МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6

по дисциплине: Исследование операций

тема: «Нахождение седловой точки в смешанных стратегиях для матричной игры с нулевой суммой»

Выполнил: ст. группы ПВ-233
Мороз Роман Алексеевич

Проверил:
Вирченко Юрий Петрович

Белгород 2025 г.

Цель работы: освоить метод нахождения седловой точки в смешанных стратегиях с помощью построения пары двойственных задач ЛП.

Задания для подготовки к работе

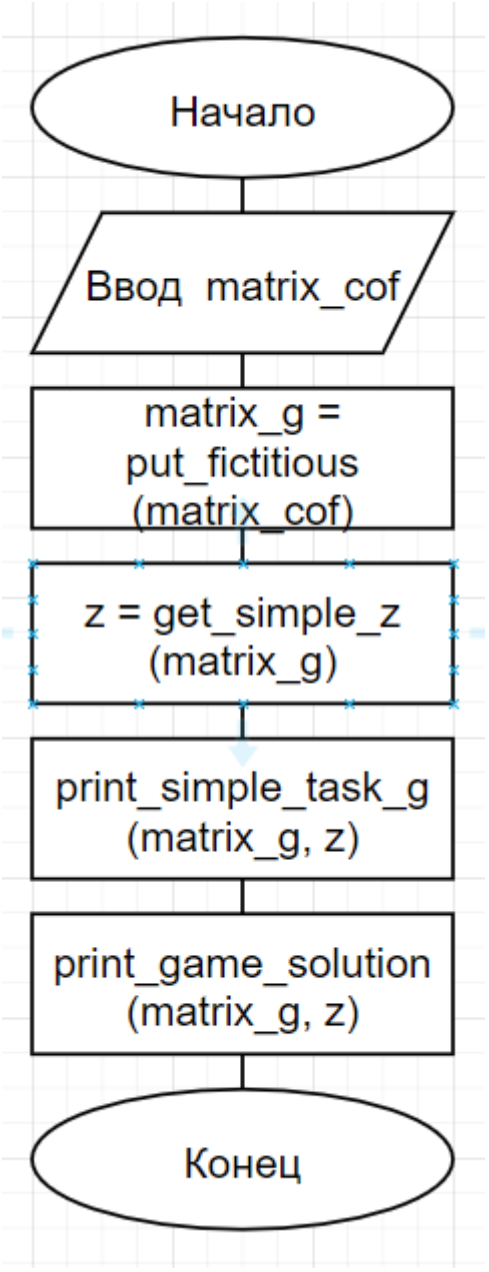
1. Изучить основные понятия теории матричных игр двух игроков с нулевой суммой, анализ игры в чистых стратегиях, понятие смешанной стратегии и седловой точки в смешанных стратегиях, а также метод нахождения седловой точки в смешанных стратегиях с помощью построения пары двойственных задач ЛП.
2. Составить и отладить программу для нахождения седловой точки игры с помощью решения пары симметрично двойственных задач ЛП.
3. Для подготовки тестовых данных решить вручную одну из следующих ниже задач.

Вариант- 9

9.

$$\begin{pmatrix} 2 & 5 \\ 7 & 9 \\ 10 & 7 \\ 3 & 4 \\ 8 & 15 \end{pmatrix}$$

Блок-схема программы:



Листинг программы:

```
# Функция получает на вход таблицу и определяет индекс опорного столбца.
def get_op_st(table):
    # Изначально устанавливается индекс первого столбца после базисных переменных.
    max_ = 2
    # Затем происходит итерация по столбцам начиная с третьего (индекс 2).
    for i in range(2, len(table[0])):
        # Если значение элемента строки с индексом, равным последней строке
        # таблицы (целевая функция),
        # меньше нуля, и при этом меньше или равно значению элемента с индексом
        # max_, то max_ принимает значение i.
        if table[len(table) - 1][i] < 0:
            if table[len(table) - 1][i] <= table[len(table) - 1][max_]:
                max_ = i
    # В конце функция возвращает индекс опорного столбца.
    return max_


# Функция для выполнения следующего шага симплекс-метода
def next_step(table):
    # Получаем индекс опорного столбца
    index_min = get_op_st(table)
    index_change = 0
    min_ = 9999999
    # Проходим по всем строкам кроме первой и последней
    for i in range(1, len(table) - 1):
        # Если значение элемента в опорном столбце больше 0
        if table[i][index_min] > 0:
            # Если значение элемента в столбце свободных членов делённое на
            # значение элемента в опорном столбце меньше min_
            if table[i][1] / table[i][index_min] < min_:
                min_ = table[i][1] / table[i][index_min]
                index_change = i
    # Если индекс изменения равен 0, выводим сообщение об ошибке и завершаем
    # программу с кодом 5
    if index_change == 0:
        print("Функция не ограничена")
        exit(5)
    # Вычисляем делитель
    delit = table[index_change][index_min]
    # Делим все элементы строки с индексом index_change на делитель
    for i in range(1, len(table[index_change])):
        table[index_change][i] /= delit
    # Создаем новую таблицу на основе старой
    new_table = [[y for y in x] for x in table]
    # Заменяем метку опорной переменной
    new_table[index_change][0] = 'X' + str(index_min - 1)
    # Обновляем остальные элементы таблицы
    for i in range(1, len(table)):
        if i != index_change:
            cof = -table[i][index_min]
            for j in range(1, len(table[i])):
                new_table[i][j] = table[i][j] + table[index_change][j] * cof
    return new_table


# Функция для печати таблицы
def print_table(table):
    # Печатаем границу таблицы
    print('=' * 120)
    doc_table = []
    # Проходим по каждой строке таблицы
    for s in table:
        row = []
```

```
        # Проходим по каждому элементу строки
        for el in s:
            # Если элемент является числом с плавающей точкой, округляем его до
            # трёх знаков после запятой
            if type(el) is float:
                el = str(round(el, 3))
            # Выводим элемент с отступом в 6 символов
            print("%-6s" % el, end=" ")
            row.append(el)
        # Переходим на следующую строку
        print()
        doc_table.append(row)
    # Печатаем границу таблицы
    print('=' * 120)

    # Записываем заголовки столбцов таблицы
    cols = doc_table[0]
    doc_table = doc_table[1:]

    # Заполняем заголовки столбцов таблицы
    for j in range(len(cols)):
        table.cell(0, j).text = cols[j]

    # Заполняем таблицу данными
    for i, row in enumerate(doc_table):
        for j, text in enumerate(row):
            # Получаем ячейку таблицы
            cell = table.cell(i + 1, j)
            # Записываем в ячейку данные
            cell.text = str(text)


# Функция добавляет фиктивные переменные
def put_fictitious(matrix):
    # Создаем новую матрицу, расширенную на количество фиктивных переменных
    new_matrix = [[0.0 for _ in range(len(matrix[0]) + len(matrix))]
                       for _ in range(len(matrix))]
    # Заполняем новую матрицу
    for i in range(len(matrix)):
        for j in range(len(matrix[0]) - 1):
            # Если элемент столбца свободных членов меньше нуля, меняем знак
            # элементов строки
            if matrix[i][len(matrix[0]) - 1] < 0:
                new_matrix[i][j] = -matrix[i][j]
            else:
                new_matrix[i][j] = matrix[i][j]
    # Добавляем в новую матрицу фиктивные переменные
    for i in range(len(matrix)):
        new_matrix[i][len(matrix[0]) + i - 1] = 1.0
    # Заполняем столбец свободных членов новой матрицы
    for i in range(len(matrix)):
        if matrix[i][len(matrix[0]) - 1] < 0:
            new_matrix[i][len(new_matrix[0]) - 1] = -matrix[i][len(matrix[0]) - 1]
        else:
            new_matrix[i][len(new_matrix[0]) - 1] = matrix[i][len(matrix[0]) - 1]
    return new_matrix


# Функция печатает коэффициенты целевой функции
def print_fg(f):
    print("f= ", end="")
    print(f[0], end=" ")
    for i in range(1, len(f)):
        if f[i] >= 0:
            print("+", end="")
        print(f[i], end="*y")
```

```
        print(i, end=" ")
    print("--> max")


# Функция печатает условия задачи
def print_g_task(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[0]) - 1):
            if matrix[i][j] >= 0:
                print("+", end="")
            print(matrix[i][j], end="*y")
            print(j + 1, end=" ")
        print("<=", end=" ")
        print(matrix[i][len(matrix[i]) - 1])


# Функция печатает простую задачу
def print_simple_task_g(matrix, z):
    print_fg(z)
    print_g_task(matrix)


# Функция печатает решение игровой задачи
def print_solution_game(table):
    # Вычисляем количество стратегий игрока A (m) и игрока B (n)
    m = len(table) - 2
    n = len(table[0]) - 2 - m
    # Инициализируем список для стратегий игрока A
    solution_a = [0 for _ in range(m)]
    # Вычисляем стратегии игрока A
    print("P = ", end="")
    for i in range(m):
        solution_a[i] = table[-1][2 + n + i] / table[len(table) - 1][1]
    print(solution_a)
    # Инициализируем список для стратегий игрока B
    solution_b = [0 for _ in range(len(table[0]) - 2)]
    # Вычисляем стратегии игрока B
    print("Q = ", end="")
    for i in range(1, len(table) - 1):
        solution_b[int(table[i][0][1]) - 1] = table[i][1] /
                                              table[len(table) - 1][1]
    solution_b = [solution_b[i] for i in range(n)]
    print(solution_b)
    # Выводим значение минимизированной функции (F_min)
    print("F_min =", table[len(table) - 1][1])


# Функция определяет базис и строку, в которой он находится
def get_basic_list(matrix_cof):
    # Инициализируем список базисных переменных и списка строк,
    # в которых они находятся
    basic_list = []
    usage_string = []
    # Проходим по каждому столбцу матрицы коэффициентов
    for i in range(len(matrix_cof[0])):
        # Инициализируем счетчики
        count_one = 0
        count_another = 0
        one_index = 0
        # Проходим по каждой строке в текущем столбце
        for j in range(len(matrix_cof)):
            # Если встречаем единицу, увеличиваем счетчик и
            # запоминаем индекс строки
            if matrix_cof[j][i] == 1:
                count_one += 1
                one_index = j
```

```
            # Если встречаем ненулевое число (не единицу), увеличиваем счетчик
            elif matrix_cof[j][i] != 0:
                count_another += 1



        # Если в столбце есть только одна единица и нули, добавляем индекс столбца
        # в список базисных переменных
        # и индекс строки, в которой находится эта единица, в список строк с
        # базисными переменными
        if (count_one == 1) & (count_another == 0):
            basic_list.append(i)
            usage_string.append(one_index)
    # Возвращаем списки базисных переменных и строк с базисными переменными
    return [basic_list, usage_string]


# Функция создает начальную таблицу симплекс-метода
def get_first_table(matrix_cof, z):
    # Получаем списки базисных переменных и строк с базисными переменными
    basic, string_basic = get_basic_list(matrix_cof)
    # Оставляем только базисные переменные и строки с базисными переменными,
    # соответствующие исходной задаче
    basic = [basic[i] for i in range(len(basic) - len(matrix_cof), len(basic))]
    string_basic = [string_basic[i] for i in range(len(string_basic) -
                                   len(matrix_cof), len(string_basic))]
    # Создаем таблицу с нулями
    table = [[0 for _ in range(len(matrix_cof[0]) + 1)]
                for _ in range(len(basic) + 2)]
    # Заполняем первую строку таблицы метками "Б.п." и "С.п." и метками переменных
    table[0][0] = "Б.п."
    table[0][1] = "С.п."
    for i in range(2, len(table[0])):
        table[0][i] = 'X' + str(i - 1)
    # Заполняем столбец метками переменных базиса
    for x in range(len(basic)):
        table[string_basic[x] + 1][0] = 'X' + str(basic[x] + 1)
    # Заполняем столбец свободных членов
    for i in range(1, len(table) - 1):
        table[i][1] = matrix_cof[i - 1][len(matrix_cof[0]) - 1]
    # Заполняем остальные элементы таблицы коэффициентами матрицы коэффициентов
    for i in range(len(matrix_cof)):
        for j in range(len(matrix_cof[0]) - 1):
            table[i + 1][j + 2] = matrix_cof[i][j]
    # Заполняем последнюю строку таблицы коэффициентами целевой функции
    # и их знаками
    table[len(table) - 1][0] = 'Z'
    table[len(table) - 1][1] = z[0]
    for i in range(1, len(z)):
        table[len(table) - 1][i + 1] = -z[i]
    return table




# Функция проверяет условие остановки симплекс-метода
def check_solution(table):
    # Проходим по столбцам, начиная со второго
    for i in range(2, len(table[0])):
        # Если элемент последней строки (целевая функция) отрицателен,
        # возвращаем False
        if table[len(table) - 1][i] < 0:
            return False
    # Если все элементы последней строки неотрицательны, возвращаем True
```

```
    return True


# Функция печатает решение игровой задачи
def print_game_solution(matrix, z):
    # Добавляем фиктивные переменные к матрице выигрышей игры
    matrix = put_fictitious(matrix)
    # Создаем начальную таблицу симплекс-метода
    table = get_first_table(matrix, z)
    # Печатаем начальную таблицу
    print_table(table)
    # Пока не выполнено условие остановки симплекс-метода, выполняем следующий шаг
    # симплекс-метода и печатаем таблицу
    while not check_solution(table):
        table = next_step(table)
        print_table(table)
    # Печатаем решение игровой задачи
    print_solution_game(table)


# Функция добавляет в матрицу фиктивные переменные
def get_simple_matrix(matrix):
    for i in range(len(matrix)):
        matrix[i].append(1)
    return matrix


# Функция создает вектор коэффициентов целевой функции для простой задачи
def get_simple_z(matrix):
    a = [0]
    for i in range(len(matrix[0]) - 1):
        a.append(1)
    return a


# Основная функция программы
def main():
    # Матрица коэффициентов для игровой задачи
    matrix_cof = [[8, 5, 7, 6],
                  [9, 8, 10, 7],
                  [12, 6, 4, 3],
                  [7, 13, 5, 2]]
    # Получаем матрицу симплекс-метода, добавляя к исходной матрице фиктивные
    # переменные
    matrix_g = put_fictitious(matrix_cof)

    # Получаем коэффициенты целевой функции
    z = get_simple_z(matrix_g)

    print("Задача для решения двойственным симплекс-методом")
    print_simple_task_g(matrix_g, z)

    # Печатаем решение игровой задачи с помощью симплекс-метода
    print_game_solution(matrix_g, z)


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

Результат работы программы:

Задача для решения двойственным симплекс-методом

$$f = y_1 + y_2 + y_3 + y_4 \to max$$

$$5y_1 + 7y_2 + 9y_3 + 8y_4 \leq 1$$

$$3y_1 + 6y_2 + 4y_3 + 5y_4 \leq 1$$

$$10y_1 + 4y_2 + 12y_3 + 8y_4 \leq 1$$

$$y_1, y_2, y_3, y_4 \geq 0$$

1. **Начальная симплекс-таблица:**

Баз.пер.	Св.чл.	X1	X2	X3	X4	X5	X6	X7
x5	1	5	7	9	8	1	0	0
x6	1	3	6	4	5	0	1	0
x7	1	10	4	12	8	0	0	1
Z	0	-1	-1	-1	-1	0	0	0

2. **Первая итерация:**
   Разрешающий столбец — X1 (минимальный коэффициент в Z).
   Разрешающая строка — x7 (min(1/10, 1/3, 1/10)).
   Новый базис: x7 → X1.
   Преобразованная таблица:

Баз.пер.	Св.чл.	X1	X2	X3	X4	X5	X6	X7
x5	0.5	0	5	3	4	1	0	-0.5
x6	0.7	0	4.8	0.4	2.6	0	1	-0.3
x1	0.1	1	0.4	1.2	0.8	0	0	0.1
Z	0.1	0	-0.6	0.2	-0.2	0	0	0.1

3. **Вторая итерация:**
   Разрешающий столбец — X2.
   Разрешающая строка — x5 (min(0.5/5, 0.7/4.8, 0.1/0.4)).

Новый базис: x5 → X2.
Преобразованная таблица:

Баз.пер.	Св.чл.	X1	X2	X3	X4	X5	X6	X7
x2	0.1	0	1	0.6	0.8	0.2	0	-0.1
x6	0.22	0	0	-2.48	-1.24	-0.96	1	0.18
x1	0.06	1	0	0.96	0.48	-0.08	0	0.14
Z	0.16	0	0	0.56	0.28	0.12	0	0.04

4. **Оптимальное решение:**
   Все коэффициенты в строке $Z \geq 0$.
   Максимум $f = 0.16 = 4/25$.
   Переменные:
   a. $y_1 = 0.06 = 3/50$
   b. $y_2 = 0.1 = 1/10$
   c. $y_3 = y_4 = 0$.

Оптимальные стратегии:

- **Игрок B (Q):**
  Цена игры $v = 1/f = 25/4$.
  Нормированные вероятности:
  $q_1 = y_1 \cdot v = (3/50) \cdot (25/4) = 3/8$
  $q_2 = y_2 \cdot v = (1/10) \cdot (25/4) = 5/8$
  $q_3 = q_4 = 0$
  Q = [3/8, 5/8, 0, 0]
- **Игрок A (P):**
  Из двойственной задачи:
  $p_1 = 3/4$, $p_2 = 0$, $p_3 = 1/4$
  P = [3/4, 0, 1/4]

Цена игры:

v = 4/25 (или 0.16).

Вывод: в ходе выполнения лабораторной работы были освоены метод

нахождения седловой точки в смешанных стратегиях с помощью построения пары двойственных задач ЛП.

Проведённые задания по подготовке к работе помогли разработать программу для решения методом нахождения седловой точки в смешанных стратегиях с помощью построения пары двойственных задач ЛП. Результатом работы программы являются функция в минимальном значении с переменными. Была создана программа, которая правильно реализует нахождения седловой точки в смешанных стратегиях с помощью построения пары двойственных задач ЛП, что в свою очередь является главным в данной лабораторной работе.