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 ВТ-231

Масленников Даниил

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

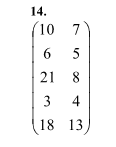
игры с помощью решения пары симметрично двойственных задач

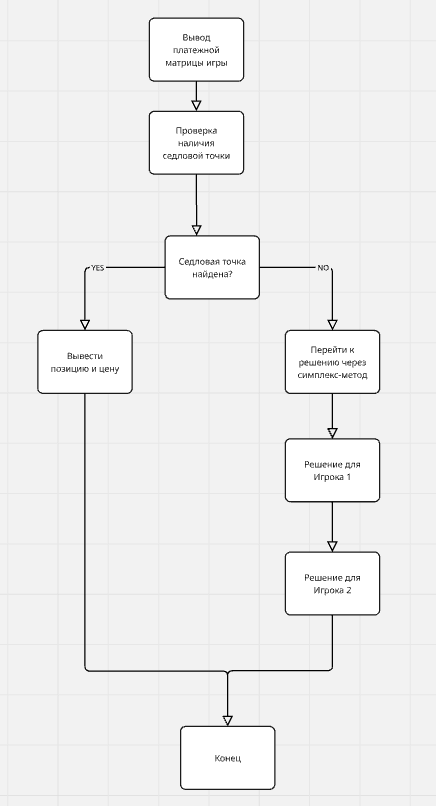
ЛП.

3. Для подготовки тестовых данных решить вручную одну из

следующих ниже задач.

# Вариант – 14

****

**Блок-схема программы**

**Код программы**

use std::f64::EPSILON as EPS;

// Функция для проверки наличия седловой точки в матрице

fn check\_saddle\_point(payoff\_matrix: &[Vec<i32>]) -> bool {

    let mut a = Vec::new(); // массив минимальных значений строк (игрок 2)

    let mut b = Vec::new(); // массив максимальных значений столбцов (игрок 1)

    // Заполняем массив a (минимальные значения строк)

    for row in payoff\_matrix {

        a.push(\*row.iter().min().unwrap());

    }

    // Заполняем массив b (максимальные значения столбцов)

    for j in 0..payoff\_matrix[0].len() {

        let mut max\_val = payoff\_matrix[0][j];

        for row in payoff\_matrix.iter().skip(1) {

            if row[j] > max\_val {

                max\_val = row[j];

            }

        }

        b.push(max\_val);

    }

    let a\_max = \*a.iter().max().unwrap(); // нижняя цена игры

    let b\_min = \*b.iter().min().unwrap(); // верхняя цена игры

    println!("\nАнализ седловой точки:");

    println!("----------------------");

    println!("Нижняя цена игры: {}", a\_max);

    println!("Верхняя цена игры: {}", b\_min);

    // Проверяем наличие седловой точки

    if a\_max == b\_min {

        let row = a.iter().position(|&x| x == a\_max).unwrap() + 1;

        let col = b.iter().position(|&x| x == b\_min).unwrap() + 1;

        println!("\nНайдена седловая точка в позиции ({}, {})", row, col);

        println!("Цена игры (v): {}", a\_max);

        true

    } else {

        println!("\nСедловая точка не найдена, игра требует смешанных стратегий");

        false

    }

}

// Структура для реализации симплекс-метода

struct Simplex {

    table: Vec<Vec<f64>>,

    m: usize,

    n: usize,

}

impl Simplex {

    fn new(a: &[Vec<f64>], b: &[f64], c: &[f64]) -> Self {

        let m = b.len();

        let n = c.len();

        let mut table = vec![vec![0.0; n + m + 1]; m + 1];

        for j in 0..n {

            table[0][j] = -c[j];

        }

        for i in 0..m {

            for j in 0..n {

                table[i + 1][j] = a[i][j];

            }

            table[i + 1][n + i] = 1.0;

            table[i + 1][n + m] = b[i];

        }

        Simplex { table, m, n }

    }

    fn pivot(&mut self, p: usize, q: usize) {

        for i in 0..=self.m {

            for j in 0..=self.n + self.m {

                if i != p && j != q {

                    self.table[i][j] -= self.table[p][j] \* self.table[i][q] / self.table[p][q];

                }

            }

        }

        for i in 0..=self.m {

            if i != p {

                self.table[i][q] = 0.0;

            }

        }

        for j in 0..=self.n + self.m {

            if j != q {

                self.table[p][j] /= self.table[p][q];

            }

        }

        self.table[p][q] = 1.0;

    }

    fn solve(&mut self, x: &mut Vec<f64>) -> bool {

        loop {

            let mut q = 0;

            for j in 1..self.n + self.m {

                if self.table[0][j] < self.table[0][q] {

                    q = j;

                }

            }

            if self.table[0][q] >= -EPS {

                break;

            }

            let mut p = None;

            let mut min\_ratio = f64::MAX;

            for i in 1..=self.m {

                if self.table[i][q] > EPS {

                    let ratio = self.table[i][self.n + self.m] / self.table[i][q];

                    if ratio < min\_ratio {

                        min\_ratio = ratio;

                        p = Some(i);

                    }

                }

            }

            let p = match p {

                Some(p) => p,

                None => return false,

            };

            self.pivot(p, q);

        }

        x.resize(self.n, 0.0);

        for j in 0..self.n {

            let mut basic = false;

            let mut p = None;

            for i in 1..=self.m {

                if (self.table[i][j] - 1.0).abs() < EPS {

                    if p.is\_none() {

                        p = Some(i);

                        basic = true;

                    } else {

                        basic = false;

                        break;

                    }

                } else if self.table[i][j].abs() > EPS {

                    basic = false;

                    break;

                }

            }

            if basic && p.is\_some() {

                x[j] = self.table[p.unwrap()][self.n + self.m];

            }

        }

        true

    }

    fn get\_objective(&self) -> f64 {

        self.table[0][self.n + self.m]

    }

}

// Функция для вывода стратегии

fn print\_strategy(strategy: &[f64], player\_name: &str) {

    println!("\nОптимальная стратегия для {}:", player\_name);

    for (i, &val) in strategy.iter().enumerate() {

        print!(" p{} = {:.4}", i + 1, val);

        if i % 3 == 2 || i == strategy.len() - 1 {

            println!();

        } else {

            print!(" |");

        }

    }

}

// Решение для игрока 1 (максимизация выигрыша)

fn solve\_player\_1(matrix: &[Vec<i32>]) {

    let rows = matrix.len();

    let cols = matrix[0].len();

    let mut a = vec![vec![0.0; rows + 1]; cols];

    let mut b = vec![0.0; cols];

    let mut c = vec![0.0; rows + 1];

    c[rows] = 1.0;

    for j in 0..cols {

        for i in 0..rows {

            a[j][i] = -(matrix[i][j] as f64);

        }

        a[j][rows] = 1.0;

    }

    let mut eq\_row = vec![0.0; rows + 1];

    for i in 0..rows {

        eq\_row[i] = 1.0;

    }

    a.push(eq\_row);

    b.push(1.0);

    let mut simplex = Simplex::new(&a, &b, &c);

    let mut solution = Vec::new();

    if !simplex.solve(&mut solution) {

        eprintln!("Ошибка решения для Игрока 1");

        return;

    }

    let v = simplex.get\_objective();

    let x = solution[..rows].to\_vec();

    println!("\nРезультаты для Игрока 1 (максимизация):");

    println!("--------------------------------------");

    println!("Цена игры: {:.4}", v);

    print\_strategy(&x, "Игрока 1 (x)");

    println!();

}

// Решение для игрока 2 (минимизация проигрыша)

fn solve\_player\_2(matrix: &[Vec<i32>]) {

    let rows = matrix.len();

    let cols = matrix[0].len();

    let mut a = vec![vec![0.0; cols + 1]; rows];

    let mut b = vec![0.0; rows];

    let mut c = vec![0.0; cols + 1];

    c[cols] = 1.0;

    for i in 0..rows {

        for j in 0..cols {

            a[i][j] = matrix[i][j] as f64;

        }

        a[i][cols] = -1.0;

    }

    let mut eq\_row = vec![0.0; cols + 1];

    for j in 0..cols {

        eq\_row[j] = 1.0;

    }

    a.push(eq\_row);

    b.push(1.0);

    let mut simplex = Simplex::new(&a, &b, &c);

    let mut solution = Vec::new();

    if !simplex.solve(&mut solution) {

        eprintln!("Ошибка решения для Игрока 2");

        return;

    }

    let v = simplex.get\_objective();

    let y = solution[..cols].to\_vec();

    println!("\nРезультаты для Игрока 2 (минимизация):");

    println!("--------------------------------------");

    println!("Цена игры: {:.4}", v);

    print\_strategy(&y, "Игрока 2 (y)");

    println!();

}

fn main() {

    let payoff\_matrix = vec![

        vec![10, 7],

        vec![6, 5],

        vec![21, 8],

        vec![3, 4],

        vec![18, 13],

    ];

    println!("Платежная матрица игры:");

    println!("-----------------------");

    for row in &payoff\_matrix {

        for val in row {

            print!("{:4}", val);

        }

        println!();

    }

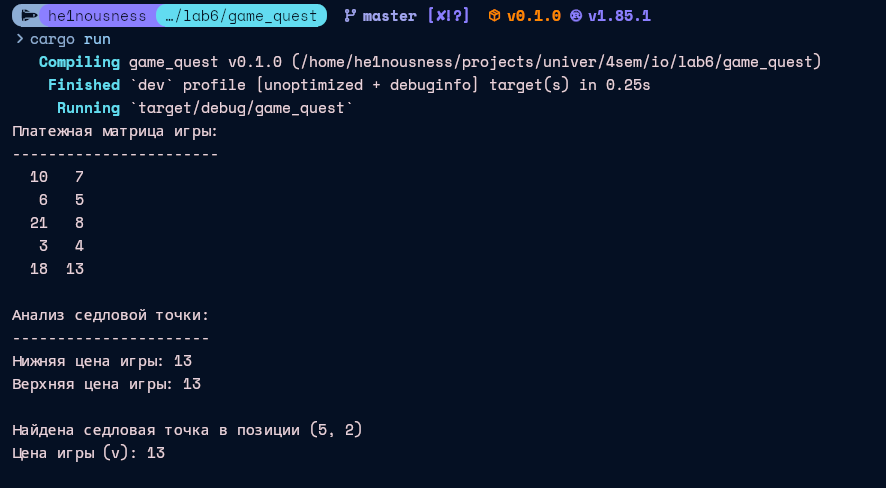
    if !check\_saddle\_point(&payoff\_matrix) {

        solve\_player\_1(&payoff\_matrix);

        solve\_player\_2(&payoff\_matrix);

    }

}

 **Аналитическое решение:**

**Платежная матрица игры:**

|  | B1 | B2 |
| --- | --- | --- |
| **A1** | 10 | 7 |
| **A2** | 6 | 5 |
| **A3** | 21 | 8 |
| **A4** | 3 | 4 |
| **A5** | 18 | 13 |

#### Анализ седловой точки:

1. **Минимальные значения по строкам (игрок II):**
   * Строка A1: min(10, 7) = 7
   * Строка A2: min(6, 5) = 5
   * Строка A3: min(21, 8) = 8
   * Строка A4: min(3, 4) = 3
   * Строка A5: min(18, 13) = 13
2. **Максимальные значения по столбцам (игрок I):**
   * Столбец B1: max(10, 6, 21, 3, 18) = 21
   * Столбец B2: max(7, 5, 8, 4, 13) = 13

#### **Результаты анализа:**

* **Нижняя цена игры (α):** max(7, 5, 8, 3, 13) = 13
* **Верхняя цена игры (β):** min(21, 13) = 13
* **Седловая точка:** (A5, B2) с значением 13

#### **Оптимальные стратегии:**

* Для **игрока I:** чистая стратегия **A5** (5-я строка).
* Для **игрока II:** чистая стратегия **B2** (2-й столбец).
* **Цена игры:** v = 13

#### **Проверка решения:**

* Для игрока I: выбор A5 гарантирует выигрыш не менее 13.
* Для игрока II: выбор B2 ограничивает проигрыш не более 13.
* Значения совпадают, решение оптимально.

#### Вывод:

Результат аналитического решения подтверждает наличие седловой точки в позиции (A5, B2) с ценой игры 13. Это означает, что оптимальные стратегии для обоих игроков являются чистыми, и игра имеет решение в чистых стратегиях.