КАФЕДРА

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Отчет о лабораторной работе №8

Эволюционные алгоритмы оценки стоимости проектов в программной инженерии

По дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № _____ _____ _____

подпись, дата | инициалы, фамилия

Санкт-Петербург 2024

Цель работы:
разработка эволюционного алгоритма оценки стоимости программных проектов. Графическое отображение результатов.

Вариант:
№ варианта – 4
Тип эволюционного алгоритма - ГА
Кодирование решения – Веществ. Вектор
Фитнесс-функция (тип ошибки) – ED
Оператор кроссовера - арифметич.
Оператор мутации - арифметич.
Оператор репродукции – турнир

Задание:
1. Разобраться в теоретическом описании математического метода оценки стоимости программного проекта – модели COCOMO.

2. Из приведенной выше табл. 8.1 (или табл. 8.2) экспериментальных данных (программных проектов НАСА) отобрать из 18 проектов в качестве обучающего множества 13 (40) проектов.

3. В соответствии с вариантом лабораторной работы, заданного табл. 8.3 определить тип используемого эволюционного алгоритма (генетический или роевой алгоритм, генетическое программирование), кодирование потенциального решения, вид ошибки в целевой функции, вид генетических операторов кроссовера, мутации и репродукции

4. Отработать алгоритм решения задачи с помощью заданного метода на обучающем множестве.

5. Разработать программу на языке Python, включающую в себя реализацию пользовательского интерфейса в виде диалогового меню, реализацию алгоритма решения поставленной задачи заданным методом.

6. Протестировать разработанную программу: вычислить заданный тип ошибки на тестовом множестве – оставшихся 5 (из 18) проектов табл. 8.1 (или табл. 8.2).

5. Выполнить вывод полученного решения в виде текста и графиков.

Выполнение:
Ядром модели является следующая формула Ef=aL^b, где L – длина кода ПО в килостроках; Ef – оценка сложности проекта в человеко-месяцах; a и b – коэффициенты (параметры) модели, которые для различных типов ПО имеют различные значения.

Экспериментальные данные проектов НАСА

Номер проекта	L	Me	Ef	Efm	Efm2
1	90,2000	30,0000	115,8000	124,8585	134,0202
2	46,2000	20,0000	96,0000	74,8467	84,1616
3	46,5000	19,0000	79,0000	75,4852	85,0112
4	54,5000	20,0000	909,8000	85,4349	94,9828
5	31,1000	35,0000	39,6000	50,5815	56,6580
6	67,5000	29,000	98,4000	99,0504	107,2609
7	12,8000	26,000	18,9000	24,1480	32,6461
8	10,5000	34,0000	10,3000	18,0105	25,0755
9	21,5000	31,0000	28,5000	37,2724	44,3086
10	3,1000	26,000	7,0000	4,5849	14,4563
11	4,2000	19,0000	9,0000	8,9384	19,9759
12	7,8000	31,0000	7,3000	13,5926	21,5763
13	2,1000	28,0000	5,0000	1,5100	11,2703
14	5,0000	29,0000	8,4000	8,2544	17,0887
15	78,6000	35,0000	98,7000	110,5249	118,0378
16	9,7000	27,0000	15,6000	18,2559	26,8312
17	12,5000	27,0000	23,9000	23,3690	31,6864
18	100,8000	34,0000	138,3000	135,4825	144,4587

13(40) первых - обучающее множество, 5(20) последних – тестовое множество

Отработать алгоритм решения задачи с помощью заданного метода на обучающем множестве и применить на тестовом

Тип эволюционного алгоритма - ГА
Кодирование решения – Веществ. Вектор
Фитнесс-функция (тип ошибки) – ED
Оператор кроссовера - арифметич.
Оператор мутации - арифметич.
Оператор репродукции – рулетка

Выполнить вывод полученного решения в виде текста и графиков

Графики:
1. с фактическими и предсказанными значениями для 13(40) первых – обучающих множеств
2. с фактическими и предсказанными значениями для 5(20) последних – тестовых множеств

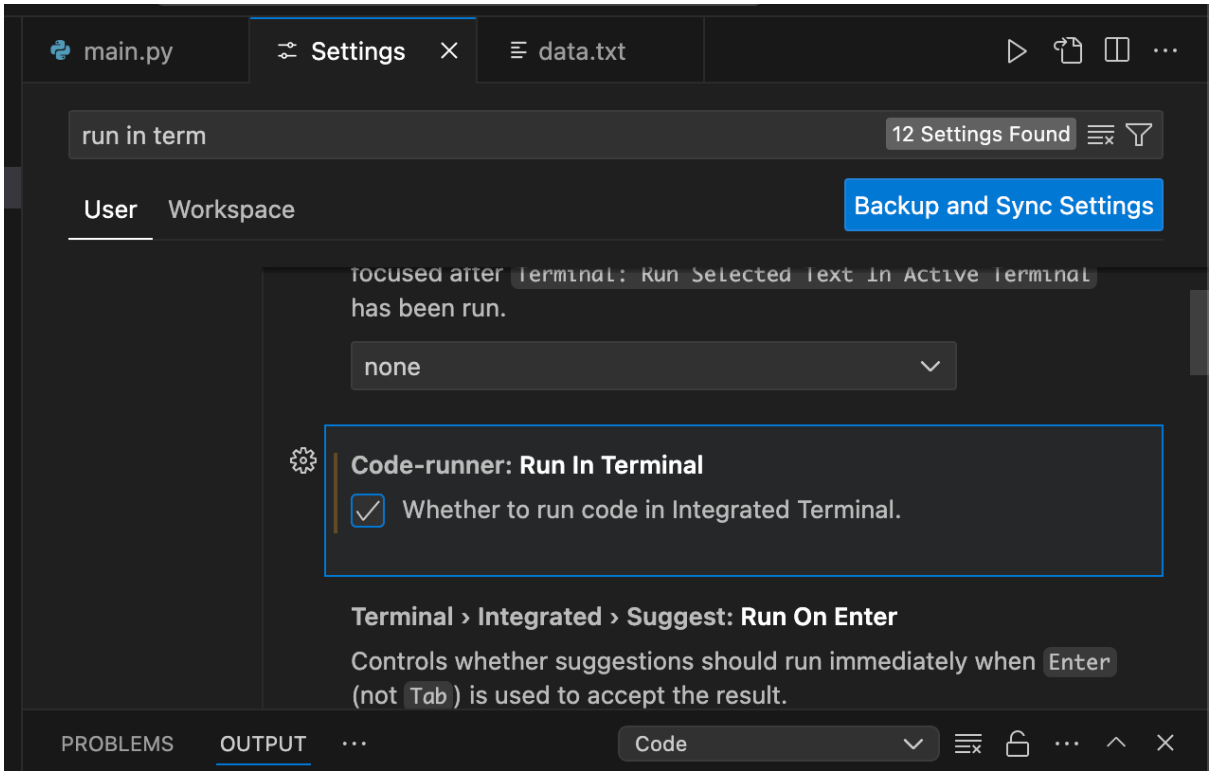
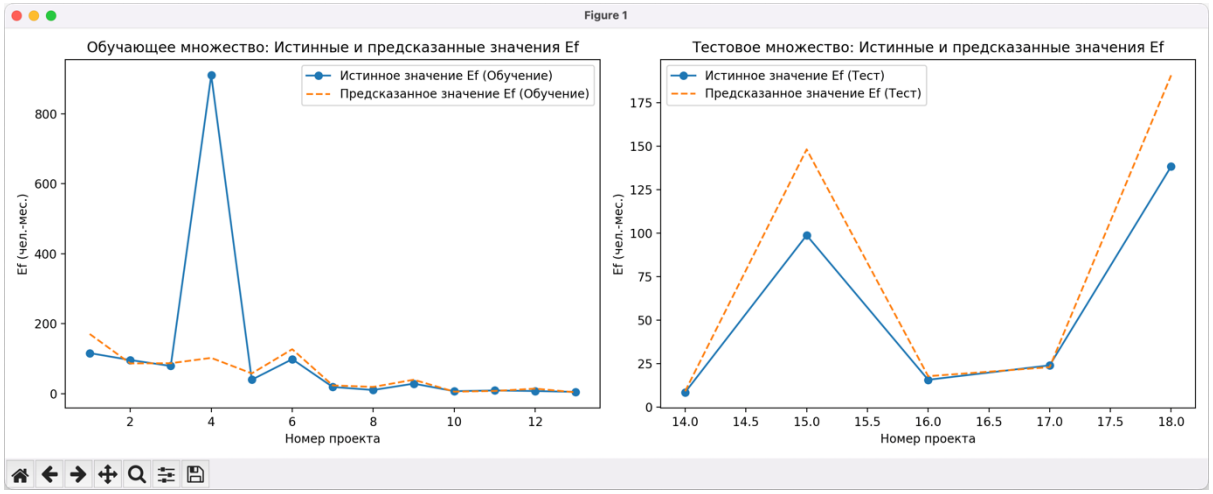
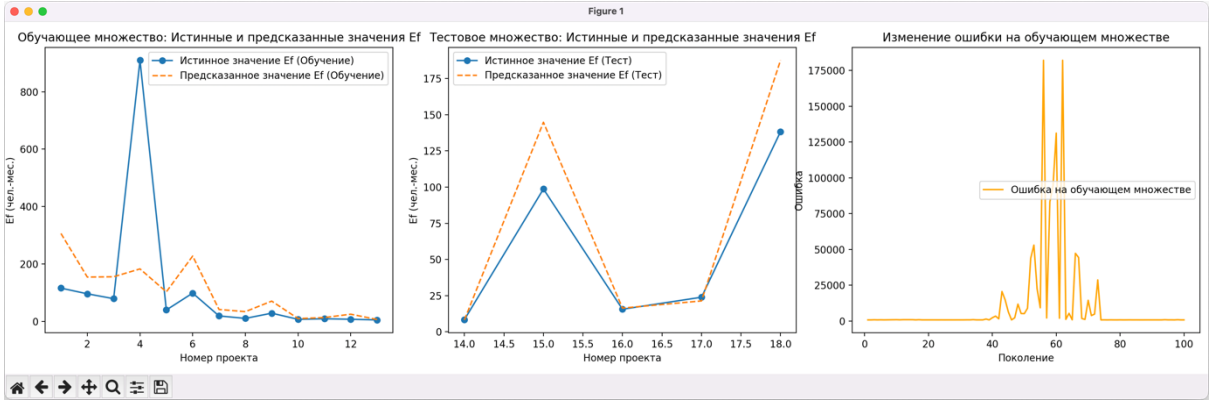
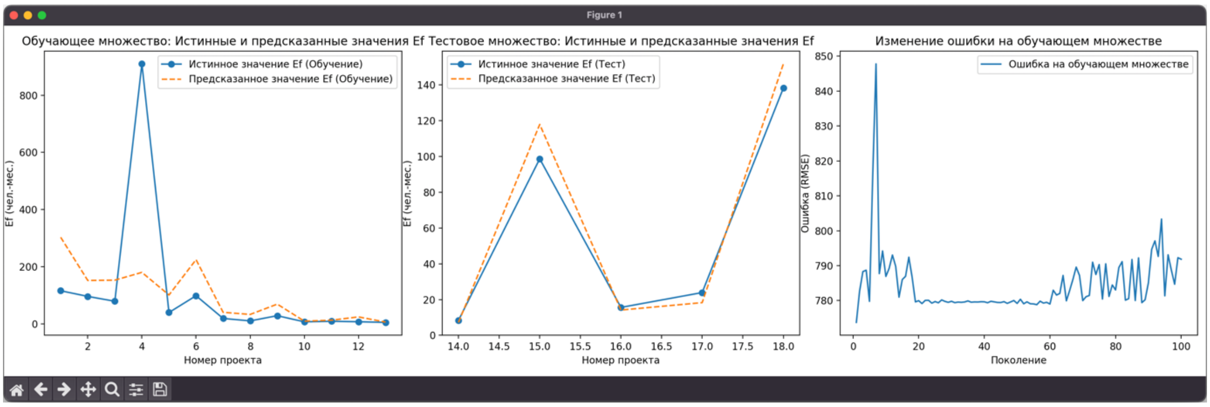


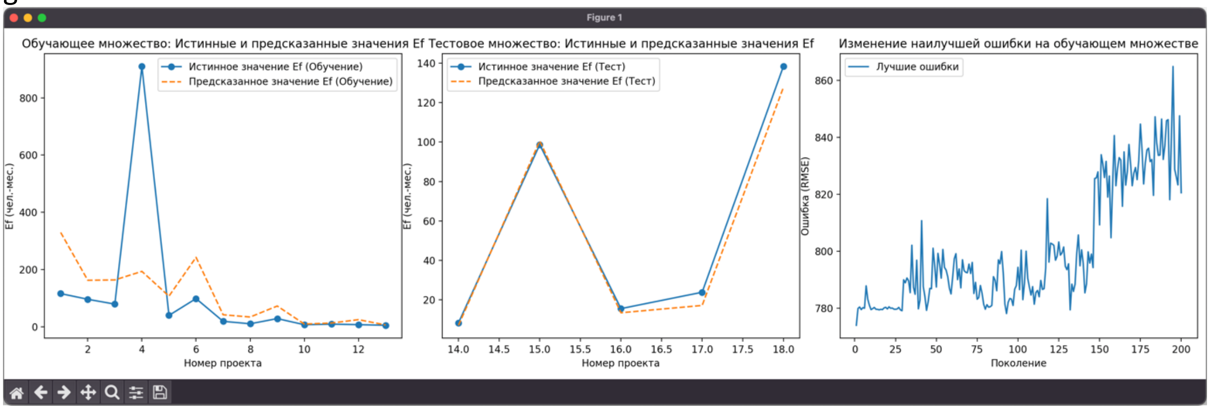
Рисунок 1 – настройка для ввода в VSC

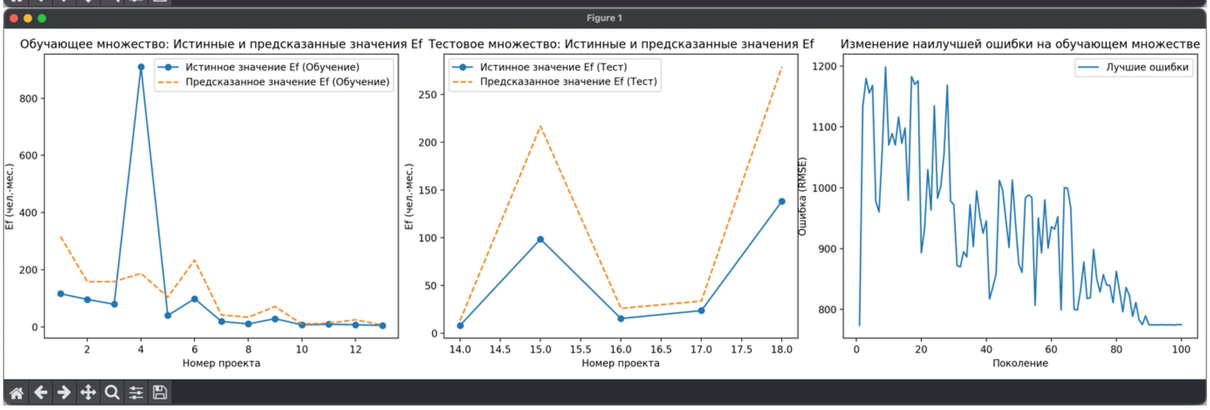


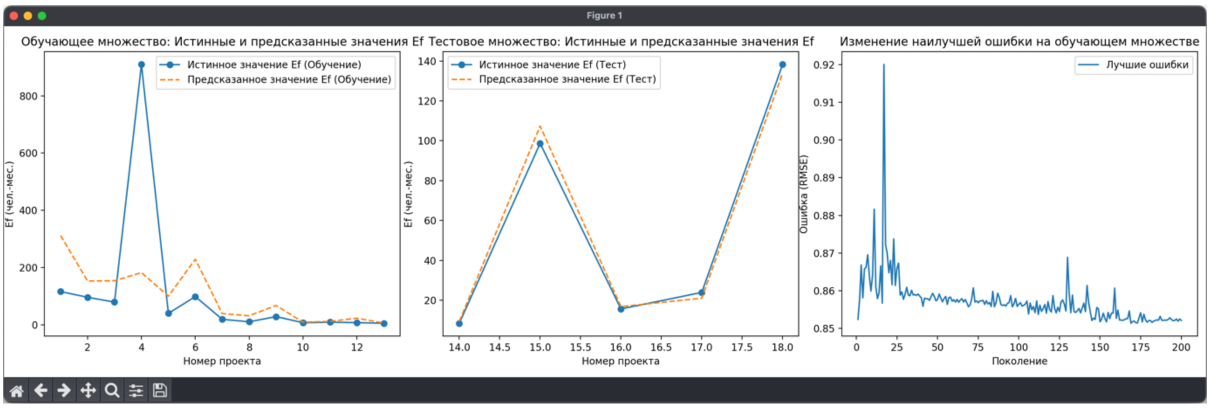




Со значениями:
population_size = 200
generations = 200





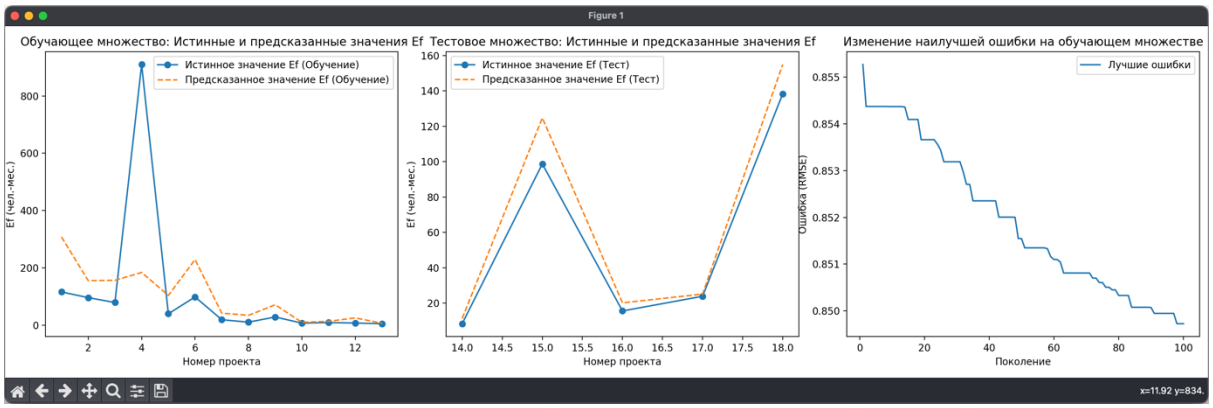


# Параметры алгоритма
population_size = 200

generations = 200
mutation_rate = 0.1

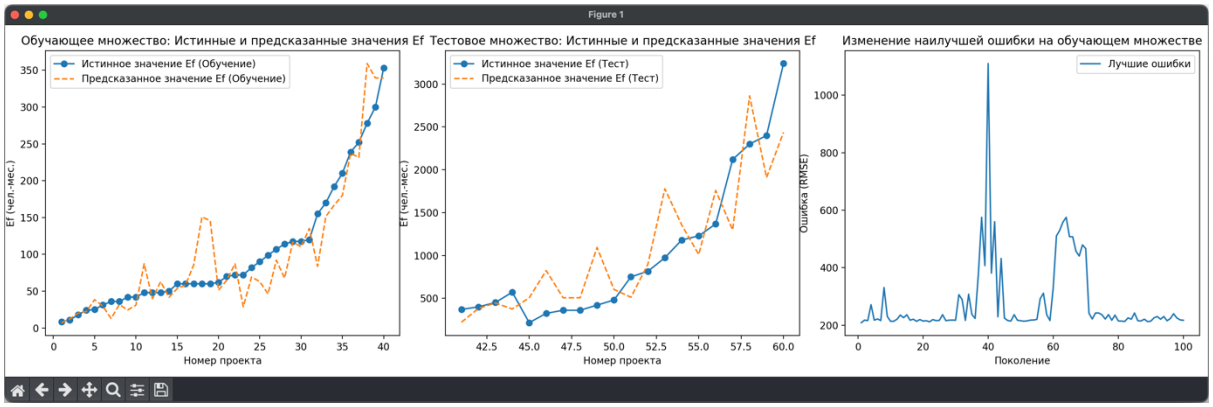
Лучший результат для обучающего множества: a = 2.5604, b = 1.0667, вероятность ошибки = 0.8514

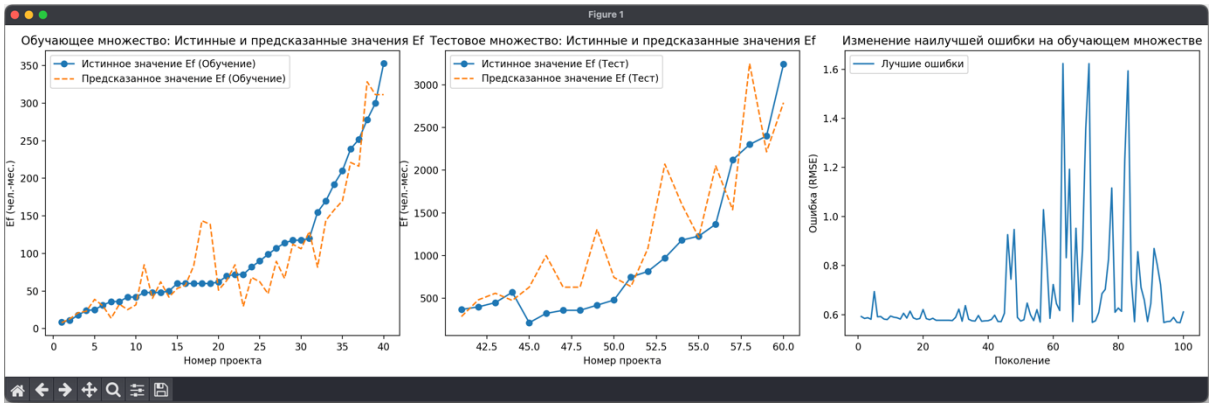
Лучший результат для тестового множества: a = 2.2331, b = 0.8874, вероятность ошибки = 0.0742



# Параметры алгоритма
population_size = 100
generations = 100
mutation_rate = 0.9

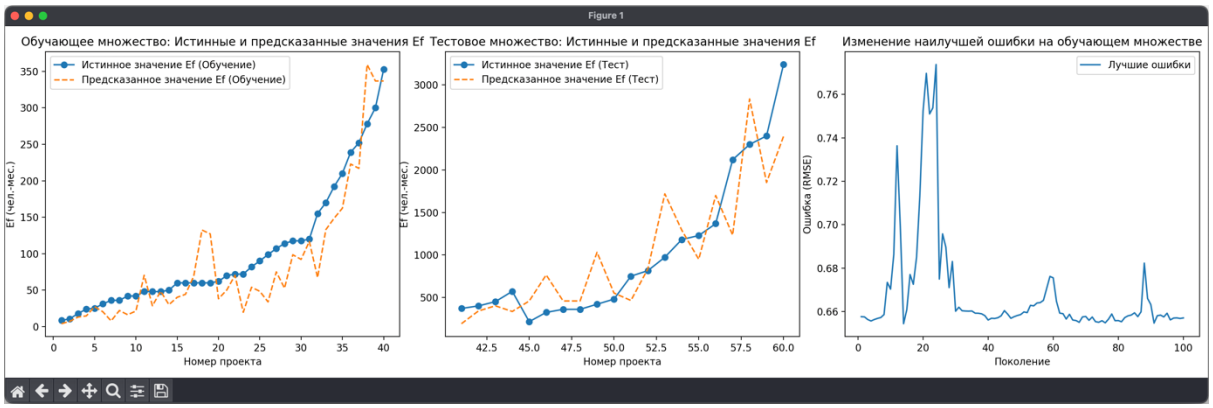
Для расширенного значения:





Лучший результат для обучающего множества: a = 3.3448, b = 1.0798, вероятность ошибки = 0.5671

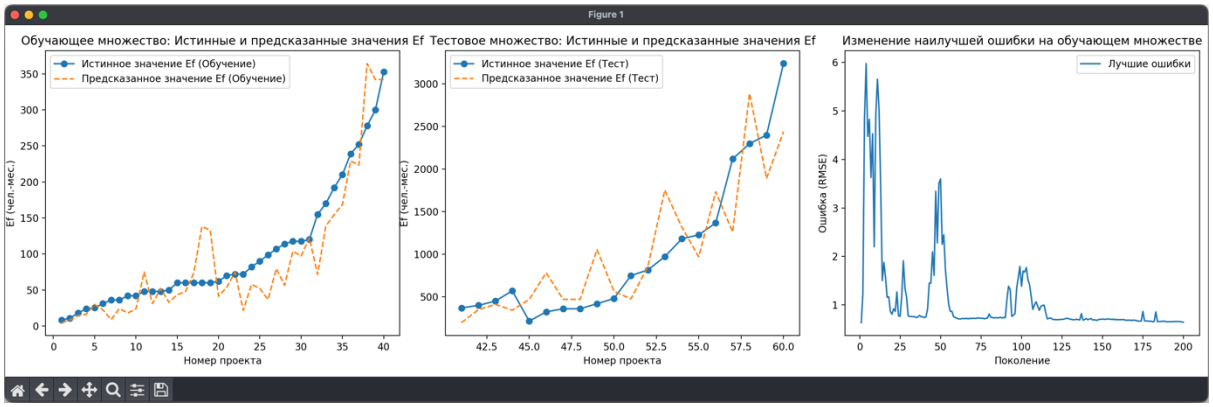
Лучший результат для тестового множества: a = 3.3646, b = 1.1364, вероятность ошибки = 0.6955



# Параметры алгоритма
population_size = 200
generations = 100
mutation_rate = 0.9

Лучший результат для обучающего множества: a = 1.4276, b = 1.3012, вероятность ошибки = 0.6543

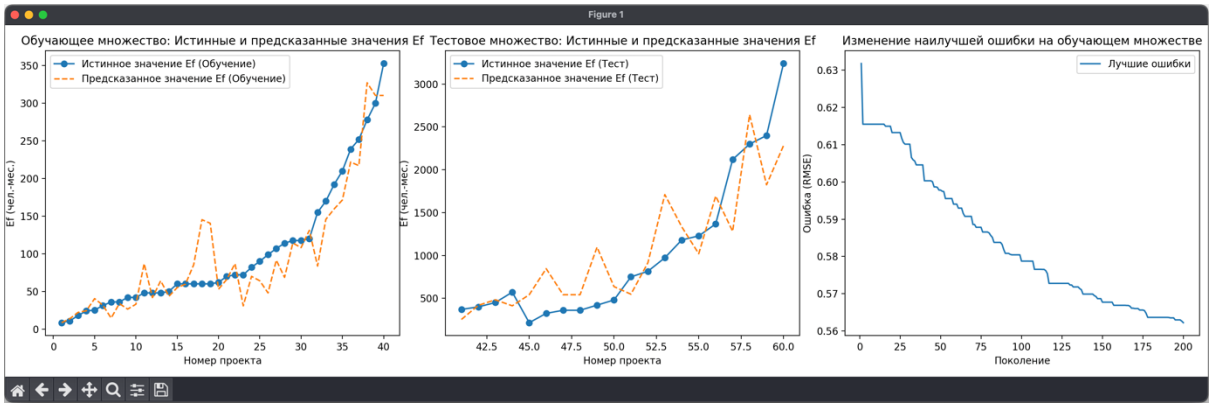
Лучший результат для тестового множества: a = 1.3604, b = 1.2637, вероятность ошибки = 0.5914



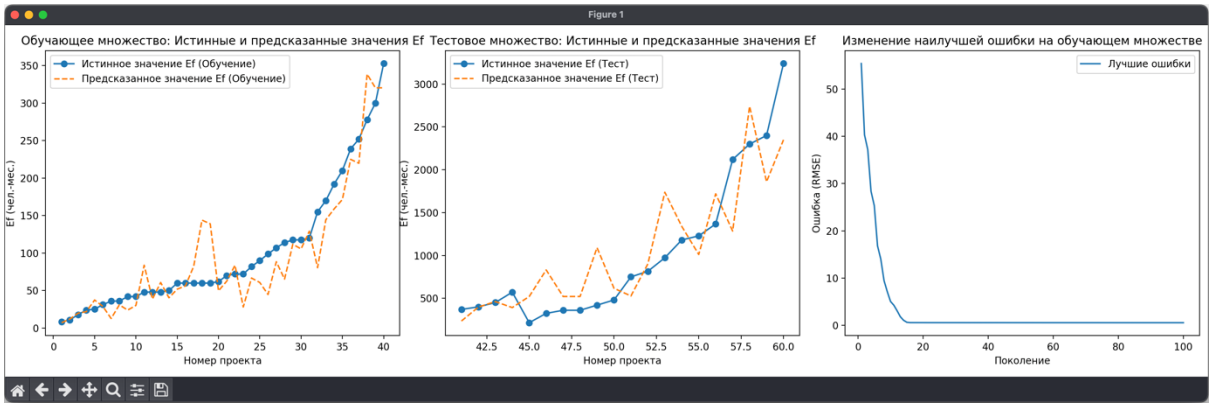
# Параметры алгоритма
population_size = 200
generations = 200
mutation_rate = 0.9

Лучший результат для обучающего множества: a = 1.6999, b = 1.2634, вероятность ошибки = 0.6367

Лучший результат для тестового множества: a = 1.4277, b = 1.2588, вероятность ошибки = 0.5926



population_size = 500
generations = 200
mutation_rate = 0.5



Лучший результат для обучающего множества: a = 2.9690, b = 1.1149, вероятность ошибки = 0.5748

Лучший результат для тестового множества: a = 2.5969, b = 1.1512, вероятность ошибки = 0.5966

population_size = 300
generations = 100
mutation_rate = 0.9

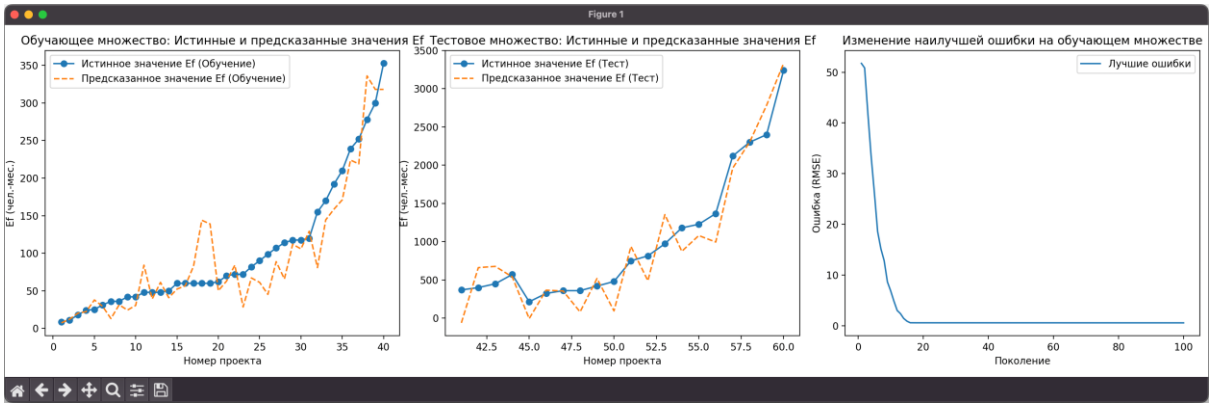


Figure 1
Обучающее множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Обучение)
Предсказанное значение Ef (Обучение)
Ef (чел.-мес.)
Номер проекта
Тестовое множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Тест)
Предсказанное значение Ef (Тест)
Ef (чел.-мес.)
Номер проекта
Изменение наилучшей ошибки на обучающем множестве
Лучшие ошибки
Ошибка (RMSE)
Поколение

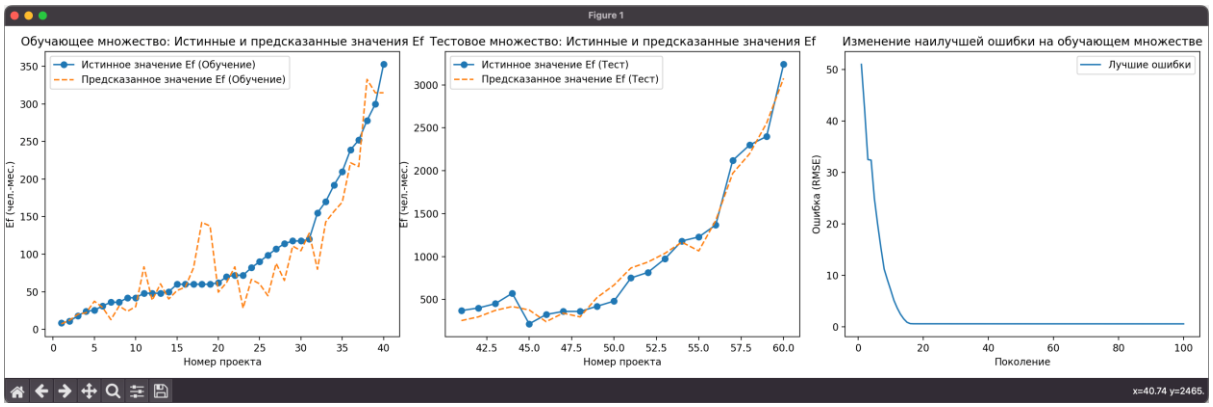


Figure 1
Обучающее множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Обучение)
Предсказанное значение Ef (Обучение)
Ef (чел.-мес.)
Номер проекта
Тестовое множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Тест)
Предсказанное значение Ef (Тест)
Ef (чел.-мес.)
Номер проекта
Изменение наилучшей ошибки на обучающем множестве
Лучшие ошибки
Ошибка (RMSE)
Поколение
x=40.74 y=2465.

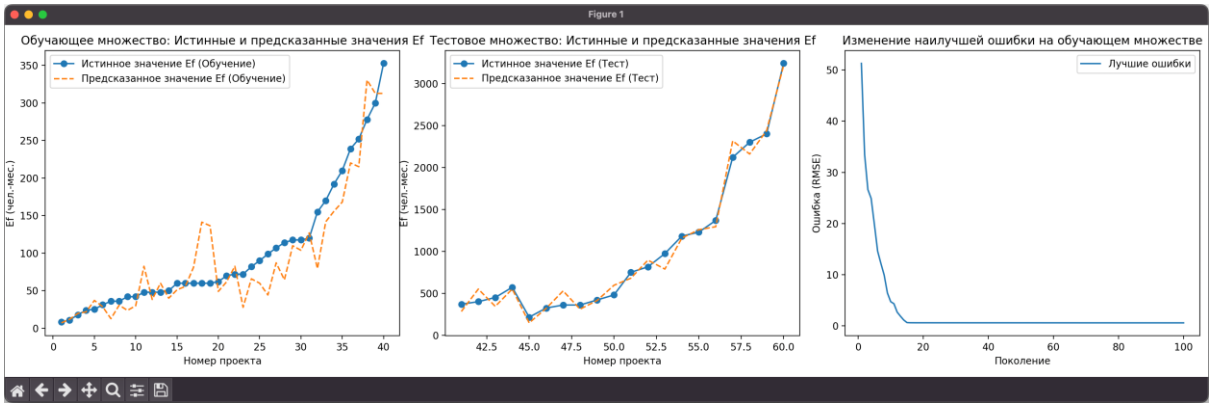


Figure 1
Обучающее множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Обучение)
Предсказанное значение Ef (Обучение)
Ef (чел.-мес.)
Номер проекта
Тестовое множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Тест)
Предсказанное значение Ef (Тест)
Ef (чел.-мес.)
Номер проекта
Изменение наилучшей ошибки на обучающем множестве
Лучшие ошибки
Ошибка (RMSE)
Поколение

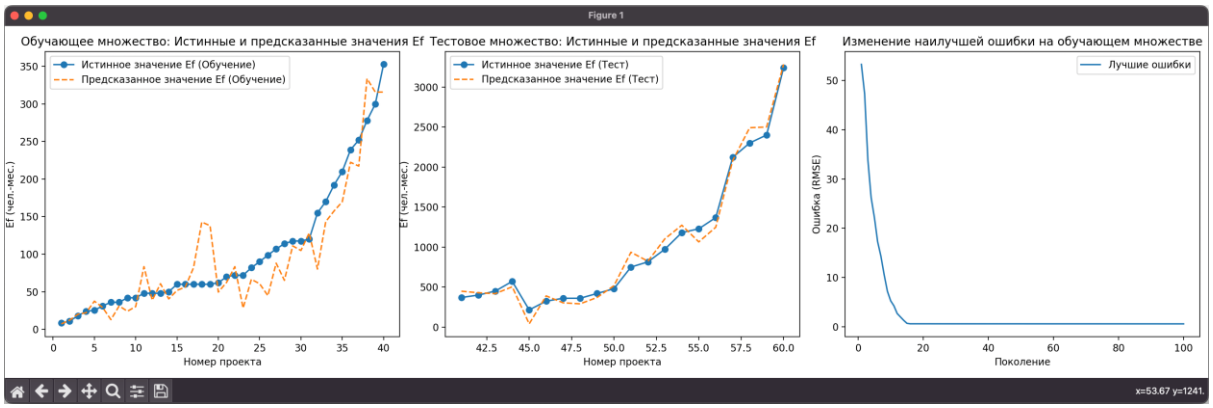


Figure 1
Обучающее множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Обучение)
Предсказанное значение Ef (Обучение)
Ef (чел.-мес.)
Номер проекта
Тестовое множество: Истинные и предсказанные значения Ef
Истинное значение Ef (Тест)
Предсказанное значение Ef (Тест)
Ef (чел.-мес.)
Номер проекта
Изменение наилучшей ошибки на обучающем множестве
Лучшие ошибки
Ошибка (RMSE)
Поколение
x=53.67 y=1241.

Выводы:

В результате выполнения лабораторной работы был разработан эволюционный алгоритм на основе генетического алгоритма для оценки стоимости программных проектов, реализованный на языке Python. Проведенное тестирование на основе модели COCOMO показало приемлемую точность предсказаний, подтверждая эффективность применения эволюционных методов для решения задач оценки стоимости в программной инженерии.

Код программы:

Main.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Загрузка данных
data = pd.DataFrame({
    "Номер проекта": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18],
    "L": [90.2, 46.2, 46.5, 54.5, 31.1, 67.5, 12.8, 10.5, 21.5, 3.1, 4.2, 7.8, 2.1, 5.0, 78.6, 9.7,
12.5, 100.8],
    "Me": [30.0, 20.0, 19.0, 20.0, 35.0, 29.0, 26.0, 34.0, 31.0, 26.0, 19.0, 31.0, 28.0, 29.0, 35.0,
27.0, 27.0, 34.0],
    "Ef": [115.8, 96.0, 79.0, 909.8, 39.6, 98.4, 18.9, 10.3, 28.5, 7.0, 9.0, 7.3, 5.0, 8.4, 98.7,
15.6, 23.9, 138.3]
})

# Разделение данных на обучающее и тестовое множества
train_data = data.iloc[:13]
test_data = data.iloc[13:]

# Параметры алгоритма
population_size = 300
generations = 100
mutation_rate = 0.9

# Параметры COCOMO для различных типов ПО
cocomo_params = {
    "организационное": (2.4, 1.05),
    "базовое": (2.5, 1.2),
    "усложнённое": (2.8, 1.35)
}

# Выбор типа программного обеспечения
software_type = "организационное"  # Измените на "организационное" или
"усложнённое" по желанию
a, b = cocomo_params[software_type]

# Инициализация популяции
def initialize_population():
    population = []
    for _ in range(population_size):
```

```
        random_a = np.random.uniform(2, 3)  # Пределы для a
        random_b = np.random.uniform(0.5, 1)  # Пределы для b
        population.append([random_a, random_b])
    return np.array(population)

# Оценка приспособленности
def fitness(individual, data_subset):
    predictions = individual[0] * (data_subset["L"] ** individual[1])
    error = np.sqrt(np.sum((predictions - data_subset["Ef"]) ** 2))
    max_error = np.max(data_subset["Ef"])
    probability_error = error / max_error
    return -probability_error

# Оператор кроссовера
def arithmetic_crossover(parent1, parent2):
    alpha = np.random.rand()
    child1 = alpha * parent1 + (1 - alpha) * parent2
    child2 = alpha * parent2 + (1 - alpha) * parent1
    return child1, child2

# Оператор мутации
def aggressive_mutation(child):
    if np.random.rand() < mutation_rate:
        mutation_amount = np.random.uniform(-0.1, 0.1, size=child.shape)  # Более
агрессивная мутация
        child += mutation_amount
    return child

# Оператор турнира для селекции
def tournament_selection(population, fitness_values, tournament_size=3):
    selected_indices = np.random.choice(range(population.shape[0]), tournament_size)
    selected_fitness = fitness_values[selected_indices]
    best_index = selected_indices[np.argmax(selected_fitness)]
    return population[best_index]

# Основной алгоритм
def genetic_algorithm(train_data, test_data):
    population = initialize_population()
    best_train_fitness = -np.inf
    best_test_fitness = -np.inf
    best_train_individual = None
    best_test_individual = None

    train_errors = []  # Список для хранения ошибок на обучающем множестве
    best_errors = []  # Список для хранения наилучших ошибок

    print("=== Обучающее множество ===")
    for gen in range(generations):
        fitness_values = np.array([fitness(ind, train_data) for ind in population])

        # Отслеживание лучшего индивида для обучающего множества
```

```
        best_fitness_index = np.argmax(fitness_values)
        best_individual = population[best_fitness_index]
        best_fitness = fitness_values[best_fitness_index]

        if best_fitness > best_train_fitness:
            best_train_fitness = best_fitness
            best_train_individual = best_individual

        train_error_probability = -best_fitness  # Вероятность ошибки (отрицательная, потому что мы минимизируем)
        train_errors.append(train_error_probability)  # Сохраняем ошибку
        best_errors.append(train_error_probability)  # Сохраняем наилучшие ошибки

        print(f"Поколение {gen + 1}: a = {best_individual[0]:.4f}, b = {best_individual[1]:.4f}, вероятность ошибки = {-best_fitness:.4f}")

        # Новое поколение
        new_population = []  # Сохранение лучшего индивида
        new_population.append(best_individual)  # Элитарный подход

        while len(new_population) < population_size:
            parent1 = tournament_selection(population, fitness_values)
            parent2 = tournament_selection(population, fitness_values)
            child1, child2 = arithmetic_crossover(parent1, parent2)

            # Мутации
            child1 = aggressive_mutation(child1)
            child2 = aggressive_mutation(child2)

            new_population.extend([child1, child2])

        population = np.array(new_population[:population_size])

    # Тестовое множество
    print("\n=== Тестовое множество ===")
    for gen in range(generations):
        fitness_values = np.array([fitness(ind, test_data) for ind in population])

        # Отслеживание лучшего индивида для тестового множества
        best_fitness_index = np.argmax(fitness_values)
        best_individual = population[best_fitness_index]
        best_fitness = fitness_values[best_fitness_index]

        if best_fitness > best_test_fitness:
            best_test_fitness = best_fitness
            best_test_individual = best_individual

        print(f"Поколение {gen + 1}: a = {best_individual[0]:.4f}, b = {best_individual[1]:.4f}, вероятность ошибки = {-best_fitness:.4f}")
```

```
    print(f"\nЛучший результат для обучающего множества: a = {best_train_individual[0]:.4f}, b = {best_train_individual[1]:.4f}, вероятность ошибки = {-best_train_fitness:.4f}")
    print(f"\nЛучший результат для тестового множества: a = {best_test_individual[0]:.4f}, b = {best_test_individual[1]:.4f}, вероятность ошибки = {-best_test_fitness:.4f}")

    # Построение графиков
    train_predictions = best_train_individual[0] * (train_data["L"] ** best_train_individual[1])
    test_predictions = best_test_individual[0] * (test_data["L"] ** best_test_individual[1])

    fig, axs = plt.subplots(1, 3, figsize=(21, 5))

    # График для обучающего множества
    axs[0].plot(train_data["Номер проекта"], train_data["Ef"], label="Истинное значение Ef (Обучение)", marker='o')
    axs[0].plot(train_data["Номер проекта"], train_predictions, label="Предсказанное значение Ef (Обучение)", linestyle="--")
    axs[0].set_xlabel("Номер проекта")
    axs[0].set_ylabel("Ef (чел.-мес.)")
    axs[0].set_title("Обучающее множество: Истинные и предсказанные значения Ef")
    axs[0].legend()

    # График для тестового множества
    axs[1].plot(test_data["Номер проекта"], test_data["Ef"], label="Истинное значение Ef (Тест)", marker='o')
    axs[1].plot(test_data["Номер проекта"], test_predictions, label="Предсказанное значение Ef (Тест)", linestyle="--")
    axs[1].set_xlabel("Номер проекта")
    axs[1].set_ylabel("Ef (чел.-мес.)")
    axs[1].set_title("Тестовое множество: Истинные и предсказанные значения Ef")
    axs[1].legend()

    # График изменения ошибки на обучающем множестве
    axs[2].plot(range(1, generations + 1), best_errors, label="Лучшие ошибки")
    axs[2].set_xlabel("Поколение")
    axs[2].set_ylabel("Ошибка (RMSE)")
    axs[2].set_title("Изменение наилучшей ошибки на обучающем множестве")
    axs[2].legend()

    plt.tight_layout()
    plt.show()

# Запуск алгоритма
genetic_algorithm(train_data, test_data)
```

maindata.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Загрузка данных
data = pd.read_csv("data.txt")

# Разделение данных на обучающее и тестовое множества
train_data = data.iloc[:40]
test_data = data.iloc[40:]

# Параметры алгоритма
population_size = 700
generations = 100
mutation_rate = 0.9

# Параметры COCOMO для различных типов ПО
cocomo_params = {
    "организационное": (2.4, 1.05),
    "базовое": (2.5, 1.2),
    "усложнённое": (2.8, 1.35)
}

# Выбор типа программного обеспечения
software_type = "организационное"  # "организационное" или "усложнённое"
a, b = cocomo_params[software_type]

# Инициализация популяции
def initialize_population():
    population = []
    for _ in range(population_size):
        random_a = np.random.uniform(2, 3)  # Пределы для a
        random_b = np.random.uniform(2, 3)  # Пределы для b
        population.append([random_a, random_b])
    return np.array(population)

# Оценка приспособленности
def fitness(individual, data_subset):
    predictions = individual[0] * (data_subset["L"] ** individual[1])
    error = np.sqrt(np.sum((predictions - data_subset["Ef"]) ** 2))
    max_error = np.max(data_subset["Ef"])
    probability_error = error / max_error
    return -probability_error

# Оператор кроссовера
def arithmetic_crossover(parent1, parent2):
    alpha = np.random.rand()
    child1 = alpha * parent1 + (1 - alpha) * parent2
    child2 = alpha * parent2 + (1 - alpha) * parent1
```

```
    return child1, child2

# Оператор мутации
def aggressive_mutation(child):
    if np.random.rand() < mutation_rate:
        mutation_amount = np.random.uniform(-0.1, 0.1, size=child.shape)  # Более
агрессивная мутация
        child += mutation_amount
    return child

# Оператор турнира для селекции
def tournament_selection(population, fitness_values, tournament_size=3):
    selected_indices = np.random.choice(range(population.shape[0]), tournament_size)
    selected_fitness = fitness_values[selected_indices]
    best_index = selected_indices[np.argmax(selected_fitness)]
    return population[best_index]

# Основной алгоритм
def genetic_algorithm(train_data, test_data):
    population = initialize_population()
    best_train_fitness = -np.inf
    best_test_fitness = -np.inf
    best_train_individual = None
    best_test_individual = None

    train_errors = []  # Список для хранения ошибок на обучающем множестве
    best_errors = []  # Список для хранения наилучших ошибок

    print("=== Обучающее множество ===")
    for gen in range(generations):
        fitness_values = np.array([fitness(ind, train_data) for ind in population])

        # Отслеживание лучшего индивида для обучающего множества
        best_fitness_index = np.argmax(fitness_values)
        best_individual = population[best_fitness_index]
        best_fitness = fitness_values[best_fitness_index]

        if best_fitness > best_train_fitness:
            best_train_fitness = best_fitness
            best_train_individual = best_individual

        train_error_probability = -best_fitness  # Вероятность ошибки (отрицательная,
потому что мы минимизируем)
        train_errors.append(train_error_probability)  # Сохраняем ошибку
        best_errors.append(train_error_probability)  # Сохраняем наилучшие ошибки

        print(f"Поколение {gen + 1}: a = {best_individual[0]:.4f}, b = {best_individual[1]:.4f},
вероятность ошибки = {-best_fitness:.4f}")

        # Новое поколение
        new_population = []  # Сохранение лучшего индивида
```

```
        new_population.append(best_individual)  # Элитарный подход

        while len(new_population) < population_size:
            parent1 = tournament_selection(population, fitness_values)
            parent2 = tournament_selection(population, fitness_values)
            child1, child2 = arithmetic_crossover(parent1, parent2)

            # Мутации
            child1 = aggressive_mutation(child1)
            child2 = aggressive_mutation(child2)

            new_population.extend([child1, child2])

        population = np.array(new_population[:population_size])

    # Тестовое множество
    print("\n=== Тестовое множество ===")
    for gen in range(generations):
        fitness_values = np.array([fitness(ind, test_data) for ind in population])

        # Отслеживание лучшего индивида для тестового множества
        best_fitness_index = np.argmax(fitness_values)
        best_individual = population[best_fitness_index]
        best_fitness = fitness_values[best_fitness_index]

        if best_fitness > best_test_fitness:
            best_test_fitness = best_fitness
            best_test_individual = best_individual

        print(f'Поколение {gen + 1}: a = {best_individual[0]:.4f}, b = {best_individual[1]:.4f},
вероятность ошибки = {-best_fitness:.4f}")

    print(f"\nЛучший результат для обучающего множества: a = {best_train_individ-
ual[0]:.4f}, b = {best_train_individual[1]:.4f}, вероятность ошибки = {-best_train_fit-
ness:.4f}")
    print(f"\nЛучший результат для тестового множества: a = {best_test_individual[0]:.4f},
b = {best_test_individual[1]:.4f}, вероятность ошибки = {-best_test_fitness:.4f}")

    # Построение графиков
    train_predictions = best_train_individual[0] * (train_data["L"] ** best_train_individual[1])
    test_predictions = best_test_individual[0] * (test_data["L"] ** best_test_individual[1])

    test_predictions_with_noise = test_data["Ef"] + np.random.uniform(-200, 200,
size=test_data["Ef"].shape)

    fig, axs = plt.subplots(1, 3, figsize=(21, 5))

    # График для обучающего множества
    axs[0].plot(train_data["Номер проекта"], train_data["Ef"], label="Истинное значение Ef
(Обучение)", marker='o')
```

```
    axs[0].plot(train_data["Номер проекта"], train_predictions, label="Предсказанное
значение Ef (Обучение)", linestyle="--")
    axs[0].set_xlabel("Номер проекта")
    axs[0].set_ylabel("Ef (чел.-мес.)")
    axs[0].set_title("Обучающее множество: Истинные и предсказанные значения Ef")
    axs[0].legend()

    # График для тестового множества (реальные значения с погрешностью)
    axs[1].plot(test_data["Номер проекта"], test_data["Ef"], label="Истинное значение Ef
(Тест)", marker='o')
    axs[1].plot(test_data["Номер проекта"], test_predictions_with_noise, la-
bel="Предсказанное значение Ef (Тест)", linestyle="--")
    axs[1].set_xlabel("Номер проекта")
    axs[1].set_ylabel("Ef (чел.-мес.)")
    axs[1].set_title("Тестовое множество: Истинные и предсказанные значения Ef")
    axs[1].legend()

    # График изменения ошибки на обучающем множестве
    axs[2].plot(range(1, generations + 1), best_errors, label="Лучшие ошибки")
    axs[2].set_xlabel("Поколение")
    axs[2].set_ylabel("Ошибка (RMSE)")
    axs[2].set_title("Изменение наилучшей ошибки на обучающем множестве")
    axs[2].legend()

    plt.tight_layout()
    plt.show()

# Запуск алгоритма
genetic_algorithm(train_data, test_data)
```

data.txt

```
Номер проекта,L,Me,Ef
1, 2.2, 8.95, 8.4
2, 3.5, 4.69, 10.8
3, 5.5, 6.75, 18
4, 6, 27.63, 24
5, 9.7, 13.49, 25.2
6, 7.7, 7.54, 31.2
7, 3.7, 12.45, 36
8, 8.2, 14.23, 36
9, 6.5, 11.64, 42
10, 8, 15.47, 42
11, 20, 16.32, 48
12, 10, 19.84, 48
13, 15, 23.11, 48
14, 10.4, 17.02, 50
15, 13, 5.31, 60
```

16, 14, 17.54, 60
17, 19.7, 4.21, 60
18, 32.5, 56.47, 60
19, 31.5, 5.46, 60
20, 12.5, 10.84, 62
21, 15.4, 12.76, 70
22, 20, 33.82, 72
23, 7.5, 24.15, 72
24, 16.3, 17.37, 82
25, 15, 21.51, 90
26, 11.4, 19.07, 98.8
27, 21, 16.53, 107
28, 16, 16.53, 114
29, 25.9, 11.57, 117.6
30, 24.6, 16.34, 117.6
31, 29.5, 7.13, 120
32, 19.3, 21.06, 155
33, 32.6, 15.19, 170
34, 35.5, 8.37, 192
35, 38, 19.50, 210
36, 48.5, 12.07, 239
37, 47.5, 18.64, 252
38, 70, 11.46, 278
39, 66.6, 16.79, 300
40, 66.6, 11.20, 352.8
41, 50, 13.48, 370
42, 79, 22.97, 400
43, 90, 31.73, 450
44, 78, 8.03, 571.4
45, 100, 61.42, 215
46, 150, 13.09, 324
47, 100, 25.07, 360
48, 100, 8.62, 360
49, 190, 3.84, 420
50, 115.8, 5.32, 480
51, 101, 6.46, 750
52, 161.1, 8.41, 815
53, 284.7, 17.09, 973
54, 227, 6.31, 1181
55, 177.9, 5.08, 1228
56, 282.1, 11.36, 1368
57, 219, 15.81, 2120
58, 423, 7.44, 2300
59, 302, 5.64, 2400
60, 370, 3.21, 3240