

**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**

**В РАМКАХ 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](#_Toc193280244)

[**Алгоритм исключения заведомо неэффективных решений** 4](#_Toc193280245)

[**Алгоритм вычисления индекса эффективности** 7](#_Toc193280246)

[**Кластеризация на основе индекса эффективности** 9](#_Toc193280247)

[**Работа программы на заданном объеме** 11](#_Toc193280248)

[**Приложение 1** 14](#_Toc193280249)

# **Цели работы**

Рассматривается N проектов цифровой платформы, каждый из которых оценивается векторным показателем эффективности , где:

*f*1​ – показатель информационной безопасности;

*f*2​ – показатель надежности цифровой платформы.

Требуется:

1. Определить множество эффективных проектов, имеющих максимальные значения *f*1​ и *f*2​.
2. Сгенерировать 200 случайных точек в пространстве (*f*1​, *f*2​), удовлетворяющих заданным ограничениям, где n- номер в списке группы:
3. Использовать алгоритм исключения заведомо неэффективных решений для определения Парето-оптимальных проектов.
4. Выполнить кластеризацию проектов на основе индекса эффективности с центрами *K*1​=1, K2​=0.85, K3​=0.75.

**Цель работы** – разработка программного обеспечения для анализа и кластеризации проектов цифровой платформы.

Задачи:

1. Разработать алгоритм генерации исходных данных.
2. Реализовать алгоритм исключения неэффективных решений.
3. Реализовать алгоритм расчёта индекса эффективности
4. Реализовать алгоритм кластеризации на основе индекса эффективности.
5. Визуализировать результаты анализа.

# **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 20 сгенерированных точек, значения которых отображены в таблице 1.



Таблица 1 - Сгенерированные точки

Шаги алгоритма и выявление множество Парето-оптимальных точек отображено на рисунке 1.

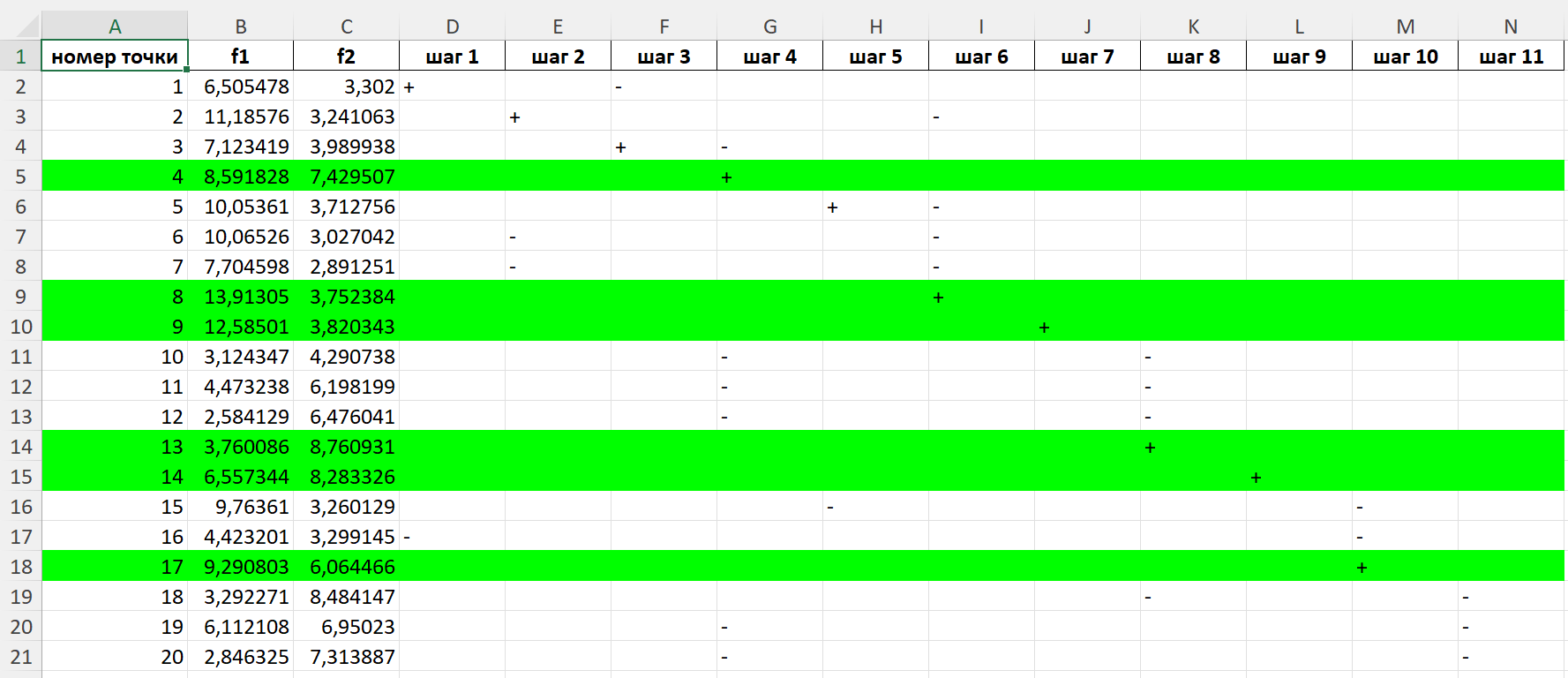


Рисунок 1 - Шаги алгоритма

Работа алгоритма визуализирована на графике, представленном на рисунке 2.

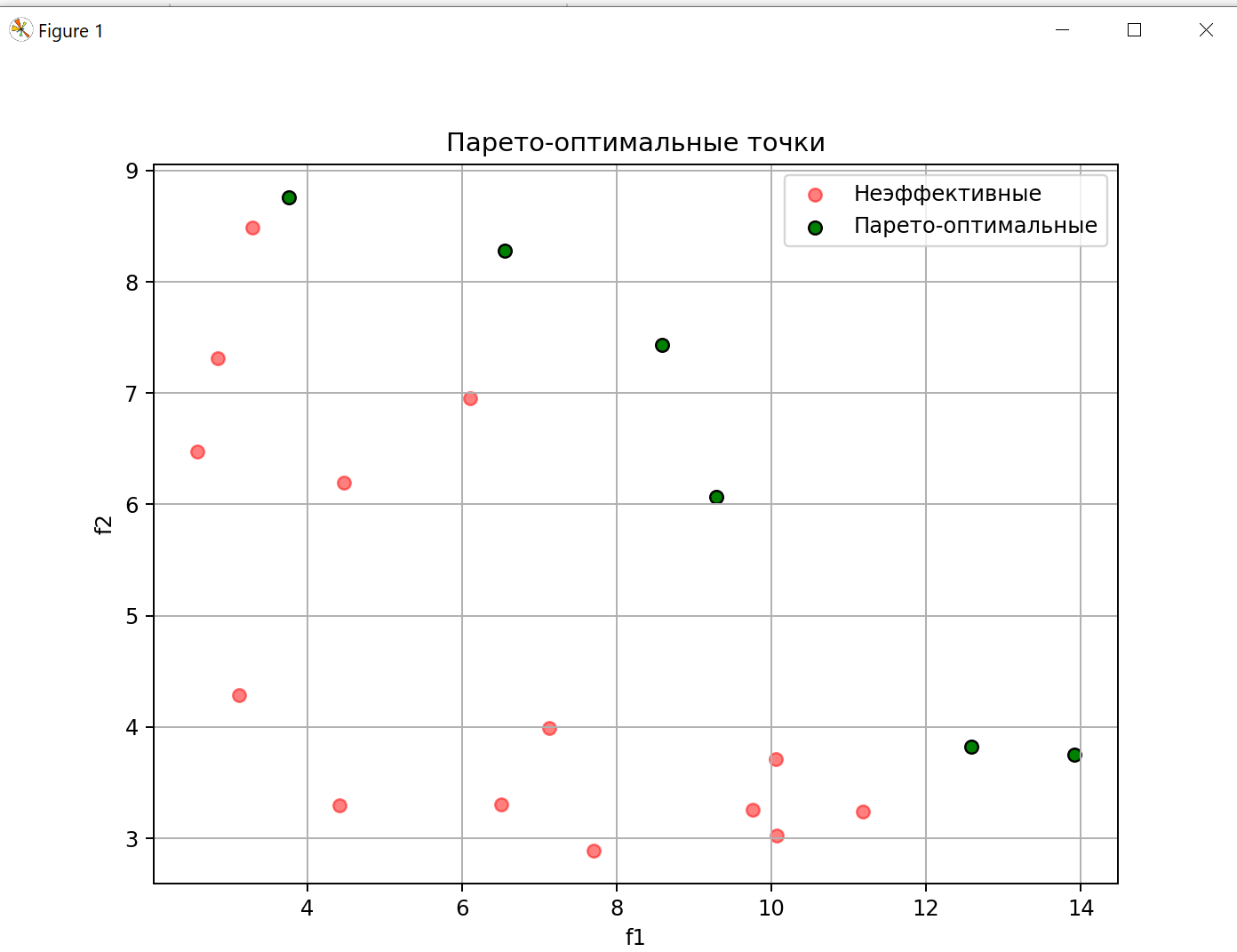


Рисунок 2 - График отображения Парето-оптимальных точек

# **Алгоритм вычисления индекса эффективности**

Индекс эффективности *F* для каждого проекта вычисляется по формуле:

где bi – количество проектов, которые доминируют над текущим проектом.

Рассмотрим те же сгенерированные точки и проведем ранжирование на основе индекса эффективности. Алгоритм состоит из следующих шагов

1. Полагаем .
2. Вычисляем параметр bi – число точек, для которых выполняется условие .
3. Вычисляем значение индекса эффективности в виде
4. Если , то полагаем что и переходим к шагу 2. Иначе, переходим к шагу 5.
5. Из множества точек множества формируем множество парето-оптимальных по правилу . Результат работы отображены в таблице 2.



Таблица 2 - Подсчет индекса эффективности точек

# **Кластеризация на основе индекса эффективности**

В задаче многокритериальной кластеризации будем предполагать, что требуется множество допустимых решений на три кластера по значению индекса эффективности. Центры кластеров задами как К1 = 1,К2 = 0.85, К3 = 0.75.

Для каждого альтернативного решения определяем кластер в соответствии со следующим алгоритмом.

1. В пространстве признаков вычисляем расстояние от xi до центров кластеров
2. Вычисляем минимально расстояние
3. Определяем номер кластера n, к которому принадлежит xi, где

Результаты работы алгоритма представлены в таблице 3, путем раскраски соответствующих строк таблицы в цвета соответствующие кластерам (К1 - зеленый,К2 - желтый, К3 – красный)



Таблица 3 - Кластеризация точек

Визуализация точек на графике отображена на рисунке 3.

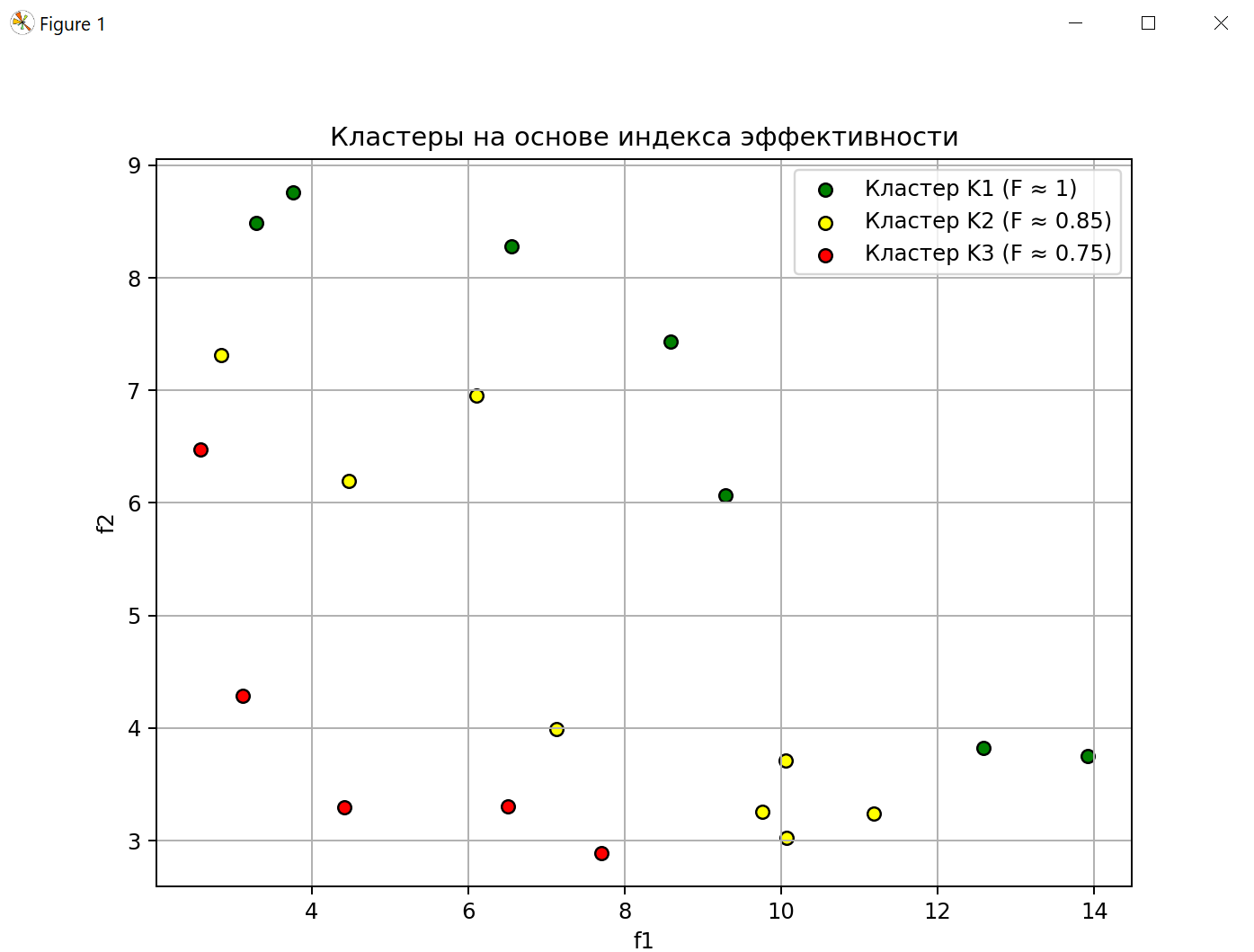


Рисунок 3 - Кластеры на графике

# **Работа программы на заданном объеме**

Работа программы на массиве данных в 200 точек отображена на рисунках 4-6.

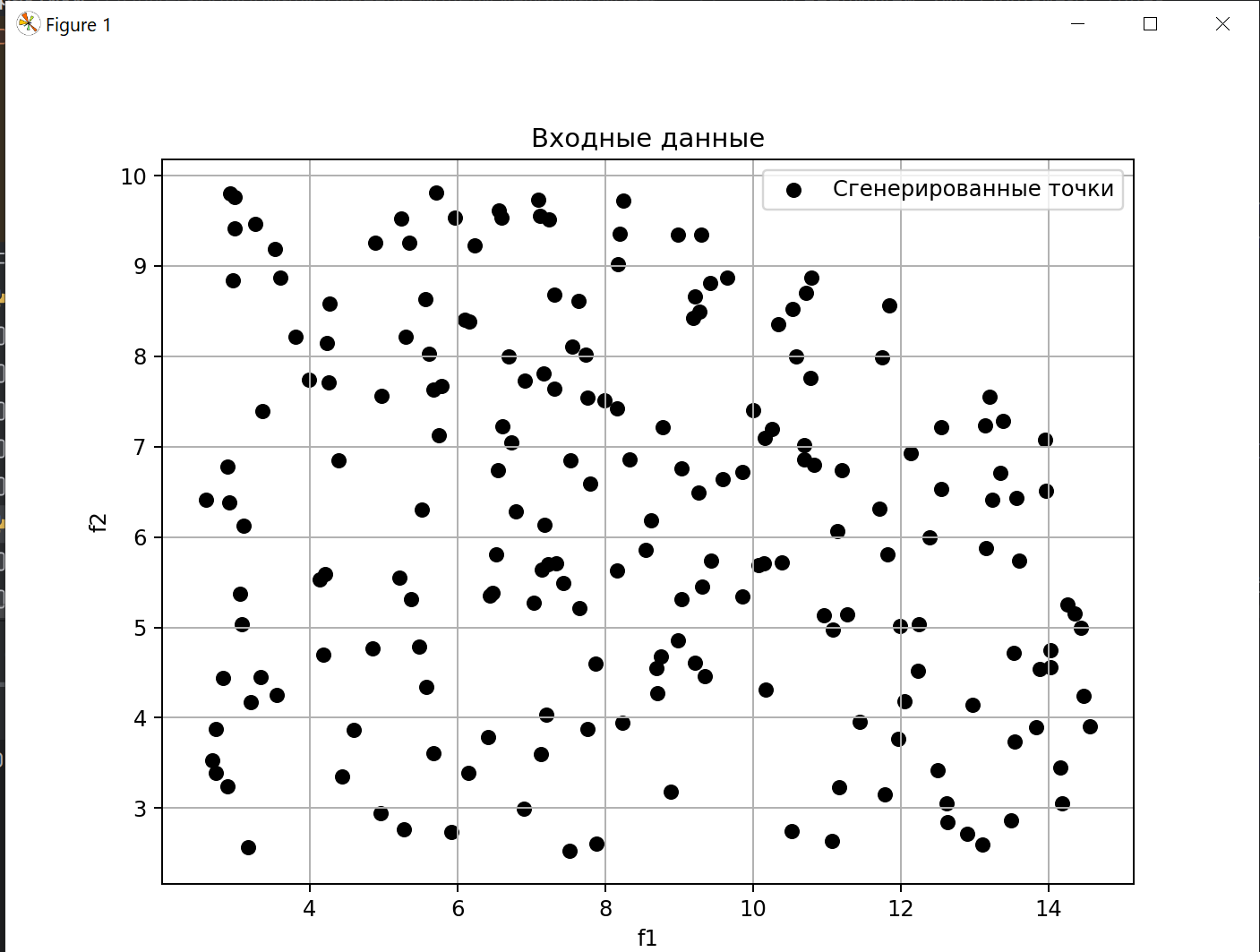


Рисунок 4 - Входные данные

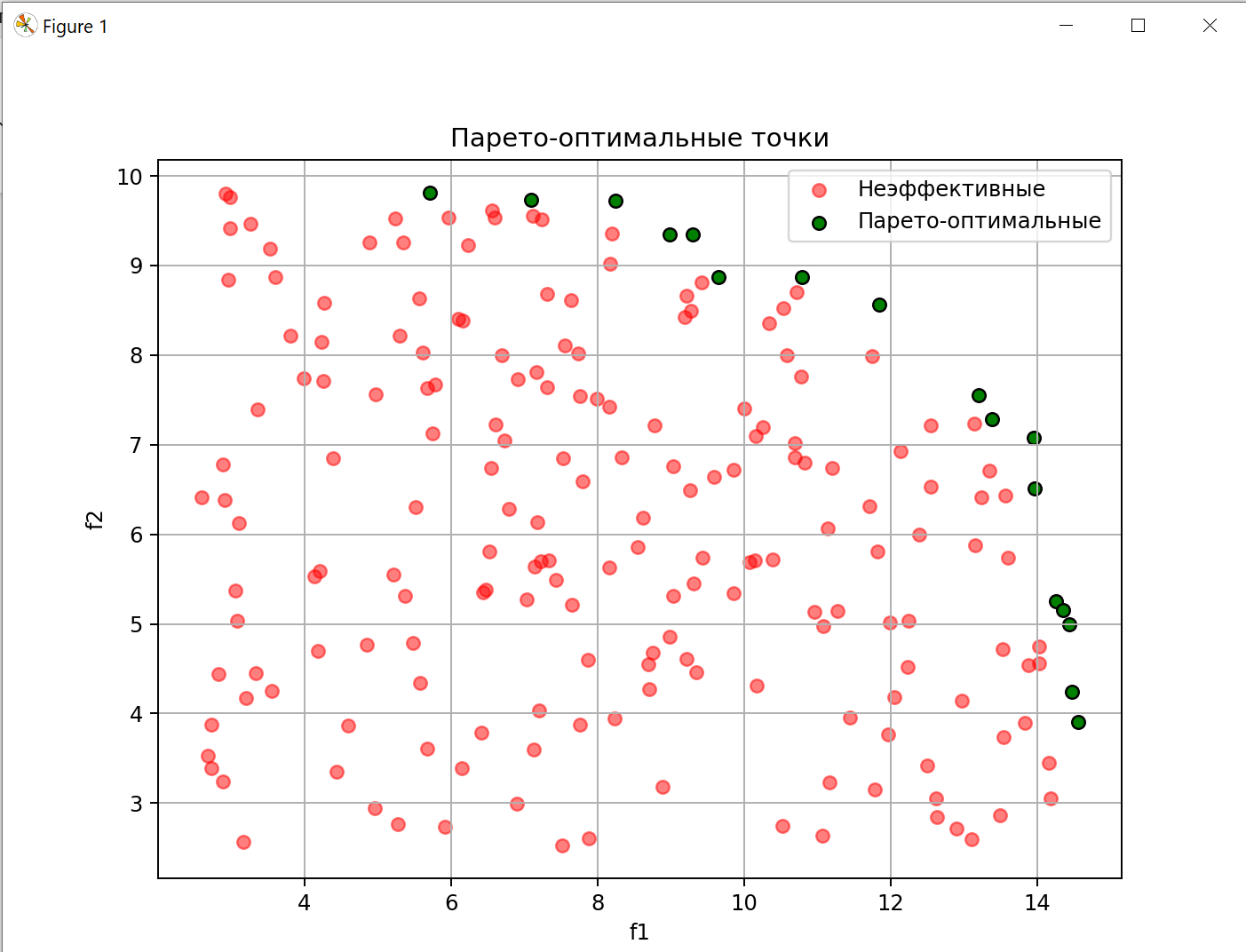


Рисунок 5 - Нахождение парето-оптимального множество точек

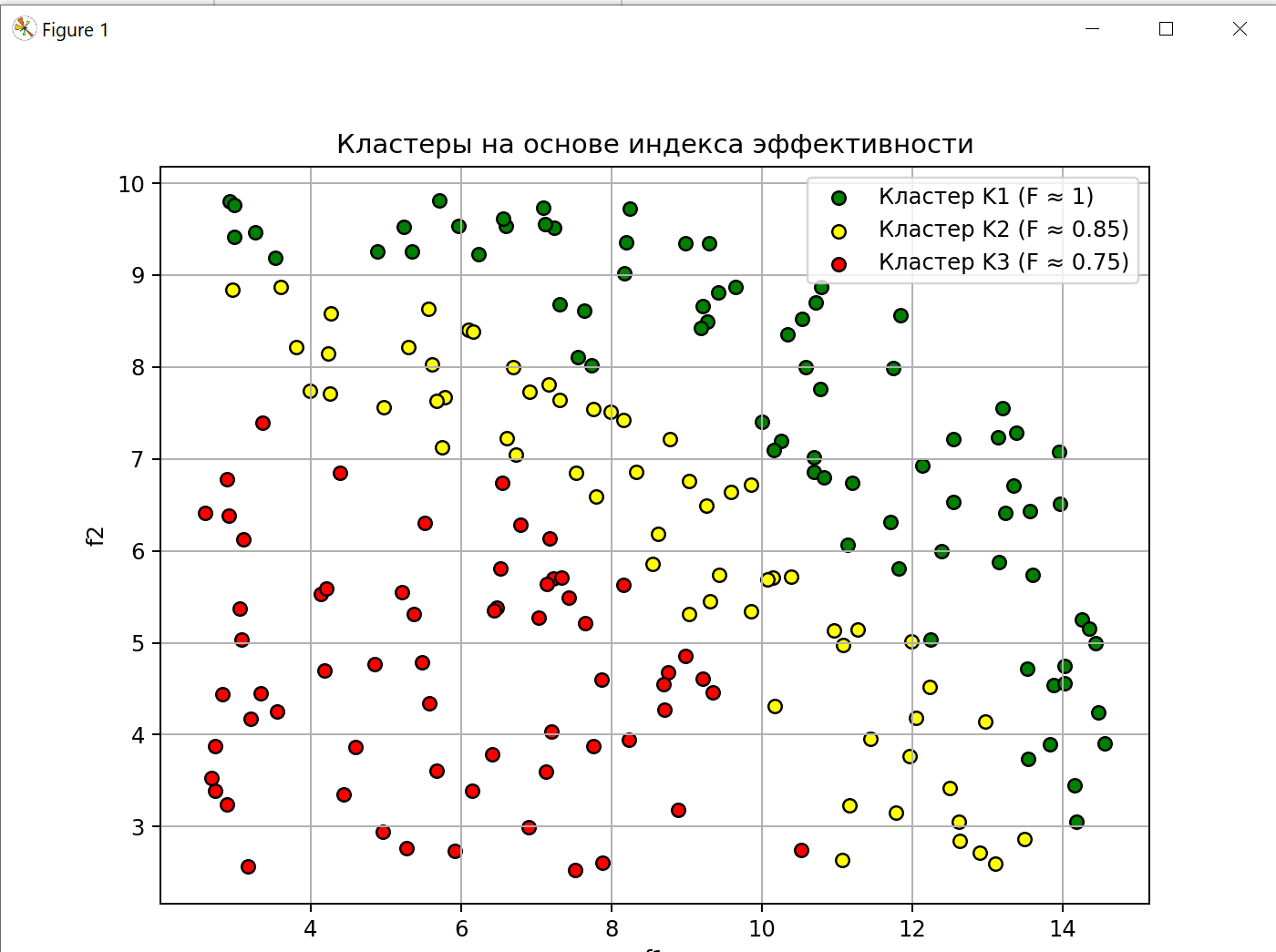


Рисунок 6 - Распределение точек по заданным кластерам

# **Приложение 1**

В приложении 1 представлен листинг кода программы

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import os  
from openpyxl import load\_workbook  
from openpyxl.styles import PatternFill  
  
#Устанавливаем параметр n  
n = 5  
  
#Определяем границы для f1 и f2  
f1\_min, f1\_max = n / 2, 3 \* n  
f2\_min, f2\_max = n / 2, 2 \* n  
  
  
#Функция проверки, лежит ли точка в заданной области  
def is\_inside\_region(f1, f2):  
 term1 = ((f1 - n) \*\* 2) / (4 \* n \*\* 2)  
 term2 = ((f2 - n) \*\* 2) / (n \*\* 2)  
 return (term1 + term2) <= 1  
  
  
#Генерируем 200 случайных точек  
num\_points = 200  
points = []  
statuses = []  
  
while len(points) < num\_points:  
 f1 = np.random.uniform(f1\_min, f1\_max)  
 f2 = np.random.uniform(f2\_min, f2\_max)  
 if is\_inside\_region(f1, f2):  
 points.append((f1, f2))  
 statuses.append(0)  
  
points = np.array(points)  
statuses = np.array(statuses, dtype=object)  
  
#Создаем DataFrame для Excel  
df = pd.DataFrame({  
 "номер точки": np.arange(1, len(points) + 1),  
 "f1": points[:, 0],  
 "f2": points[:, 1]  
})  
  
# ---------------------------------------------  
# Алгоритм исключения заведомо не оптимальных точек  
# ---------------------------------------------  
  
i = 0  
step = 1 # Шаг алгоритма  
while i < len(points):  
 column\_name = f"шаг {step}"  
  
 if column\_name not in df:  
 df[column\_name] = ""  
  
 if statuses[i] == "-":  
 df.at[i, column\_name] = "-" # Записываем статус в таблицу  
 i += 1  
 continue  
  
 statuses[i] = "+"  
 df.at[i, column\_name] = "+"  
  
 selected\_f1, selected\_f2 = points[i]  
  
 for j in range(len(points)):  
 if i == j or statuses[j] == "-":  
 continue  
  
 f1, f2 = points[j]  
  
 if f1 <= selected\_f1 and f2 <= selected\_f2 and (f1 < selected\_f1 or f2 < selected\_f2):  
 statuses[j] = "-"  
 df.at[j, column\_name] = "-"  
  
 i += 1  
 step += 1  
  
#Определение финальных Парето-оптимальных точек  
final\_pareto\_points = set()  
for idx in range(len(points)):  
 row\_statuses = df.iloc[idx, 3:].values # Берем только статусы из шагов  
 last\_plus\_index = None  
  
 for step\_index, status in enumerate(row\_statuses):  
 if status == "+":  
 last\_plus\_index = step\_index  
  
 if last\_plus\_index is not None and all(s != "-" for s in row\_statuses[last\_plus\_index + 1:]):  
 final\_pareto\_points.add(idx)  
  
#Сохранение в Excel  
file\_name = "КДЗ\_1.xlsx"  
  
# Удаляем старый файл перед сохранением нового  
if os.path.exists(file\_name):  
 os.remove(file\_name)  
  
# Создаем Excel-файл и записываем данные  
with pd.ExcelWriter(file\_name, engine="openpyxl") as writer:  
 df.to\_excel(writer, index=False, sheet\_name="Заведомо не оптимальные")  
  
#Раскрашивание финальных Парето-оптимальных точек в зеленый  
wb = load\_workbook(file\_name)  
ws = wb["Заведомо не оптимальные"]  
  
# Определяем зеленую заливку  
green\_fill = PatternFill(start\_color="00FF00", end\_color="00FF00", fill\_type="solid")  
  
# Закрашиваем строки финальных Парето-оптимальных точек  
for idx in final\_pareto\_points:  
 for cell in ws[idx + 2]: # Excel-индексация с 1 (плюс заголовок)  
 cell.fill = green\_fill  
  
# ---------------------------------------------  
# Алгоритм ранжирования на основе индекса эффективности  
# ---------------------------------------------  
  
def calculate\_efficiency\_index(points):  
 N = len(points)  
 efficiency\_indices = []  
  
 for i, (f1\_i, f2\_i) in enumerate(points):  
 bi = 0  
 for j, (f1\_j, f2\_j) in enumerate(points):  
 if f1\_j >= f1\_i and f2\_j >= f2\_i and (f1\_j > f1\_i or f2\_j > f2\_i):  
 bi += 1  
 F = 1 / (1 + (bi / (N - 1)))  
 efficiency\_indices.append(F)  
  
 return np.array(efficiency\_indices)  
  
  
# Вычисляем индексы эффективности  
efficiency\_indices = calculate\_efficiency\_index(points)  
  
# Кластеризация по индексам эффективности  
clusters = np.zeros(len(points), dtype=int)  
k1, k2, k3 = 1.0, 0.85, 0.75  
  
for i, F in enumerate(efficiency\_indices):  
 if abs(F - k1) < abs(F - k2) and abs(F - k1) < abs(F - k3):  
 clusters[i] = 1 # Зеленый  
 elif abs(F - k2) < abs(F - k1) and abs(F - k2) < abs(F - k3):  
 clusters[i] = 2 # Желтый  
 else:  
 clusters[i] = 3 # Красный  
  
#Раскрашивание точек по кластерам  
colors = {1: "00FF00", 2: "FFFF00", 3: "FF0000"}  
ws\_eff = wb.create\_sheet("Индекс эффективности")  
ws\_eff.append(["Номер точки", "f1", "f2", "F"])  
for i, (f1, f2, F) in enumerate(zip(points[:, 0], points[:, 1], efficiency\_indices)):  
 ws\_eff.append([i + 1, f1, f2, F])  
 fill = PatternFill(start\_color=colors[clusters[i]], end\_color=colors[clusters[i]], fill\_type="solid")  
 for cell in ws\_eff[i + 2]:  
 cell.fill = fill  
wb.save(file\_name)  
  
# ---------------------------------------------  
# Отображение графиков  
# ---------------------------------------------  
# Отображение всех точек  
plt.figure(figsize=(8, 6))  
inefficient = points[statuses == "-"]  
plt.scatter(inefficient[:, 0], inefficient[:, 1], color="black")  
optimal = points[statuses == "+"]  
plt.scatter(optimal[:, 0], optimal[:, 1],color="black",label="Сгенерированные точки")  
plt.xlabel("f1")  
plt.ylabel("f2")  
plt.title("Входные данные")  
plt.legend()  
plt.grid()  
plt.show()  
  
#Отображение графика Парето-оптимальных точек  
plt.figure(figsize=(8, 6))  
inefficient = points[statuses == "-"]  
plt.scatter(inefficient[:, 0], inefficient[:, 1], color="red", alpha=0.5, label="Неэффективные")  
optimal = points[statuses == "+"]  
plt.scatter(optimal[:, 0], optimal[:, 1], color="green", edgecolor="black", label="Парето-оптимальные")  
plt.xlabel("f1")  
plt.ylabel("f2")  
plt.title("Парето-оптимальные точки")  
plt.legend()  
plt.grid()  
plt.show()  
  
# Отображение кластеров на графике  
plt.figure(figsize=(8, 6))  
colors = {1: "green", 2: "yellow", 3: "red"}  
labels = {1: "Кластер K1 (F ≈ 1)", 2: "Кластер K2 (F ≈ 0.85)", 3: "Кластер K3 (F ≈ 0.75)"}  
  
for cluster\_id in [1, 2, 3]:  
 cluster\_points = points[clusters == cluster\_id]  
 plt.scatter(cluster\_points[:, 0], cluster\_points[:, 1], color=colors[cluster\_id], label=labels[cluster\_id],  
 edgecolor="black")  
  
plt.xlabel("f1")  
plt.ylabel("f2")  
plt.title("Кластеры на основе индекса эффективности")  
plt.legend()  
plt.grid()  
plt.show()