#### **Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управления (ИУ) КАФЕДРА Информационная безопасность (ИУ8)

**Теория игр и исследование операций**

**Рубежный контроль №1**

**«Монотонный итеративный алгоритм»**

**Вариант: 9**

**Студент:**

Киселев Владислав Александрович группа ИУ8-104

(5 курс) (подпись, дата)

**Преподаватель:**

к.т.н., доцент кафедры ИУ8

Коннова Наталья Сергеевна [(подпись, дата)](#_bookmark0)

Москва, 2024 г.

# Задание.

1. Для 10 агентов случайным образом сгенерировать стохастическую матрицу доверия.
2. Получить результирующую матрицу доверия.
3. Случайным образом выбрать номера агентов из общего числа агентов для первого и второго игроков.
4. Определить функции выигрыша, целевые функции, точку утопии, найти аналитическое решение игры с не противоположными интересами двух игроков.

| **Номер варианта** | **a** | **b** | **c** | **d** | **g\_f** | **g\_s** |
| --- | --- | --- | --- | --- | --- | --- |
| 9 | 4 | 4 | 1 | 2 | 2 | 2 |

# Решение 1. СТОХАСТИЧЕСКАЯ МАТРИЦА ДОВЕРИЯ A:

Размер матрицы: (10, 10)

Проверка: сумма каждой строки должна быть равна 1

Суммы строк: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Матрица A:

[[0.077 0.169 0.134 0.113 0.041 0.041 0.025 0.156 0.113 0.13 ]

[0.024 0.216 0.188 0.063 0.057 0.057 0.082 0.126 0.107 0.079]

[0.142 0.048 0.078 0.093 0.111 0.177 0.06 0.123 0.138 0.029]

[0.116 0.044 0.027 0.172 0.175 0.149 0.066 0.032 0.129 0.089]

[0.044 0.118 0.027 0.2 0.071 0.151 0.082 0.123 0.128 0.056]

[0.158 0.13 0.154 0.147 0.103 0.151 0.028 0.044 0.022 0.063]

[0.086 0.065 0.164 0.081 0.067 0.113 0.042 0.159 0.031 0.192]

[0.157 0.054 0.019 0.165 0.145 0.149 0.157 0.031 0.083 0.039]

[0.154 0.116 0.069 0.026 0.066 0.068 0.133 0.118 0.158 0.092]

[0.04 0.148 0.156 0.12 0.158 0.108 0.113 0.096 0.023 0.038]]

Сходимость достигнута на итерации 13

**2. РЕЗУЛЬТИРУЮЩАЯ МАТРИЦА A∞:**

Матрица A∞ (первые 3 строки):

[[0.102 0.111 0.1 0.12 0.1 0.117 0.077 0.098 0.095 0.079]

[0.102 0.111 0.1 0.12 0.1 0.117 0.077 0.098 0.095 0.079]

[0.102 0.111 0.1 0.12 0.1 0.117 0.077 0.098 0.095 0.079]]

Вектор r (стационарное распределение): [0.102 0.111 0.1 0.12 0.1 0.117 0.077 0.098 0.095 0.079]

Сумма элементов r: 1.000000

**3. РАСПРЕДЕЛЕНИЕ АГЕНТОВ (ВАРИАНТ 9):**

Агенты первого игрока (F): [4 5 6]

Агенты второго игрока (S): [1 2 3]

Нейтральные агенты: [0 7 8 9]

**4. ВЫЧИСЛЕНИЕ КЛЮЧЕВЫХ ПАРАМЕТРОВ:**

r\_f = сумма r для агентов первого игрока = 0.295

r\_s = сумма r для агентов второго игрока = 0.331

Нейтральные агенты: [0 7 8 9]

Их начальные мнения: [0.475 0.59 0.546 0.52 ]

X^0 = 0.199

**5. ФУНКЦИИ ВЫИГРЫША И ОПТИМАЛЬНЫЕ МНЕНИЯ:**

H\_f(x) = 4\*x - 4\*x^2

H\_s(x) = 1\*x - 2\*x^2

Оптимальное мнение первого игрока X\_max\_f = a/(2b) = 0.500

Оптимальное мнение второго игрока X\_max\_s = c/(2d) = 0.250

Решение системы уравнений для равновесия Нэша:

∂Φ\_f/∂u = a\*r\_f - 2\*b\*r\_f\*(r\_f\*u + r\_s\*v + X0) - gf\*u = 0

∂Φ\_s/∂v = c\*r\_s - 2\*d\*r\_s\*(r\_f\*u + r\_s\*v + X0) - gs\*v = 0

Система уравнений:

2.695\*u + 0.782\*v = 0.710

0.391\*u + 2.440\*v = 0.068

Определитель матрицы: 6.270

Решение на границе, используем численную оптимизацию...

Численная сходимость достигнута на итерации 4

**6. РАВНОВЕСИЕ НЭША:**

Оптимальное управление первого игрока: u\* = 0.263

Оптимальное управление второго игрока: v\* = 0.000

**7. РЕЗУЛЬТАТЫ:**

Итоговое мнение агентов: X = 0.277

Точка утопии первого игрока: X\_max\_f = 0.500

Точка утопии второго игрока: X\_max\_s = 0.250

Расстояние до точки утопии первого игрока: Δ\_f = 0.223

Расстояние до точки утопии второго игрока: Δ\_s = 0.027

ПОБЕДИТЕЛЬ: Второй игрок (меньшее расстояние до точки утопии)

**РАСШИРЕННЫЙ АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ:**

1. ДЕЙСТВИЯ В ОДИНОЧКУ:

Первый игрок один: X = 0.347, Δ = 0.153

Второй игрок один: X = 0.282, Δ = 0.032

2. КООПЕРАТИВНОЕ РЕШЕНИЕ (Парето-оптимум):

u\_coop = 0.295, v\_coop = 0.295

X\_coop = 0.384, Δ\_f = 0.116, Δ\_s = 0.134

3. ВЫИГРЫШ ОТ КООПЕРАЦИИ (равновесие Нэша vs одиночная игра):

Первый игрок: -0.070 (проигрыш)

Второй игрок: +0.005 (улучшение)

4. ЭФФЕКТИВНОСТЬ ПО ПАРЕТО:

Эффективность: 100.00%

# Вывод

Параметры варианта 9: a=4, b=4, c=1, d=2, gf=2, gs=2

Агенты влияния: Player1=[4 5 6], Player2=[1 2 3]

Коэффициенты влияния: r\_f=0.295, r\_s=0.331

Начальное мнение нейтральных агентов: X0=0.199

Равновесие Нэша: u\*=0.263, v\*=0.000

Итоговое мнение: X=0.277

Точки утопии: X\_max\_f=0.500, X\_max\_s=0.250

Расстояния: Δ\_f=0.223, Δ\_s=0.027

РЕЗУЛЬТАТ: Второй игрок

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import minimize

# =============================================================================

# 1. ГЕНЕРАЦИЯ СТОХАСТИЧЕСКОЙ МАТРИЦЫ ДОВЕРИЯ ДЛЯ 10 АГЕНТОВ

# =============================================================================

def generate\_stochastic\_matrix(n, seed=42):

    """Генерирует стохастическую матрицу с более равномерным распределением"""

    np.random.seed(seed)

    matrix = np.random.rand(n, n) + 0.1  # Добавляем 0.1 чтобы избежать нулей

    stochastic\_matrix = matrix / matrix.sum(axis=1, keepdims=True)

    return stochastic\_matrix

# Генерируем матрицу доверия для 10 агентов

n\_agents = 10

A = generate\_stochastic\_matrix(n\_agents)

print("1. СТОХАСТИЧЕСКАЯ МАТРИЦА ДОВЕРИЯ A:")

print("Размер матрицы:", A.shape)

print("Проверка: сумма каждой строки должна быть равна 1")

print("Суммы строк:", np.sum(A, axis=1).round(3))

print("\nМатрица A:")

print(A.round(3))

# =============================================================================

# 2. ВЫЧИСЛЕНИЕ РЕЗУЛЬТИРУЮЩЕЙ МАТРИЦЫ A∞ (ПРЕДЕЛЬНОЕ РАСПРЕДЕЛЕНИЕ)

# =============================================================================

def compute\_limiting\_matrix(A, tolerance=1e-10, max\_iterations=1000):

    """Вычисляет предельную матрицу A∞ методом итерационного возведения в степень"""

    A\_current = A.copy()

    for i in range(max\_iterations):

        A\_next = np.dot(A\_current, A)

        max\_diff = np.max(np.abs(A\_next - A\_current))

        if max\_diff < tolerance:

            print(f"Сходимость достигнута на итерации {i+1}")

            break

        A\_current = A\_next

    return A\_current

# Вычисляем A∞

A\_inf = compute\_limiting\_matrix(A)

print("\n" + "="\*80)

print("2. РЕЗУЛЬТИРУЮЩАЯ МАТРИЦА A∞:")

print("Все строки должны быть одинаковыми (стационарное распределение)")

print("\nМатрица A∞ (первые 3 строки):")

print(A\_inf[:3].round(3))

# Вектор r - первая строка матрицы A∞ (все строки одинаковы)

r = A\_inf[0, :]

print(f"\nВектор r (стационарное распределение): {r.round(3)}")

print(f"Сумма элементов r: {r.sum():.6f}")

# =============================================================================

# 3. ВЫБОР АГЕНТОВ ВЛИЯНИЯ ДЛЯ ИГРОКОВ (ВАРИАНТ 9)

# =============================================================================

# Для варианта 9 фиксируем конкретных агентов

player1\_agents = np.array([4, 5, 6])  # Агенты первого игрока

player2\_agents = np.array([1, 2, 3])  # Агенты второго игрока

neutral\_agents = np.array([0, 7, 8, 9])  # Нейтральные агенты

print("\n" + "="\*80)

print("3. РАСПРЕДЕЛЕНИЕ АГЕНТОВ (ВАРИАНТ 9):")

print(f"Агенты первого игрока (F): {player1\_agents}")

print(f"Агенты второго игрока (S): {player2\_agents}")

print(f"Нейтральные агенты: {neutral\_agents}")

# =============================================================================

# 4. ВЫЧИСЛЕНИЕ ПАРАМЕТРОВ r\_f, r\_s, X^0

# =============================================================================

# Параметры из варианта 9

a, b, c, d = 4, 4, 1, 2

gf, gs = 2, 2

# Вычисляем r\_f и r\_s

r\_f = np.sum(r[player1\_agents])

r\_s = np.sum(r[player2\_agents])

# Генерируем начальные мнения для нейтральных агентов в диапазоне [0.4, 0.6] для баланса

np.random.seed(42)

neutral\_opinions = np.random.uniform(0.4, 0.6, len(neutral\_agents))

# Вычисляем X^0

X0 = np.sum(r[neutral\_agents] \* neutral\_opinions)

print("\n" + "="\*80)

print("4. ВЫЧИСЛЕНИЕ КЛЮЧЕВЫХ ПАРАМЕТРОВ:")

print(f"r\_f = сумма r для агентов первого игрока = {r\_f:.3f}")

print(f"r\_s = сумма r для агентов второго игрока = {r\_s:.3f}")

print(f"Нейтральные агенты: {neutral\_agents}")

print(f"Их начальные мнения: {neutral\_opinions.round(3)}")

print(f"X^0 = {X0:.3f}")

# =============================================================================

# 5. ОПРЕДЕЛЕНИЕ ФУНКЦИЙ ВЫИГРЫША И ЦЕЛЕВЫХ ФУНКЦИЙ

# =============================================================================

def final\_opinion(u, v, r\_f, r\_s, X0):

    """Вычисляет итоговое мнение агентов: X(u,v) = r\_f\*u + r\_s\*v + X0"""

    return r\_f \* u + r\_s \* v + X0

def objective\_f(u, v, r\_f, r\_s, X0, a, b, gf):

    """Целевая функция первого игрока: Φ\_f(u,v) = a\*X - b\*X^2 - gf\*u^2/2"""

    X = final\_opinion(u, v, r\_f, r\_s, X0)

    return -(a \* X - b \* X\*\*2 - gf \* u\*\*2 / 2)  # Минус для минимизации

def objective\_s(u, v, r\_f, r\_s, X0, c, d, gs):

    """Целевая функция второго игрока: Φ\_s(u,v) = c\*X - d\*X^2 - gs\*v^2/2"""

    X = final\_opinion(u, v, r\_f, r\_s, X0)

    return -(c \* X - d \* X\*\*2 - gs \* v\*\*2 / 2)  # Минус для минимизации

# Оптимальные мнения (точки утопии)

X\_max\_f = a / (2 \* b)  # Максимум H\_f(x)

X\_max\_s = c / (2 \* d)  # Максимум H\_s(x)

print("\n" + "="\*80)

print("5. ФУНКЦИИ ВЫИГРЫША И ОПТИМАЛЬНЫЕ МНЕНИЯ:")

print(f"H\_f(x) = {a}\*x - {b}\*x^2")

print(f"H\_s(x) = {c}\*x - {d}\*x^2")

print(f"Оптимальное мнение первого игрока X\_max\_f = a/(2b) = {X\_max\_f:.3f}")

print(f"Оптимальное мнение второго игрока X\_max\_s = c/(2d) = {X\_max\_s:.3f}")

# =============================================================================

# 6. ПОИСК РАВНОВЕСИЯ НЭША (УЛУЧШЕННАЯ ВЕРСИЯ)

# =============================================================================

def solve\_analytical(r\_f, r\_s, X0, a, b, c, d, gf, gs):

    """Аналитическое решение системы уравнений с проверкой устойчивости"""

    print("\nРешение системы уравнений для равновесия Нэша:")

    print("∂Φ\_f/∂u = a\*r\_f - 2\*b\*r\_f\*(r\_f\*u + r\_s\*v + X0) - gf\*u = 0")

    print("∂Φ\_s/∂v = c\*r\_s - 2\*d\*r\_s\*(r\_f\*u + r\_s\*v + X0) - gs\*v = 0")

    print()

    # Коэффициенты системы уравнений:

    A11 = 2\*b\*r\_f\*\*2 + gf

    A12 = 2\*b\*r\_f\*r\_s

    B1 = a\*r\_f - 2\*b\*r\_f\*X0

    A21 = 2\*d\*r\_f\*r\_s

    A22 = 2\*d\*r\_s\*\*2 + gs

    B2 = c\*r\_s - 2\*d\*r\_s\*X0

    print(f"Система уравнений:")

    print(f"{A11:.3f}\*u + {A12:.3f}\*v = {B1:.3f}")

    print(f"{A21:.3f}\*u + {A22:.3f}\*v = {B2:.3f}")

    # Решаем систему

    A\_matrix = np.array([[A11, A12], [A21, A22]])

    B\_vector = np.array([B1, B2])

    # Проверяем условие устойчивости (определитель)

    det = A11 \* A22 - A12 \* A21

    print(f"Определитель матрицы: {det:.3f}")

    if abs(det) < 1e-10:

        print("Внимание: система близка к вырожденной!")

        # Используем псевдообратную матрицу

        solution = np.linalg.pinv(A\_matrix) @ B\_vector

    else:

        solution = np.linalg.solve(A\_matrix, B\_vector)

    u\_opt, v\_opt = solution[0], solution[1]

    # Применяем ограничения [0, 1] более мягко

    u\_opt = np.clip(u\_opt, 0, 1)

    v\_opt = np.clip(v\_opt, 0, 1)

    # Если решение на границе, пытаемся найти внутреннее решение численно

    if u\_opt in [0, 1] or v\_opt in [0, 1]:

        print("Решение на границе, используем численную оптимизацию...")

        u\_opt, v\_opt = find\_numerical\_equilibrium(r\_f, r\_s, X0, a, b, c, d, gf, gs)

    return u\_opt, v\_opt

def find\_numerical\_equilibrium(r\_f, r\_s, X0, a, b, c, d, gf, gs, max\_iter=100, tol=1e-6):

    """Численный метод поиска равновесия Нэша"""

    # Начальные приближения

    u, v = 0.5, 0.5

    for iteration in range(max\_iter):

        # Игрок 1 оптимизирует u при фиксированном v

        res1 = minimize(lambda u\_val: objective\_f(u\_val, v, r\_f, r\_s, X0, a, b, gf),

                       u, bounds=[(0, 1)], method='L-BFGS-B')

        u\_new = res1.x[0]

        # Игрок 2 оптимизирует v при фиксированном u

        res2 = minimize(lambda v\_val: objective\_s(u\_new, v\_val, r\_f, r\_s, X0, c, d, gs),

                       v, bounds=[(0, 1)], method='L-BFGS-B')

        v\_new = res2.x[0]

        # Проверяем сходимость

        if abs(u\_new - u) < tol and abs(v\_new - v) < tol:

            print(f"Численная сходимость достигнута на итерации {iteration+1}")

            return u\_new, v\_new

        u, v = u\_new, v\_new

    print(f"Численный метод сошелся за {max\_iter} итераций (точность: {abs(u\_new - u):.2e}, {abs(v\_new - v):.2e})")

    return u, v

# Находим равновесие Нэша

u\_opt, v\_opt = solve\_analytical(r\_f, r\_s, X0, a, b, c, d, gf, gs)

print("\n" + "="\*80)

print("6. РАВНОВЕСИЕ НЭША:")

print(f"Оптимальное управление первого игрока: u\* = {u\_opt:.3f}")

print(f"Оптимальное управление второго игрока: v\* = {v\_opt:.3f}")

# =============================================================================

# 7. ВЫЧИСЛЕНИЕ ИТОГОВОГО МНЕНИЯ И РАССТОЯНИЙ ДО ТОЧКИ УТОПИИ

# =============================================================================

# Итоговое мнение при оптимальных управлениях

X\_final = final\_opinion(u\_opt, v\_opt, r\_f, r\_s, X0)

# Расстояния до точек утопии

distance\_f = abs(X\_final - X\_max\_f)

distance\_s = abs(X\_final - X\_max\_s)

# Определяем победителя

winner = "Первый игрок" if distance\_f < distance\_s else "Второй игрок"

print("\n" + "="\*80)

print("7. РЕЗУЛЬТАТЫ:")

print(f"Итоговое мнение агентов: X = {X\_final:.3f}")

print(f"Точка утопии первого игрока: X\_max\_f = {X\_max\_f:.3f}")

print(f"Точка утопии второго игрока: X\_max\_s = {X\_max\_s:.3f}")

print(f"Расстояние до точки утопии первого игрока: Δ\_f = {distance\_f:.3f}")

print(f"Расстояние до точки утопии второго игрока: Δ\_s = {distance\_s:.3f}")

print(f"ПОБЕДИТЕЛЬ: {winner} (меньшее расстояние до точки утопии)")

# =============================================================================

# 8. УЛУЧШЕННЫЙ АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ

# =============================================================================

def analyze\_sensitivity(r\_f, r\_s, X0, X\_max\_f, X\_max\_s):

    """Расширенный анализ чувствительности"""

    print("\n" + "="\*80)

    print("РАСШИРЕННЫЙ АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ:")

    # Сценарий 1: Каждый игрок действует в одиночку

    X\_solo\_f = final\_opinion(X\_max\_f, 0, r\_f, r\_s, X0)

    X\_solo\_s = final\_opinion(0, X\_max\_s, r\_f, r\_s, X0)

    delta\_solo\_f = abs(X\_solo\_f - X\_max\_f)

    delta\_solo\_s = abs(X\_solo\_s - X\_max\_s)

    print(f"\n1. ДЕЙСТВИЯ В ОДИНОЧКУ:")

    print(f"   Первый игрок один: X = {X\_solo\_f:.3f}, Δ = {delta\_solo\_f:.3f}")

    print(f"   Второй игрок один: X = {X\_solo\_s:.3f}, Δ = {delta\_solo\_s:.3f}")

    # Сценарий 2: Кооперативное решение (Парето-оптимум)

    # Ищем u, v которые минимизируют сумму расстояний

    def cooperative\_objective(x):

        u, v = x

        X = final\_opinion(u, v, r\_f, r\_s, X0)

        return abs(X - X\_max\_f) + abs(X - X\_max\_s)

    res\_coop = minimize(cooperative\_objective, [0.5, 0.5], bounds=[(0,1), (0,1)])

    u\_coop, v\_coop = res\_coop.x

    X\_coop = final\_opinion(u\_coop, v\_coop, r\_f, r\_s, X0)

    delta\_coop\_f = abs(X\_coop - X\_max\_f)

    delta\_coop\_s = abs(X\_coop - X\_max\_s)

    print(f"\n2. КООПЕРАТИВНОЕ РЕШЕНИЕ (Парето-оптимум):")

    print(f"   u\_coop = {u\_coop:.3f}, v\_coop = {v\_coop:.3f}")

    print(f"   X\_coop = {X\_coop:.3f}, Δ\_f = {delta\_coop\_f:.3f}, Δ\_s = {delta\_coop\_s:.3f}")

    # Сравнение выигрышей

    improvement\_f = delta\_solo\_f - distance\_f

    improvement\_s = delta\_solo\_s - distance\_s

    print(f"\n3. ВЫИГРЫШ ОТ КООПЕРАЦИИ (равновесие Нэша vs одиночная игра):")

    print(f"   Первый игрок: {improvement\_f:+.3f} (улучшение)" if improvement\_f > 0 else

          f"   Первый игрок: {improvement\_f:+.3f} (проигрыш)")

    print(f"   Второй игрок: {improvement\_s:+.3f} (улучшение)" if improvement\_s > 0 else

          f"   Второй игрок: {improvement\_s:+.3f} (проигрыш)")

    # Эффективность по Парето

    pareto\_efficiency = (distance\_f + distance\_s) / (delta\_coop\_f + delta\_coop\_s)

    print(f"\n4. ЭФФЕКТИВНОСТЬ ПО ПАРЕТО:")

    print(f"   Эффективность: {pareto\_efficiency:.2%}")

analyze\_sensitivity(r\_f, r\_s, X0, X\_max\_f, X\_max\_s)

# =============================================================================

# 9. ФИНАЛЬНЫЕ РЕЗУЛЬТАТЫ

# =============================================================================

print("\n" + "="\*80)

print("ИТОГОВЫЕ РЕЗУЛЬТАТЫ ЛАБОРАТОРНОЙ РАБОТЫ №6 (ВАРИАНТ 9)")

print("="\*80)

print(f"Параметры варианта 9: a={a}, b={b}, c={c}, d={d}, gf={gf}, gs={gs}")

print(f"Агенты влияния: Player1={player1\_agents}, Player2={player2\_agents}")

print(f"Коэффициенты влияния: r\_f={r\_f:.3f}, r\_s={r\_s:.3f}")

print(f"Начальное мнение нейтральных агентов: X0={X0:.3f}")

print(f"Равновесие Нэша: u\*={u\_opt:.3f}, v\*={v\_opt:.3f}")

print(f"Итоговое мнение: X={X\_final:.3f}")

print(f"Точки утопии: X\_max\_f={X\_max\_f:.3f}, X\_max\_s={X\_max\_s:.3f}")

print(f"Расстояния: Δ\_f={distance\_f:.3f}, Δ\_s={distance\_s:.3f}")

print(f"РЕЗУЛЬТАТ: {winner}")