

**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 №1**

**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**

Представлены 16 проектов цифровой платформы (ЦП). Эффективность каждого проекта векторным критерием:

- ожидаемая экономическая эффективность

- степень информационной безопасности ЦП, и зависит от состояния внешней среды.

Предполагается, что выделено 4 различных состояния внешней среды, каждое из которых означает определенное сочетание внешних факторов, влияющих на эффективность ЦП. Значения критериев эффективности заданы матрицей Q.

Задачи

1. Принять решение о выборе типа ЦП, используя принцип векторного максимина.
2. Принять решение о выборе типа ЦП, используя принцип векторного минимаксного сожаления.
3. Принять решение о выборе типа ЦП по показателю , используя принципы Вальда, Сэвиджа, Гурвица, Лапласа.
4. Принять решение о выборе типа ЦП по показателю , , используя принципы Вальда, Сэвиджа, Гурвица, Лапласа.
5. Принять окончательное решение на основе анализа результатов пп. 1-4 и построения матрицы «голосования». Для формирования матрицы Q выбрать из таблицы 4 строки с номерами, соответствующими варианту задания.

Исходная матрица Q, заданная согласно варианту 7.



Таблица 1 - Исходная матрица Q

# **Принцип векторного минимакса**

Каждому решению поставим в соответствие вектор , компоненты которого определяются в виде .

Вектор определяет на множестве точку «крайнего пессимизма». Таким образом для определения векторных максиминов необходимо решить вспомогательную детерминированную многокритериальную задачу , в которой множество с компонентами в виде определено и его требуется минимизировать. Множество эффективных решений такой задачи является множеством векторных максиминов исходной задачи.

В контексте предоставленной задачи точки определяющие множество находятся в таблице 1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 |
| F1 | 2 | 4 | 1 | 4 | 1 | 2 | 2 | 3 |
| F2 | 6 | 4 | 3 | 1 | 2 | 3 | 3 | 3 |
|  | X9 | X10 | X11 | X12 | X13 | X14 | X15 | X16 |
| F1 | 1 | 4 | 3 | 1 | 2 | 2 | 3 | 4 |
| F2 | 4 | 4 | 2 | 3 | 3 | 6 | 6 | 7 |

Таблица 2 - Множество

Решаем задачу максимизации при помощи алгоритма исключения заведомо не оптимальных решений.

На полученном множестве оптимальных решений находим наиболее эффективное решая задачу при помощи функции Гермейера. Постановка задачи в таком случае выглядит как

Отображение данной задачи представлено в таблице 3.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 |
|  | 2 | 4 | 1 | 1 | 1 | 2 | 2 | 3 |
|  | X9 | X10 | X11 | X12 | X13 | X14 | X15 | X16 |
|  | 1 | 4 | 2 | 1 | 2 | 2 | 3 | 4 |

Таблица 3 - Значения функции Гермейера

Результат работы алгоритма представлен на рисунке 1.

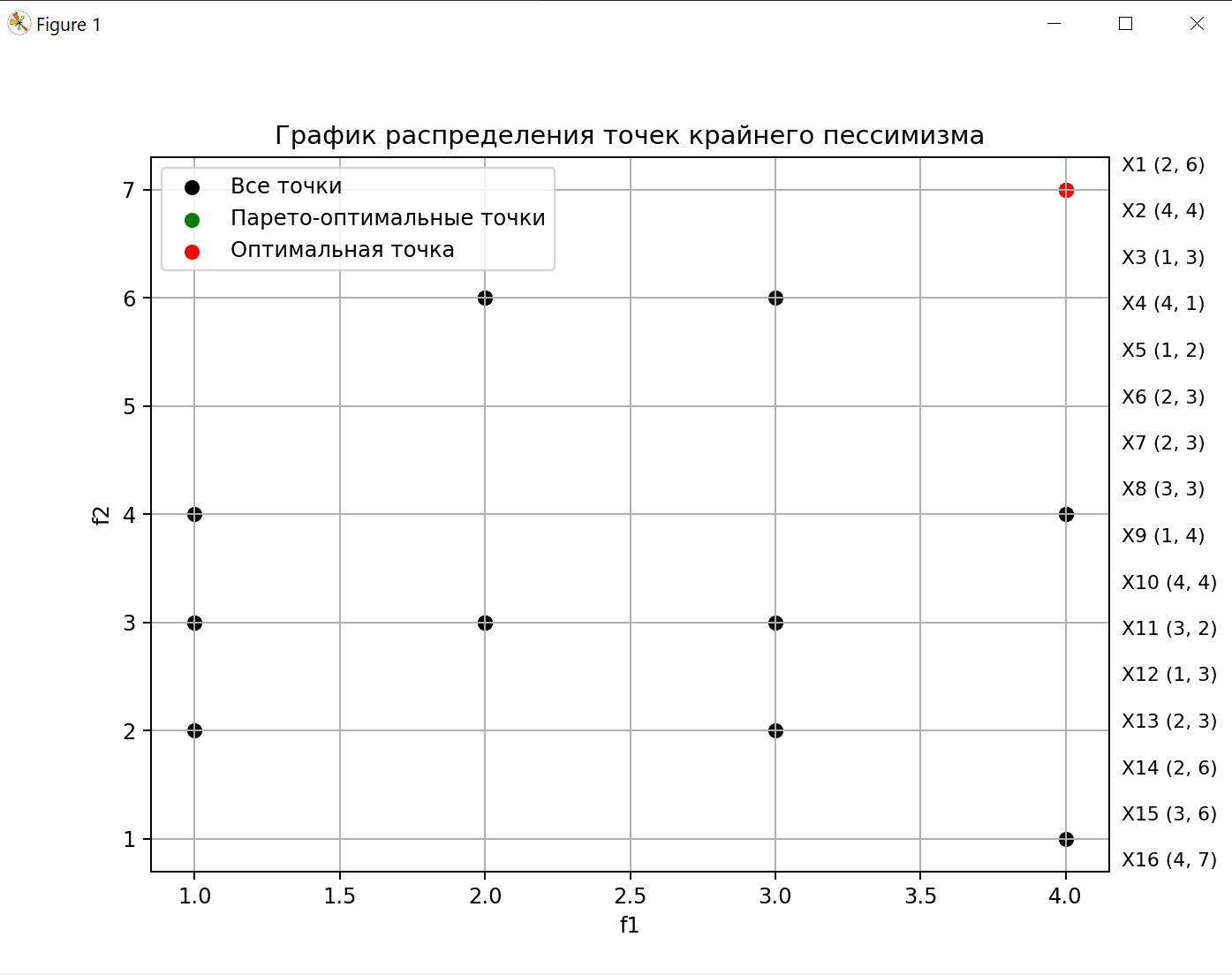


Рисунок 1 - Результат работы алгоритма

Из рисунка видно, что по критерию векторного минимакса оптимальным решением является проект под номером 16.

# **Принцип векторного минимаксного сожаления**

Для решения задачи необходимо построить множество «идеальных точек», каждая из которых представляет собой вид , для каждого возможного состояния среды. Значения точек представлены в таблице 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Z1 | Z2 | Z3 | Z4 |
| F1 | 11 | 9 | 10 | 7 |
| F2 | 12 | 10 | 12 | 11 |

Таблица 4 - Идеальные точки

Далее построим таблицу значений функции векторных рисков . Значения представленные в таблице 4.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | F1 | F2 | F1 | F2 | F1 | F2 | F1 | F2 |
| X1 | 6 | 2 | 7 | 3 | 2 | 6 | 2 | 2 |
| X2 | 7 | 8 | 4 | 3 | 1 | 6 | 3 | 7 |
| X3 | 7 | 4 | 8 | 1 | 3 | 9 | 0 | 6 |
| X4 | 7 | 11 | 3 | 6 | 1 | 10 | 0 | 9 |
| X5 | 10 | 9 | 6 | 5 | 6 | 10 | 4 | 8 |
| X6 | 6 | 8 | 7 | 4 | 7 | 9 | 5 | 7 |
| X7 | 0 | 7 | 7 | 3 | 6 | 7 | 5 | 8 |
| X8 | 7 | 9 | 6 | 3 | 0 | 8 | 4 | 3 |
| X9 | 10 | 2 | 7 | 4 | 7 | 6 | 5 | 7 |
| X10 | 7 | 2 | 2 | 5 | 4 | 8 | 3 | 3 |
| X11 | 3 | 5 | 0 | 3 | 7 | 10 | 2 | 8 |
| X12 | 10 | 9 | 5 | 0 | 8 | 6 | 5 | 6 |
| X13 | 6 | 9 | 5 | 0 | 8 | 6 | 5 | 6 |
| X14 | 9 | 3 | 6 | 4 | 7 | 0 | 3 | 3 |
| X15 | 8 | 6 | 4 | 2 | 5 | 5 | 4 | 0 |
| X16 | 4 | 0 | 5 | 1 | 3 | 5 | 3 | 3 |

Таблица 5 - Таблица векторных сожалений

Решим вспомогательную задачу максимизации возможных рисков. Результаты представлены в таблице 6.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 |
| F1 | 7 | 7 | 8 | 7 | 10 | 7 | 7 | 7 |
| F2 | 6 | 8 | 9 | 11 | 10 | 9 | 8 | 9 |
|  | X9 | X10 | X11 | X12 | X13 | X14 | X15 | X16 |
| F1 | 10 | 7 | 7 | 10 | 8 | 9 | 8 | 5 |
| F2 | 7 | 8 | 10 | 9 | 9 | 4 | 6 | 5 |

Таблица 6 - Точки крайнего пессимизма

Решаем задачу минимизации при помощи алгоритма исключения заведомо не оптимальных решений.

На полученном множестве оптимальных решений находим наиболее эффективное решая задачу при помощи функции Гермейера. Постановка задачи в таком случае выглядит как Отображение данной задачи представлено в таблице 7.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 |
|  | 7 | 8 | 9 | 11 | 10 | 9 | 8 | 9 |
|  | X9 | X10 | X11 | X12 | X13 | X14 | X15 | X16 |
|  | 10 | 8 | 10 | 10 | 9 | 9 | 8 | 5 |

Таблица 7 - значение функции Гермейера

Результат работы программы отображен на рисунке 2.

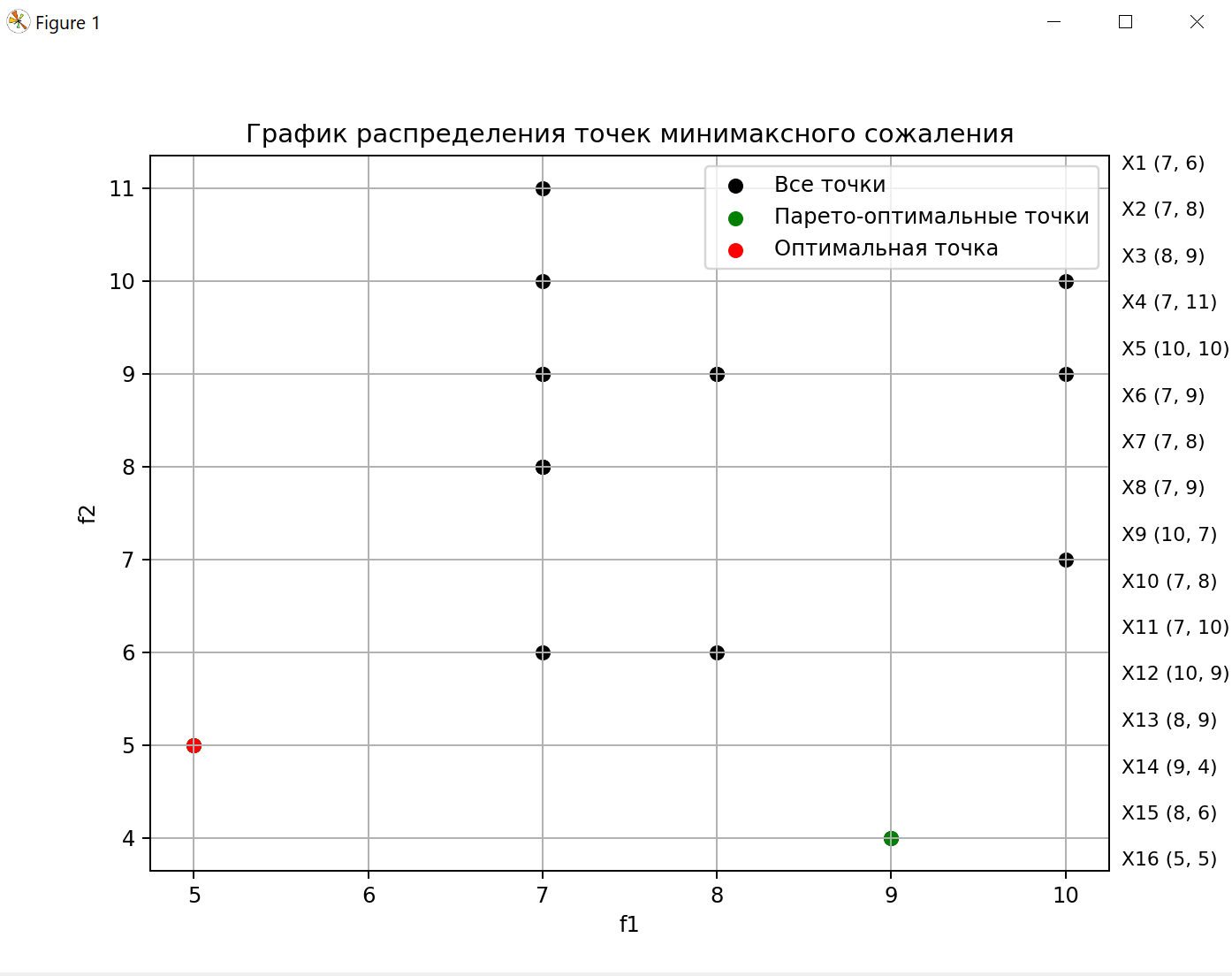


Рисунок 2 - Результат работы программы

Из рисунка видно, что оптимальным решением по критерию векторного минимаксного сожаления будет вариант проекта под номером 16.

# **Критерии принятия решений**

## Критерий Вальда

Математическая запись критерия строится следующим образом

, далее вычисляем – оптимальное решение.

## Критерий Сэвиджа

Математическая запись критерия строится следующим образом

, далее вычисляем – оптимальное решение.

## Критерий Гурвица

Математическая запись критерия строится следующим образом

, далее вычисляем – оптимальное решение. В нашем случае .

## Критерий Лапласа

Математическая запись критерия строится следующим образом

Критерий предполагает, что вероятности состояния среды () равновероятны.

Затем вычисляем – оптимальное решение.

По описанным критериям проведем подсчет и результаты запишем в таблицу 8.



Рисунок 3 - Оптимальные проекты по каждому критерию

Результат работы программы показан на рисунке 4.

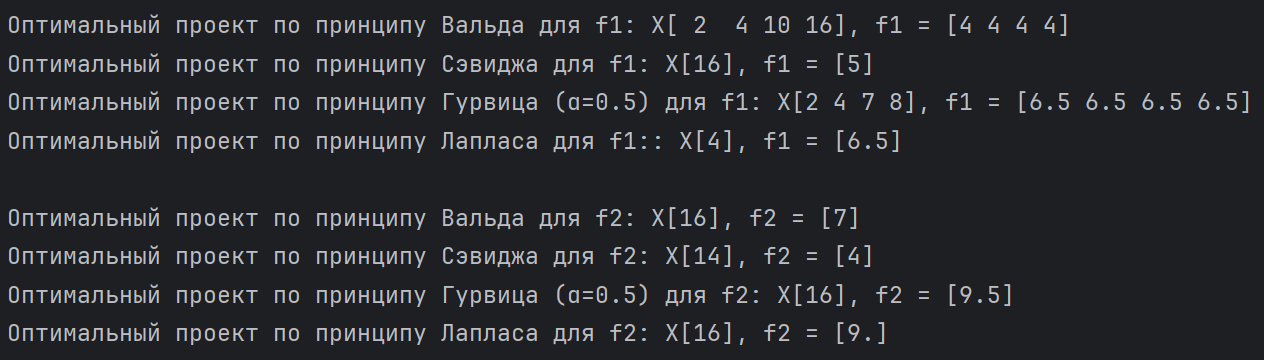


Рисунок 4 - Оптимальные проекты по каждому критерию и параметру

Матрица голосования, построенная программой показана на рисунке 5.

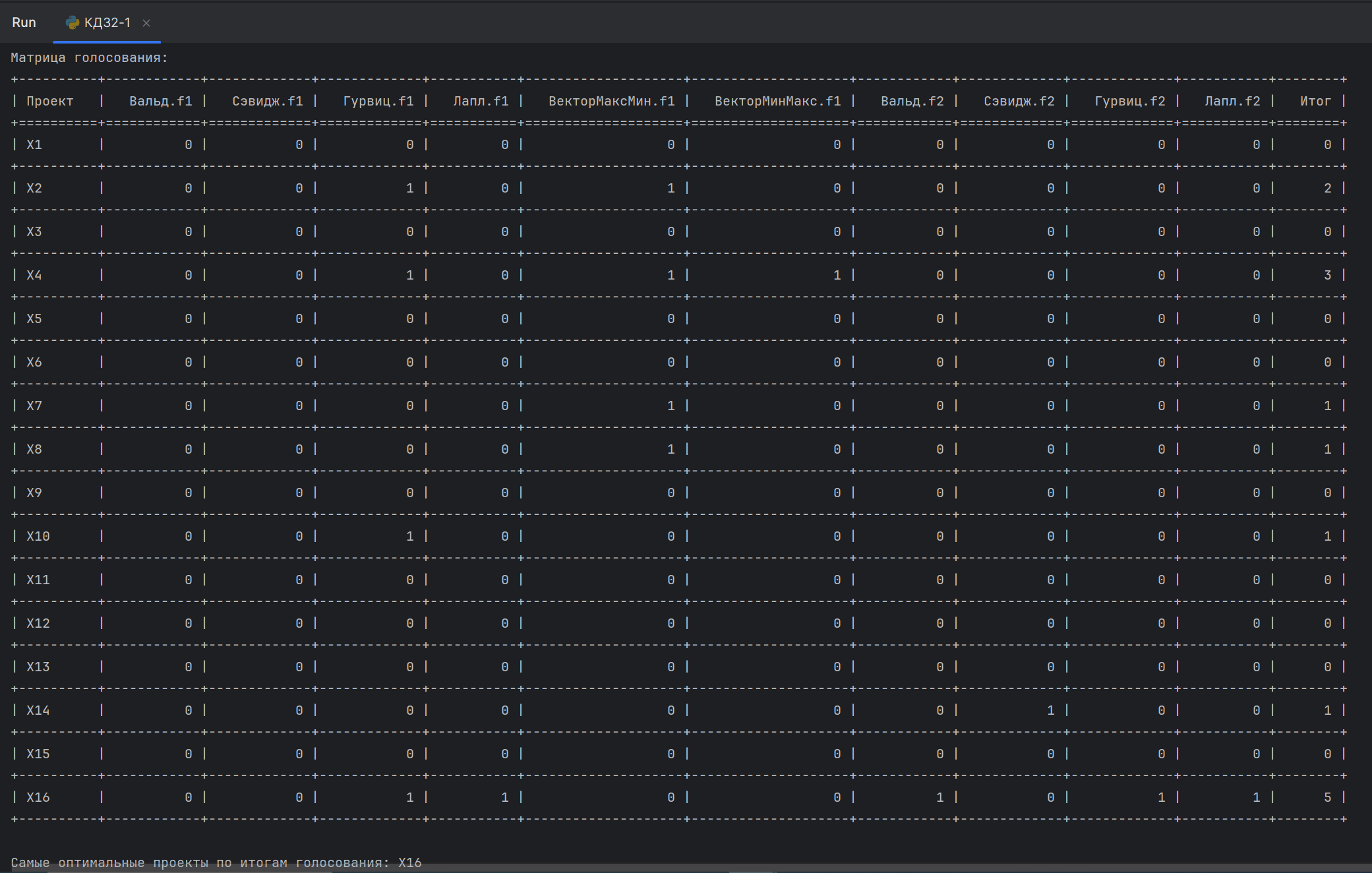


Рисунок 5 - Матрица голосования и искомый проект

Самый оптимальный проект имеет номер 16.

# **Приложение 1**

В приложении 1 представлен листинг кода программы

import matplotlib.pyplot as plt  
import numpy as np  
from tabulate import tabulate  
  
# Исходная матрица Q (значения f\_1 и f\_2 для каждого проекта и состояния внешней среды)  
Q = [  
 [(5, 10), (2, 7), (8, 6), (5, 9)],  
 [(4, 4), (5, 7), (9, 6), (4, 4)],  
 [(4, 8), (1, 9), (7, 3), (7, 5)],  
 [(4, 1), (6, 4), (9, 2), (7, 2)],  
 [(1, 3), (3, 5), (4, 2), (3, 3)],  
 [(5, 4), (2, 6), (3, 3), (2, 4)],  
 [(11, 5), (2, 7), (4, 5), (2, 3)],  
 [(4, 3), (3, 7), (10, 4), (3, 8)],  
 [(1, 10), (2, 6), (3, 6), (2, 4)],  
 [(4, 10), (7, 5), (6, 4), (4, 8)],  
 [(8, 7), (9, 7), (3, 2), (5, 3)],  
 [(1, 3), (4, 8), (2, 6), (2, 5)],  
 [(5, 3), (4, 10), (2, 6), (3, 4)],  
 [(2, 9), (3, 6), (3, 12), (4, 8)],  
 [(3, 6), (5, 8), (5, 7), (3, 11)],  
 [(7, 12), (4, 9), (7, 7), (4, 8)]  
]  
  
# Инициализация списков для хранения результатов  
min\_points\_f1 = []  
min\_points\_f2 = []  
  
  
# Обрабатываем каждый проект  
for project in range(len(Q)):  
 # Для каждого проекта находим минимальные значения по f\_1 и f\_2 для каждого состояния среды  
 min\_values\_f1 = []  
 min\_values\_f2 = []  
  
 for state in range(len(Q[project])):  
 # Для каждого состояния добавляем минимальные значения f\_1 и f\_2  
 min\_values\_f1.append(Q[project][state][0])  
 min\_values\_f2.append(Q[project][state][1])  
  
 # Точка максимального пессимизма для f\_1 и f\_2 (минимум по каждой координате)  
 min\_f1 = min(min\_values\_f1)  
 min\_f2 = min(min\_values\_f2)  
  
 # Добавляем результат в списки  
 min\_points\_f1.append(min\_f1)  
 min\_points\_f2.append(min\_f2)  
  
  
# 1) Вершины и рёбра по Грею  
def generate\_vertices(mu\_min, mu\_max):  
 return [  
 np.array([mu\_min[0], mu\_min[1]]), # код 00  
 np.array([mu\_max[0], mu\_min[1]]), # код 01  
 np.array([mu\_max[0], mu\_max[1]]), # код 11  
 np.array([mu\_min[0], mu\_max[1]]), # код 10  
 ]  
  
  
def build\_edges():  
 return [(0, 1), (1, 2), (2, 3), (3, 0)]  
  
  
# 2) Пересечение ребра с линией μ1+μ2=1  
def intersect\_edge(p, q):  
 Lp, Lq = p.sum() - 1, q.sum() - 1  
 if Lp \* Lq > 0:  
 return None  
 # вертикальное ребро?  
 if np.isclose(p[0], q[0]):  
 x = p[0];  
 y = 1 - x  
 else:  
 y = p[1];  
 x = 1 - y  
 return np.array([x, y])  
  
  
# 3) Построение матрицы B для заданного интервала весов  
def construct\_polyhedral\_cone(mu\_min, mu\_max):  
 if np.allclose(mu\_min, mu\_max):  
 return np.array([mu\_min], dtype=float)  
 verts = generate\_vertices(mu\_min, mu\_max)  
 B = []  
 for i, j in build\_edges():  
 P = intersect\_edge(verts[i], verts[j])  
 if P is None: continue  
 if mu\_min[0] <= P[0] <= mu\_max[0] and mu\_min[1] <= P[1] <= mu\_max[1]:  
 B.append(P)  
 return np.unique(np.array(B), axis=0) if B else np.empty((0, 2))  
  
  
# 4) Проверка: Fi доминируется Fj по конусу B? (для максимизации)  
def is\_dominated\_by(Fi, Fj, B,type):  
 if B.size == 0:  
 return False  
 if type == 'max':  
 diff = Fi - Fj # для максимизации  
 ge = [np.dot(b, diff) <= 0 for b in B] # знак неравенства изменился на <= для максимизации  
 gt = [np.dot(b, diff) < 0 for b in B] # строгое неравенство для максимизации  
 return all(ge) and any(gt)  
 if type == 'min':  
 diff = Fi - Fj # для максимизации  
 ge = [np.dot(b, diff) >= 0 for b in B] # знак неравенства изменился на >= для минимизации  
 gt = [np.dot(b, diff) > 0 for b in B] # строгое неравенство для минимизации  
 return all(ge) and any(gt)  
  
  
# 5) Однократное построение парето через B0 = B([0,0],[1,1])  
def build\_pareto(fx, type):  
 B0 = construct\_polyhedral\_cone([0, 0], [1, 1])  
 pareto = []  
 for Fi in fx:  
 if not any(is\_dominated\_by(Fi, Fj, B0,type) for Fj in fx if not np.allclose(Fi, Fj)):  
 pareto.append(Fi)  
 return np.array(pareto)  
  
def calculate\_ideal\_point(Q,column):  
 f1\_max,f2\_max = 0,0  
 for i in range (len(Q)):  
 if f1\_max <= Q[i][column][0]:  
 f1\_max = Q[i][column][0]  
 if f2\_max <= Q[i][column][1]:  
 f2\_max = Q[i][column][1]  
 return f1\_max,f2\_max  
  
def build\_regret\_matrix(Q):  
 regret\_matrix = [[list(state) for state in project] for project in Q]  
 for i in range (len(Q)):  
 for j in range(len(Q[i])):  
 ideal\_point = list(calculate\_ideal\_point(Q,j))  
 regret\_matrix[i][j][0]=ideal\_point[0]-Q[i][j][0]  
 regret\_matrix[i][j][1] = ideal\_point[1] - Q[i][j][1]  
 return np.array(regret\_matrix)  
  
def build\_ideal\_points\_for\_projects(regret\_matrix):  
 ideal\_points = []  
 for regret\_row in regret\_matrix:  
 ideal\_point\_project = np.max(regret\_row, axis=0) # Для каждого проекта находим максимум среди состояний  
 ideal\_points.append(ideal\_point\_project)  
 return np.array(ideal\_points)  
  
# Строим множество Парето  
pareto\_points = build\_pareto(np.column\_stack((min\_points\_f1, min\_points\_f2)), type='max')  
  
  
# 6) Поиск оптимальной точки, сравнивая минимумы по каждой компоненте  
def find\_optimal\_solution(pareto\_points):  
 best\_solution = None  
 max\_min\_value = -float('inf')  
  
 for point in pareto\_points:  
 min\_value = min(point[0], point[1]) # минимальная компонента  
 if min\_value > max\_min\_value:  
 max\_min\_value = min\_value  
 best\_solution = point  
  
 return best\_solution  
  
  
# 1) Принцип Вальда  
def valda\_criterion(f\_values):  
 min\_f\_values = np.min(f\_values, axis=1) # Находим минимальное значение f1 для каждого проекта  
 max\_min\_f = np.max(min\_f\_values) # Максимальное минимальное значение f1  
 optimal\_projects = np.where(min\_f\_values == max\_min\_f)[0] # Находим все проекты, которые соответствуют максимальному минимальному f1  
 return optimal\_projects, min\_f\_values[optimal\_projects]  
  
# 2) Принцип Сэвиджа  
def savage\_criterion(f\_values):  
 ideal\_f = np.max(f\_values, axis=0) # Находим идеальные значения для f1 по каждому состоянию  
 regret\_matrix = ideal\_f - f\_values # Строим матрицу сожалений (разница между идеальной и текущей точкой)  
 regret\_values = np.max(regret\_matrix, axis=1) # Находим максимальное сожаление для каждого проекта  
 min\_regret = np.min(regret\_values) # Минимальное сожаление  
 optimal\_projects = np.where(regret\_values == min\_regret)[0] # Находим все проекты с минимальным сожалением  
 return optimal\_projects, regret\_values[optimal\_projects]  
  
# 3) Принцип Гурвица  
def hurwicz\_criterion(f\_values, alpha=0.5):  
 hurwicz\_values = []  
 for f in f\_values:  
 hurwicz\_value = alpha \* np.max(f) + (1 - alpha) \* np.min(f) # Применяем формулу Гурвица  
 hurwicz\_values.append(hurwicz\_value)  
 hurwicz\_values = np.array(hurwicz\_values) # Преобразуем список в массив  
 max\_hurwicz\_value = np.max(hurwicz\_values) # Максимальное значение  
 optimal\_projects = np.where(hurwicz\_values == max\_hurwicz\_value)[0] # Находим все проекты с максимальным значением  
 return optimal\_projects, hurwicz\_values[optimal\_projects]  
  
# 4) Принцип Лапласа  
def laplace\_criterion(f\_values):  
 laplace\_values = np.mean(f\_values, axis=1) # Среднее значение f1 по всем состояниям  
 max\_laplace\_value = np.max(laplace\_values) # Максимальное среднее значение  
 optimal\_projects = np.where(laplace\_values == max\_laplace\_value)[0] # Находим все проекты с максимальным средним значением  
 return optimal\_projects, laplace\_values[optimal\_projects]  
  
#-------------------------ЗАДНАИЕ1-----------------------------------------------------------  
  
# Найдем оптимальную точку  
optimal\_point = find\_optimal\_solution(pareto\_points)  
  
  
#График с оптимальной точкой по критерию векторного максимина  
plt.figure(figsize=(8, 6)) # Устанавливаем размер графика  
  
# Подписываем каждую точку рядом с графиком  
x\_offset = 1.2 # Смещение по оси X для текста  
y\_offset = 0.05 # Смещение по оси Y для текста  
  
# Печатаем подписи рядом с графиком  
for i in range(len(min\_points\_f1)):  
 plt.figtext(0.91, 0.93 - (i + 1) \* y\_offset, f"X{i + 1} ({min\_points\_f1[i]}, {min\_points\_f2[i]})", ha='left',  
 va='top', fontsize=9)  
 if min\_points\_f1[i] == optimal\_point[0] and min\_points\_f2[i] == optimal\_point[1]:  
 solution\_pont\_mimnax = i+1  
# Рисуем все точки чёрным цветом  
plt.scatter(min\_points\_f1, min\_points\_f2, color='black', marker='o', label="Все точки")  
  
# Рисуем только Парето-оптимальные точки зелёным  
plt.scatter(pareto\_points[:, 0], pareto\_points[:, 1], color='green', marker='o', label="Парето-оптимальные точки")  
  
# Выделяем оптимальную точку как красную точку  
plt.scatter(optimal\_point[0], optimal\_point[1], color='red', marker='o', label="Оптимальная точка")  
  
# Настройки графика  
plt.xlabel('f1') # Подпись оси x  
plt.ylabel('f2') # Подпись оси y  
plt.title('График распределения точек крайнего пессимизма')  
  
# Показать сетку  
plt.grid(True)  
plt.legend()  
plt.show()  
  
#-------------------------ЗАДНАИЕ2-----------------------------------------------------------  
  
# Строим матрицу сожалений  
regret\_matrix = build\_regret\_matrix(Q)  
  
  
# Строим идеальные точки для каждого проекта  
ideal\_points\_for\_projects = build\_ideal\_points\_for\_projects(regret\_matrix)  
  
# Построение полиэдрального конуса и нахождение Парето-оптимальных точек  
pareto\_points = build\_pareto(ideal\_points\_for\_projects, type = 'min')  
  
# Поиск оптимальной точки из множества Парето-оптимальных  
optimal\_point = find\_optimal\_solution(pareto\_points)  
  
# Оптимальная точка для по критерияю минимаксного сожаления  
plt.figure(figsize=(8, 6)) # Устанавливаем размер графика  
  
# Рисуем все точки чёрным цветом  
plt.scatter(ideal\_points\_for\_projects[:, 0], ideal\_points\_for\_projects[:, 1], color='black', marker='o', label="Все точки")  
  
# Рисуем Парето-оптимальные точки зелёным  
plt.scatter(pareto\_points[:, 0], pareto\_points[:, 1], color='green', marker='o', label="Парето-оптимальные точки")  
  
# Выделяем оптимальную точку как красную точку  
plt.scatter(optimal\_point[0], optimal\_point[1], color='red', marker='o', label="Оптимальная точка")  
  
# Добавляем подписи к точкам  
x\_offset = 1.2 # Смещение по оси X для текста  
y\_offset = 0.05 # Смещение по оси Y для текста  
  
# Подпись для всех точек  
for i, point in enumerate(ideal\_points\_for\_projects):  
 plt.figtext(0.91, 0.93 - (i + 1) \* y\_offset, f"X{i + 1} ({point[0]}, {point[1]})", ha='left', va='top', fontsize=9)  
 if point[0] == optimal\_point[0] and point[1] == optimal\_point[1]:  
 solution\_pont\_maxmin = i+1  
# Настройки графика  
plt.xlabel('f1') # Подпись оси x  
plt.ylabel('f2') # Подпись оси y  
plt.title('График распределения точек минимаксного сожаления')  
  
# Показать сетку  
plt.grid(True)  
plt.legend()  
plt.show()  
  
#-------------------------ЗАДНАИЕ3-----------------------------------------------------------  
# Извлекаем только f1 значения из матрицы Q для дальнейшего анализа  
f1\_values = np.array([[project[0] for project in row] for row in Q])  
  
# Применяем принципы  
optimal\_project\_valda\_f1, valda\_value = valda\_criterion(f1\_values)  
optimal\_project\_savage\_f1, savage\_value = savage\_criterion(f1\_values)  
optimal\_project\_hurwicz\_f1, hurwicz\_value = hurwicz\_criterion(f1\_values, alpha=0.5)  
optimal\_project\_laplace\_f1, laplace\_value = laplace\_criterion(f1\_values)  
  
# Выводим оптимальные проекты по каждому критерию  
print(f"Оптимальный проект по принципу Вальда для f1: X{optimal\_project\_valda\_f1 + 1}, f1 = {valda\_value}")  
print(f"Оптимальный проект по принципу Сэвиджа для f1: X{optimal\_project\_savage\_f1 + 1}, f1 = {savage\_value}")  
print(f"Оптимальный проект по принципу Гурвица (α=0.5) для f1: X{optimal\_project\_hurwicz\_f1 + 1}, f1 = {hurwicz\_value}")  
print(f"Оптимальный проект по принципу Лапласа для f1:: X{optimal\_project\_laplace\_f1 + 1}, f1 = {laplace\_value} \n")  
  
#-------------------------ЗАДНАИЕ4-----------------------------------------------------------  
  
# Извлекаем только f2 значения из матрицы Q для дальнейшего анализа  
f2\_values = np.array([[project[1] for project in row] for row in Q])  
  
# Применяем принципы  
optimal\_project\_valda\_f2, valda\_value = valda\_criterion(f2\_values)  
optimal\_project\_savage\_f2, savage\_value = savage\_criterion(f2\_values)  
optimal\_project\_hurwicz\_f2, hurwicz\_value = hurwicz\_criterion(f2\_values, alpha=0.5)  
optimal\_project\_laplace\_f2, laplace\_value = laplace\_criterion(f2\_values)  
  
# Выводим оптимальные проекты по каждому критерию  
print(f"Оптимальный проект по принципу Вальда для f2: X{optimal\_project\_valda\_f2 + 1}, f2 = {valda\_value}")  
print(f"Оптимальный проект по принципу Сэвиджа для f2: X{optimal\_project\_savage\_f2 + 1}, f2 = {savage\_value}")  
print(f"Оптимальный проект по принципу Гурвица (α=0.5) для f2: X{optimal\_project\_hurwicz\_f2 + 1}, f2 = {hurwicz\_value}")  
print(f"Оптимальный проект по принципу Лапласа для f2: X{optimal\_project\_laplace\_f2 + 1}, f2 = {laplace\_value} \n")  
  
#-------------------------МАТРИЦА ГОЛОСОВАНИЯ-----------------------------------------------------------  
voting\_matrix = np.zeros((len(Q), 11), dtype=int)  
max\_total = 0  
for i in range (len(Q)):  
 total = 0  
 if i== solution\_pont\_mimnax:  
 voting\_matrix[i][0] = 1  
 total += 1  
 if i== solution\_pont\_maxmin:  
 voting\_matrix[i][1] = 1  
 total += 1  
 if i in optimal\_project\_valda\_f1:  
 voting\_matrix[i][2] = 1  
 total += 1  
 if i in optimal\_project\_savage\_f1:  
 voting\_matrix[i][3] = 1  
 total += 1  
 if i in optimal\_project\_hurwicz\_f1:  
 voting\_matrix[i][4] = 1  
 total += 1  
 if i in optimal\_project\_laplace\_f1:  
 voting\_matrix[i][5] = 1  
 total += 1  
 if i in optimal\_project\_valda\_f2:  
 voting\_matrix[i][6] = 1  
 total += 1  
 if i in optimal\_project\_savage\_f2:  
 voting\_matrix[i][7] = 1  
 total += 1  
 if i in optimal\_project\_hurwicz\_f2:  
 voting\_matrix[i][8] = 1  
 total += 1  
 if i in optimal\_project\_laplace\_f2:  
 voting\_matrix[i][9] = 1  
 total += 1  
 voting\_matrix[i][10] = total  
 if max\_total <= total:  
 max\_total = total  
# Формируем заголовки столбцов  
headers = [  
 "Вальд.f1", "Сэвидж.f1", "Гурвиц.f1", "Лапл.f1", "ВекторМаксМин.f1", "ВекторМинМакс.f1",  
 "Вальд.f2", "Сэвидж.f2", "Гурвиц.f2", "Лапл.f2", "Итог"  
]  
  
# Преобразуем матрицу голосования в формат для `tabulate`  
table = []  
for i, row in enumerate(voting\_matrix):  
 row\_data = row.tolist() # Преобразуем строку в список  
 table.append([f"X{i+1}"] + row\_data) # Добавляем индекс проекта  
  
# Выводим таблицу с результатами  
print("Матрица голосования:")  
print(tabulate(table, headers=["Проект"] + headers, tablefmt="grid"))  
  
# Выводим итоговый проект  
max\_votes = np.max(voting\_matrix[:, 10])  
optimal\_projects\_final = np.where(voting\_matrix[:, 10] == max\_votes)[0]  
optimal\_projects\_final = ', '.join([f"X{x+1}" for x in optimal\_projects\_final])  
print(f"\nСамые оптимальные проекты по итогам голосования: {optimal\_projects\_final}")