**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине: **«Оптимизация проектных решений»**

на тему: **Симплекс-метод решения задач линейного программирования**

Выполнил: студент гр. ИТП-31

Солодков М.А

Приняла: преподаватель-стажер

Васюкова В.О.

Гомель 2020

**Цель**: изучение симплекс-метода решения задач линейного программирования.

**Ход работы**

**Вариант 6**

# **Задание:**

1. Изучить симплекс-метод решения задач линейного программирования.

2. Разработать программу решения задачи линейного программирования симплекс-методом. Решить задачу ЛП, рассматривая в качестве начального базисного плана план, приведенный в задании.

3. Оформить отчет о выполнении задания с приведением условия задачи, описания алгоритма реализации и фрагментов программ, результатов работы программы и выводов.

Вариант задания приведен на рисунке 15.

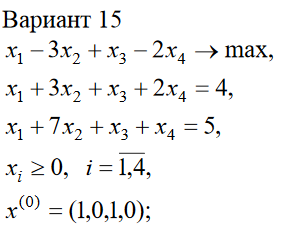


Рисунок 1 – Целевая функция согласно варианту

Код программы приведен в приложении А.

Результат применения симплекс-метода приведен на рисунке 2.

Верификация симплекс-метода приведена на рисунке 3.

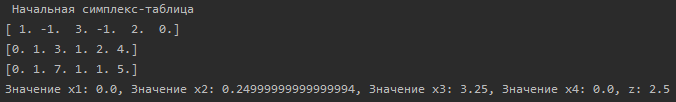


Рисунок 2 – Численный результат нахождения значения целевой функции

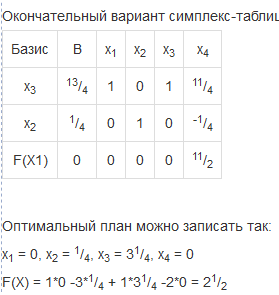


Рисунок 3 – Верификация симплекс-метода

**Вывод:** в результате выполнения лабораторной работы был изучен симплекс-метод решения задач линейного программирования

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Код программы**

import numpy as np

def create\_table\_and\_variables(a, b, f):

number\_of\_variables = len(a[0])

number\_of\_restrictions = len(b)

basis\_indexes = np.arange(number\_of\_variables, number\_of\_variables + number\_of\_restrictions)

table = []

new = np.zeros(number\_of\_restrictions)

for i in range(number\_of\_restrictions):

t = new.copy()

t[i] = 1

table.append([b[i]] + a[i] + t.tolist())

table.append([0] + (-np.array(f)).tolist() + [0] \* number\_of\_restrictions)

table.append([0] \* (number\_of\_variables + 1) + [1] \* number\_of\_restrictions)

table = np.array(table)

for i in range(number\_of\_restrictions):

table[-1] -= table[i]

return table, number\_of\_variables, number\_of\_restrictions, basis\_indexes

def get\_basic\_plan(table, number\_of\_variables, number\_of\_restrictions, basis\_indexes):

basic\_plan = [0] \* (number\_of\_variables + number\_of\_restrictions)

for i, j in zip(basis\_indexes, range(number\_of\_restrictions)):

basic\_plan[i] = table[j, 0]

return basic\_plan

def is\_optimal\_solution(table, number\_of\_variables):

return (table[-1][1 : number\_of\_variables + 1] >= 0).all() and (table[-2][1 : number\_of\_variables + 1] >= 0).all()

def get\_permutations(table, number\_of\_variables, number\_of\_restrictions):

k = -2

if (table[-1] >= 0).all():

k = -3

in\_idx = np.argmin(table[k + 1, 1:number\_of\_variables + 1])

mask = table[:number\_of\_restrictions, in\_idx + 1] > 0

out\_idx = np.argmin(table[:number\_of\_restrictions, 0][mask] / table[:number\_of\_restrictions, in\_idx + 1][mask])

if (table[-1] >= 0).all() and not mask[0]:

out\_idx += len(table[:number\_of\_restrictions, 0]) - sum(mask)

return in\_idx, out\_idx

def recalculation(table, in\_idx, out\_idx):

leading\_column = in\_idx + 1

leading\_row = out\_idx

for i in range(len(table)):

if i != leading\_row:

table[i] = table[i] - np.divide(table[i][leading\_column], table[leading\_row][leading\_column],

out=np.zeros\_like(table[leading\_row][leading\_column]),

where=table[leading\_row][leading\_column] != 0) \* table[leading\_row]

table[leading\_row] = table[leading\_row] / table[leading\_row][leading\_column]

return table

def solve(a, b, f):

table, number\_of\_variables, number\_of\_restrictions, basis\_indexes = create\_table\_and\_variables(a, b, f)

basic\_plan = get\_basic\_plan(table, number\_of\_variables, number\_of\_restrictions, basis\_indexes)

print("Начальная таблица")

print(table)

number = 1

while not is\_optimal\_solution(table, number\_of\_variables):

in\_idx, out\_idx = get\_permutations(table, number\_of\_variables, number\_of\_restrictions)

basis\_indexes[out\_idx] = in\_idx

table = recalculation(table, in\_idx, out\_idx)

basic\_plan = get\_basic\_plan(table, number\_of\_variables, number\_of\_restrictions, basis\_indexes)

print("\n\nШаг ", number)

print("Таблица")

print(np.round(table, 3))

print("Базисный план")

print(np.round(basic\_plan, 3))

number += 1

return basic\_plan

def get\_value(f, basic\_plan):

s = 0

for i, j in zip(f, basic\_plan):

s += i \* j

return s

a = [[1, 2, 2, 1],

[2, 0, -2, 1]]

b = [5, 1]

f = [1, 3, 2, 1]

basic\_plan = solve(a, b, f)

print("\nЗначение целевой функции:", get\_value(f, basic\_plan))