

**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](#_Toc195618060)

[Алгоритм исключения заведомо неэффективных решений 4](#_Toc195618061)

[Алгоритм построения полиэдрального конуса доминирования 5](#_Toc195618062)

[Геометрическая интерпретация 7](#_Toc195618063)

[Сравнительный анализ множеств. 10](#_Toc195618064)

[Приложение 1 11](#_Toc195618065)

# **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 |
| 4 | 0.3 | 0.3 | 0.7 | 0.7 |

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 .

# **Алгоритм построения полиэдрального 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 100 точек рассмотрим работу алгоритма для весовых коэффициентов варианта 3.

# **Решение задачи**

Рассмотрим построение полиэдрального конуса доминирования с геометрической точки зрения для варианта 1 весовых коэффициентов µ1min= 0.2, µ1max= 0.6, µ2min= 0.4, µ2max= 0.8. Построим таблицу .

Определяем расположение вершин относительно прямой :

Определяем координаты точек пересечения ребер с прямой .

. Определим какая координата ребра остается неизменной. .

. Определим какая координата ребра остается неизменной. .

,. Определим какая координата ребра остается неизменной. .

,. Определим какая координата ребра остается неизменной. .

Запишем матрицу B.

Так как в матрице есть повторяющиеся и . Получим конечную матрицу полиэдрального конуса доминирования.

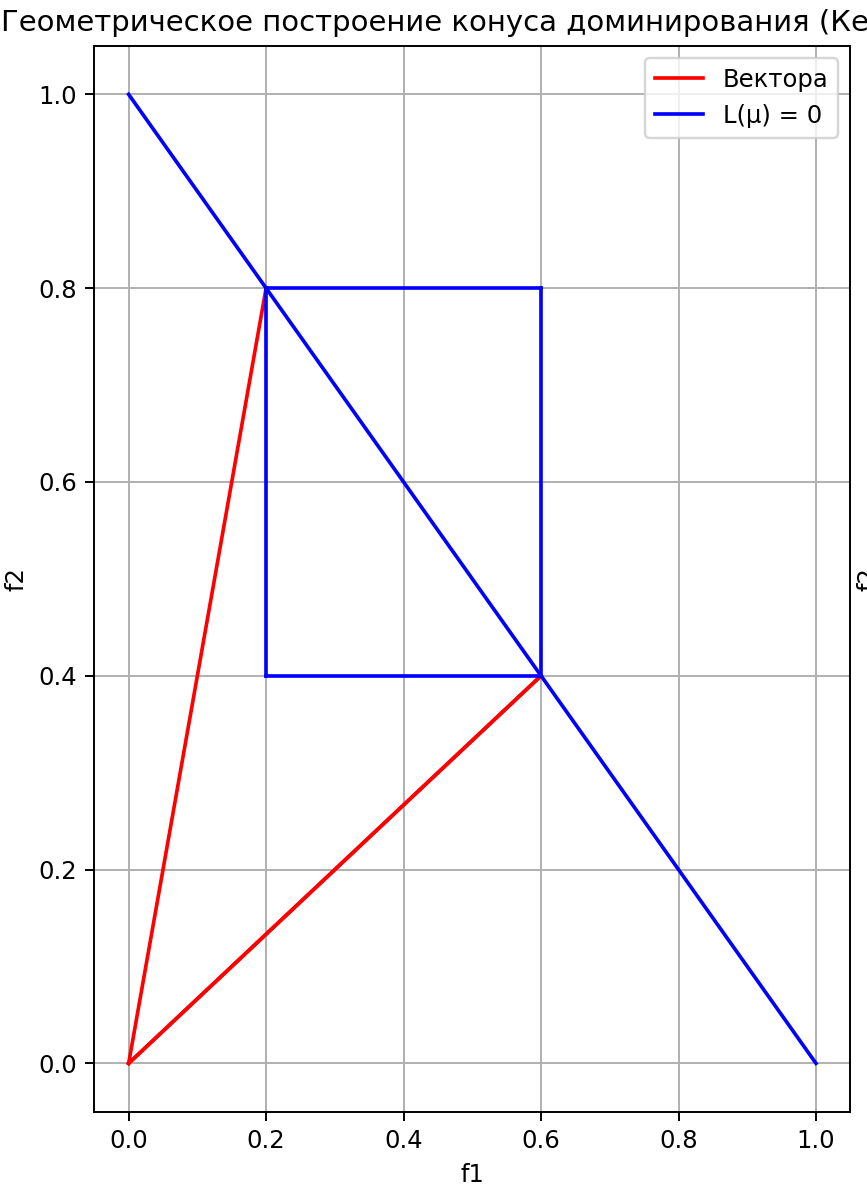


Рисунок 1 - Геометрическая интерпретация

Данный гиперпарраллелепипед пересекает прямая . Точки пересечения прямой ребер гиперпараллелепипеда становятся концами векторов b. Сами вектора b образуют полиэдральный конус доминирования Ω.

Для построения дискретной аппроксимации множества Ω - оптимальных решений сгенерируем конечное множество допустимых точек таких, что . Далее на множестве этих точек построим дискретную аппроксимацию Парето-оптимальных решений используя алгоритм исключения заведомо не оптимальных решений. Результат представлен на рисунке 2.

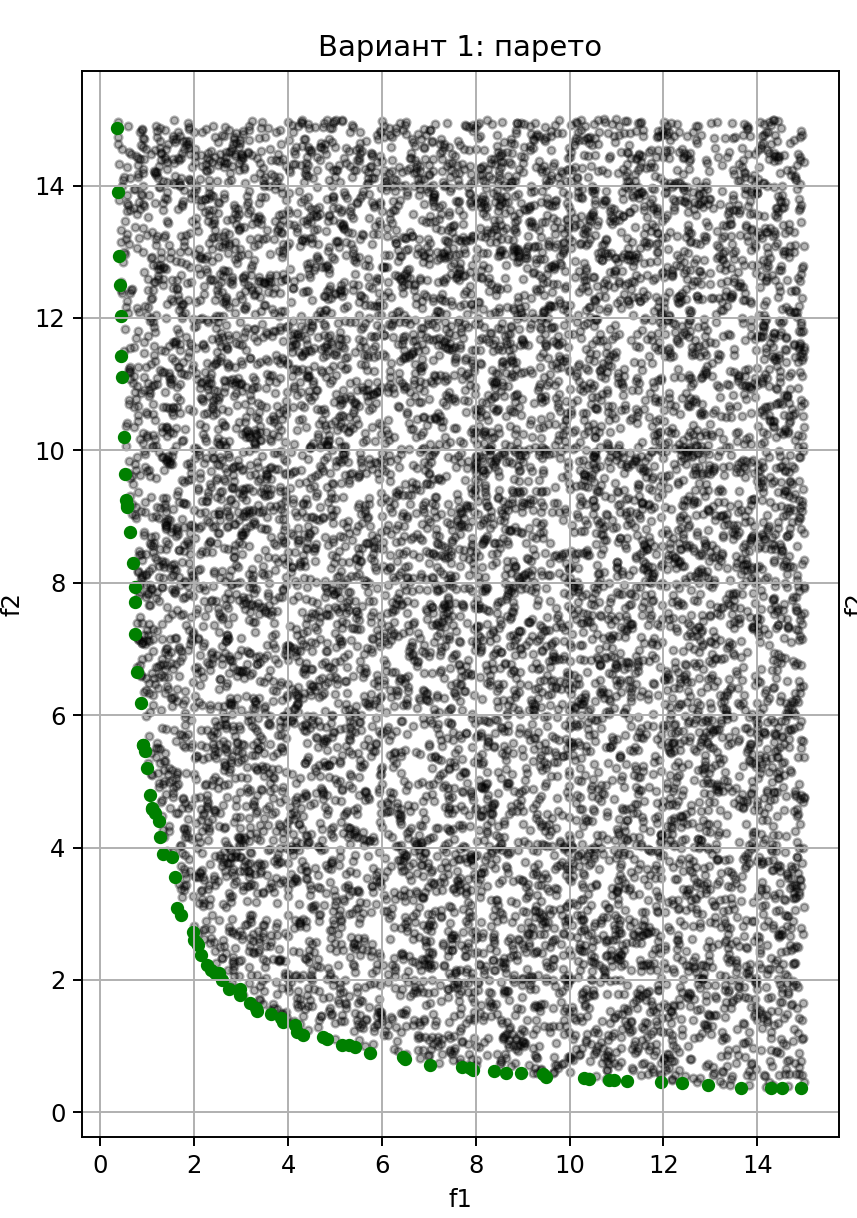


Рисунок 2 - Дискретная аппроксимация множества парето-оптимальных решений

Для построения дискретной аппроксимации множества Ω - оптимальных решений используем алгоритм исключения заведомо не эффективных решений. В качестве условия исключения точки рассматриваем выполнение неравенства . Результатом алгоритма становится получение множества , изображенное на рисунке 3.

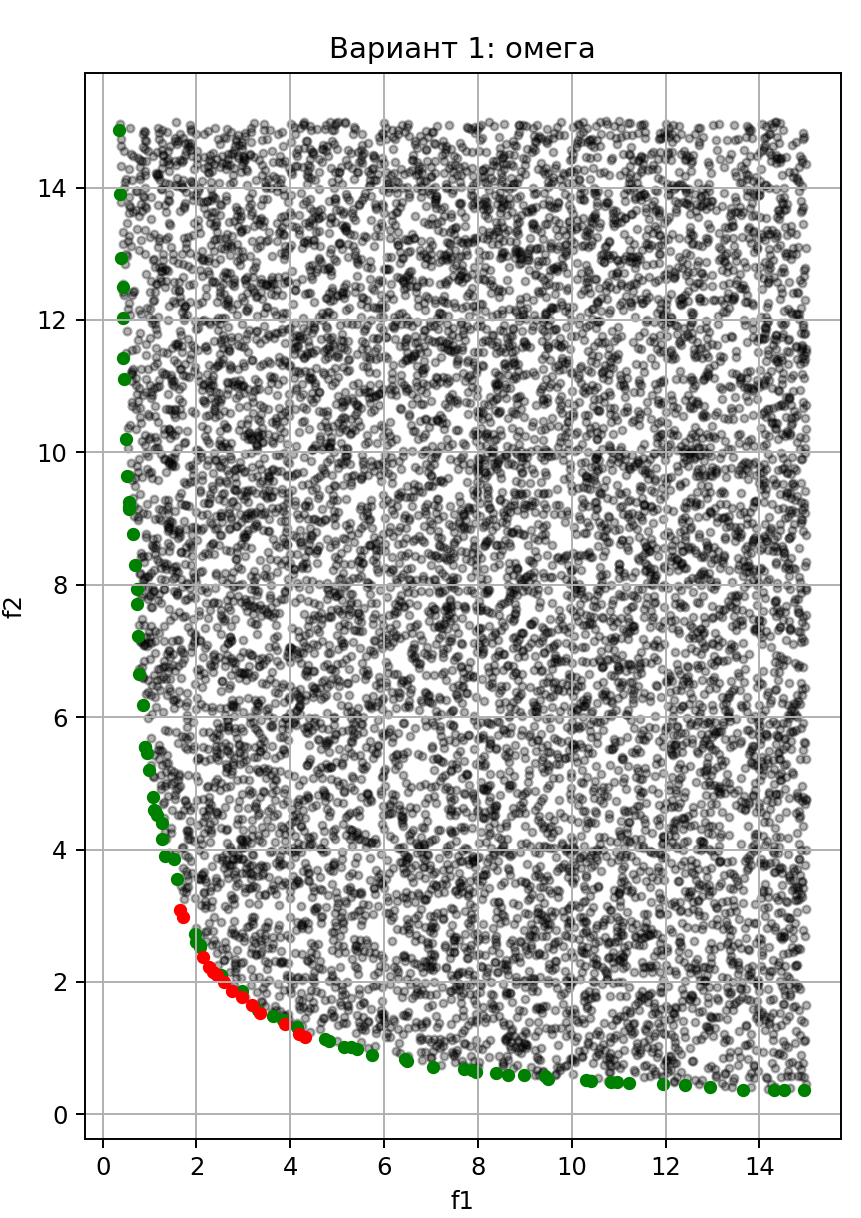


Рисунок 3 - дискретная аппроксимация множества Ω-оптимальных решений

# **Сравнительный анализ множеств.**

Для сравнительного анализа были взяты результаты генерации конечного множества точек, где . Результаты сравнительного анализа мощности множеств приведены в таблице 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | µ1min | µ1max | µ2min | µ2max |  |
| 10000 | 69 | 0.2 | 0.6 | 0.4 | 0.8 | 14 |
| 10000 | 69 | 0.4 | 0.8 | 0.2 | 0.6 | 14 |
| 10000 | 69 | 0.3 | 0.6 | 0.3 | 0.6 | 7 |
| 10000 | 85 | 0.3 | 0.3 | 0.7 | 0.7 | 1 |

Сравнение множеств представлено на рисунках 4-6

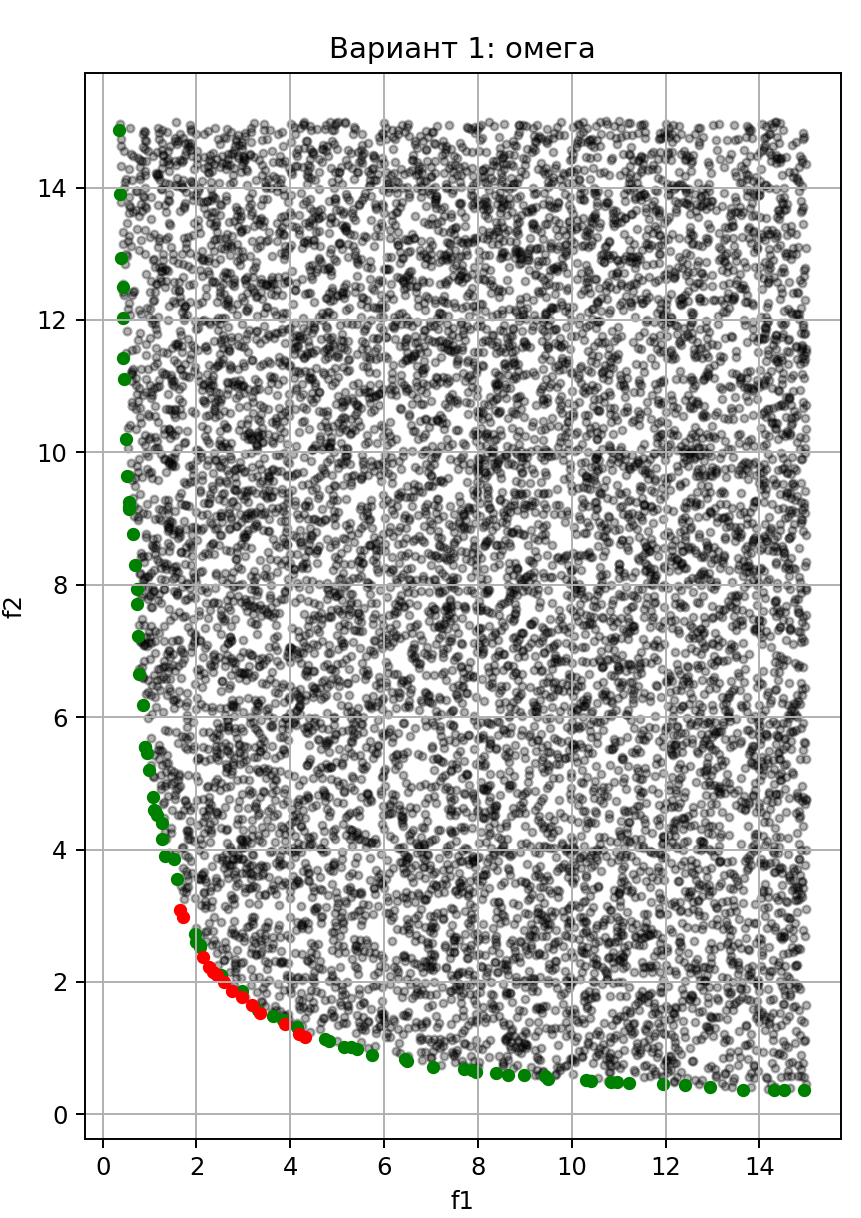


Рисунок 4 - Множество с весами варианта 1

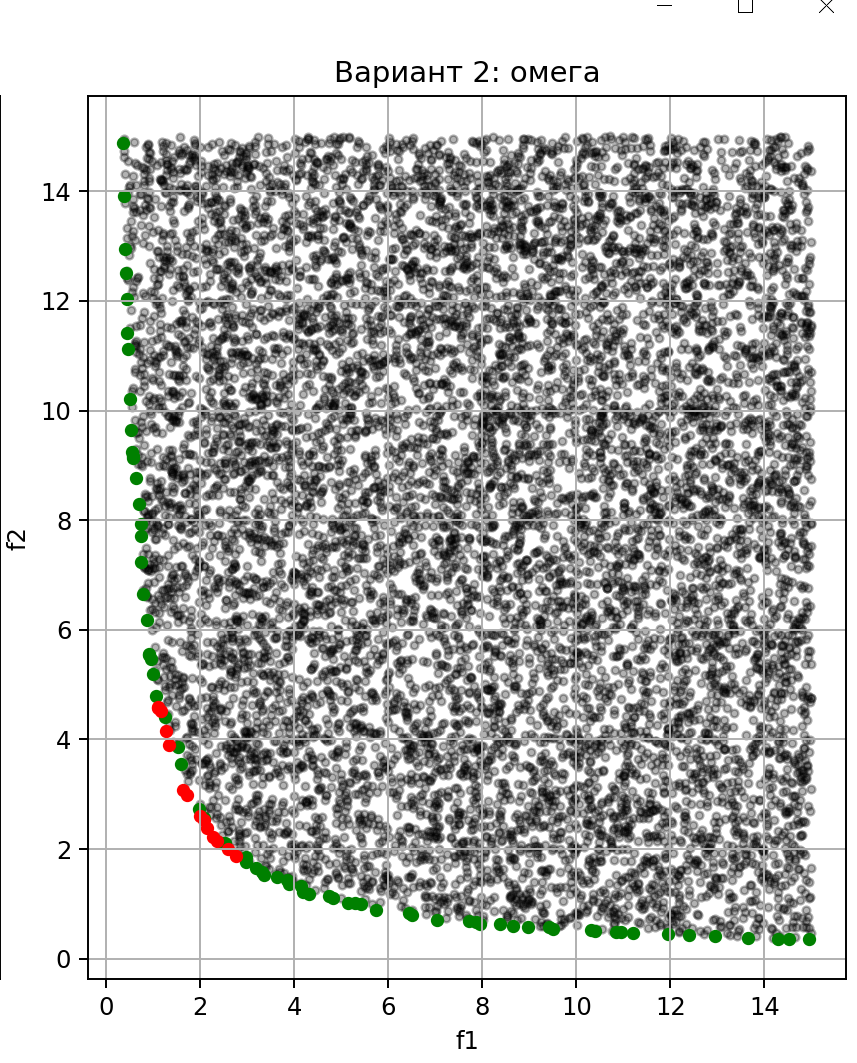


Рисунок 5 - Множество с весами варианта 2

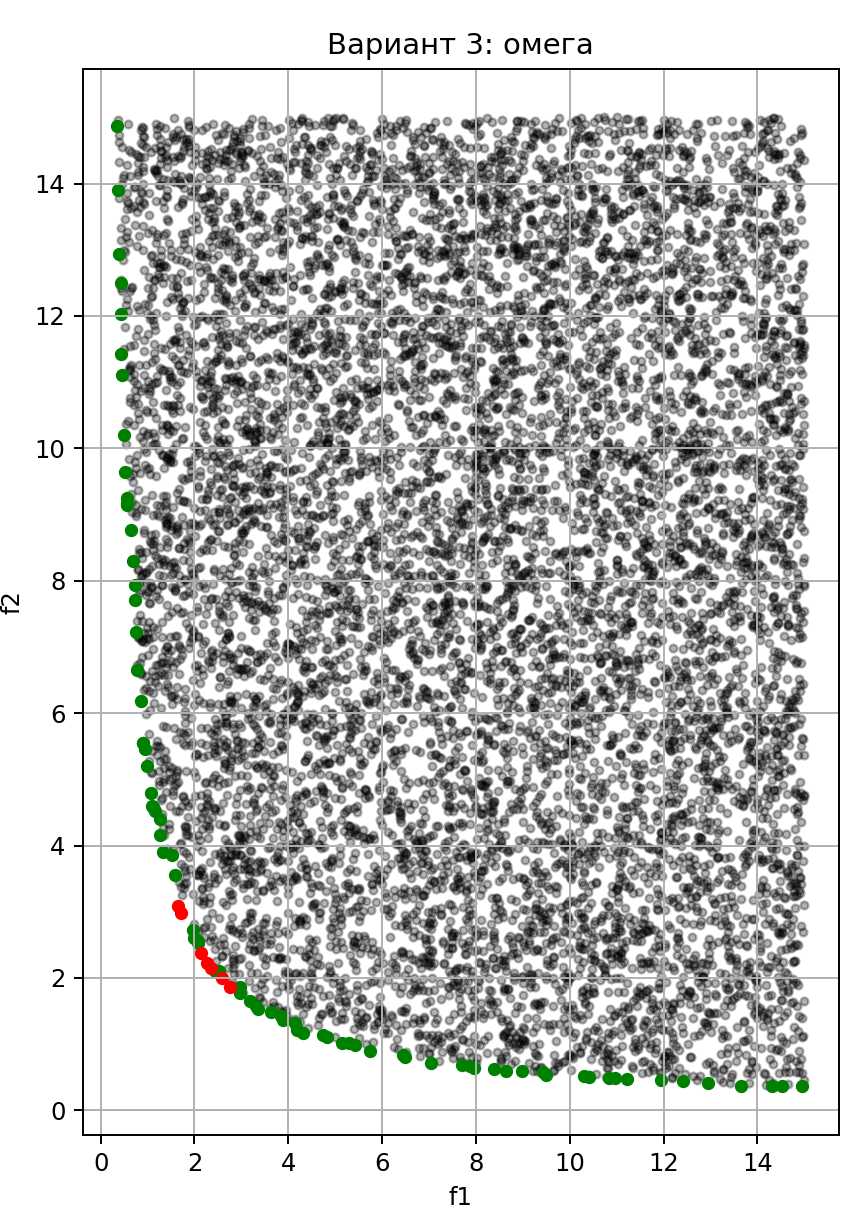


Рисунок 6 - Множество с весами варианта 3

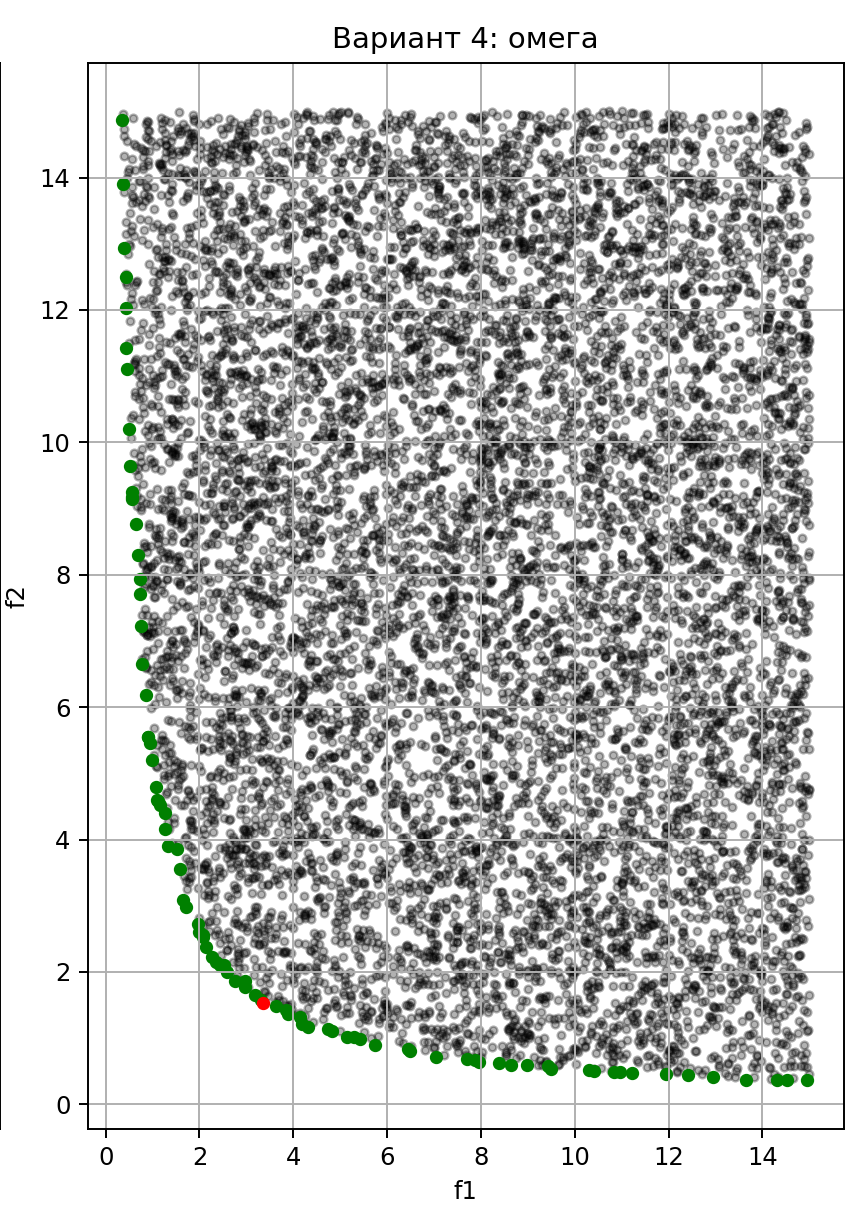


Рисунок 7 - Множество с весами варианта 4

# **Приложение 1**

В приложении 1 представлен листинг кода программы

import numpy as np  
import matplotlib.pyplot as plt  
  
# Порог и границы для генерации точек  
n = 5  
f1\_min, f1\_max, f2\_min, f2\_max = 0, 3\*n, 0, 3\*n  
  
# 1) Генерация допустимых точек  
def generate\_feasible\_points(N=100):  
 pts = []  
 while len(pts) < N:  
 x = np.random.uniform(f1\_min, f1\_max)  
 y = np.random.uniform(f2\_min, f2\_max)  
 if x \* y >= n:  
 pts.append([x, y])  
 return np.array(pts)  
  
# 2) 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)]  
  
# 3) 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])  
  
# 4) Построение матрицы B для заданного интервала весов  
def construct\_polyhedral\_cone(mu\_min, mu\_max):  
 # случай точных весов  
 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))  
  
# 5) Проверка: Fi доминируется Fj по конусу B?  
def is\_dominated\_by(Fi, Fj, B):  
 if B.size == 0:  
 return False  
 diff = Fi - Fj # для минимизации  
 ge = [np.dot(b, diff) >= 0 for b in B]  
 gt = [np.dot(b, diff) > 0 for b in B]  
 return all(ge) and any(gt)  
  
# 6) Однократное построение парето через B0 = B([0,0],[1,1])  
def build\_pareto(fx):  
 B0 = construct\_polyhedral\_cone([0,0], [1,1])  
 pareto = []  
 for Fi in fx:  
 if not any(is\_dominated\_by(Fi, Fj, B0) for Fj in fx if not np.allclose(Fi, Fj)):  
 pareto.append(Fi)  
 return np.array(pareto)  
  
# 7) Ω‑оптимальные через B для текущих весов  
def build\_omega(pareto, B):  
 omega = []  
 for Fi in pareto:  
 if not any(is\_dominated\_by(Fi, Fj, B) for Fj in pareto if not np.allclose(Fi, Fj)):  
 omega.append(Fi)  
 return np.array(omega)  
  
# 8) Рисуем три графика  
def plot\_all(fx, pareto, omega, B, mu\_min, mu\_max, case\_num):  
 fig, axs = plt.subplots(1,3,figsize=(18,6))  
  
 # 1) конус  
 ax = axs[0]  
 ax.set\_title(f'Кейс {case\_num}: конус')  
 for v in B:  
 ax.plot([0,v[0]],[0,v[1]],'r-')  
 ax.plot([0,1],[1,0],'b-')  
 ax.plot([mu\_min[0],mu\_max[0]],[mu\_min[1],mu\_min[1]],'b-')  
 ax.plot([mu\_min[0],mu\_max[0]],[mu\_max[1],mu\_max[1]],'b-')  
 ax.plot([mu\_min[0],mu\_min[0]],[mu\_min[1],mu\_max[1]],'b-')  
 ax.plot([mu\_max[0],mu\_max[0]],[mu\_min[1],mu\_max[1]],'b-')  
 ax.set\_xlabel('f1'); ax.set\_ylabel('f2'); ax.grid(True)  
  
 # 2) парето  
 ax = axs[1]  
 ax.set\_title(f'Кейс {case\_num}: парето')  
 ax.scatter(fx[:,0], fx[:,1], s=10, alpha=0.3, color='black')  
 if pareto.size:  
 ax.scatter(pareto[:,0], pareto[:,1], s=20, color='green')  
 ax.set\_xlabel('f1'); ax.set\_ylabel('f2'); ax.grid(True)  
  
 # 3) омега  
 ax = axs[2]  
 ax.set\_title(f'Кейс {case\_num}: омега')  
 ax.scatter(fx[:,0], fx[:,1], s=10, alpha=0.3, color='black')  
 non\_omega = np.array([p for p in pareto if not any(np.allclose(p,w) for w in omega)])  
 if non\_omega.size:  
 ax.scatter(non\_omega[:,0], non\_omega[:,1], s=20, color='green')  
 if omega.size:  
 ax.scatter(omega[:,0], omega[:,1], s=20, color='red')  
 ax.set\_xlabel('f1'); ax.set\_ylabel('f2'); ax.grid(True)  
  
 plt.tight\_layout()  
 plt.show()  
  
# 9) Основной анализ  
def analyze\_all():  
 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),  
 (0.3, 0.3, 0.7, 0.7)  
 ]  
 for N in [1000]:  
 fx = generate\_feasible\_points(N)  
 pareto = build\_pareto(fx) # только один раз  
 print(f"N={N}, |F|={len(fx)}, |Pareto|={len(pareto)}")  
 for idx,(a1,b1,a2,b2) in enumerate(cases,1):  
 mu\_min, mu\_max = [a1,a2],[b1,b2]  
 B = construct\_polyhedral\_cone(mu\_min, mu\_max)  
 omega = build\_omega(pareto, B)  
 print(f" Case {idx}: |Ω|={len(omega)}")  
 plot\_all(fx, pareto, omega, B, mu\_min, mu\_max, idx)  
  
if \_\_name\_\_=="\_\_main\_\_":  
 analyze\_all()