

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**Федеральное государственное автономное образовательное учреждение**  
**высшего образования**  
**«Санкт-Петербургский государственный университет**  
**аэрокосмического приборостроения»**

---

КАФЕДРА 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
РУКОВОДИТЕЛЬ

Д-р техн. наук, профессор  
\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

Ю.А. Скобцов  
\_\_\_\_\_  
инициалы, фамилия

Отчет о лабораторной работе №4

Генетическое программирование

По дисциплине: Эволюционные методы проектирования программно-  
информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4133К

\_\_\_\_\_  
подпись, дата

А.С. Захаров  
\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2024

**Цель работы:**

Решение задачи символьной регрессии. Графическое отображение результатов оптимизации.

**Вариант:**

Вид функции, Кол-во пер-ых N, Промежуток исследования

14	$fGold(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14 \cdot x_1 + 3 \cdot x_1^2 - 14 \cdot x_2 + 6 \cdot x_1 \cdot x_2 + 3 \cdot x_2^2)] \cdot [30 + (2 \cdot x_1 - 3 \cdot x_2)^2 \cdot (18 - 32 \cdot x_1 + 12 \cdot x_1^2 + 48 \cdot x_2 - 36 \cdot x_1 \cdot x_2 + 27 \cdot x_2^2)]$	2	$-2 \leq x(i) \leq 2, i=1:2.$
----	---	---	-------------------------------

Вид функции:  $fGold(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14 \cdot x_1 + 3 \cdot x_1^2 - 14 \cdot x_2 + 6 \cdot x_1 \cdot x_2 + 3 \cdot x_2^2)] \cdot [30 + (2 \cdot x_1 - 3 \cdot x_2)^2 \cdot (18 - 32 \cdot x_1 + 12 \cdot x_1^2 + 48 \cdot x_2 - 36 \cdot x_1 \cdot x_2 + 27 \cdot x_2^2)]$

Кол-во пер-ых N: 2

Промежуток исследования:  $-2 \leq x(i) \leq 2, i=1:2$

**Задание:**

1. Разработать эволюционный алгоритм, реализующий ГП для нахождения заданной по варианту функции (таб. 4.1).
  - Структура для представления программы – древовидное представление.
  - Терминальное множество: переменные  $x_1, x_2, x_3, \dots, x_n$ , и константы в соответствии с заданием по варианту.
  - Функциональное множество:  $+, -, *, /, \text{abs}(), \sin(), \cos(), \exp()$ , возведение в степень,
  - Фитнесс-функция – мера близости между реальными значениями выхода и требуемыми.
2. Представить графически найденное решение на каждой итерации.
3. Сравнить найденное решение с представленным в условии задачи.

**Общий алгоритм генетического программирования**

Таким образом, для решения задачи с помощью ГП необходимо выполнить описанные выше предварительные этапы:

- 1) Определить терминальное множество;
- 2) Определить функциональное множество;
- 3) Определить фитнесс-функцию;
- 4) Определить значения параметров, такие как мощность популяции, максимальный размер особи, вероятности кроссинговера и мутации, способ отбора родителей, критерий окончания эволюции (например, максимальное число поколений) и т. п. После этого можно разрабатывать непосредственно сам эволюционный алгоритм, реализующий ГП для конкретной задачи. Например, решение задачи на основе ГП можно представить следующей последовательностью действий.
  - 1) установка параметров эволюции;
  - 2) инициализация начальной популяции;
  - 3)  $t = 0$ ;
  - 4) оценка особей, входящих в популяцию;
  - 5)  $t = t + 1$ ;
  - 6) отбор родителей;
  - 7) создание потомков выбранных пар родителей – выполнение оператора кроссинговера;
  - 8) мутация новых особей;
  - 9) расширение популяции новыми порожденными особями;

- 10) сокращение расширенной популяции до исходного размера;
  - 11) если критерий останова алгоритма выполнен, то выбор лучшей особи в конечной популяции – результат работы алгоритма.
- Иначе переход на шаг 4.

### Выполнение:

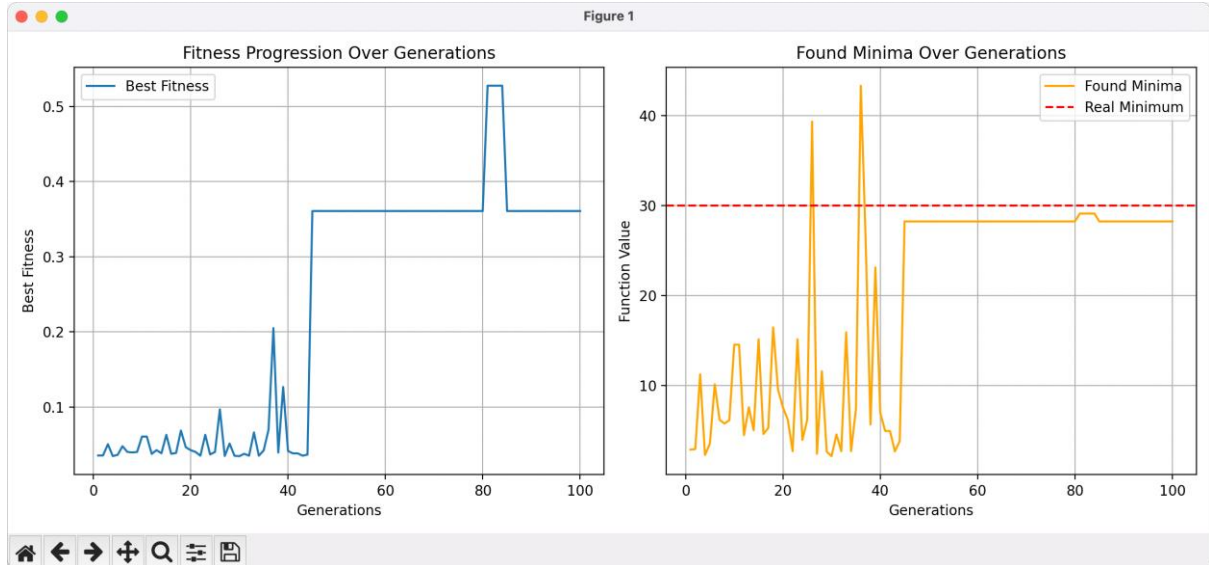


Рисунок 1 – вывод графиков

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
'exp'
Generation 99: Best Fitness = 0.3609, Best Value = 28.2294
Best Individual Tree Structure:
'exp'
  '-'
    '/'
      -0.44174379856440016
      'sin'
        '*'
          -1.031764946498898
          0.22600807288277247
        'x1'
      -1.428712213179677
    '+'
      0.07580680893561142
      '-'
        'x1'
        'sin'
          'x2'
          '*'
            'cos'
              'x1'
              1.7940857939329304
            'x2'

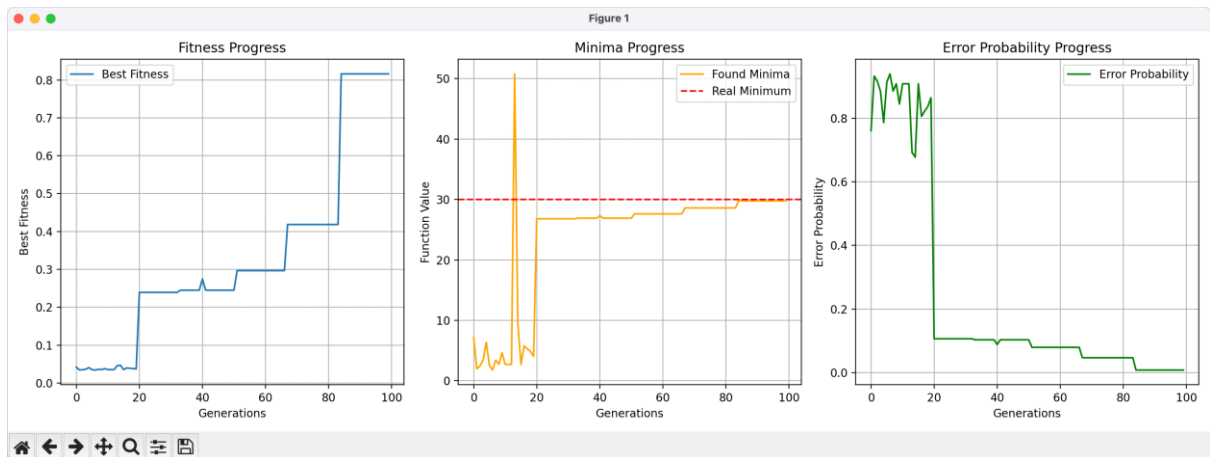
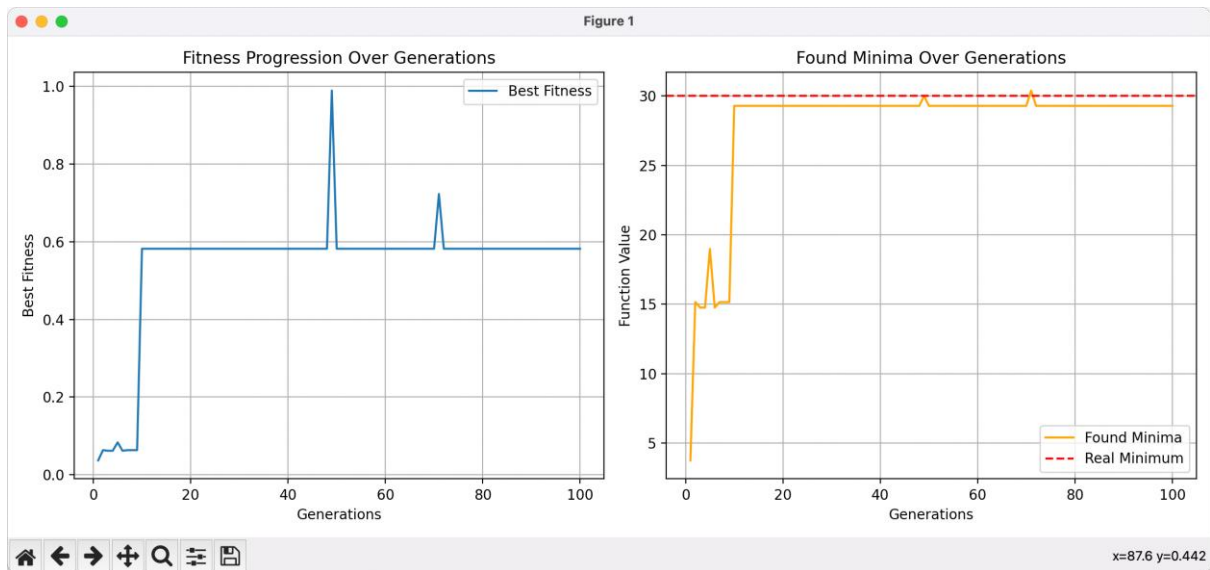
Generation 100: Best Fitness = 0.3609, Best Value = 28.2294
Best Individual Tree Structure:
'exp'
  '-'
    '/'
      -0.44174379856440016
      'sin'
        '*'
          -1.031764946498898
          0.22600807288277247
        'x1'
      -1.428712213179677
    'x2'

Best Overall Individual:
'abs'
  'exp'
    'exp'
      1.2151829104856962
      -1.1084582316051428
      -1.4285287594678362
      1.87475815147062

Best Overall Value: 29.1050
Best Overall Structure: 0.5777
hrs 15 mins Coding, 32 mins Browsing  -> Java: Ready

```

Рисунок 2 – вывод в виде дерева



Best Overall Individual (Generation 85):

```
'+'
'*'
'abs'
'exp'
'x2'
'abs'
'sin'
-0.7531380587091716
'x1'
'*'
-1.3323160353712384
'x2'
'cos'
'+'
'x1'
0.6282595687969681
0.9979052314445584
'exp'
'exp'
```

Best Overall Value: 29.7744  
Best Overall Fitness: 0.8160



## 1. Импорт библиотек

- **random**: Для генерации случайных чисел.
- **numpy**: Для выполнения математических операций.
- **matplotlib.pyplot**: Для визуализации графиков.
- **scipy.optimize**: Для нахождения реального минимума функции.

## 2. Настройки параметров алгоритма

- Определяются параметры генетического алгоритма, такие как количество поколений, размер популяции, максимальная глубина дерева и вероятность мутации.

## 3. Определение функции fGold

- **fGold(x1, x2)**: Определяет целевую функцию, которую нужно минимизировать. Функция имеет сложную формулу и зависит от двух переменных  $x_1$  и  $x_2$ .

## 4. Поиск реального минимума функции

- **find\_real\_minimum()**: Использует метод `minimize` из `scipy` для нахождения реального минимума функции `fGold` в заданных границах.

## 5. Класс узла дерева Node

- Определяет структуру для представления узлов в дереве (дерево решений), включая методы для оценки выражений, построенных из узлов.

## 6. Генерация случайного дерева

- **generate\_tree(max\_depth, current\_depth=0)**: Рекурсивно создает случайное дерево заданной глубины, используя случайные функции и значения.

## 7. Оценка фитнеса особи

- **fitness(individual, real\_minimum)**: Вычисляет “фитнес” (качество) индивидуумов в популяции на основе того, насколько близко их оценка к реальному минимуму.

## 8. Вероятность ошибки

- **error\_probability(found\_value, real\_minimum)**: Рассчитывает вероятность ошибки, определяя, насколько найденное значение отличается от реального минимума.

## 9. Отбор родителей

- **select\_parents(population, real\_minimum):** Выбирает двух родителей из популяции на основе их фитнеса с использованием вероятностного отбора.

## 10. Кроссинговер

- **crossover(parent1, parent2):** Создает двух детей, комбинируя части генетического материала (дерева) родителей.

## 11. Мутация

- **mutate(individual, max\_depth):** Случайным образом изменяет индивидуум, создавая новое дерево, если происходит мутация.

## 12. Визуализация дерева

- **plot\_tree(node, pos, level, delta\_x, ax):** Рекурсивно визуализирует дерево, создавая графическое представление структуры узлов.

## 13. Основной алгоритм генетического программирования

- **genetic\_programming():** Основная функция, реализующая генетический алгоритм. Она инициализирует популяцию, проводит эволюцию через отбор, кроссинговер и мутацию, а также отслеживает прогресс, фитнес и вероятности ошибок для каждой генерации.

## 14. Вывод результатов

- После завершения всех поколений выводится информация о наилучшем индивидууме и его фитнесе, а также строятся графики для визуализации прогресса и наилучшего дерева.

## 15. Запуск программы

- Вся логика генетического программирования выполняется при запуске скрипта, вызывая функцию `genetic_programming()`.

### Выводы:

В данной работе был реализован алгоритм генетического программирования для поиска минимума функции, используя деревья для представления математических выражений. Проведён анализ полученных решений, где удалось найти выражение, приближающееся к реальному минимуму функции, что подтвердило эффективность использованного подхода и его потенциал для решения задач оптимизации в различных областях.

### Код программы:

```
import random
```

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# Параметры алгоритма
params = {
    'generations': 100,    # Количество поколений
    'population_size': 300, # Размер популяции
    'max_depth': 8,        # Максимальная глубина дерева
    'mutation_chance': 0.7 # Вероятность мутации
}

# Функция для оценки (первоначальная функция)
def fGold(x1, x2):
    return (1 + (x1 + x2 + 1)**2 * (19 - 14*x1 + 3*x1**2 - 14*x2 + 6*x1*x2 + 3*x2**2)) * \
        (30 + (2*x1 - 3*x2)**2 * (18 - 32*x1 + 12*x1**2 + 48*x2 - 36*x1*x2 + 27*x2**2))

# Поиск реального минимума функции
def find_real_minimum():
    bounds = [(-2, 2), (-2, 2)]
    result = minimize(lambda x: fGold(x[0], x[1]), [0, 0], bounds=bounds)
    return result.fun, result.x

# Класс для узла дерева
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def evaluate(self, x1, x2):
        if self.value in ['+', '-', '*', '/', 'abs', 'sin', 'cos', 'exp']:
            try:
                left_value = self.left.evaluate(x1, x2) if self.left else None
                right_value = self.right.evaluate(x1, x2) if self.right else None

                if left_value is None or right_value is None:
                    return None # Пропускаем, если нет значений

                if self.value == '+':
                    return left_value + right_value
                elif self.value == '-':
                    return left_value - right_value
                elif self.value == '*':
                    return left_value * right_value
                elif self.value == '/':
                    if right_value == 0:
                        return None # Пропускаем деление на 0

```



```

        return left_value / right_value
    elif self.value == 'abs':
        return abs(left_value)
    elif self.value == 'sin':
        return np.sin(left_value)
    elif self.value == 'cos':
        return np.cos(left_value)
    elif self.value == 'exp':
        return np.exp(left_value)
    except Exception:
        return None # Возвращаем None при ошибке
else:
    return float(self.value) if isinstance(self.value, (int, float)) else (x1 if self.value == 'x1'
else x2)

```

```

def __str__(self, level=0):
    ret = "\t" * level + repr(self.value) + "\n"
    if self.left: ret += self.left.__str__(level + 1)
    if self.right: ret += self.right.__str__(level + 1)
    return ret

```

# Генерация случайного дерева

```

def generate_tree(max_depth, current_depth=0):
    if current_depth < max_depth and random.random() > 0.5:
        # Генерация внутреннего узла
        func = random.choice(['+', '-', '*', '/', 'abs', 'sin', 'cos', 'exp'])
        left = generate_tree(max_depth, current_depth + 1)
        right = generate_tree(max_depth, current_depth + 1)
        return Node(func, left, right)
    else:
        # Генерация терминала (переменные или константы)
        return Node(random.choice([random.uniform(-2, 2), 'x1', 'x2']))

```

# Оценка фитнеса особи

```

def fitness(individual, real_minimum):
    output = individual.evaluate(0, 0) # Используем 0, 0 как заглушку
    if output is None: # Проверяем на None
        return 0
    # Стремимся к реальному минимуму (30)
    return 1 / (1 + abs(output - real_minimum))

```

# Отбор родителей

```

def select_parents(population, real_minimum):
    weights = [fitness(ind, real_minimum) for ind in population]
    return random.choices(population, weights=weights, k=2)

```

# Кроссинговер

```

def crossover(parent1, parent2):

```

```

# Обмен поддеревьями
child1 = Node(parent1.value)
child2 = Node(parent2.value)

child1.left, child1.right = parent1.left, parent2.right
child2.left, child2.right = parent2.left, parent1.right
return child1, child2

# Мутация
def mutate(individual, max_depth):
    if random.random() < params['mutation_chance']: # Шанс мутации
        return generate_tree(max_depth) # Генерируем новое дерево
    return individual # Если мутация не произошла, возвращаем оригинал

# Основной алгоритм генетического программирования
def genetic_programming():
    population = [generate_tree(params['max_depth']) for _ in
range(params['population_size'])]
    best_fitnesses = []
    found_minima = []

    # Находим реальный минимум
    real_minimum, _ = find_real_minimum()
    print(f"Real Minimum Value: {real_minimum}")

    # Переменные для хранения наилучшего индивидуума
    best_overall_individual = None
    best_overall_fitness = 0

    for generation in range(params['generations']):
        new_population = []
        for _ in range(params['population_size'] // 2):
            parent1, parent2 = select_parents(population, real_minimum)
            child1, child2 = crossover(parent1, parent2)
            new_population.extend([mutate(child1, params['max_depth']), mutate(child2,
params['max_depth'])])

        population = new_population

        # Находим лучший фитнес в текущем поколении
        best_individual = max(population, key=lambda ind: fitness(ind, real_minimum))
        found_minima.append(best_individual.evaluate(0, 0)) # Вычисляем значение
функции
        best_fitness = fitness(best_individual, real_minimum)
        best_fitnesses.append(best_fitness)

        # Обновляем наилучшего индивидуума за все поколения
        if best_fitness > best_overall_fitness:

```

```

    best_overall_fitness = best_fitness
    best_overall_individual = best_individual

    # Выводим древовидное представление лучшего индивидуума
    print(f"\nGeneration {generation + 1}: Best Fitness = {best_fitness:.4f}, Best Value = {found_minima[-1]:.4f}")
    print("Best Individual Tree Structure:")
    print(best_individual)

    # Вывод наилучшего результата из всех поколений
    print("\nBest Overall Individual:")
    print(best_overall_individual)
    print(f"Best Overall Value: {best_overall_individual.evaluate(0, 0):.4f}")
    print(f"Best Overall Fitness: {best_overall_fitness:.4f}")

    # Отображение графиков
    plt.figure(figsize=(12, 5))

    # График 1: Прогресс фитнеса
    plt.subplot(1, 2, 1)
    plt.plot(range(1, params['generations'] + 1), best_fitnesses, label='Best Fitness')
    plt.xlabel('Generations')
    plt.ylabel('Best Fitness')
    plt.title('Fitness Progression Over Generations')
    plt.grid()
    plt.legend()

    # График 2: Найденные минимумы и реальный минимум
    plt.subplot(1, 2, 2)
    plt.plot(range(1, params['generations'] + 1), found_minima, label='Found Minima',
color='orange')
    plt.axhline(real_minimum, color='red', linestyle='--', label='Real Minimum')
    plt.xlabel('Generations')
    plt.ylabel('Function Value')
    plt.title('Found Minima Over Generations')
    plt.grid()
    plt.legend()

    plt.tight_layout()
    plt.show()

    # Запуск алгоритма
    genetic_programming()

```