

**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 №3**

**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**

Дана многокритериальная аналитическая задача:

При ограничениях:

Решить поставленную задачу методом пороговой оптимизации.

Условия задачи заданы в таблице.

|  |  |  |
| --- | --- | --- |
|  | – главный критерий |  |

При ограничениях

# **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.

# **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.

# **Геометрическая интерпретация**

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. Данные значения весовых коэффициентов образуют гиперпараллелепипед изображенный на рисунке 1.

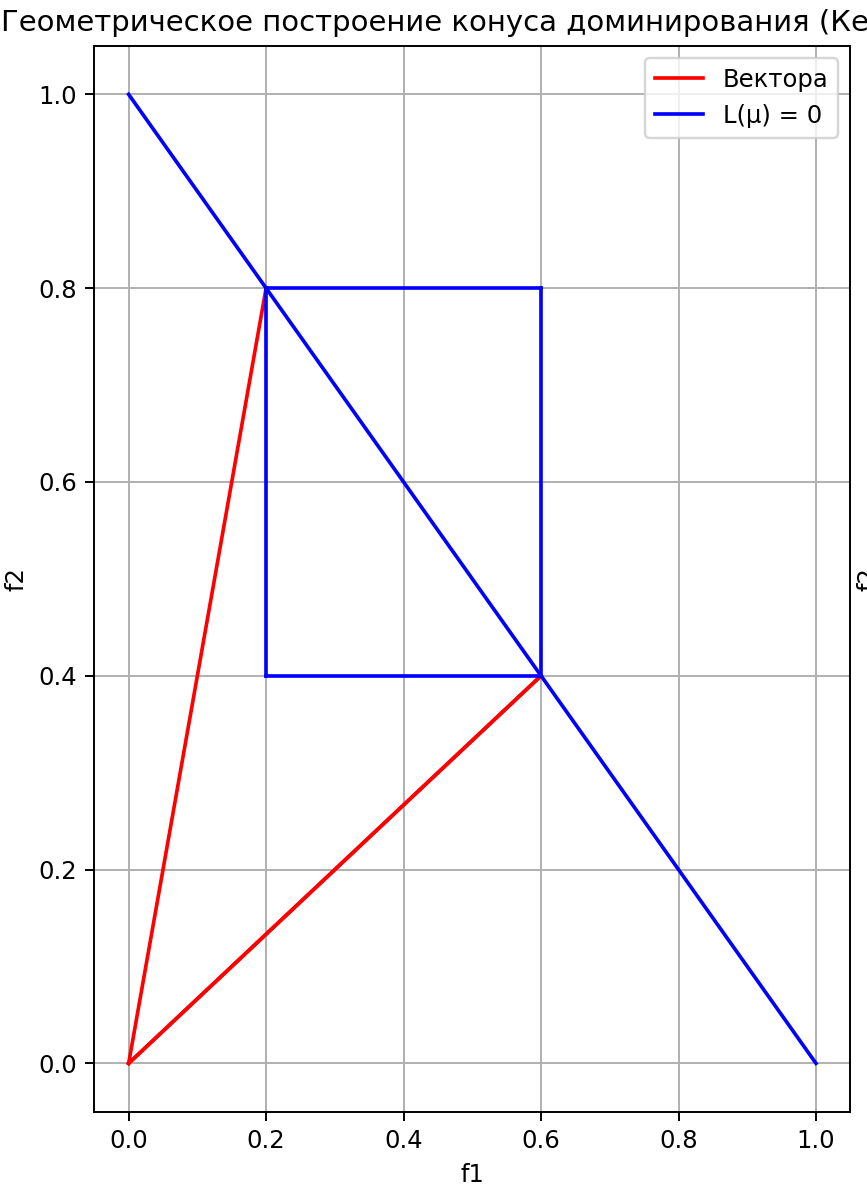


Рисунок 1 - Геометрическая интерпретация

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.

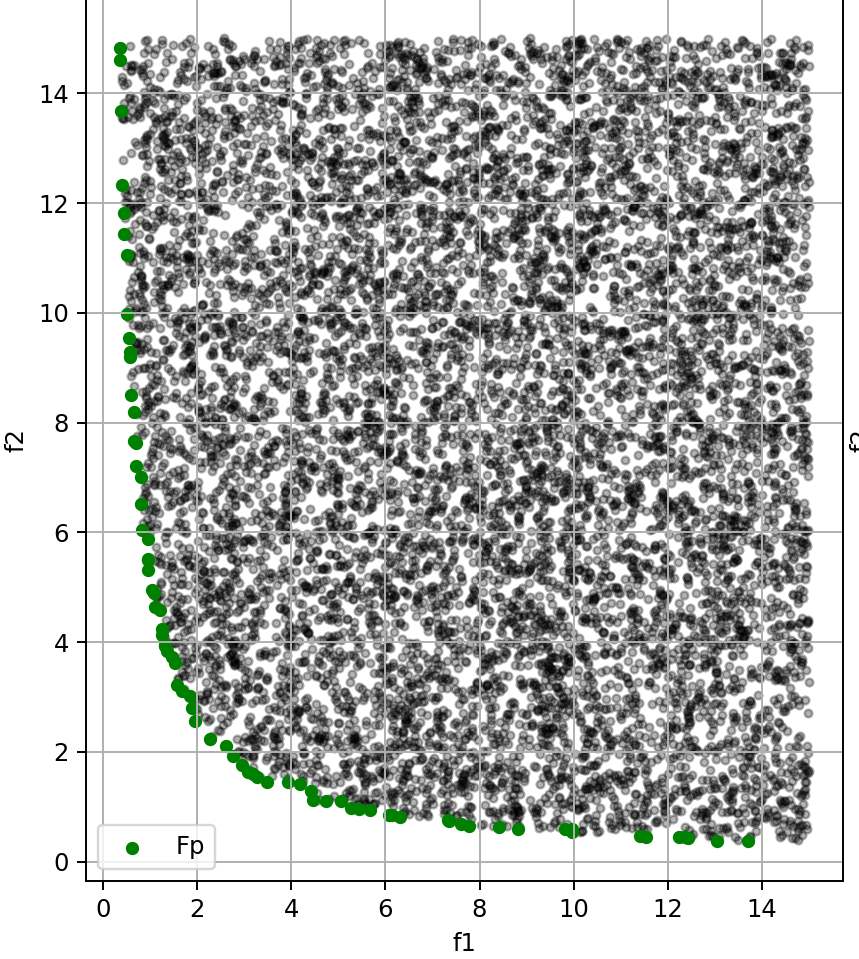


Рисунок 2 - Дискретная аппроксимация множества парето-оптимальных решений

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.

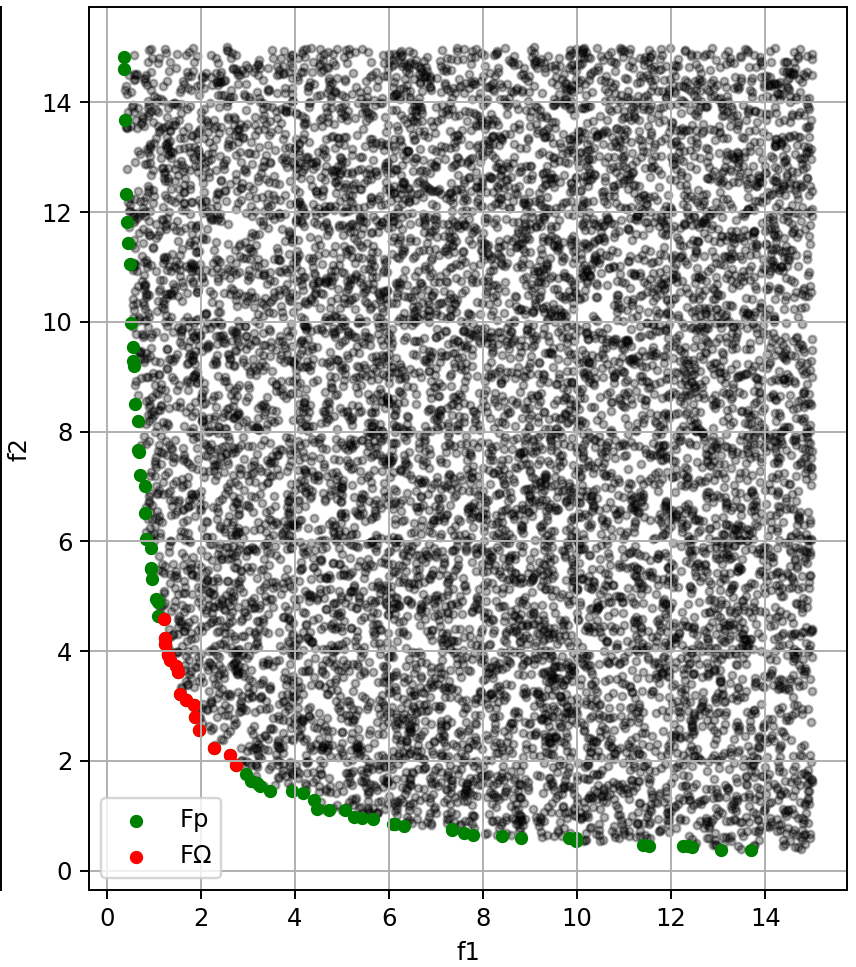


Рисунок 3 - дискретная аппроксимация множества Ω-оптимальных решений

# **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 | 89 | 0.2 | 0.6 | 0.4 | 0.8 | 23 |
| 10000 | 89 | 0.4 | 0.8 | 0.2 | 0.6 | 19 |
| 10000 | 89 | 0.3 | 0.6 | 0.3 | 0.6 | 9 |

# **Приложение 1**

В приложении 1 представлен листинг кода программы

import numpy as np  
import matplotlib.pyplot as plt  
  
  
n = 5  
f1\_min,f1\_max,f2\_min,f2\_max=0,3\*n,0,3\*n  
# === Генерация допустимых точек (условие f1\*f2 >= 5) ===  
def generate\_feasible\_points(N=100):  
 *"""Генерирует N точек, удовлетворяющих условию f1 \* f2 >= 5."""* points = []  
 while len(points) < N:  
 f1 = np.random.uniform(f1\_min, f1\_max)  
 f2 = np.random.uniform(f2\_min, f2\_max)  
 if f1 \* f2 >= n: # Условие минимизации  
 points.append([f1, f2])  
 return np.array(points)  
  
  
# === Шаг 2. Построение полиэдрального конуса доминирования (с динамическим пересчетом векторов) ===  
def find\_intersection(mu\_min, mu\_max):  
 *"""Находим точку пересечения прямой L(μ) = 0 с ребрами гиперпараллелепипеда."""* # Прямая L(μ) = 0: соединение точек (0,1) и (1,0)  
 # Линия y = 1 - x (так как проходящая через (0,1) и (1,0))  
  
 intersections = []  
  
 # Пересечение с ребром, где f1 = μ1min  
 x\_intersect1 = mu\_min[0]  
 y\_intersect1 = 1 - x\_intersect1 # y = 1 - x  
 if mu\_min[1] <= y\_intersect1 <= mu\_max[1]:  
 intersections.append([x\_intersect1, y\_intersect1])  
  
 # Пересечение с ребром, где f1 = μ1max  
 x\_intersect2 = mu\_max[0]  
 y\_intersect2 = 1 - x\_intersect2 # y = 1 - x  
 if mu\_min[1] <= y\_intersect2 <= mu\_max[1]:  
 intersections.append([x\_intersect2, y\_intersect2])  
  
 # Пересечение с ребром, где f2 = μ2min  
 y\_intersect3 = mu\_min[1]  
 x\_intersect3 = 1 - y\_intersect3 # x = 1 - y  
 if mu\_min[0] <= x\_intersect3 <= mu\_max[0]:  
 intersections.append([x\_intersect3, y\_intersect3])  
  
 # Пересечение с ребром, где f2 = μ2max  
 y\_intersect4 = mu\_max[1]  
 x\_intersect4 = 1 - y\_intersect4 # x = 1 - y  
 if mu\_min[0] <= x\_intersect4 <= mu\_max[0]:  
 intersections.append([x\_intersect4, y\_intersect4])  
  
 return np.array(intersections)  
  
  
# === Вычисление угла между вектором и осью абсцисс ===  
def compute\_angle(v):  
 *"""Вычисляет угол между вектором v и осью абсцисс."""* return np.arctan2(v[1], v[0]) # arctan2 учитывает и знак угла  
  
  
# === Шаг 3. Проверка попадания точки в полиэдральный конус ===  
def is\_point\_in\_cone(F, B):  
 *"""Проверяет, попадает ли точка F в полиэдральный конус, определенный матрицей B."""* # Вычисляем углы для всех векторов в B  
 angles = [compute\_angle(b) for b in B]  
  
 # Нахождение минимального и максимального углов  
 fi\_min = min(angles)  
 fi\_max = max(angles)  
  
 # Вычисляем угол для точки F  
 fi = compute\_angle(F)  
  
 # Точка попадает в полиэдральный конус, если угол лежит в пределах [fi\_min, fi\_max]  
 return fi\_min <= fi <= fi\_max  
  
  
# === Шаг 4. Проверка доминирования между двумя точками для минимизации ===  
def is\_dominating(F\_i, F\_j):  
 *"""Проверяет, доминирует ли точка F\_i над точкой F\_j в задаче минимизации."""* return (F\_i[0] <= F\_j[0] and F\_i[1] <= F\_j[1]) and (F\_i[0] < F\_j[0] or F\_i[1] < F\_j[1])  
  
  
# === Шаг 5. Нахождение точек, принадлежащих полиэдральному конусу доминирования ===  
def find\_points\_in\_cone(fx, B):  
 *"""Находим все точки, которые принадлежат полиэдральному конусу доминирования."""* points\_in\_cone = []  
 for F in fx:  
 if is\_point\_in\_cone(F, B):  
 # Проверяем, что точка не доминируется другими точками  
 is\_efficient = True  
 for F\_j in fx:  
 if F is not F\_j and is\_dominating(F\_j, F):  
 is\_efficient = False  
 break  
 if is\_efficient:  
 points\_in\_cone.append(F)  
 return np.array(points\_in\_cone)  
  
  
# === Шаг 6. Построение полиэдрального конуса доминирования (с динамическим пересчетом векторов) ===  
def construct\_polyhedral\_cone(mu\_min, mu\_max):  
 *"""Строим полиэдральный конус доминирования с учетом пересечений прямой L(μ) = 0 и ребер гиперпараллелепипеда."""* B = []  
  
 # Пересекаем гиперпараллелепипед с прямой L(μ) = 0  
 intersections = find\_intersection(mu\_min, mu\_max)  
  
 # Добавляем все пересечения в список B (векторы от (0, 0) к точкам пересечения)  
 for point in intersections:  
 B.append(point)  
  
 return np.array(B)  
  
  
# === Отображение графиков ===  
def plot\_lights\_and\_optimal\_points(fx, pareto, f\_omega, B, mu\_min, mu\_max, case\_num):  
 fig, axes = plt.subplots(1, 3, figsize=(18, 6))  
  
 # 1-й график: Геометрическое построение конуса доминирования  
 axes[0].set\_title(f'Геометрическое построение конуса доминирования (Вариант {case\_num})')  
 axes[0].set\_xlabel('f1')  
 axes[0].set\_ylabel('f2')  
  
 # Рисуем лучи для каждого пересечения  
 mem=0  
 for b in B:  
 if mem == 0:  
 axes[0].plot([0, b[0]], [0, b[1]], 'r-', label='Вектора') # Непрерывные красные линии  
 mem +=1  
 else:  
 axes[0].plot([0, b[0]], [0, b[1]], 'r-')  
 # Рисуем прямую L(μ) = 0 (от (0, 1) до (1, 0))  
 axes[0].plot([0, 1], [1, 0], 'b-', label='L(μ) = 0') # Прямая L(μ) = 0 (синий)  
  
 # Рисуем гиперпараллелепипед (прямоугольник с синими линиями)  
 axes[0].plot([mu\_min[0], mu\_max[0]], [mu\_min[1], mu\_min[1]], 'b-') # Нижняя граница  
 axes[0].plot([mu\_min[0], mu\_max[0]], [mu\_max[1], mu\_max[1]], 'b-') # Верхняя граница  
 axes[0].plot([mu\_min[0], mu\_min[0]], [mu\_min[1], mu\_max[1]], 'b-') # Левая граница  
 axes[0].plot([mu\_max[0], mu\_max[0]], [mu\_min[1], mu\_max[1]], 'b-') # Правая граница  
  
 # 2-й график: Все точки (черные) и парето-оптимальные точки (зеленые)  
 axes[1].scatter(fx[:, 0], fx[:, 1], s=10, alpha=0.3, color='black')  
 if pareto.size > 0:  
 axes[1].scatter(pareto[:, 0], pareto[:, 1], s=20, color='green', label = "Fp")  
 axes[1].set\_title(f'Все точки и Парето-оптимальные (Вариант {case\_num})')  
 axes[1].set\_xlabel('f1')  
 axes[1].set\_ylabel('f2')  
  
 # 3-й график: Все точки (черные), парето-оптимальные точки, омега-оптимальные (красные)  
 axes[2].scatter(fx[:, 0], fx[:, 1], s=10, alpha=0.3, color='black')  
 if pareto.size > 0:  
 axes[2].scatter(pareto[:, 0], pareto[:, 1], s=20, color='green', label= 'Fp')  
 if f\_omega.size > 0:  
 axes[2].scatter(f\_omega[:, 0], f\_omega[:, 1], s=20, color='red', label = 'FΩ')  
 axes[2].set\_title(f'Парето и Омега-оптимальные (Вариант {case\_num})')  
 axes[2].set\_xlabel('f1')  
 axes[2].set\_ylabel('f2')  
  
 for ax in axes:  
 ax.grid(True)  
 ax.legend()  
  
 plt.tight\_layout()  
 plt.show()  
  
  
# === Шаг 7. 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 [10000]:  
 print(f"\n==== Анализ для N = {N} ====")  
 fx = generate\_feasible\_points(N)  
 pareto = find\_points\_in\_cone(fx,construct\_polyhedral\_cone([0, 0], [1,1])) # для парето-оптимальных точек использовать веса 0,1,0,1  
 print(f"|F(X)| = {len(fx)}")  
 print(f"|Fp(X)| = {len(pareto)}")  
  
 for i, (mu1min, mu1max, mu2min, mu2max) in enumerate(cases):  
 B = construct\_polyhedral\_cone([mu1min, mu2min], [mu1max, mu2max])  
 f\_omega = find\_points\_in\_cone(fx, B)  
  
 print(f"Кейс {i + 1}: |FΩ(X)| = {len(f\_omega)}")  
  
 # Визуализируем результаты  
 plot\_lights\_and\_optimal\_points(fx, pareto, f\_omega, B, [mu1min, mu2min], [mu1max, mu2max], i + 1)  
  
  
# === Запуск анализа ===  
analyze\_all\_cases()