

**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 3-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

Оглавление

[**Цели работы** 3](#_Toc194853884)

[**Алгоритм исключения заведомо неэффективных решений** 4](#_Toc194853885)

[**Алгоритм построения конуса доминирования** 6](#_Toc194853886)

[**Кластеризация на основе индекса эффективности** 8](#_Toc194853887)

[**Работа программы на заданном объеме** 10](#_Toc194853888)

[**Приложение 1** 13](#_Toc194853889)

# **Цели работы**

В многокритериальной аналитической задаче множество достижимых векторных оценок задано системой неравенств

1. Построить множество парето-оптимальных решений. Для этого сгенерировать на множестве F(X) N=100 равномерно распределенных точек. Применить алгоритм исключения заведомо не эффективных решений.
2. Построить множество решений, оптимальных относительно полиэдрального конуса доминирования Ω. Применить алгоритм исключения заведомо неэффективных решений, используя представление Ω, как функции интервалов неопределенности весовых коэффициентов компонент векторного критерия . Рассмотреть следующие варианты интервалов неопределенности весовых коэффициентов, представленных в таблице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | µ1min | µ1max | µ2min | µ2max |
| 1 | 0.2 | 0.6 | 0.4 | 0.8 |
| 2 | 0.4 | 0.8 | 0.2 | 0.6 |
| 3 | 0.3 | 0.6 | 0.3 | 0.6 |

1. Построить множество парето-оптимальных решений и решений, оптимальных относительно полиэдрального конуса доминирования Ω для N = 1000 Провести сравнительный анализ мощности множеств:

# **Алгоритм исключения заведомо неэффективных решений**

Алгоритм предполагает проход по к точкам, и исключение заведомо неоптимальных решений.

1. Полагаем к =1
2. Выбираем элемент . Если имеет статус заведомо неоптимального решения, то переходим к шагу 4. Иначе переходим к шагу 3.
3. Для всех , проверяем выполнение условия . Все элементы , для которых выполняется данное условие считаются заведомо неэффективными и переходим к шагу 4.
4. Если , то полагаем что и переходим к шагу 2. Иначе переходим к шагу 5.
5. Просматриваем таблицу значений и удаляем из нее элементы, имеющие статус заведомо неоптимальных. Получаем таблицу . Полагаем, что множество Ω- оптимальных решений .

В качестве примера возьмем 100 сгенерированных точек. Их расположение показано на рисунке 1.

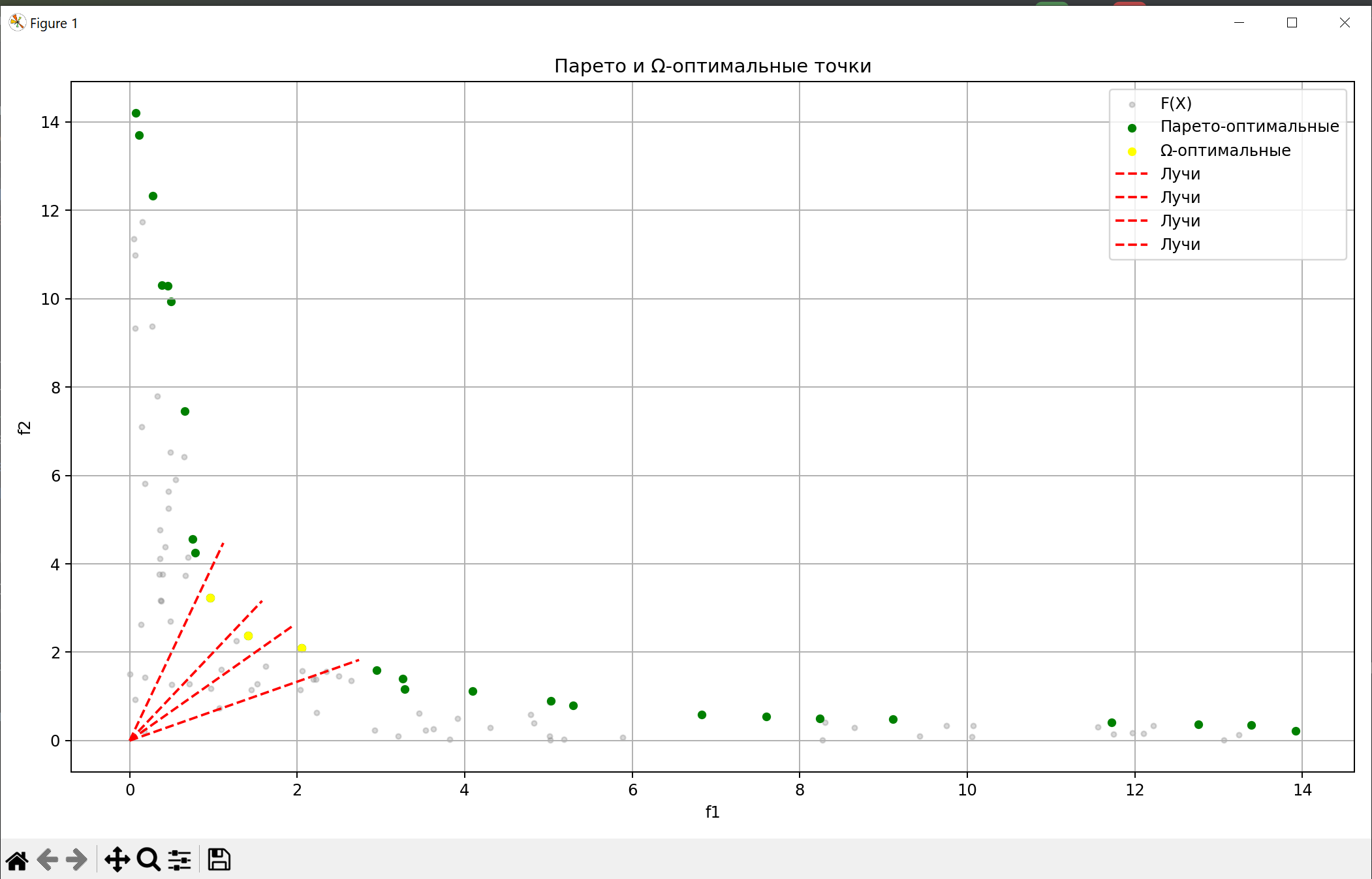


Рисунок 1 - График отображения Парето-оптимальных точек

# **Алгоритм построения конуса доминирования**

1. В пространстве весовых коэффициентов построим гиперпараллелепипед Π, все точки которого удовлетворяют условиям: и . Пронумеруем все вершины с помощью бинарного кода Грея(БКГ). Все БКГ представим в виде строк , таблицы ТΠ. Будем считать, что если в какой-либо строке j-й разряд справа , то j-ая координата i-ой вершины гиперпараллелепипеда Π равна . Если , то j-ая координата i-ой вершины Π равна . Далее полагаем что i = 0 и переходим к шагу 2.
2. Осуществляем попарное сравнение строк . Если выявлены строки tk и tl, отличающиеся друг от друга одним разрядом , то данной паре (tk,tl) соответствует вершина в таблице Π, образующее ребро в Π. Переходим к шагу 3, иначе переходим к шагу 5.
3. Осуществим анализ положения вершин tk и tl относительно гиперплоскости , определяющий условие компонентов вектора весовых коэффициентов. Если , то это означает что гиперплоскость пересекает ребро . Полагаем что и переходим к шагу 4.
4. Вычисляем точку пересечения ребра с гиперплоскостью и записываем ее координаты в виде строки с номером i в матрицу B. Переходим к шагу 5.
5. Проверяем, все ли пары строк таблицы ТΠ просмотрены. Если да то полагаем что , заканчиваем формирование матрицы B размером и переходим к шагу 6. Иначе переходим к шагу 2.
6. Построим конус доминирования в виде . Ω является полярным к выпуклому конусу ℧ , где система векторов образует строки матрицы B

Для демонстрации возьмем те же сгенерированные точки рассмотрим работу алгоритма для весовых коэффициентов варианта 3.

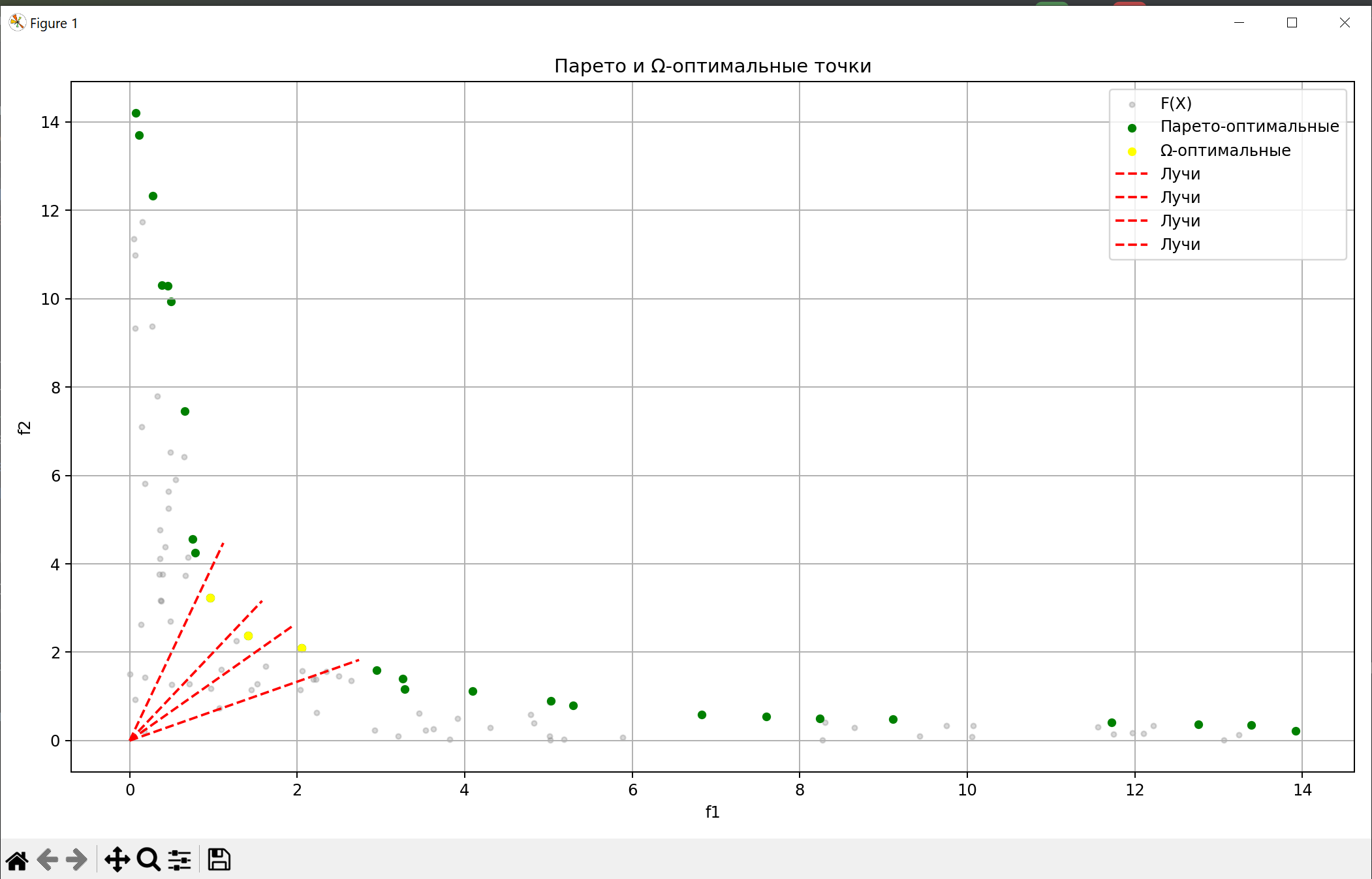


Рисунок 2 - Омега оптимальные точки

# Работа алгоритма на полных значениях

Для 1000 точек проведем генерацию точек и отображение Парето-оптимальных и Ω-оптимальных точек и проведем сравнительный анализ мощности множеств.

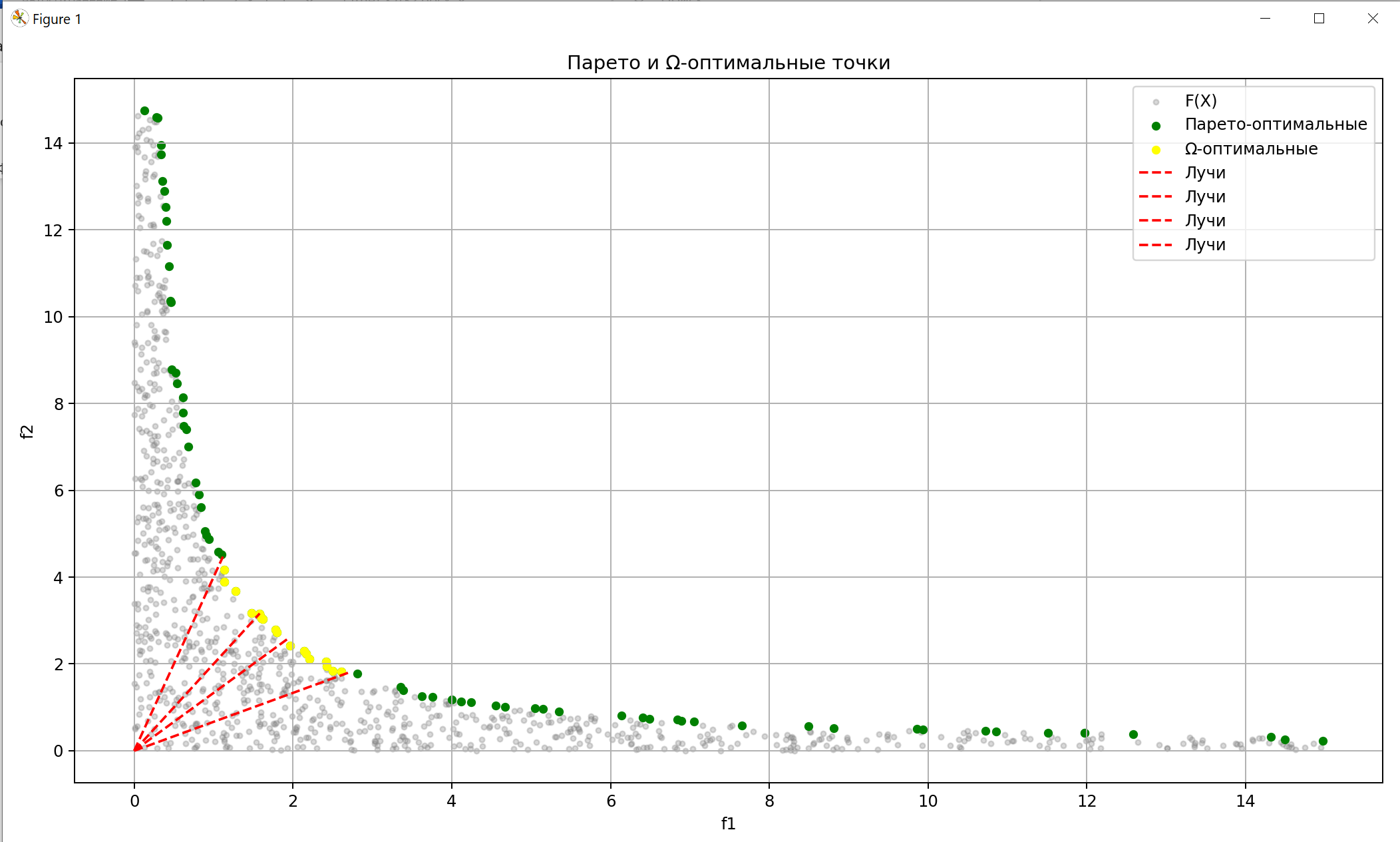


Рисунок 3 - Вариант 1

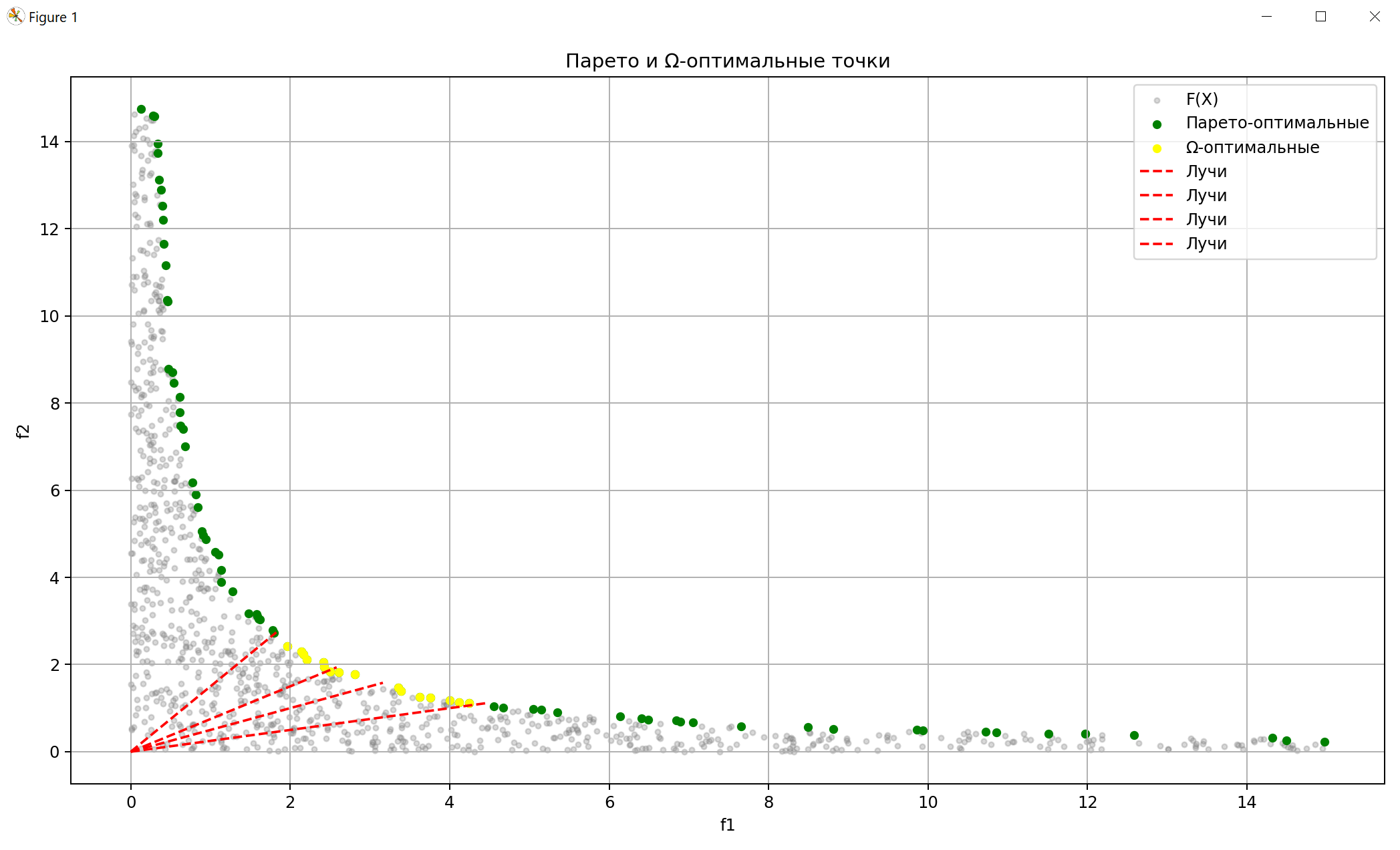


Рисунок 4- Вариант 2

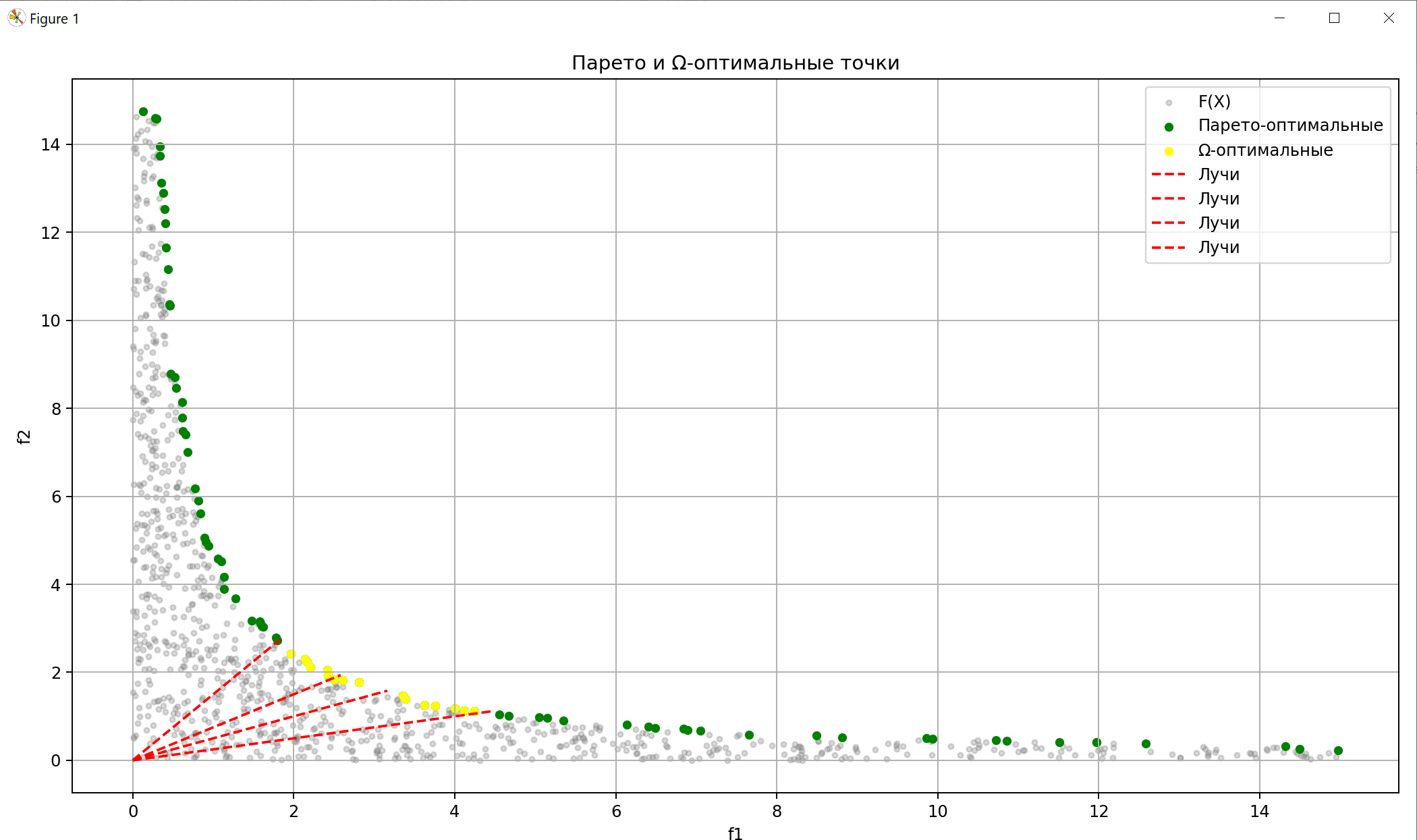


Рисунок 5 - Вариант 3

Сравнительный анализ множеств.

# **Приложение 1**

В приложении 1 представлен листинг кода программы

import numpy as np  
import matplotlib.pyplot as plt  
  
n = 5  
f1min,f1max,f2min,f2max=0,3\*n,0,3\*n  
# === Генерация бинарных кодов Грея ===  
def gray\_codes(n):  
 if n == 0:  
 return ['']  
 prev = gray\_codes(n - 1)  
 return ['0' + code for code in prev] + ['1' + code for code in reversed(prev)]  
  
  
# === Шаг 1. Генерация допустимых точек ===  
def generate\_feasible\_points(N=1000):  
 *"""Генерирует N точек, удовлетворяющих условию f1 \* f2 <= 5."""* points = []  
 while len(points) < N:  
 f1 = np.random.uniform(f1min, f1max)  
 f2 = np.random.uniform(f1min, f2max)  
 if f1 \* f2 <= n:  
 points.append([f1, f2])  
 return np.array(points)  
  
  
# === Шаг 2. Нахождение пересечений с гиперболой f1 \* f2 = 5 ===  
def find\_intersection\_with\_hyperbola(vector, origin=(0, 0)):  
 *"""  
 Для заданного вектора находим точку пересечения с гиперболой f1 \* f2 = 5.  
 Вектор задается как координаты (dx, dy).  
 """* dx, dy = vector  
 if dy == 0 or dx == 0: # Если вектор параллелен осям  
 return None  
 x\_intersect = np.sqrt(n \* dx / dy) # решаем для x  
 y\_intersect = (dy / dx) \* x\_intersect # решаем для y  
 return (x\_intersect, y\_intersect)  
  
  
# === Шаг 3. Нахождение угла между двумя векторами ===  
def angle\_between\_vectors(v1, v2):  
 *"""Вычисляет угол между двумя векторами в радианах."""* dot\_product = np.dot(v1, v2)  
 norm\_v1 = np.linalg.norm(v1)  
 norm\_v2 = np.linalg.norm(v2)  
 cos\_theta = dot\_product / (norm\_v1 \* norm\_v2)  
 cos\_theta = np.clip(cos\_theta, -1.0, 1.0) # Для числовой стабильности  
 return np.arccos(cos\_theta)  
  
  
# === Шаг 4. Нахождение Парето-оптимальных точек (максимизация) ===  
def pareto\_front\_maximization(points):  
 pareto = []  
 for i, p in enumerate(points):  
 if not any(np.all(q >= p) and np.any(q > p) for j, q in enumerate(points) if j != i):  
 pareto.append(p)  
 return np.array(pareto)  
  
  
# === Шаг 5. Построение матрицы B с учетом интервала неопределенности для весов ===  
def construct\_B(mu\_min, mu\_max):  
 *"""Строим матрицу B, содержащую векторы, определяющие конус."""* # В этой матрице будем хранить вектора (направления)  
 B = []  
  
 # Генерируем все возможные комбинации для 2D (mu\_min и mu\_max)  
 v1 = np.array([mu\_min[0], mu\_min[1]])  
 v2 = np.array([mu\_min[0], mu\_max[1]])  
 v3 = np.array([mu\_max[0], mu\_min[1]])  
 v4 = np.array([mu\_max[0], mu\_max[1]])  
  
 # Добавляем все возможные вектора в список B  
 B.append(v1)  
 B.append(v2)  
 B.append(v3)  
 B.append(v4)  
  
 return np.array(B)  
  
  
# === Шаг 6. Нахождение Ω-оптимальных точек с использованием углов ===  
def find\_omega\_optimal\_points(fx, B, pareto):  
 omega\_opt = []  
  
 # Шаг 1: Вычисляем углы для каждого вектора  
 angles = []  
 for b in B:  
 angle = angle\_between\_vectors(np.array([1, 0]), b) # угол с осью абсцисс  
 angles.append(angle)  
  
 # Шаг 2: Находим минимальный и максимальный углы для конуса  
 min\_angle = min(angles)  
 max\_angle = max(angles)  
  
 # Шаг 3: Проверяем, попадает ли угол прямой, идущей из каждой точки, в диапазон углов конуса  
 for p in pareto: # Только по парето-оптимальным точкам  
 vector\_p = np.array(p)  
 angle\_p = angle\_between\_vectors(np.array([1, 0]), vector\_p) # угол с осью абсцисс  
  
 if min\_angle <= angle\_p <= max\_angle:  
 omega\_opt.append(p)  
  
 return np.array(omega\_opt)  
  
  
# === Шаг 7. Отображение лучей и Ω-оптимальных точек ===  
def plot\_lights\_and\_optimal\_points(fx, pareto, f\_omega, B):  
 fig, axes = plt.subplots(1, 1, figsize=(12, 7))  
  
 # График Парето и Ω-оптимальных точек  
 axes.scatter(fx[:, 0], fx[:, 1], s=10, alpha=0.3, color='gray', label='F(X)')  
 if pareto.size > 0:  
 axes.scatter(pareto[:, 0], pareto[:, 1], s=20, color='green', label='Парето-оптимальные')  
 if f\_omega.size > 0:  
 axes.scatter(f\_omega[:, 0], f\_omega[:, 1], s=20, color='yellow', label='Ω-оптимальные')  
  
 # Добавляем лучи  
 for b in B:  
 intersection = find\_intersection\_with\_hyperbola(b)  
 if intersection:  
 axes.plot([0, intersection[0]], [0, intersection[1]], 'r--', label='Лучи')  
  
 axes.set\_title(f'Парето и Ω-оптимальные точки')  
 axes.set\_xlabel('f1')  
 axes.set\_ylabel('f2')  
 axes.legend()  
 axes.grid(True)  
  
 plt.tight\_layout()  
 plt.show()  
  
  
# === Шаг 8. Основной анализ с учетом пересечений и разбиений области Парето ===  
def analyze\_all\_cases():  
 cases = [  
 (0.2, 0.6, 0.4, 0.8),  
 (0.4, 0.8, 0.2, 0.6),  
 (0.3, 0.6, 0.3, 0.6)  
 ]  
  
 for N in [100, 1000]:  
 print(f"\n==== Анализ для N = {N} ====")  
 fx = generate\_feasible\_points(N)  
 pareto = pareto\_front\_maximization(fx)  
 print(f"|F(X)| = {len(fx)}")  
 print(f"|Fp(X)| = {len(pareto)}")  
  
 for i, (mu1min, mu1max, mu2min, mu2max) in enumerate(cases):  
 B = construct\_B([mu1min, mu2min], [mu1max, mu2max])  
 f\_omega = find\_omega\_optimal\_points(fx, B, pareto)  
  
 print(f"Кейс {i + 1}: |FΩ(X)| = {len(f\_omega)}")  
  
 # Визуализируем лучи и Ω-оптимальные точки на одном графике  
 plot\_lights\_and\_optimal\_points(fx, pareto, f\_omega, B)  
  
  
# === Запуск анализа ===  
analyze\_all\_cases()