Министерство науки и высшего образования Российской Федерации
Калужский филиал федерального государственного автономного образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК Информатика и управление

КАФЕДРА ИУК4 Программное обеспечение ЭВМ, информационные технологии

ЛАБОРАТОРНАЯ РАБОТА

«Марковские модели принятие решений»

по дисциплине: «*Методы принятия решений в программной инженерии*»

Выполнил: студент группы ИУК4-72Б _____ Губин Е.В.
(Подпись) (И.О. Фамилия)

Проверил: _____ Никитенко У.В.
(Подпись) (И.О. Фамилия)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

Цель: изучить марковские модели принятия решений и освоение методов нахождения оптимальных стратегий управления в стохастических системах при конечном и бесконечном горизонтах планирования.

Задачи:

1. Построить марковскую модель принятия решений для задачи управления сбытом продукции фирмы.
2. Исследовать и сравнить методы нахождения оптимальных стратегий при конечном и бесконечном горизонтах планирования.
3. Реализовать и применить алгоритмы полного перебора, итераций по стратегиям и линейного программирования для решения задачи.

Вариант №1

Формулировка задачи №1:

Фирма ежегодно оценивает положение со сбытом своей продукции как удовлетворительное (состояние S1) или неудовлетворительное (состояние S2). Необходимо принять решение о целесообразности рекламирования продукции с целью расширения её сбыта при конечном (задано N) и (или) бесконечном горизонтах планирования, если матрицы

$$P_1 = \begin{pmatrix} 0,9 & 0,1 \\ 0,6 & 0,4 \end{pmatrix},\ P_2 = \begin{pmatrix} 0,7 & 0,3 \\ 0,2 & 0,8 \end{pmatrix}$$

определяют переходные вероятности рассматриваемой системы при наличии рекламы (допустимое решение X1) и без неё (допустимое решение X2) в течение любого года, а соответствующие им доходы заданы матрицами

$$R_1 = \begin{pmatrix} 2 & -1 \\ 1 & -3 \end{pmatrix},\ R_2 = \begin{pmatrix} 4 & 1 \\ 2 & -1 \end{pmatrix},$$

при этом необходимо учитывать коэффициент дисконтирования (если он задан).

1. Построить марковскую модель принятия решений.

2. В случае конечного горизонта планирования:
 – решить задачу методом полного перебора;
 – решить задачу методом итераций по стратегиям.

3. В случае бесконечного горизонта планирования:
 – найти и перечислить все стационарные стратегии;
 – решить задачу методом полного перебора;
 – решить задачу методом итераций по стратегиям;
 – решить задачу методами линейного программирования.

Решить задачу как для конечного горизонта (N=3), так и для бесконечного горизонтов планирования. В обоих случаях без коэффициента дисконтирования.

Результаты выполнения работы:

```
--- КОНЕЧНЫЙ ГОРИЗОНТ: ПОЛНЫЙ ПЕРЕБОР ---
Лучшая стратегия: (0, 1, 1)
Функция ценности: [5.69375 2.0235 ]

--- КОНЕЧНЫЙ ГОРИЗОНТ: ИТЕРАЦИИ ПО СТРАТЕГИЯМ ---
Оптимальная политика:
[[1 0]
 [1 0]
 [1 1]]
Функция ценности: [6.46645 2.4051 ]

Все стационарные стратегии: [(0, 0), (0, 1), (1, 0), (1, 1)]

--- БЕСКОНЕЧНЫЙ ГОРИЗОНТ: ПОЛНЫЙ ПЕРЕБОР ---
Лучшая стратегия: (1, 0)
Функция ценности: [20.02196897 15.95603491]

--- БЕСКОНЕЧНЫЙ ГОРИЗОНТ: ИТЕРАЦИИ ПО СТРАТЕГИЯМ ---
Оптимальная стратегия: [1 0]
Функция ценности: [20.02196897 15.95603491]

--- БЕСКОНЕЧНЫЙ ГОРИЗОНТ: ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ ---
Функция ценности: [11.89189189  6.21621622]
```

Рисунок 1 Решение задачи

Листинг программы:

```
import numpy as np
import itertools
from scipy.optimize import linprog

states = [0, 1]
actions = [0, 1]
N = 3
gamma = 0.9

P = {
    0: np.array([[0.9, 0.1],
                 [0.6, 0.4]]),
    1: np.array([[0.7, 0.3],
                 [0.2, 0.8]])
}

R = {
    0: np.array([[2, -1],
                 [1, -3]]),
    1: np.array([[4, 1],
                 [2, -1]])
}

def expected_reward(s, a):
    return np.sum(P[a][s] * R[a][s])

def brute_force_finite():
    print("\n--- КОНЕЧНЫЙ ГОРИЗОНТ: ПОЛНЫЙ ПЕРЕБОР ---")
    strategies = list(itertools.product(actions, repeat=N))
    results = []

    for strategy in strategies:
        V = np.zeros(len(states))
        for t in reversed(range(N)):
```

```
            new_V = np.zeros(len(states))
            for s in states:
                a = strategy[t]
                new_V[s] = expected_reward(s, a) + gamma * np.sum(P[a][s] * V)
            V = new_V
        results.append((strategy, V.copy()))

    best = max(results, key=lambda x: x[1].sum())
    print("Лучшая стратегия:", best[0])
    print("Функция ценности:", best[1])
    return best

def policy_iteration_finite():
    print("\n--- КОНЕЧНЫЙ ГОРИЗОНТ: ИТЕРАЦИИ ПО СТРАТЕГИЯМ ---")
    policy = np.zeros((N, len(states)), dtype=int)
    stable = False

    while not stable:
        V = np.zeros((N + 1, len(states)))
        for t in reversed(range(N)):
            for s in states:
                a = policy[t, s]
                V[t, s] = expected_reward(s, a) + gamma * np.sum(P[a][s] * V[t + 1])

        stable = True
        for t in range(N):
            for s in states:
                values = []
                for a in actions:
                    val = expected_reward(s, a) + gamma * np.sum(P[a][s] * V[t + 1])
                    values.append(val)
                best_a = np.argmax(values)
                if best_a != policy[t, s]:
                    policy[t, s] = best_a
                    stable = False

    print("Оптимальная политика:")
    print(policy)
    print("Функция ценности:", V[0])
    return policy, V[0]

def stationary_strategies():
    return list(itertools.product(actions, repeat=len(states)))

def evaluate_policy(policy, eps=1e-6):
    V = np.zeros(len(states))
    while True:
        new_V = np.zeros(len(states))
        for s in states:
            a = policy[s]
            new_V[s] = expected_reward(s, a) + gamma * np.sum(P[a][s] * V)
        if np.max(np.abs(new_V - V)) < eps:
            break
        V = new_V
    return V

def brute_force_infinite():
    print("\n--- БЕСКОНЕЧНЫЙ ГОРИЗОНТ: ПОЛНЫЙ ПЕРЕБОР ---")
    strategies = stationary_strategies()
    results = []

    for policy in strategies:
        V = evaluate_policy(policy)
        results.append((policy, V))

    best = max(results, key=lambda x: x[1].sum())
    print("Лучшая стратегия:", best[0])
    print("Функция ценности:", best[1])
    return best

def policy_iteration_infinite():
```

```
    print("\n--- БЕСКОНЕЧНЫЙ ГОРИЗОНТ: ИТЕРАЦИИ ПО СТРАТЕГИЯМ ---")
    policy = np.zeros(len(states), dtype=int)
    stable = False

    while not stable:
        V = evaluate_policy(policy)
        stable = True
        for s in states:
            values = []
            for a in actions:
                val = expected_reward(s, a) + gamma * np.sum(P[a][s] * V)
                values.append(val)
            best_a = np.argmax(values)
            if best_a != policy[s]:
                policy[s] = best_a
                stable = False

    print("Оптимальная стратегия:", policy)
    print("Функция ценности:", V)
    return policy, V

def linear_programming_solution():
    print("\n--- БЕСКОНЕЧНЫЙ ГОРИЗОНТ: ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ ---")
    c = [-1, -1]
    A = []
    b = []

    for s in states:
        for a in actions:
            row = [0, 0]
            row[s] = 1
            row -= gamma * P[a][s]
            A.append(row)
            b.append(expected_reward(s, a))

    res = linprog(c, A_ub=A, b_ub=b, method="highs")
    print("Функция ценности:", res.x)
    return res.x

if __name__ == "__main__":
    brute_force_finite()
    policy_iteration_finite()
    print("\nВсе стационарные стратегии:", stationary_strategies())
    brute_force_infinite()
    policy_iteration_infinite()
    linear_programming_solution()
```

Вывод: в ходе выполнения лабораторной работы были получены навыки построения и анализа марковских моделей принятия решений, а также практического применения методов оптимального управления стохастическими системами.