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 18

Преподаватель:
Коннова Н.С.

Студент:
Ожогин М.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):

$$F = cx \to max,$$
$$Ax \leq b,$$
$$x \geq 0$$

Здесь $x = [x_1, x_2, \ldots, x_n]^T$ – искомый вектор решения;

$c = [c_1, c_2, \ldots, c_n]$ – вектор коэффициентов целевой функции (ЦФ) F;

$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$ – матрица системы ограничений;

$b = [b_1, b_2, \ldots, b_m]^T$ – вектор правой части системы ограничений.

$$F = \sum_{i=1}^{n} c_i x_i \to max,$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2, \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m, \end{cases}$$

$$x_i \geq 0, i = 1, \ldots, n.$$

Ход работы

Решим следующую задачу ЛП:

$$F = 7x_1 + 7x_2 + 6x_3 \to max,$$

$$\begin{cases} 2x_1 + x_2 + x_3 \leq 8, \\ x_1 + 2x_2 \leq 2, \\ 0{,}5x_2 + 4x_3 \leq 6, \end{cases}$$

$$x_1, x_2, x_3 \geq 0$$

Используя фиктивные переменные x_4, x_5, x_6 приведем исходную задачу к каноническому виду:

$$F = 7x_1 + 7x_2 + 6x_3 \to max,$$

$$\begin{cases} 2x_1 + x_2 + x_3 + x_4 = 8, \\ x_1 + 2x_2 + x_5 = 2, \\ 0{,}5x_2 + 4x_3 + x_6 = 6, \end{cases}$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

Пусть x4, x5, x6 – базисные переменные, x1, x2 – свободные переменные.

Тогда решение будет:

	b	x1	x2	x3
x4	8.00	2.00	1.00	1.00
x5	2.00	1.00	2.00	0.00
x6	6.00	0.00	0.50	4.00
F	0.00	7.00	7.00	6.00

```
Resolving element located: [1, 1, 0]
```

	b	x5	x2	x3
x4	4.00	-2.00	-3.00	1.00
x1	2.00	1.00	2.00	0.00
x6	6.00	-0.00	0.50	4.00
F	-14.00	-7.00	-7.00	6.00

```
Resolving element located: [4.0, 2, 2]

Optimal Solution Found:
```

	b	x5	x2	x6
x4	2.50	-2.00	-3.12	-0.25
x1	2.00	1.00	2.00	-0.00
x3	1.50	-0.00	0.12	0.25
F	-23.00	-7.00	-7.75	-1.50

```
The function is maximized.
Answer: 23.0
```

Выполним проверку полученных значений:

$$F = 7 * 2 + 7 * 0 + 6 * 1{,}5 = 23,$$

$$\begin{cases} 2 * 2 + 1 * 0 + 1 * 1{,}5 \leq 8, \\ 2 + 2 * 0 \leq 2, \\ 0{,}5 * 0 + 4 * 1{,}5 \leq 6, \end{cases}$$

$$F = 23 = 23,$$

$$\begin{cases} 5{,}5 \le 8, \\ 2 \le 2, \\ 6 \le 6, \end{cases}$$

Все неравенства верны, тогда итоговый ответ будет:

x = [2, 0, 1,5],

F = 23

Вывод

В ходе выполнения работы была изучена теория и практика применения симплекс-метода для решения задач линейного программирования. Симплекс-метод позволяет эффективно находить оптимальное решение задачи ЛП при наличии ограничений, представленных в виде линейных неравенств. В процессе работы были рассмотрены ключевые этапы алгоритма симплекс-метода: приведение задачи к канонической форме, построение симплекс-таблиц и выполнение итераций для нахождения оптимального решения. Полученные результаты подтверждают эффективность симплекс-метода для решения задач ЛП и его широкое применение в различных областях, таких как экономика, логистика и управление ресурсами.

Приложение.

Файл “main.py”:

```
from simp_solv import execute_simplex

def main():
    minimize = False
    c = [7, 7, 6]
    A = [[2, 1, 1],
         [1, 2, 0],
         [0, 0.5, 4]]
    b = [8, 2, 6]
    f = 0
    print("Answer:", execute_simplex(c, A, b, f, minimize))


if __name__ == "__main__":
    main()
```

Файл “simp_solv.py”:

```
columns = []
rows = []

def init_headers(c, A, b):
    num_columns = len(c)
    num_rows = len(b)
    global columns
    global rows
    columns = ['b'] + [f'x{i + 1}' for i in range(num_columns)]
    rows = []
    for i in range(num_rows):
        temp_row = [f'x{i + num_columns + 1}']
        rows.append(temp_row)
    rows.append('F')

def validate_simplex_input(c, A, b):
    len_A = len(A[0])
    return (all(len(row) == len_A for row in A) and
            len(c) == len_A and
            len(b) == len(A))

def check_solution_existence(c, A, b):
    if all(x == 0 for x in c):
        return False
    return all(b[row] >= 0 or min(A[row]) < 0 for row in range(len(b)))

def construct_simplex_table(c, A, b, f):
    table = []
    for i in range(len(A)):
        table.append([b[i]] + A[i])
    table.append([f] + c)
    return table

def format_header_value(value):
    return " ".join(str(v) for v in value) if isinstance(value, list) else str(value)

def display_simplex_table(simplex_table):
    print()
    formatted_columns = [format_header_value(col) for col in columns]
```

```
    formatted_rows = [format_header_value(row) for row in rows]
    max_width = max(len(str(float(j))) for row in simplex_table for j in row if
isinstance(j, (int, float))) + 2
    full_headers = [""] + formatted_columns  # Добавляем 'b' перед заголовками
столбцов
    print(" | ".join(f"{header:>{max_width}}" for header in full_headers))
    print("-" * (max_width * len(full_headers) + 3 * (len(full_headers) - 1)))
    for i, row in enumerate(simplex_table):
        row_values = [formatted_rows[i]] + list(row)  # Добавляем заголовок для
строки 'b'
        print(" | ".join(
            f"{float(j):>{max_width}.2f}" if isinstance(j, (int, float)) else
str(j).ljust(max_width) for j in
            row_values))

def locate_resolving_element(c, A, b):
    if not check_solution_existence(c, A, b):
        return ["not_er"]
    for row in range(len(b)):
        if b[row] < 0:
            for col in range(len(A[0])):
                if A[row][col] < 0:
                    try:
                        return calculate_min_ratio(A, b, col)
                    except:
                        return ["inf_er"]
    if max(c) < 0:
        return ["not_er"]
    c_max_index = c.index(max(c))
    return calculate_min_ratio(A, b, c_max_index)

def calculate_min_ratio(A, b, ratio_col):
    min_ratio = float("inf")
    min_ratio_row = -1
    for row in range(len(A)):
        if A[row][ratio_col] == 0:
            continue
        ratio = b[row] / A[row][ratio_col]
        if 0 < ratio < min_ratio:
            min_ratio = ratio
            min_ratio_row = row
    if min_ratio_row == -1:
        raise ValueError("No valid resolving element found.")
    return [A[min_ratio_row][ratio_col], min_ratio_row, ratio_col]

def perform_simplex_iteration(c, A, b, f, res_el):
    global columns, rows
    new_resolving_element = 1 / res_el[0]
    new_b = [0] * len(b)
    new_A = [[0] * len(A[0]) for _ in A]
    for i in range(len(A)):
        new_A[i][res_el[2]] = (
            new_resolving_element if i == res_el[1]
            else A[i][res_el[2]] / res_el[0] * -1
        )

    new_c = [0] * len(c)
    new_c[res_el[2]] = c[res_el[2]] / res_el[0] * -1
    for i in range(len(A[0])):
        if i != res_el[2]:
            new_A[res_el[1]][i] = A[res_el[1]][i] / res_el[0]
```

```
    new_b[res_el[1]] = b[res_el[1]] / res_el[0]
    for i in range(len(c)):
        if i == res_el[2]:
            continue
        new_c[i] = c[i] - (A[res_el[1]][i] * c[res_el[2]]) / (res_el[0])

    for i in range(len(b)):
        if i == res_el[1]:
            continue
        new_b[i] = b[i] - ((A[i][res_el[2]] * b[res_el[1]]) / res_el[0])

    for i in range(len(A)):
        for j in range(len(A[0])):
            if (i == res_el[1]) or (j == res_el[2]):
                continue
            new_A[i][j] = A[i][j] - ((A[i][res_el[2]] * A[res_el[1]][j]) / res_el[0])

    new_f = f - ((c[res_el[2]] * b[res_el[1]]) / res_el[0])
    temp_row = rows[res_el[1]]
    rows[res_el[1]] = columns[1+res_el[2]]
    columns[1+res_el[2]] = temp_row
    return new_c, new_A, new_b, new_f

def execute_simplex(c, A, b, f, minimize):
    if validate_simplex_input(c, A, b):
        print("Input Validation: Passed")
        init_headers(c, A, b)
        if minimize:
            c = [-x for x in c]
        while (max(c) > 0) or (min(b) < 0):
            simplex_table = construct_simplex_table(c, A, b, f)
            display_simplex_table(simplex_table)
            resolving_element = locate_resolving_element(c, A, b)
            if resolving_element == ["not"]:
                print("No feasible solution exists.")
                return 1
            if resolving_element == ["inf"]:
                print("Infinite solutions detected.")
                return 1
            print("Resolving element located:", resolving_element)
            c, A, b, f = perform_simplex_iteration(c, A, b, f, resolving_element)

        print("\nOptimal Solution Found:")
        simplex_table = construct_simplex_table(c, A, b, f)
        display_simplex_table(simplex_table)
    else:
        print("Input Validation: Failed")
        return 1

    if minimize:
        print("The function is minimized.")
        return f
    print("The function is maximized.")
    return f * -1
```