**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**Факультет прикладной математики – процессов управления**

**отчет**

**по лабораторной работе**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Исследование генетического алгоритма. Изучение различных кодировок генотипа»**

**1 вариант**

|  |  |  |
| --- | --- | --- |
| Студент гр. 22.Б15 |  | Агишев А.Б. |
| Преподаватель |  | Дик А.Г. |

**Санкт-Петербург**

**2023 г.**

**Содержание**

[1. Цель работы 3](#_Toc152641675)

[2. Задача 3](#_Toc152641676)

[3. Теоретическая часть 3](#_Toc152641677)

[4. Описание алгоритма 4](#_Toc152641678)

[5. Описание программы 6](#_Toc152641679)

[5.1 Описание классов 6](#_Toc152641680)

[5.2. Описание функций 6](#_Toc152641681)

[5.3. Описание переменных 7](#_Toc152641682)

[6. Рекомендации пользователю 8](#_Toc152641683)

[7. Рекомендации программисту 8](#_Toc152641684)

[8. Контрольный пример 9](#_Toc152641685)

[9. Заключение 10](#_Toc152641686)

[10. Приложения 10](#_Toc152641687)

# **Цель работы**

Разработать алгоритм и программу, которая использует генетический алгоритм для поиска минимального значения заданной математической функции.

# **Задача**

1) Изучить теоретические основы в области генетических алгоритмов, методов их оптимизаций.

2) Изучить два основных способа кодирования хромосом в генетическом алгоритме.

3) Написать программу поиска минимума функции.

4) Протестировать программу на выбранной тестовой функции.

5) Проанализировать работу алгоритма в зависимости от методов оптимизаций, способов кодирования хромосом, параметров: размера популяции, вероятности мутаций и др.

# **Теоретическая часть**

*Генетический алгоритм* представляет собой оптимизационный метод, вдохновленный принципами естественного отбора и генетики. Он использует механизмы эволюции, такие как мутация, скрещивание и селекция, для поиска приближенных решений задач оптимизации.

Основные функции в генетическом алгоритме:

*Инициализация популяции*: создание начальной группы хромосом со случайными значениями генов.

*Расчет пригодности (fitness)*: оценка каждой хромосомы, вычисление значения функции приспособленности.

*Селекция*: выбор лучших хромосом для следующего поколения на основе их пригодности.

*Скрещивание*: образование новых хромосом путем комбинирования генов двух родителей.

*Мутация*: случайное изменение генов с целью внесения разнообразия в популяцию.

Оптимизация через *стратегию эволюционных стратегий* для скрещивания:

В генетическом алгоритме применяется стратегия эволюционных стратегий для скрещивания. Это подразумевает использование взвешенной смеси генов родителей с коэффициентом α. Этот параметр позволяет более гибко управлять влиянием каждого родителя на потомство.

Два типа кодирования хромосом:

*Бинарное кодирование*: гены хромосомы представлены в виде бинарных строк, что особенно полезно для задач с дискретным пространством поиска.

*Вещественное кодирование*: гены хромосомы представлены в виде вещественных чисел, что позволяет работать с непрерывным пространством поиска.

# **Описание алгоритма**

1. Получение основных характеристик через интерфейс программы: функция, вероятность мутации, количество хромосом, минимальное значение гена, максимальное значение гена, количество поколений, тип кодирования.
2. Запуск генетического алгоритма с заданными параметрами.
3. Отображение лучшего решения вместе с точкой. Вывод таблицы с информацией о популяции, включая номер хромосомы, результат, значения генов.

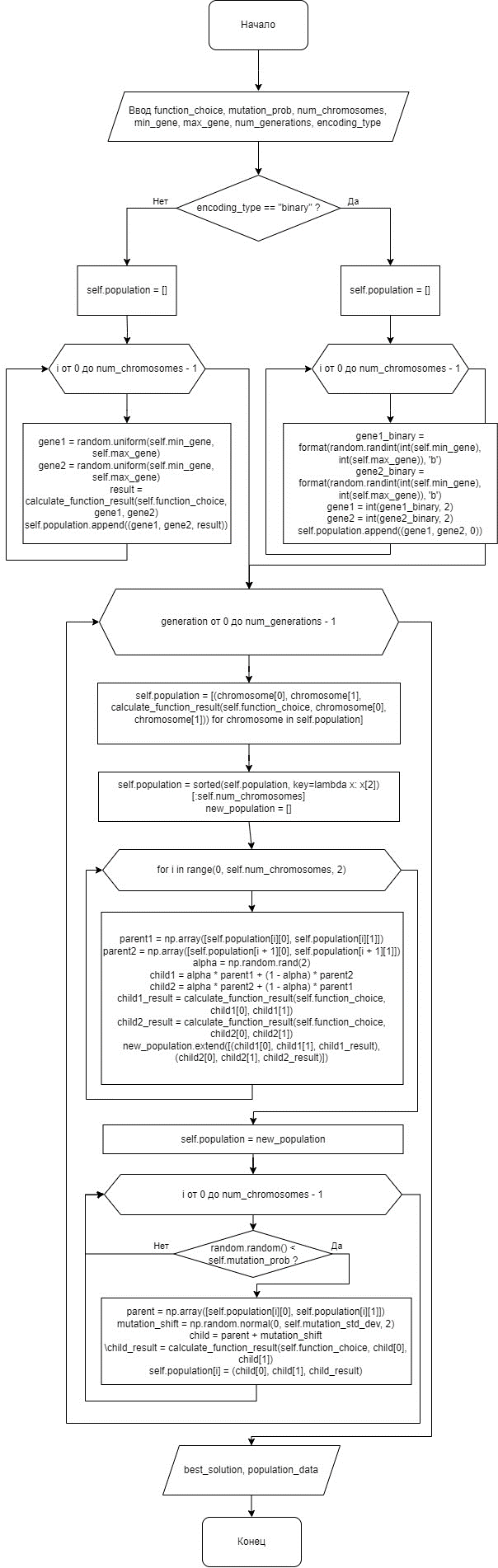


Рисунок 4.1. Блок-схема алгоритма.

# **Описание программы**

## **Описание классов**

В программе используется 2 класса: 1 связанный с интерфейсом программы, 1 связанный с генетическим алгоритмом. В таблице 5.1 представлено описание классов.

*Таблица 5.1. Описание классов*

|  |  |  |
| --- | --- | --- |
| Имя класса | Наследование | Описание |
| GeneticAlgorithm | — | Создание интерфейса |
| GeneticAlgorithmInterface | — | Поиск минимального значения функции с помощью генетического алгоритма |

## **5.2. Описание функций**

Описание функций класса *«GeneticAlgorithm»* представлено в таблице 5.2.

*Таблица 5.2. Описание функций класса «GeneticAlgorithm»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| Выбор функции, вероятность мутации, количество хромосом, минимальное значение гена, максимальное значение гена, количество поколений, тип кодировки | \_\_init\_\_ | Инициализация класса |
| — | run | Запуск генетического алгоритма |
| — | binary\_encoding | Бинарное кодирование хромосом |
| — | real\_encoding | Вещественное кодирование хромосом |  |
| — | initialize\_population | Инициализация начальной популяции с учетом выбора кодирования |  |
| — | calculate\_fitness | Расчет значения функции для каждой хромосомы в текущей популяции |  |
| — | selection | Сортировка популяции по значению функции и выбор лучших хромосом |  |
| — | crossover | Скрещивание пар хромосом для создания новых потомков |  |
| — | mutation | Мутация случайных хромосом в текущей популяции |  |
| — | get\_best\_solution | Получение лучшего решения из текущей популяции |  |
| — | get\_population | Получение информации о популяции для отображения в интерфейсе |  |

## **5.3. Описание переменных**

*Таблица 5.3. Описание переменных класса «GeneticAlgorithm»*

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Описание |
| str | function\_choice | Выбранная функция |
| float | mutation\_prob | Вероятность мутации в долях |
| int | num\_chromosomes | Количество хромосом в популяции |
| float | min\_gene | Минимальное значение гена |
| float | max\_gene | Максимальное значение гена |
| int | num\_generations | Количество поколений |
| float | mutation\_std\_dev | Новый параметр для стратегии эволюционных стратегий |
| List[] | population | Хранение текущей популяции хромосом |
| str | encoding\_type | Добавление параметра для хранения типа кодирования |

Полный код программы представлен в Приложении 1.

# **Рекомендации пользователю**

Поле *«Функция»* позволяет выбрать математическую функцию из выпадающего списка. Поля *«Вероятность мутации (%)»*, *«Количество хромосом»*, *«Минимальное значение гена»*, *«Максимальное значение гена»*, *«Количество поколений»* — позволяют настроить параметры генетического алгоритма. В поле *«Тип кодирования»* доступен выбор кодировок хромосом для алгоритма.

Кнопка *«Рассчитать»* отвечает за запуск работы программы. Далее лучшее решение и значение выводится в поле *«Результат»*, а в поле *«Таблица хромосом»* заполняется данными хромосом.

Для завершения работы нажмите на крестик в левом верхнем углу.

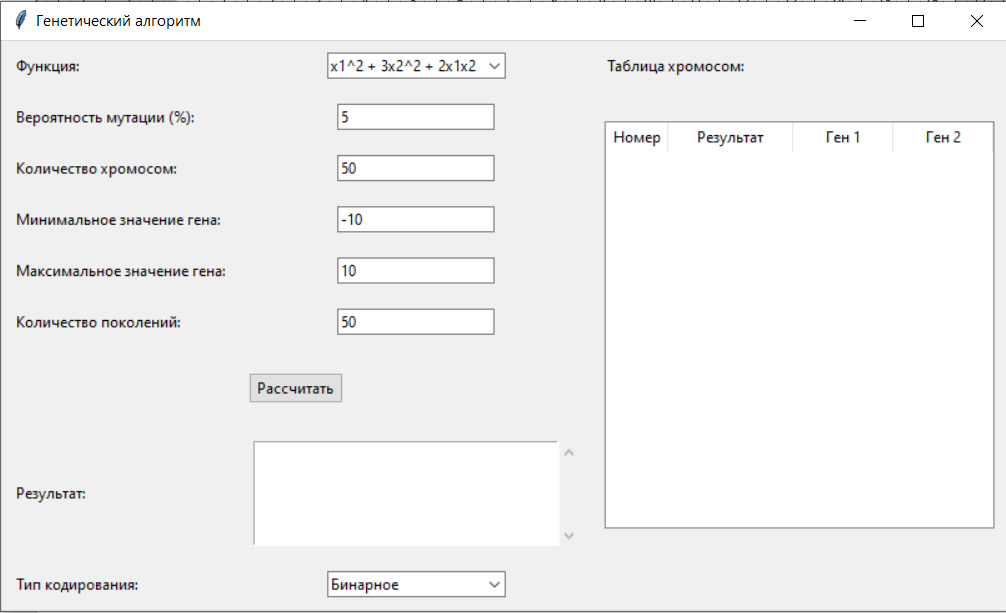
# **Рекомендации программисту**

Для запуска программы необходим Python версии не ниже 3.10.6, а также 64-битная операционная система Windows/Linux/macOS. Предварительно необходимо установить библиотеку numpy версии не ниже 1.24.2. Для работы с кодом необходим PyCharm версии не ниже 2022.2.1.

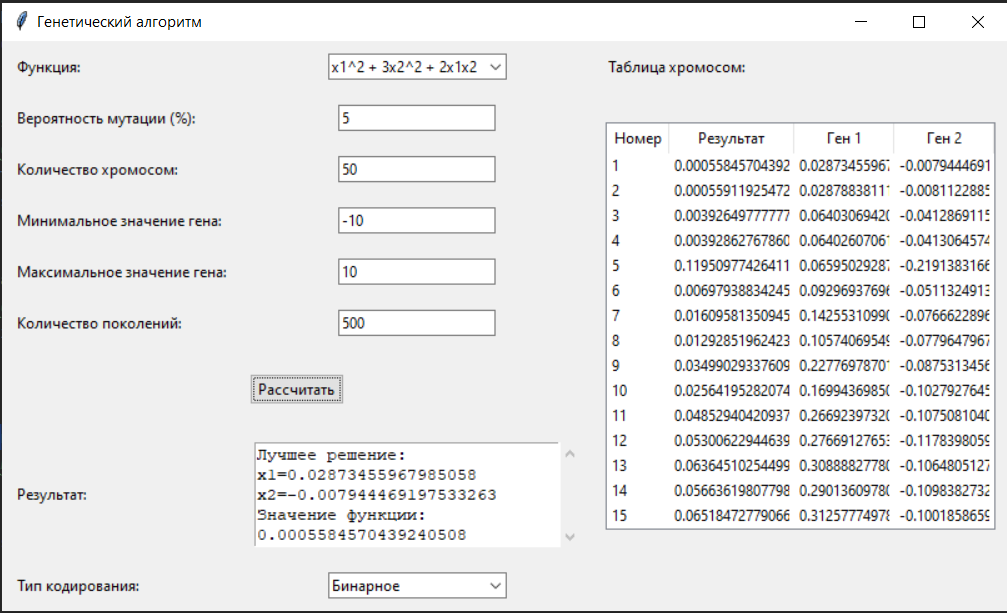
Минимальное необходимое место на диске: 0.5 МБ. Минимальное необходимое количество оперативной памяти: 100 МБ.

# **Контрольный пример**

В данном разделе представлен контрольный пример, демонстрирующий способность программы искать минимальное значение функции с помощью генетического алгоритма.



*Рисунок 8.1. Интерфейс программы*



*Рисунок 8.2. Результат работы программы*

# **Заключение**

Разработанная программа, основанная на генетическом алгоритме для оптимизации математических функций, продемонстрировала значительную эффективность в решении задач оптимизации. Результаты экспериментов на тестовой функции подтверждают успешное функционирование генетического алгоритма в достижении оптимальных значений.

Особое внимание стоит уделить внедрению двух типов кодирования хромосом — бинарного и вещественного. Эта дополнительная функциональность предоставляет пользователю возможность выбора подходящего типа кодировки в зависимости от характера задачи оптимизации, обеспечивая более гибкое и универсальное применение генетического алгоритма.

Одним из ключевых улучшений является внедрение стратегии эволюционных стратегий для скрещивания. Это улучшение способствует повышению разнообразия потомков, улучшая способность алгоритма к изучению пространства поиска решений.

Следует отметить, что достижение оптимальных результатов требует внимательной настройки параметров генетического алгоритма, что подчеркивает важность подбора параметров для конкретной задачи.

# **Приложения**

import random  
import numpy as np  
import tkinter as tk  
from tkinter import ttk, scrolledtext  
  
  
def calculate\_function\_result(function\_choice, x1, x2):  
 if function\_choice == "x1^2 + 3x2^2 + 2x1x2":  
 return x1 \*\* 2 + 3 \* x2 \*\* 2 + 2 \* x1 \* x2  
 elif function\_choice == "4(x1 - 5)^2+(x2 - 6)^2":  
 return 4 \* (x1 - 5) \*\* 2 + (x2 - 6) \*\* 2  
  
  
class GeneticAlgorithm:  
 *""" Класс, отвечающий за генетический алгоритм """* def \_\_init\_\_(self, function\_choice, mutation\_prob, num\_chromosomes, min\_gene, max\_gene, num\_generations, encoding\_type):  
 *""" Инициализация параметров генетического алгоритма """* self.function\_choice = function\_choice # Выбранная функция  
 self.mutation\_prob = mutation\_prob / 100.0 # Вероятность мутации (перевод в доли)  
 self.num\_chromosomes = num\_chromosomes # Количество хромосом в популяции  
 self.min\_gene = min\_gene # Минимальное значение гена  
 self.max\_gene = max\_gene # Максимальное значение гена  
 self.num\_generations = num\_generations # Количество поколений  
 self.mutation\_std\_dev = 0.2 # Новый параметр для стратегии эволюционных стратегий  
 self.population = [] # Хранение текущей популяции хромосом  
 self.encoding\_type = encoding\_type # Добавление параметра для хранения типа кодирования  
  
 def run(self):  
 *""" Запуск генетического алгоритма """* self.initialize\_population()  
  
 for generation in range(self.num\_generations):  
 self.calculate\_fitness()  
 self.selection()  
 self.crossover()  
 self.mutation()  
  
 def binary\_encoding(self):  
 *""" Бинарное кодирование хромосом """* self.population = []  
 for i in range(self.num\_chromosomes):  
 gene1\_binary = format(random.randint(int(self.min\_gene), int(self.max\_gene)), 'b')  
 gene2\_binary = format(random.randint(int(self.min\_gene), int(self.max\_gene)), 'b')  
 gene1 = int(gene1\_binary, 2)  
 gene2 = int(gene2\_binary, 2)  
 self.population.append((gene1, gene2, 0)) # 0 - временное значение результата  
  
 def real\_encoding(self):  
 *""" Вещественное кодирование хромосом """* self.population = []  
 for i in range(self.num\_chromosomes):  
 gene1 = random.uniform(self.min\_gene, self.max\_gene)  
 gene2 = random.uniform(self.min\_gene, self.max\_gene)  
 result = calculate\_function\_result(self.function\_choice, gene1, gene2)  
 self.population.append((gene1, gene2, result))  
  
 def initialize\_population(self):  
 *""" Инициализация начальной популяции с учетом выбора кодирования """* if self.encoding\_type == "binary":  
 self.binary\_encoding()  
 elif self.encoding\_type == "real":  
 self.real\_encoding()  
  
 def calculate\_fitness(self):  
 *""" Расчет значения функции для каждой хромосомы в текущей популяции """* for i, chromosome in enumerate(self.population):  
 result = calculate\_function\_result(self.function\_choice, chromosome[0], chromosome[1])  
 self.population[i] = (chromosome[0], chromosome[1], result)  
  
 def selection(self):  
 *""" Сортировка популяции по значению функции и выбор лучших хромосом """* self.population = sorted(self.population, key=lambda x: x[2])[:self.num\_chromosomes]  
  
 def crossover(self):  
 *""" Скрещивание пар хромосом для создания новых потомков """* new\_population = []  
 for i in range(0, self.num\_chromosomes, 2):  
 parent1 = np.array([self.population[i][0], self.population[i][1]])  
 parent2 = np.array([self.population[i + 1][0], self.population[i + 1][1]])  
  
 """ Используем стратегию эволюционных стратегий для скрещивания """  
 alpha = np.random.rand(2)  
 child1 = alpha \* parent1 + (1 - alpha) \* parent2  
 child2 = alpha \* parent2 + (1 - alpha) \* parent1  
  
 child1\_result = calculate\_function\_result(self.function\_choice, child1[0], child1[1])  
 child2\_result = calculate\_function\_result(self.function\_choice, child2[0], child2[1])  
  
 new\_population.extend([(child1[0], child1[1], child1\_result), (child2[0], child2[1], child2\_result)])  
  
 self.population = new\_population  
  
 def mutation(self):  
 *""" Мутация случайных хромосом в текущей популяции """* for i in range(self.num\_chromosomes):  
 if random.random() < self.mutation\_prob:  
 parent = np.array([self.population[i][0], self.population[i][1]])  
  
 """ Используем стратегию эволюционных стратегий для мутации """  
 mutation\_shift = np.random.normal(0, self.mutation\_std\_dev, 2)  
 child = parent + mutation\_shift  
  
 child\_result = calculate\_function\_result(self.function\_choice, child[0], child[1])  
 self.population[i] = (child[0], child[1], child\_result)  
  
 def get\_best\_solution(self):  
 *""" Получение лучшего решения из текущей популяции """* best\_solution = min(self.population, key=lambda x: x[2])  
 return best\_solution[0], best\_solution[1], best\_solution[2]  
  
 def get\_population(self):  
 *""" Получение информации о популяции для отображения в интерфейсе """* return [(i + 1, chromosome[2], chromosome[0], chromosome[1]) for i, chromosome in enumerate(self.population)]  
  
  
class GeneticAlgorithmInterface:  
 *""" Класс, отвечающий за интерфейс на библиотеке tkinter """* def \_\_init\_\_(self, root: tk.Tk):  
 self.root = root  
 self.root.title("Генетический алгоритм")  
  
 """ Добавление элементов интерфейса (выбор функции, вероятность мутации и т.д.) """  
 self.function\_label = ttk.Label(self.root, text="Функция:")  
 self.function\_combobox = ttk.Combobox(self.root, values=["x1^2 + 3x2^2 + 2x1x2", "4(x1 - 5)^2+(x2 - 6)^2"], state="readonly")  
 self.mutation\_label = ttk.Label(self.root, text="Вероятность мутации (%):")  
 self.mutation\_entry = ttk.Entry(self.root)  
 self.mutation\_entry.insert(0, "5")  
 self.num\_chromosomes\_label = ttk.Label(self.root, text="Количество хромосом:")  
 self.num\_chromosomes\_entry = ttk.Entry(self.root)  
 self.num\_chromosomes\_entry.insert(0, "50")  
 self.min\_gene\_label = ttk.Label(self.root, text="Минимальное значение гена:")  
 self.min\_gene\_entry = ttk.Entry(self.root)  
 self.min\_gene\_entry.insert(0, "-10")  
 self.max\_gene\_label = ttk.Label(self.root, text="Максимальное значение гена:")  
 self.max\_gene\_entry = ttk.Entry(self.root)  
 self.max\_gene\_entry.insert(0, "10")  
 self.num\_generations\_label = ttk.Label(self.root, text="Количество поколений:")  
 self.num\_generations\_entry = ttk.Entry(self.root)  
 self.num\_generations\_entry.insert(0, "50")  
 self.calculate\_button = ttk.Button(self.root, text="Рассчитать", command=self.run\_genetic\_algorithm)  
 self.result\_label = ttk.Label(self.root, text="Результат:")  
 self.result\_text = scrolledtext.ScrolledText(self.root, width=30, height=5)  
 self.result\_text.config(state=tk.DISABLED)  
 self.encoding\_label = ttk.Label(self.root, text="Тип кодирования:")  
 self.encoding\_combobox = ttk.Combobox(self.root, values=["Бинарное", "Вещественное"], state="readonly")  
 self.encoding\_combobox.current(0)  
  
 self.table\_label = ttk.Label(self.root, text="Таблица хромосом:")  
 self.tree = ttk.Treeview(self.root, columns=('Number', 'Result', 'Gene1', 'Gene2'), show='headings', height=15)  
 self.tree.heading('Number', text='Номер')  
 self.tree.heading('Result', text='Результат')  
 self.tree.heading('Gene1', text='Ген 1')  
 self.tree.heading('Gene2', text='Ген 2')  
 self.tree.column('Number', width=50)  
 self.tree.column('Result', width=100)  
 self.tree.column('Gene1', width=80)  
 self.tree.column('Gene2', width=80)  
  
 """ Размещение элементов в окне """  
 self.function\_label.grid(row=0, column=0, padx=10, pady=10, sticky="w")  
 self.function\_combobox.grid(row=0, column=1, padx=10, pady=10)  
 self.mutation\_label.grid(row=1, column=0, padx=10, pady=10, sticky="w")  
 self.mutation\_entry.grid(row=1, column=1, padx=10, pady=10)  
 self.num\_chromosomes\_label.grid(row=2, column=0, padx=10, pady=10, sticky="w")  
 self.num\_chromosomes\_entry.grid(row=2, column=1, padx=10, pady=10)  
 self.min\_gene\_label.grid(row=3, column=0, padx=10, pady=10, sticky="w")  
 self.min\_gene\_entry.grid(row=3, column=1, padx=10, pady=10)  
 self.max\_gene\_label.grid(row=4, column=0, padx=10, pady=10, sticky="w")  
 self.max\_gene\_entry.grid(row=4, column=1, padx=10, pady=10)  
 self.num\_generations\_label.grid(row=5, column=0, padx=10, pady=10, sticky="w")  
 self.num\_generations\_entry.grid(row=5, column=1, padx=10, pady=10)  
 self.calculate\_button.grid(row=6, column=0, columnspan=2, pady=20)  
 self.result\_label.grid(row=7, column=0, padx=10, pady=10, sticky="w")  
 self.result\_text.grid(row=7, column=1, padx=10, pady=10, columnspan=1)  
 self.table\_label.grid(row=0, column=2, padx=10, pady=10, sticky="w")  
 self.tree.grid(row=0, column=2, padx=10, pady=10, columnspan=2, rowspan=9)  
 self.encoding\_label.grid(row=8, column=0, padx=10, pady=10, sticky="w")  
 self.encoding\_combobox.grid(row=8, column=1, padx=10, pady=10)  
  
 def run\_genetic\_algorithm(self):  
 *""" Получение параметров из интерфейса """* function\_choice = self.function\_combobox.get()  
 mutation\_prob = float(self.mutation\_entry.get())  
 num\_chromosomes = int(self.num\_chromosomes\_entry.get())  
 min\_gene = float(self.min\_gene\_entry.get())  
 max\_gene = float(self.max\_gene\_entry.get())  
 num\_generations = int(self.num\_generations\_entry.get())  
 encoding\_type = "binary" if self.encoding\_combobox.get() == "Бинарное" else "real"  
  
 """ Создание экземпляра генетического алгоритма и запуск """  
 genetic\_algorithm = GeneticAlgorithm(function\_choice, mutation\_prob, num\_chromosomes, min\_gene, max\_gene, num\_generations, encoding\_type)  
 genetic\_algorithm.run()  
  
 """ Получение и вывод результатов """  
 best\_solution = genetic\_algorithm.get\_best\_solution()  
 population\_data = genetic\_algorithm.get\_population()  
  
 """ Отображение результатов в таблице """  
 self.fill\_table(population\_data)  
  
 """ Отображение результатов в виде x1 и x2 """  
 self.result\_text.config(state=tk.NORMAL)  
 self.result\_text.delete(1.0, tk.END)  
 self.result\_text.insert(tk.END, f"Лучшее решение:\nx1={best\_solution[0]}\nx2={best\_solution[1]}\n")  
 self.result\_text.insert(tk.END, f"Значение функции:\n{best\_solution[2]}")  
 self.result\_text.config(state=tk.DISABLED)  
  
 def fill\_table(self, results):  
 *""" Очистка текущих данных в таблице """* for item in self.tree.get\_children():  
 self.tree.delete(item)  
  
 """ Заполнение таблицы новыми данными """  
 for result in results:  
 self.tree.insert('', tk.END, values=result)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 window = tk.Tk()  
 GeneticAlgorithmInterface(window)  
 window.mainloop()

*Приложение 1. Листинг программы.*