fraction(0):

print(f"x{i+1}=0")

else:

print(f"x{i+1}={fsl[i]}λ")

elif fsl[i] == Fraction(0):

print(f"x{i+1}={fsl[i]}λ")

elif fsl[i] == Fraction(0) and solutions[len(solutions)-2][i] == Fraction(0):

print(f"x{i+1}=0")

else:

print(f"x{i+1}={solutions[len(solutions)-2][i]}", end="")

if fsl[i] < Fraction(0):

print(f" - {fsl[i]\*Fraction(-1)}λ")

elif fsl[i] > Fraction(0):

print(f" + {fsl[i]}λ")

else:

print("0")

def createMatrix():

with open("2.txt") as f:

functionZ\_in\_file = f.readline()

solution\_in\_file = f.readline()

matrix\_in\_file = f.readlines()

functionZ = []

functionZ.append([Fraction(x) for x in map(lambda item: int(item),

functionZ\_in\_file.split(" "))])

for i in range(0, len(functionZ[0]) - 1):

functionZ[0][i] \*= Fraction(-1)

if functionZ[0][i] != Fraction(0):

checkfunction.append(True)

else:

checkfunction.append(False)

print(functionZ)

matrix = []

for line in matrix\_in\_file:

matrix.append([Fraction(x) for x in map(

lambda item: int(item), line.split(" "))])

solution = []

solution.append([Fraction(x) for x in map(

lambda item: int(item), solution\_in\_file.split(" "))])

printMatrix(matrix, solution, functionZ)

findAndCreateBasisVar(matrix, functionZ, solution)

printMatrix(matrix, solution, functionZ)

if len(basis) != len(matrix):

createBasis(matrix, solution, functionZ)

printMatrix(matrix, solution, functionZ)

if findNegative(functionZ) != -1:

print("Двойственно недопустимая")

return

printMatrix(matrix, solution, functionZ)

print(f"\nБазис: {basis}")

solutions = []

makeSolutionX(basis, solution, solutions, matrix)

while(True):

if findSolve(solution, matrix, functionZ) != (-1, -1):

i, j = findSolve(solution, matrix, functionZ)

if j == -1:

print("Система не ограничена")

break

oldBasisVar = findPrevious(i)

basis.remove(oldBasisVar)

basis.append((i, j))

basis.sort()

transformationJordan((i, j), matrix, solution, functionZ)

makeSolutionX(basis, solution, solutions, matrix)

printMatrix(matrix, solution, functionZ)

print(f"\nБазис: {basis}")

printSol(solutions, checkfunction)

printMatrix(matrix, solution, functionZ)

else:

if checkMultipleSolution(functionZ, basis) != -1:

j = checkMultipleSolution(functionZ, basis)

i = findSimplexRatio(matrix, functionZ, j)

oldBasisVar = findPrevious(i)

basis.remove(oldBasisVar)

basis.append((i, j))

basis.sort()

transformationJordan((i, j), matrix, solution, functionZ)

makeSolutionX(basis, solution, solutions, matrix)

printMatrix(matrix, solution, functionZ)

print(f"\nbasis: {basis}")

printMulSolution(solutions, checkfunction)

print("\n Z = ", (functionZ[0][len(functionZ[0]) - 1]))

else:

printSol(solutions, checkfunction)

print("\n Z = ", (functionZ[0][len(functionZ[0]) - 1]))

break

def main():

createMatrix()

if \_\_name\_\_ == "\_\_main\_\_":

main()

import math

class Fraction:

def \_\_init\_\_(self, numerator=0, denominator=1):

self.numerator = int(numerator)

self.denominator = int(denominator)

def normalize(self):

if self.numerator == 0:

self.denominator = 1

return

gcd = math.gcd(int(self.numerator), int(self.denominator))

self.numerator /= gcd

self.denominator /= gcd

if self.denominator < 0:

self.denominator \*= -1

self.numerator \*= -1

def sign(self):

if self.numerator < 0:

return "-"

return "+"

def neg\_sign(self):

if self.numerator < 0:

return "-"

return ""

def \_\_add\_\_(self, other):

temp = Fraction()

temp.numerator = self.numerator \* other.denominator \

+ self.denominator \* other.numerator

temp.denominator = self.denominator \* other.denominator

temp.normalize()

return temp

def \_\_sub\_\_(self, other):

temp = Fraction()

temp.numerator = self.numerator \* other.denominator \

- self.denominator \* other.numerator

temp.denominator = self.denominator \* other.denominator

temp.normalize()

return temp

def \_\_mul\_\_(self, other):

temp = Fraction()

temp.numerator = self.numerator \* other.numerator

temp.denominator = self.denominator \* other.denominator

temp.normalize()

return temp

def \_\_truediv\_\_(self, other):

temp = Fraction()

temp.numerator = self.numerator \* other.denominator

temp.denominator = self.denominator \* other.numerator

temp.normalize()

return temp

def \_\_eq\_\_(self, other):

if isinstance(other, Fraction):

if self.numerator == other.numerator \

and self.denominator == other.denominator:

return True

elif isinstance(other, int):

if self.numerator == other and self.denominator == 1:

return True

return False

def \_\_ne\_\_(self, other):

return not self.\_\_eq\_\_(other)

def \_\_str\_\_(self):

if self.denominator == 1:

return f"{int(self.numerator)}"

return f"{int(self.numerator)}/{int(self.denominator)}"

def \_\_repr\_\_(self):

if self.denominator == 1:

return f"{int(self.numerator)}"

return f"{int(self.numerator)}/{int(self.denominator)}"

def \_\_lt\_\_(self, other):

if isinstance(other, Fraction):

if self.numerator\*other.denominator < other.numerator\*self.denominator:

return True

return False

def \_\_le\_\_(self, other):

if isinstance(other, Fraction):

if self.numerator\*other.denominator <= other.numerator\*self.denominator:

return True

return False

def \_\_gt\_\_(self, other):

if isinstance(other, Fraction):

if self.numerator\*other.denominator > other.numerator\*self.denominator:

return True

return False

def \_\_ge\_\_(self, other):

if isinstance(other, Fraction):

if self.numerator\*other.denominator >= other.numerator\*self.denominator:

return True

return False