**Министерство образования Российской Федерации**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**им. Н.Э. БАУМАНА**

Факультет: Информатика и системы управления

Кафедра: Информационная безопасность (ИУ8)

**МЕТОДЫ ОПТИМИЗАЦИИ**

**Лабораторная работа №1 на тему:**

«Постановка задачи линейного программирования»

Вариант 25

**Преподаватель:**

Коннова Н.С.

**Студент:**

Урнышева А. А.

**Группа:**

ИУ8-34

Москва 2024

**Цель работы**

Изучение симплекс-метода решения задачи линейного программирования (ЛП).

**Постановка задачи**

Требуется найти решение следующей задачи линейного программирования (ЛП):

F = **cx** → min,

**Ax ≤ b,**

**x ≥ 0.**

Здесь **x =** [*x1, x2, …, xn*]*T*- искомый вектор решения;

**c** = [*c1, c2, …, cn*]*T*- вектор коэффициентов целевой функции (ЦФ) F;

матрица системы ограничений

**b** = [*b1, b2, …, bm*]*T*- вектор правой части системы ограничений.

В развернутой форме задача имеет вид:

Задачу ЛП требуется записать в *канонической* форме.

Затем следует решить задачу ЛП симплекс-методом, продемонстрировав на каждом этапе все преобразования симплекс-таблиц.

Получив оптимальное решение, необходимо выполнить его проверку подстановкой.

**Ход работы**

Начальные условия:

**c:**

[8 6 2]

**A:**

[[2 1 1]

[ 1 4 0 ]

[ 0 0,5 1 ]]

**b:**

[4 3 6]

Решим следующую задачу ЛП:

Используя фиктивные переменные приведем исходную задачу к каноническому виду:

Пусть – базисные переменные, а – свободные переменные.

Тогда имеем:

Исходная симплекс-таблица:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Si0*** |  |  |  |
|  | 4 | 2 | 1 | 1 |
|  | 3 | 1 | 4 | 0 |
|  | 6 | 0 | 0.5 | 1 |
| ***F*** | 0 | 8 | 6 | 2 |

В столбце свободных членов нет отрицательных элементов, поэтому имеем опорное решение:

Целевая функция

*F* = 0

Анализируем последнюю строку симплекс-таблицы и ищем первый положительный элемент: 8 в столбце (разрешающий столбец).

Найдем минимальное положительное отношение элемента свободных членов ***Si0*** к соответствующему элементу в разрешающем столбце:

Следовательно, – разрешающая строка.

Заменим базисную переменную на свободную . Пересчитаем все коэффициенты по формулам:

Где столбец с номером *k* – **разрешающий столбец**, строка с номером *r* – **разрешающая строка**, элемент **разрешающий элемент**.

Расчеты:

1. Элементы разрешающей строки:
2. Элементы разрешающего столбца:
3. Остальные элементы:

Строка *x5*:

Строка *x6*:

Строка *F*:

Получим преобразованную симплекс-таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Si0*** |  |  |  |
|  | 2 | 0.5 | 0.5 | 0.5 |
|  | 1 | -0.5 | 3.5 | -0.5 |
|  | 6 | 0 | 0.5 | 1 |
| ***F*** | -16 | -4 | 2 | -2 |

В последней строке полученной таблицы все еще есть положительный элемент, а значит, найденное решение не оптимально. В столбце в последней строке находится положительный элемент 2, значит этот столбец разрешающий. Найдем разрешающую строку:

Следовательно, – разрешающая строка, а 3.5 – разрешающий элемент.

В таблице заменим базисную переменную на свободную . Пересчитаем все коэффициенты тем же способом, что и в предыдущей итерации:

1. Элементы разрешающей строки:
2. Элементы разрешающего столбца:
3. Остальные элементы:

Строка *x1*:

Строка *x6*:

Строка *F*:

Получим преобразованную симплекс-таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Si0*** |  |  |  |
|  | 1.86 | 0.57 | -0.14 | 0.57 |
|  | 0.29 | -0.14 | 0.29 | -0.14 |
|  | 5.86 | 0.07 | -0.14 | 1.07 |
| ***F*** | -16.57 | -3.71 | -0.57 | -1.71 |

Поскольку в последней строке больше нет положительных коэффициентов, кроме свободного члена, имеем оптимальное решение:

Целевая функция

*F* = -16.57

Но так как по заданию , а мы нашли ее минимальное значение, то в качестве ответа к задаче следует написать, что искомое значение

**Выводы**

В данной лабораторной работе мы решили прямую задачу линейного программирования с помощью симплекс-метода. Также данная задача была решена на языке Python. Предусмотрены три возможных исхода:

1. В столбце свободных членов есть отрицательный элемент, но в строке с данным отрицательным элементом все оставшиеся элементы положительны (или равны 0) – **решений нет;**
2. Все отношения свободных членов к элементам найденного разрешающего столбца, стоящих в соответствующих строках отрицательны (или 0) => нет минимального неотрицательного отношения – **решений бесконечно много;**
3. Найдено оптимальное решение.

*Файл ‘MO lab1.py’.*

# Проверка корректности данных

def check\_simplex\_table(c, A, b):

len\_A = len(A[0]) # Количество переменных

# Проверяет, что все строки матрицы A имеют одинаковую длину и содержат допустимые значения

for row in A:

if len(row) != len\_A:

return False # Длина строки не соответствует ожидаемой

# Проверяем, что длины векторов c и b соответствуют количеству строк в A

if (len(c) != len\_A) or (len(b) != len(A)):

return False

# Проверка пройдена успешно

return True

# Проверяем, существуют ли решения симплекс-метода

def check\_simplex\_response(c, A, b):

if c.count(0) == len(c):

return False # Все коэффициенты равны нулю

for row in range(len(b)):

if b[row] < 0: # Если есть отрицательный элемент в b

for col in range(len(A[0])):

if min(A[row]) >= 0:

return False # Нет подходящих коэффициентов

return True # Существуют отрицательные коэффициенты

# Проверяем корректность решения, полученного симплекс-методом

def check\_simplex\_answer(c, b, f, var\_row, var\_col):

check\_f = 0

for i in range(len(var\_row)):

if var\_row[i] in var\_col:

check\_f += b[i] \* c[var\_col.index(var\_row[i]) - 1]

if round(check\_f, 2) == round(f, 2):

return True

return False

# Создаем симплекс-таблицу

def create\_simplex\_table(c, A, b, f):

table = []

# Формируем таблицу, добавляя строки с b и A

for i in range(len(A)):

table.append([b[i]] + A[i])

# Добавляем строку с коэффициентами c

table.append([f] + c)

return table

# Создаем имена переменных для симплекс-метода

def create\_simplex\_variables(A):

var\_col = ["Si0"] + [f"x{i+1}" for i in range(len(A[0]))]

var\_row = [f"x{i+len(var\_col)}" for i in range(len(A))] + ["F "]

return var\_row, var\_col

# Выводим симплекс-таблицу

def print\_simplex\_table(simplex\_table, var\_row, var\_col):

print()

# Определяем максимальную ширину для форматирования

# max\_width = max(len(str(float(j))) for row in simplex\_table for j in row) + 2

max\_width = 6

# Выводим заголовки

headers = [var\_col[i] for i in range(len(simplex\_table[0]))]

print(" ", " | ".join(f"{header:>{max\_width}}" for header in headers))

print("----", "-" \* (max\_width \* len(headers) + 4 \* (len(headers) - 1)), sep="")

for i in range(len(simplex\_table)):

print(var\_row[i], end=" | ")

for j in simplex\_table[i]:

# Выравнивание по правому краю, 2 знака после запятой

if j == 0:

print(f"{float(0):>{max\_width}.2f}", end=" | ")

else:

print(f"{round(float(j), 2):>{max\_width}.2f}", end=" | ")

print()

# Находим разрешающий элемент. Выводим информацию о минимальном отношении

def find\_simplex\_resolve(c, A, b):

if check\_simplex\_response(c, A, b):

# Проверяем, корректна ли симплекс-таблица

for row in range(len(b)):

if b[row] < 0:

for col in range(len(A[0])):

if A[row][col] < 0:

# Возвращаем минимальное отношение для данного столбца

try:

return find\_min\_ratio(A, b, col)

except:

return ["inf"]

c\_max\_value = max(c)

if c\_max\_value < 0:

# Если максимальное значение меньше нуля

return ["not"]

c\_max\_index = c.index(c\_max\_value)

try:

return find\_min\_ratio(A, b, c\_max\_index)

except:

return ["inf"]

else:

return ["not"]

# Находим минимальное отношение для заданного столбца симплекс-таблицы

def find\_min\_ratio(A, b, min\_ratio\_col):

min\_ratio = float("inf")

min\_ratio\_row = -1

for row in range(len(A)):

if A[row][min\_ratio\_col] == 0:

continue

ratio = b[row] / A[row][min\_ratio\_col]

# Обновляем минимальное отношение, если нашли новое

if (ratio > 0) and (ratio < min\_ratio):

min\_ratio = ratio

min\_ratio\_row = row

# Если не найден подходящий индекс, выбрасываем ошибку

if min\_ratio\_row == -1:

raise ValueError("[ ! ] Нет допустимого разрешающего элемента.")

return [A[min\_ratio\_row][min\_ratio\_col], min\_ratio\_row, min\_ratio\_col]

# Меняем местами эл-ты в двух списках на основе указаний, полученных из решения симплекс-метода

def swap\_variables(var\_row, var\_col, simplex\_resolve):

var\_row[simplex\_resolve[1]], var\_col[simplex\_resolve[2] + 1] = var\_col[simplex\_resolve[2] + 1], var\_row[simplex\_resolve[1]]

return var\_row, var\_col

# Функция для выполнения одной итерации симплекс-метода

def simplex\_table\_iteration(c, A, b, f, simplex\_resolve):

# Получаем значение разрешающего элемента

new\_simplex\_resolve = 1 / simplex\_resolve[0]

new\_c = [0] \* len(c) # Инициализируем новый вектор коэффициентов целевой функции

new\_b = [0] \* len(b) # Инициализируем новый вектор правых частей

new\_A = [[0 for \_ in range(len(A[0]))] for \_ in range(len(A))] # Инициализируем новую матрицу ограничений

# Заполняем колонну A

for i in range(len(A)):

if i == simplex\_resolve[1]: # Текущая строка разрешающего элемента

new\_A[i][simplex\_resolve[2]] = new\_simplex\_resolve

else: # Остальные строки

new\_A[i][simplex\_resolve[2]] = (A[i][simplex\_resolve[2]] / simplex\_resolve[0] \* -1)

# Обновляем коэффициенты целевой функции для разрешающего столбца

new\_c[simplex\_resolve[2]] = c[simplex\_resolve[2]] / simplex\_resolve[0] \* -1

# Заполняем строку A для разрешающего элемента

for i in range(len(A[0])):

if i == simplex\_resolve[2]:

continue

new\_A[simplex\_resolve[1]][i] = A[simplex\_resolve[1]][i] / simplex\_resolve[0]

# Обновляем вектор правых частей для разрешающего элемента

new\_b[simplex\_resolve[1]] = b[simplex\_resolve[1]] / simplex\_resolve[0]

# Обновляем остальные коэффициенты целевой функции

for i in range(len(c)):

if i == simplex\_resolve[2]:

continue

new\_c[i] = c[i] - (A[simplex\_resolve[1]][i] \* c[simplex\_resolve[2]]) / (simplex\_resolve[0])

# Обновляем вектор правых частей для остальных строк

for i in range(len(b)):

if i == simplex\_resolve[1]:

continue

new\_b[i] = b[i] - ((A[i][simplex\_resolve[2]] \* b[simplex\_resolve[1]]) / simplex\_resolve[0])

# Обновляем матрицу ограничений

for i in range(len(A)):

for j in range(len(A[0])):

if (i == simplex\_resolve[1]) or (j == simplex\_resolve[2]):

continue

new\_A[i][j] = A[i][j] - ((A[i][simplex\_resolve[2]] \* A[simplex\_resolve[1]][j]) / simplex\_resolve[0])

# Обновляем значение целевой функции

new\_f = f - ((c[simplex\_resolve[2]] \* b[simplex\_resolve[1]]) / simplex\_resolve[0])

return new\_c, new\_A, new\_b, new\_f

# Основная функция симплекс-метода

# Выполняет проверки входных данных и находит решение

def simplexsus(c, A, b, f, minimize):

# Проверка условий

if check\_simplex\_table(c, A, b):

print("[ + ] Check: OK")

old\_b = b.copy()

# Инвертируем c, если ищем минимум

if minimize:

for i in range(len(c)):

c[i] \*= -1

var\_row, var\_col = create\_simplex\_variables(A) # Создание обозначений симплекс-таблицы

old\_var\_row = var\_row.copy()

while (max(c) > 0) or (min(b) < 0):

simplex\_table = create\_simplex\_table(c, A, b, f) # Создание симплекс-таблицы

print\_simplex\_table(simplex\_table, var\_row, var\_col) # Вывод симплекс-таблицы

simplex\_resolve = find\_simplex\_resolve(c, A, b) # Поиск и выбор разрешающего элемента

# Обработка результатов нахождения разрешающего элемента

if simplex\_resolve == ["not"]:

print("[ - ] There's no answer")

return 1

if simplex\_resolve == ["inf"]:

print("[ - ] Infinite number of solutions")

return 1

print("[ \* ] The resolving element is found:", round(simplex\_resolve[0], 2), simplex\_resolve[1:])

var\_row, var\_col = swap\_variables(var\_row, var\_col, simplex\_resolve)

c, A, b, f = simplex\_table\_iteration(c, A, b, f, simplex\_resolve)

# Найдено оптимальное решение

print("\n[ + ] OPTI ANS")

simplex\_table = create\_simplex\_table(c, A, b, f) # Создание симплекс-таблицы

print\_simplex\_table(simplex\_table, var\_row, var\_col) # Вывод симплекс-таблицы

else:

print("[ - ] Check: BAD TABLE")

return 1

if check\_simplex\_answer(c, old\_b, f, old\_var\_row, var\_col):

if minimize:

print("[ \* ] The function goes to the minimum")

return round(f, 2)

else:

print("[ \* ] The function goes to the maximum")

return round(f \* -1, 2)

print("[ - ] Check: BAD ANS")

return -1

# Инициализация значений и вывод решения симплекс-метода

def main():

minimize = False # Необходимость минимизации

c = [8, 6, 2] # Коэффициенты целевой функции

A = [[2, 1, 1], [1, 4, 0], [0, 0.5, 1]] # Ограничения

b = [4, 3, 6] # Правая часть ограничений

f = 0

print("[ + ] Ans:", simplexsus(c, A, b, f, minimize))

return 0

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

main()

**Вывод программы**

[ + ] Check: OK

Si0 | x1 | x2 | x3

----------------------------------------

x4 | 4.00 | 2.00 | 1.00 | 1.00 |

x5 | 3.00 | 1.00 | 4.00 | 0.00 |

x6 | 6.00 | 0.00 | 0.50 | 1.00 |

F | 0.00 | 8.00 | 6.00 | 2.00 |

[ \* ] The resolving element is found: 2 [0, 0]

Si0 | x4 | x2 | x3

----------------------------------------

x1 | 2.00 | 0.50 | 0.50 | 0.50 |

x5 | 1.00 | -0.50 | 3.50 | -0.50 |

x6 | 6.00 | 0.00 | 0.50 | 1.00 |

F | -16.00 | -4.00 | 2.00 | -2.00 |

[ \* ] The resolving element is found: 3.5 [1, 1]

[ + ] OPTI ANS

Si0 | x4 | x5 | x3

----------------------------------------

x1 | 1.86 | 0.57 | -0.14 | 0.57 |

x2 | 0.29 | -0.14 | 0.29 | -0.14 |

x6 | 5.86 | 0.07 | -0.14 | 1.07 |

F | -16.57 | -3.71 | -0.57 | -1.71 |

[ \* ] The function goes to the maximum

[ + ] Ans: 16.57

Ссылка на репозиторий с кодом и отчетом: [ArinaUrnysheva/MO-lab1](https://github.com/ArinaUrnysheva/MO-lab1)