

**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 №5**

**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:

И

Решить задачу методом идеальной точки. Использовать алгоритм Франк-Вульфа

# **Метод «идеальной» точки**

Алгоритм нахождение оптимальной точки путем сравнения точек из ОДЗ по определенной метрике относительно «идеальной» точки

1. Построение идеальной точки. Для этого последовательно решаем следующие задачи

Максимизация критерия 1.

Оптимальное решение достигается в точке

Максимизация критерия 2.

Оптимальное решение достигается в точке

Максимизация критерия 3.

Оптимальное решение достигается в точке

Результаты решения задач представлены на рисунке 1.

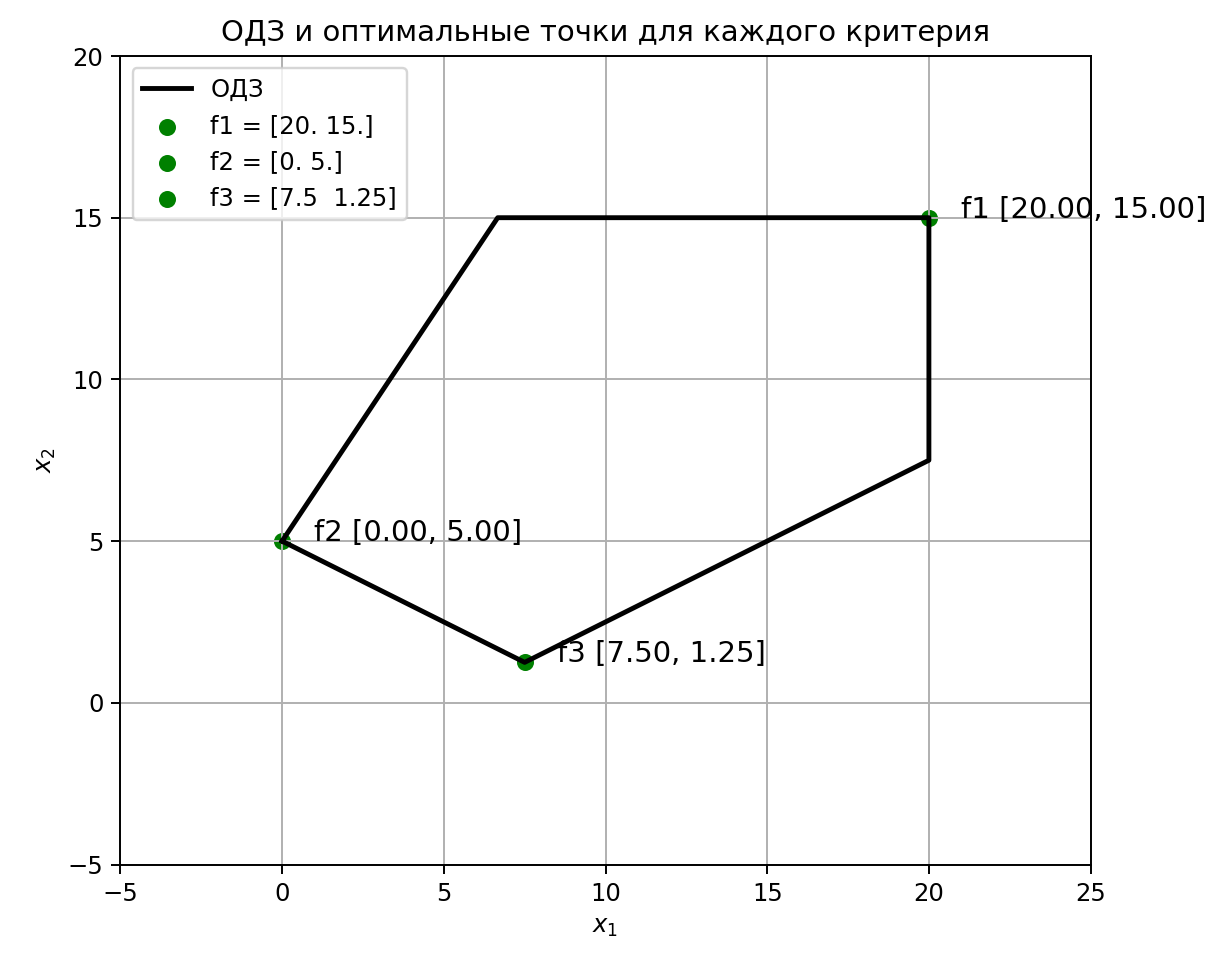


Рисунок 1 - Оптимальные точки

1. Сформируем функцию метрики
2. Решаем задачу

Для решения задачи нелинейного программирования, в целевая функция- нелинейная, а система ограничений - линейная. Целесообразно применять метод Франк-Вульфа.

# **Метод Франк-Вульфа**

1. Задать k = 0 – номер итерации. Задать х0 [5,10] – начальное приближение. Вычислить градиент целевой функции :

Выполним одну итерацию алгоритма

1. Сформируем вспомогательную функцию вида
2. Решить задачу линейного программирования

Оптимальное решение достигается в точке

1. Нати приближение к решению задачи

где определяется из решения задачи

Целевая функция в задаче зависит только от параметра и все параметры задачи известны и зафиксированы. Подставляя числовые значения

Тогда

Где

Реализация итераций алгоритма представлена на Рисунке 2

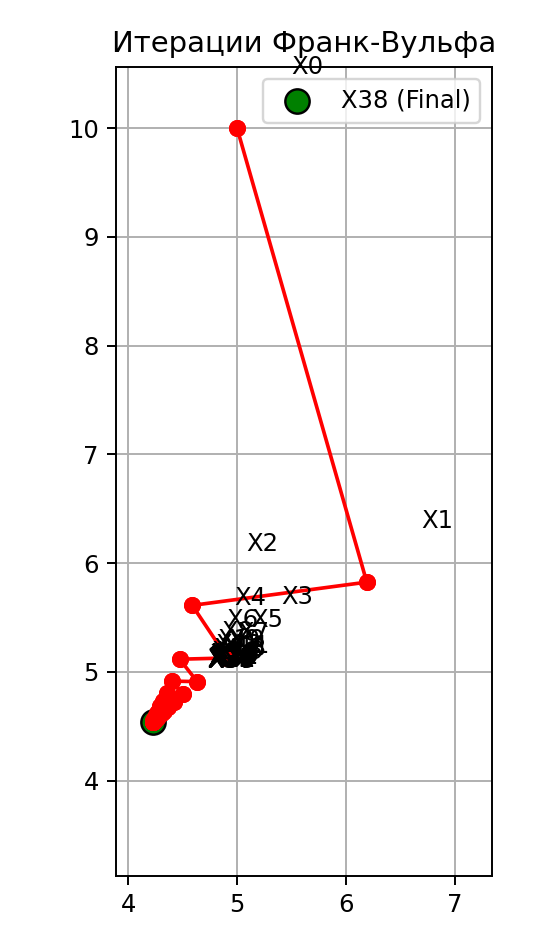


Рисунок 2 - Итерации алгоритма Франк-Вульфа

Результат итераций получен на шаге 38.

# **Приложение 1**

В приложении 1 представлен листинг кода программы

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.optimize import linprog, minimize\_scalar  
from shapely.geometry import Polygon  
from scipy.linalg import solve  
from itertools import combinations  
  
# Вариант 5  
n = 5  
  
# Матрица ограничений (левая часть Ax ≤ b)  
A = np.array([  
 [-1, -2], # -x1 - 2x2 ≤ -2n (x1 + 2x2 ≥ 2n)  
 [1, -2], # x1 - 2x2 ≤ n  
 [-3, 2], # -3x1 + 2x2 ≤ 2n  
 [-1, 0], # -x1 ≤ 0 (x1 ≥ 0)  
 [1, 0], # x1 ≤ 4n  
 [0, -1], # -x2 ≤ 0 (x2 ≥ 0)  
 [0, 1] # x2 ≤ 3n  
])  
b = np.array([  
 -2\*n,  
 n,  
 2\*n,  
 0,  
 4\*n,  
 0,  
 3\*n  
])  
  
# Критерии  
def f1(x): return x[0] + x[1]  
def f2(x): return -3\*x[0] + x[1]  
def f3(x): return x[0] - 3\*x[1]  
  
# Линейная максимизация одного критерия  
def maximize\_criterion(c):  
 res = linprog(-np.array(c), A\_ub=A, b\_ub=b, method='highs')  
 return res.x, -res.fun  
  
# Находим оптимумы по каждому критерию  
opt1, f1\_max = maximize\_criterion([1,1])  
opt2, f2\_max = maximize\_criterion([-3,1])  
opt3, f3\_max = maximize\_criterion([1,-3])  
print(f"Оптимальная точка для f1: {opt1} с макс. {f1\_max:.2f}")  
print(f"Оптимальная точка для f2: {opt2} с макс. {f2\_max:.2f}")  
print(f"Оптимальная точка для f3: {opt3} с макс. {f3\_max:.2f}")  
  
# Строим вершины ОДЗ  
vertices = []  
for i,j in combinations(range(len(A)),2):  
 M = np.array([A[i], A[j]])  
 rhs = np.array([b[i], b[j]])  
 if np.linalg.matrix\_rank(M)==2:  
 try:  
 pnt = solve(M,rhs)  
 if all(np.dot(A,pnt) <= b + 1e-6):  
 vertices.append(tuple(pnt))  
 except:  
 pass  
vertices = list(set(vertices))  
center = np.mean(vertices,axis=0)  
vertices.sort(key=lambda p: np.arctan2(p[1]-center[1], p[0]-center[0]))  
  
# Метрика ρ и её градиент (динамически по f\*\_max)  
def rho(x):  
 return (f1(x) - f1\_max)\*\*2 + (f2(x) - f2\_max)\*\*2 + (f3(x) - f3\_max)\*\*2  
  
def grad\_rho(x):  
 # градиенты критериев f1,f2,f3  
 g1 = np.array([1, 1])  
 g2 = np.array([-3, 1])  
 g3 = np.array([1, -3])  
 # производная суммы квадратов  
 return 2\*(f1(x)-f1\_max)\*g1 + 2\*(f2(x)-f2\_max)\*g2 + 2\*(f3(x)-f3\_max)\*g3  
  
# Проверка ОДЗ  
def in\_D(x):  
 return all(np.dot(A,x) <= b)  
  
# Франк–Вульфа: линейная подзадача по ρ + line-search по ρ  
def frank\_wolfe\_algorithm(x0, max\_iter=200, tol=10\*\*-3, eps=1e-4):  
 x\_old = x0.copy()  
 history = [x\_old]  
 for k in range(1, max\_iter+1):  
 # градиент метрики  
 g = grad\_rho(x\_old)  
 # линейная подзадача: min g^T x  
 res = linprog(g, A\_ub=A, b\_ub=b, bounds=[(0,4\*n),(0,3\*n)], method='highs')  
 x\_bar = res.x  
 # одномерный поиск λ ∈ [0,1]  
 d = x\_bar - x\_old  
 phi = lambda lam: rho(x\_old + lam\*d)  
 lam = minimize\_scalar(phi, bounds=(0,1), method='bounded').x  
 x\_new = x\_old + lam\*d  
  
 # проверки  
 if not in\_D(x\_new):  
 print(f"Завершено на итерации {k} (выход за ОДЗ)")  
 break  
 if np.linalg.norm(x\_new-x\_old) < tol:  
 print(f"Завершено на итерации {k} (шаг < tol={tol})")  
 break  
 if abs(rho(x\_new)-rho(x\_old)) < eps:  
 print(f"Завершено на итерации {k} (Δρ < eps={eps})")  
 break  
  
 print(f"Итерация {k}: x\_bar={x\_bar}, λ={lam:.4f}, x\_new={x\_new}")  
 history.append(x\_new)  
 x\_old = x\_new  
 else:  
 print(f"Достигнуто max\_iter={max\_iter}")  
 return x\_old, np.array(history)  
  
# Запуск алгоритма и визуализация  
x0 = np.array([n, 2\*n])  
opt\_x, hist = frank\_wolfe\_algorithm(x0)  
print("Оптимальная точка (FW):", opt\_x)  
  
# График 1: ОДЗ  
plt.figure(figsize=(6,6))  
poly = Polygon(vertices); xv,yv = poly.exterior.xy  
plt.plot(xv,yv,'k-',linewidth=2)  
plt.xlim(-1,4\*n+1); plt.ylim(-1,3\*n+1)  
plt.gca().set\_aspect('equal'); plt.title("ОДЗ"); plt.grid(True)  
plt.show()  
  
# График 2: ОДЗ + опт. точки  
plt.figure(figsize=(6,6))  
plt.plot(xv,yv,'k-')  
for name,pt in zip(['f1','f2','f3'],[opt1,opt2,opt3]):  
 plt.scatter(\*pt,color='green')  
 plt.text(pt[0]+0.5,pt[1]+0.5,f"{name}[{pt[0]:.1f},{pt[1]:.1f}]")  
plt.xlim(-1,4\*n+1); plt.ylim(-1,3\*n+1)  
plt.gca().set\_aspect('equal'); plt.title("ОДЗ и оптимальные точки"); plt.grid(True)  
plt.show()  
  
# График 3: ОДЗ + путь FW  
plt.figure(figsize=(6,6))  
plt.plot(xv,yv,'k-')  
xs, ys = hist[:,0], hist[:,1]  
plt.plot(xs, ys,'o-',color='red') # Все точки, кроме финальной  
  
# Отображаем все точки, полученные на каждой итерации  
for i, (xx, yy) in enumerate(hist):  
 if i == len(hist) - 1: # Финальная точка  
 plt.scatter(xx, yy, color='green', s=100, label=f'X{i} (Final)', edgecolors='black') # Увеличиваем размер для финальной точки  
 else:  
 plt.scatter(xx, yy, color='red') # Все остальные точки  
  
 # Подписываем каждую точку  
 plt.text(xx + 0.5, yy + 0.5, f"X{i}")  
  
plt.xlim(-1, 4 \* n + 1); plt.ylim(-1, 3 \* n + 1)  
plt.gca().set\_aspect('equal')  
plt.title("Итерации Франк-Вульфа")  
plt.grid(True)  
plt.legend()  
plt.show()