

**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

|  |
| --- |
| **РТУ МИРЭА** |
|  |
| **Институт кибербезопасности и цифровых технологий (ИКБ)** |
|  |
| КБ-2 «Информационно-аналитические системы кибербезопасности» |

**ОТЧЕТ О ВЫПОЛНЕНИИ КОНТРОЛЬНОГО ДОМАШНЕГО ЗАДАНИЯ №2**

**ПО ДИСЦИПЛИНЫ «ФОРМАЛИЗОВАННЫЕ МОДЕЛИ И МЕТОДЫ ИНФОРМАЦИОННО-АНАЛИТИЧЕСКОГО МОНИТОРИНГА»**

Выполнил:

Студент 4-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

Оглавление

[**Цели работы** 3](#_Toc209293292)

[**Принцип векторного минимакса** 5](#_Toc209293293)

[**Принцип векторного минимаксного сожаления** 7](#_Toc209293294)

[**Критерии принятия решений** 10](#_Toc209293295)

[Критерий Вальда 10](#_Toc209293296)

[Критерий Сэвиджа 10](#_Toc209293297)

[Критерий Гурвица 10](#_Toc209293298)

[Критерий Лапласа 10](#_Toc209293299)

[**Приложение 1** 13](#_Toc209293300)

# **Цели работы**

В информационной системе (ИС) можно использовать два протокола обмена информацией между сотрудниками. Известно, что злоумышленники могут использовать 5 типов DDoS-атак для перегрузки системы передачи данных и замедления ее работы. Время передачи одного сообщения в системе при использовании штатных средств защиты информации указано в таблице 1. Требуется определить оптимальную стратегию использования протоколов передачи данных, гарантирующую минимальное среднее время обработки одной операции. Представить задачу в виде матричной игры и решить ее графоаналитическим методом.



Таблица 1- Вариант исходных данных №7

# **Представление в виде матричной игры**

Для представления выбора стратегии в виде матричной игры введем следующие обозначения:

Пусть игрок А (администратор ИС) выбирает стратегии , где каждое – время передачи для протокола i, при атаке типа j. Цель игрока выбрать стратегию, которая позволяет получить .

Игрок В (злоумышленник) – выбирает варианты атак , где каждое – время передачи одного сообщения для атаки j, при использовании игроком А протокола передачи i. Цель игрока выбрать стратегию, которая позволяет получить .

В контексте задачи .

**Цена игры, седловые точки**

Нижняя цена игры (чистая цена) – определяется как . Стратегия – называется оптимальной чистой стратегией игрока A.

Верхняя цена игры (чистая цена) – определяется как . Стратегия – называется оптимальной чистой стратегией игрока B.

В случае если для чистых стратегий , то – называется чистой ценой игры, пара стратегий – называется седловой точкой, а элемент матрицы А – называется седловым элементом платежной матрицы.

В контексте задачи необходимо проверять матрицу на наличие седловой точки, так как такая точка будет являться решением задачи. Данные расчета приведены в таблице 2.



Таблица 2 - Поиск седловой точки

Из таблицы видно, что , а . В данной игре нет седловой точки необходимо использовать смешанные стратегии.

# **Графоаналитический метод, смешанные стратегии**

Графоаналитический метод применим только к матричным играм, заданным матрицами . Пусть игрок А использует смешанную стратегию, тогда для построения графика определим функцию , которая является платежной функцией для чистых стратегий игрока B, где Bi – стратегия атаки игрока B и pi – вероятность выбора игроком А стратегии Ai. Так как стратегии всего 2, то вероятность пусть p1 – вероятность выбора первой стратегии, тогда вероятность выбора второй стратегии p2 = (1 – p1) Значения матрицы A представлены в таблице 3.



Таблица 3 - Значения матрицы A

= 10p +11

= -21p +33

= -13p +28

= 7p +16

= -p +19

Построим графики полученных функций для , а также построим верхнюю огибающую по точкам, найденным как объединение максимальных точек платежный функций на интервале от [0,1].

Найдем пересечения платежных функций

Точка, принадлежащая верхней облегающей – x6

Так как точка на верхней огибающей одна, соответственно она и будет минимаксной точкой.

Результат работы программы показан на рисунке 1.

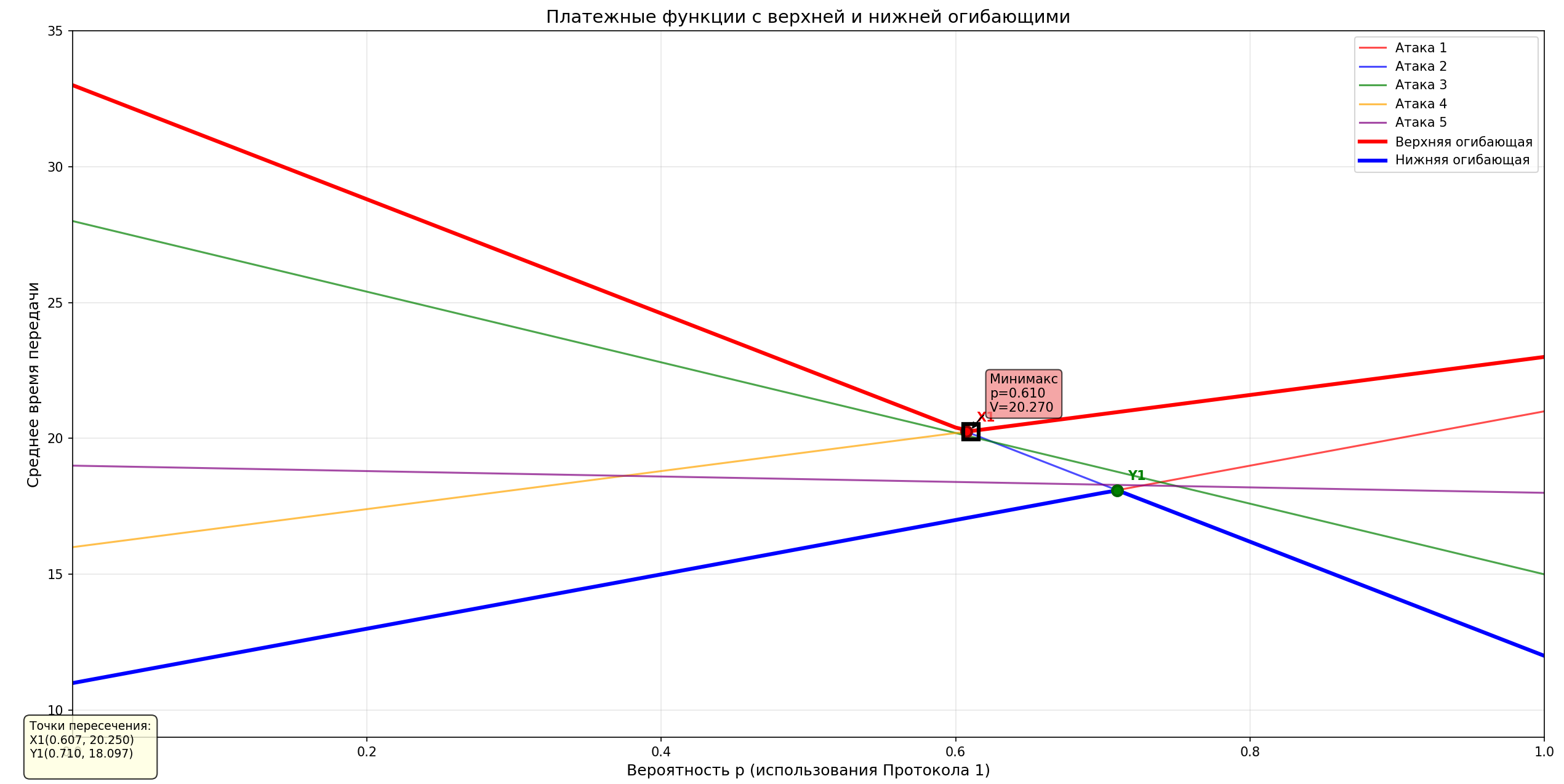


Рисунок 1 - Графики платежных функций

Вычислим значения функций в данной точке

Вычисления показывают, что при выборе любой из стратегий время передачи пакета не превысит 20.25 у.е.

# **Приложение 1**

В приложении 1 представлен листинг кода программы

import matplotlib.pyplot as plt  
import numpy as np  
from itertools import combinations  
  
  
def calculate\_payment\_functions(payment\_matrix):  
 *"""  
 Вычисляет коэффициенты платежных функций на основе платежной матрицы  
 """* num\_attacks = len(payment\_matrix[0])  
 functions = []  
  
 for j in range(num\_attacks):  
 # Получаем значения для текущей атаки  
 a1j = payment\_matrix[0][j]  
 a2j = payment\_matrix[1][j]  
  
 # Вычисляем коэффициенты линейной функции: k\*p + b  
 k = a1j - a2j  
 b = a2j  
  
 # Формируем описание функции  
 function\_info = {  
 'attack\_num': j + 1,  
 'a1j': a1j,  
 'a2j': a2j,  
 'k': k,  
 'b': b,  
 'equation': f"W(B{j + 1})(p) = {a1j}\*p + {a2j}\*(1-p) = {k}p + {b}"  
 }  
 functions.append(function\_info)  
  
 return functions  
  
  
def find\_intersections(functions):  
 intersections = []  
  
 # Перебираем все пары функций  
 for i, j in combinations(range(len(functions)), 2):  
 func1 = functions[i]  
 func2 = functions[j]  
  
 # Проверяем, не параллельны ли прямые  
 if func1['k'] != func2['k']:  
 # Находим точку пересечения: k1\*p + b1 = k2\*p + b2  
 p\_intersect = (func2['b'] - func1['b']) / (func1['k'] - func2['k'])  
  
 # Проверяем, что точка в допустимом диапазоне  
 if 0 <= p\_intersect <= 1:  
 # Вычисляем значение в точке пересечения  
 value = func1['k'] \* p\_intersect + func1['b']  
  
 intersection\_info = {  
 'p': p\_intersect,  
 'value': value,  
 'func1': func1['attack\_num'],  
 'func2': func2['attack\_num'],  
 'label': f"B{func1['attack\_num']}∩B{func2['attack\_num']}"  
 }  
 intersections.append(intersection\_info)  
  
 return intersections  
  
  
def find\_envelope\_points(intersections, functions, p\_values):  
 # Вычисляем значения всех функций для всех p  
 all\_values = []  
 for func in functions:  
 func\_values = []  
 for p in p\_values:  
 value = func['k'] \* p + func['b']  
 func\_values.append(value)  
 all\_values.append(func\_values)  
  
 # Вычисляем верхнюю огибающую (максимум в каждой точке)  
 upper\_envelope = []  
 for i in range(len(p\_values)):  
 max\_val = -float('inf')  
 for j in range(len(functions)):  
 if all\_values[j][i] > max\_val:  
 max\_val = all\_values[j][i]  
 upper\_envelope.append(max\_val)  
  
 # Вычисляем нижнюю огибающую (минимум в каждой точке)  
 lower\_envelope = []  
 for i in range(len(p\_values)):  
 min\_val = float('inf')  
 for j in range(len(functions)):  
 if all\_values[j][i] < min\_val:  
 min\_val = all\_values[j][i]  
 lower\_envelope.append(min\_val)  
  
 # Находим точки пересечения, принадлежащие огибающим  
 upper\_points = []  
 lower\_points = []  
  
 for intersection in intersections:  
 p = intersection['p']  
 value = intersection['value']  
  
 # Находим ближайшую точку в дискретном массиве  
 idx = np.argmin(np.abs(p\_values - p))  
  
 # Проверяем принадлежность к верхней огибающей (с допуском)  
 if abs(upper\_envelope[idx] - value) < 0.1:  
 upper\_points.append(intersection)  
  
 # Проверяем принадлежность к нижней огибающей (с допуском)  
 if abs(lower\_envelope[idx] - value) < 0.1:  
 lower\_points.append(intersection)  
  
 # Сортируем точки по координате p  
 upper\_points.sort(key=lambda x: x['p'])  
 lower\_points.sort(key=lambda x: x['p'])  
  
 return upper\_points, lower\_points, upper\_envelope, lower\_envelope  
  
  
def plot\_payment\_functions\_with\_envelopes(payment\_matrix):  
  
 # Вычисляем платежные функции  
 functions = calculate\_payment\_functions(payment\_matrix)  
  
 # Находим точки пересечения  
 intersections = find\_intersections(functions)  
  
 # Создаем диапазон значений p от 0 до 1  
 p\_values = np.arange(0, 1.01, 0.01)  
  
 # Находим точки огибающих  
 upper\_points, lower\_points, upper\_envelope, lower\_envelope = find\_envelope\_points(  
 intersections, functions, p\_values)  
  
 # Создаем график  
 plt.figure(figsize=(14, 10))  
  
 # Цвета для разных типов атак  
 colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown', 'pink', 'gray', 'olive', 'cyan']  
  
 # Строим графики для каждой атаки  
 for i, func in enumerate(functions):  
 # Вычисляем значения функции для всех p  
 w\_values = []  
 for p in p\_values:  
 w = func['k'] \* p + func['b']  
 w\_values.append(w)  
  
 # Строим график  
 color\_index = i % len(colors)  
 plt.plot(p\_values, w\_values,  
 color=colors[color\_index],  
 linewidth=1.5,  
 alpha=0.7,  
 label=f'Атака {func["attack\_num"]}')  
  
 # Строим верхнюю огибающую  
 plt.plot(p\_values, upper\_envelope, 'red', linewidth=3, label='Верхняя огибающая')  
  
 # Строим нижнюю огибающую  
 plt.plot(p\_values, lower\_envelope, 'blue', linewidth=3, label='Нижняя огибающая')  
  
 # Отмечаем точки верхней огибающей (красные) и нумеруем их  
 for i, point in enumerate(upper\_points):  
 plt.plot(point['p'], point['value'], 'ro', markersize=8, markeredgecolor='darkred', markeredgewidth=2)  
 plt.annotate(f'X{i + 1}',  
 xy=(point['p'], point['value']),  
 xytext=(8, 8),  
 textcoords='offset points',  
 fontsize=10,  
 fontweight='bold',  
 color='red')  
  
 # Отмечаем точки нижней огибающей (зеленые) и нумеруем их  
 for i, point in enumerate(lower\_points):  
 plt.plot(point['p'], point['value'], 'go', markersize=8, markeredgecolor='darkgreen', markeredgewidth=2)  
 plt.annotate(f'Y{i + 1}',  
 xy=(point['p'], point['value']),  
 xytext=(8, 8),  
 textcoords='offset points',  
 fontsize=10,  
 fontweight='bold',  
 color='green')  
  
 # Находим и отмечаем точку минимакса (минимум верхней огибающей)  
 min\_upper\_value = min(upper\_envelope)  
 min\_upper\_index = upper\_envelope.index(min\_upper\_value)  
 min\_upper\_p = p\_values[min\_upper\_index]  
  
 plt.plot(min\_upper\_p, min\_upper\_value, 's', markersize=12, markerfacecolor='none', markeredgecolor='black',  
 markeredgewidth=3)  
 plt.annotate(f'Минимакс\np={min\_upper\_p:.3f}\nV={min\_upper\_value:.3f}',  
 xy=(min\_upper\_p, min\_upper\_value),  
 xytext=(15, 15),  
 textcoords='offset points',  
 fontsize=10,  
 bbox=dict(boxstyle="round,pad=0.3", fc="lightcoral", alpha=0.7),  
 arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=0.2"))  
  
 # Создаем текст с координатами точек пересечения  
 intersection\_text = "Точки пересечения:\n"  
 for i, point in enumerate(upper\_points):  
 intersection\_text += f"X{i + 1}({point['p']:.3f}, {point['value']:.3f})\n"  
 for i, point in enumerate(lower\_points):  
 intersection\_text += f"Y{i + 1}({point['p']:.3f}, {point['value']:.3f})\n"  
  
 # Добавляем текст на график  
 plt.figtext(0.02, 0.02, intersection\_text, fontsize=9,  
 bbox=dict(boxstyle="round,pad=0.5", fc="lightyellow", alpha=0.8))  
  
 # Настраиваем график  
 plt.xlabel('Вероятность p (использования Протокола 1)', fontsize=12)  
 plt.ylabel('Среднее время передачи', fontsize=12)  
 plt.title('Платежные функции с верхней и нижней огибающими', fontsize=14)  
 plt.grid(True, alpha=0.3)  
 plt.legend(fontsize=10, loc='upper right')  
  
 # Устанавливаем пределы осей  
 plt.xlim(0, 1)  
  
 # Автоматически устанавливаем пределы по Y  
 y\_min = min(lower\_envelope) - 2  
 y\_max = max(upper\_envelope) + 2  
 plt.ylim(max(0, y\_min), y\_max)  
  
 # Отображаем график  
 plt.tight\_layout()  
 #plt.show()  
  
 return functions, intersections, (min\_upper\_p, min\_upper\_value), upper\_points, lower\_points  
  
  
# Основная программа  
payment\_matrix = [  
 [21, 12, 15, 23, 18], # Протокол 1  
 [11, 33, 28, 16, 19] # Протокол 2  
]  
  
print("Платежная матрица:")  
print(" B1 B2 B3 B4 B5")  
print(f"A1: {payment\_matrix[0]}")  
print(f"A2: {payment\_matrix[1]}")  
print()  
  
# Строим графики и получаем информацию о функциях и пересечениях  
functions, intersections, minimax\_point, upper\_points, lower\_points = plot\_payment\_functions\_with\_envelopes(  
 payment\_matrix)  
# Выводим уравнения платежных функций  
print("Уравнения платежных функций:")  
for func in functions:  
 print(func['equation'])  
print()  
# Выводим информацию о точках пересечения на огибающих  
print("Точки верхней огибающей:")  
for i, point in enumerate(upper\_points):  
 print(f"X{i + 1}: p = {point['p']:.3f}, значение = {point['value']:.3f} (B{point['func1']}∩B{point['func2']})")  
  
print("\nТочки нижней огибающей:")  
for i, point in enumerate(lower\_points):  
 print(f"Y{i + 1}: p = {point['p']:.3f}, значение = {point['value']:.3f} (B{point['func1']}∩B{point['func2']})")  
print()  
# Выводим информацию о минимаксной точке  
print("Минимаксная точка (оптимальная стратегия):")  
print(f"Вероятность p = {minimax\_point[0]:.4f}")  
print(f"Цена игры V = {minimax\_point[1]:.4f}")  
print(f"Оптимальная смешанная стратегия:")  
print(f" Использовать Протокол 1 с вероятностью {minimax\_point[0]:.3f}")  
print(f" Использовать Протокол 2 с вероятностью {1 - minimax\_point[0]:.3f}")