**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики - процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №5**

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

**на тему «Роевой алгоритм»**

**Вариант – 9**

**Студент гр. 23Б15-пу**

**Кубякин Н.А.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

***Оглавление***

[1. Цель работы 3](#_Toc178797423)

[2. Описание задачи (формализация задачи) 3](#_Toc178797424)

[3. Теоретическая часть 3](#_Toc178797431)

[4. Основные шаги программы 5](#_Toc178797432)

[5. Блок-схема программы 6](#_Toc178797433)

[6. Описание программы 7](#_Toc178797442)

[7. Рекомендации пользователя 7](#_Toc178797443)

[8. Рекомендации программиста 8](#_Toc178797444)

[9. Исходный код программы 8](#_Toc178797445)

[10. Контрольный пример 8](#_Toc178797447)

[11. Вывод 12](#_Toc178797448)

[12. Источники 12](#_Toc178797448)

[13. Листинг 13](#_Toc178797450)

# Цель работы

Исследование особенностей алгоритмов роевого интеллекта для решения задач глобальной оптимизации и сравнение с генетическим алгоритмом.

# Описание задачи (формализация задачи)

*Цель работы* – исследование особенностей алгоритмов роевого интеллекта для решения задач глобальной оптимизации и сравнение с генетическим алгоритмом.

Порядок выполнения лабораторной работы:

1. Изучить особенности кодирования роевого интеллекта.
2. Написать программу поиска минимума функции, выбрав вариант тестовой функции .
3. Протестировать программу на выбранной тестовой функции.

# Теоретическая часть

Метод роя частиц (PSO) является оптимизационным алгоритмом. В рамках PSO, каждая "частица" в пространстве поиска представляет собой потенциальное решение задачи оптимизации. Частицы перемещаются по пространству с целью минимизации (или максимизации) функции приспособленности.

Ниже приведен список основных понятий:

* **Частица (Particle):** Потенциальное решение задачи оптимизации, представленное в пространстве параметров.
* **Функция приспособленности (Fitness Function):** Оценочная функция, измеряющая качество решения в данной точке пространства параметров.
* **Положение частицы (Position):** Текущая координата частицы в пространстве параметров.
* **Лучшее личное положение (pbest):** Запоминает лучшую позицию, на которой данная частица достигла наилучшей приспособленности.
* **Лучшее глобальное положение (gbest):** Запоминает лучшую позицию среди всех частиц в рое.
* **Скорость частицы (Velocity):** Вектор, определяющий изменение положения частицы на каждом шаге оптимизации.
* **Коэффициенты ускорения (Acceleration Coefficients):** Параметры, определяющие влияние лучшего личного и глобального положения (также известны как когнитивный и социальный коэффициенты) на обновление скорости частицы.
* **Итерации (Generations):** Итерации оптимизационного процесса, на каждой из которых происходит обновление положения частиц.
* **Нормальное распределение (Normal Distribution, N(.)):** Распределение вероятностей, описывающее случайные величины, которые сгруппированы вокруг среднего значения.
* **Адаптивность (Adaptability):** Способность алгоритма или параметров алгоритма изменяться в зависимости от условий задачи.

Основные шаги алгоритма:

1. **Инициализация:** 
   * Инициализация частиц с случайными положениями и лучшими личными положениями.
2. **Основной цикл оптимизации:**
   * Вычисление значений функции приспособленности в текущем положении и обновление глобального лучшего положения
   * Обновление скорости частицы в соответствии с лучшим глобальным и лучшем личным положением, а также коэффициентами ускорения.
   * Обновление положения частицы в соответствии со скоростью
3. **Повторение:**
   * Повторение шага 2 заданное количество раз (генераций).
4. **Вывод результатов:**
   * Вывод результатов

# В программе реализована модификация коэффициента сжатия. Модификация подразумевает нормировку весовых коэффициентов с целью снижения зависимости сходимости от значений этих коэффициентов.

# Необходимые формулы:

# https://jenyay.net/uploads/Programming/ParticleSwarm/eqn_8.png

# https://jenyay.net/uploads/Programming/ParticleSwarm/eqn_1.png

# https://jenyay.net/uploads/Programming/ParticleSwarm/eqn_9.png

# https://jenyay.net/uploads/Programming/ParticleSwarm/eqn_10.png

# Здесь:

# vi, t – i-я компонента скорости при t-ой итерации алгоритма xi, t – i-я координата частицы при t-ой итерации алгоритма pi – i-я координата лучшего решения, найденного частицей gi – i-я координата лучшего решения, найденного всеми частицами rp, rg – случайные числа в интервале (0, 1) φp, φg – весовые коэффициенты

# Основные шаги программы

1. Инициализация частиц со случайными положениями, скоростями и лучшими личными положениями.
2. Вычисление значений функции приспособленности в текущем положении и обновление глобального лучшего положения.
3. Обновление скорости частицы в соответствии с лучшим глобальным и лучшем личным положением, а также коэффициентами ускорения.
4. Обновление положения частицы в соответствии со скоростью.
5. Повтор 2-4 заданное количество раз.
6. Вывод результатов в графическом интерфейсе.

# Блок-схема программы

# На рисунке представлена блок-схема алгоритма.

# 

# Рис. 5 Блок-схема генетического алгоритма.

# Описание программы

# Алгоритм реализован на языке Python 3.10 с использованием библиотеки PyQT для визуализации и некоторых встроенных библиотек. Вся логика программы заключена в 1 классе и 3 фуннкциях:

Таблица 6.1 Описание классов

|  |  |
| --- | --- |
| Имя класса | Назначение |
| Particle | TypedDict, представляющий структуру данных для частиц. |

Таблица 6.2 Описание функций

|  |  |
| --- | --- |
| Имя функции | Назначение |
| initialize\_particles | Инициализация частиц. |
| func | Вычисление значения целевой функции при данных значениях переменных. |
| algorithm | Реализация роевого алгоритма. |

# Рекомендации пользователя

1. Запустить main.py для запуска графического интерфейса программы.
2. Ввести в поле нужную функцию (заранее введена функция , умножение - \*, возведение в степень - \*\*)
3. Выбрать в полях соответствующие величины.
4. Поставить галочки по желанию для выбора отсутствия или присутствия модификации коэффициента сжатия.
5. Нажать кнопку “Сгенерировать рой”.
6. Нажать кнопку “Рассчитать”. В поле под кнопкой отобразится лучшее значение и соответствующие значения переменных. В большом поле снизу отобразится таблица значений частиц и график соответствующих точек.

# Рекомендации программиста

Для запуска программы необходима 64-битная операционная система Windows, Linux или macOS. Для работы с кодом необходима среда разработки, совместимая с python 3.1 и библиотеки random, struct [[1]](#struct), copy [[2]](#copy), typing [[3]](#typing), PyQt5 [[4]](#pyqt5).

Для запуска программы необходимо установить Python версии не ниже 3.10, а так же библиотеки random, struct [[1]](#struct), copy [[2]](#copy), typing [[3]](#typing), PyQt5 [[4]](#pyqt5).

# Исходный код программы

# Исходный код программы и необходимые текстовые файлы доступны по ссылке: <https://github.com/NikiTaku1/spbu_alg/tree/main/lab5>

# Контрольный пример

В данном разделе представлен пример, демонстрирующий работу программы.

1) Запуск main.py (заранее введены контрольные значения).

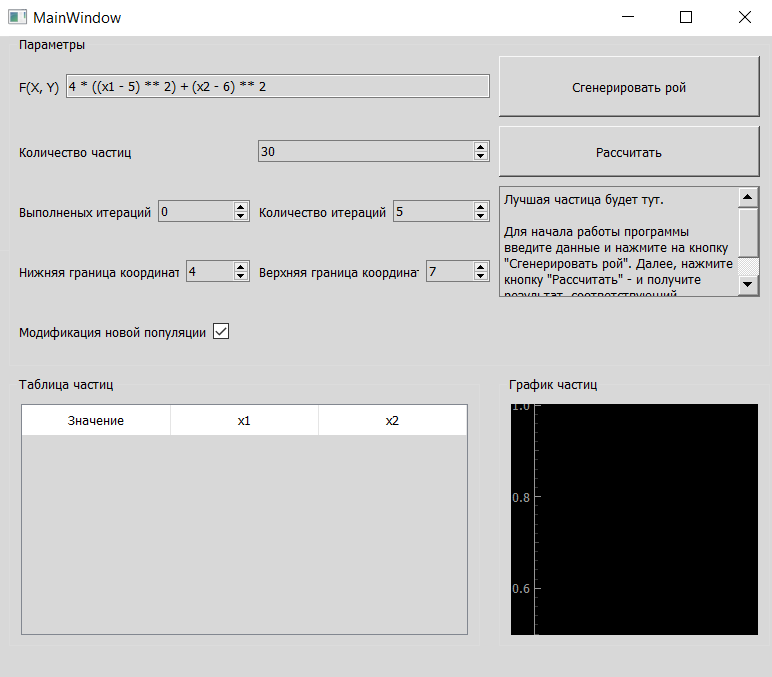


Рис 10.1 Графический интерфейс программы.

2) Нажатие кнопок “Рассчитать хромосомы” и “Рассчитать”

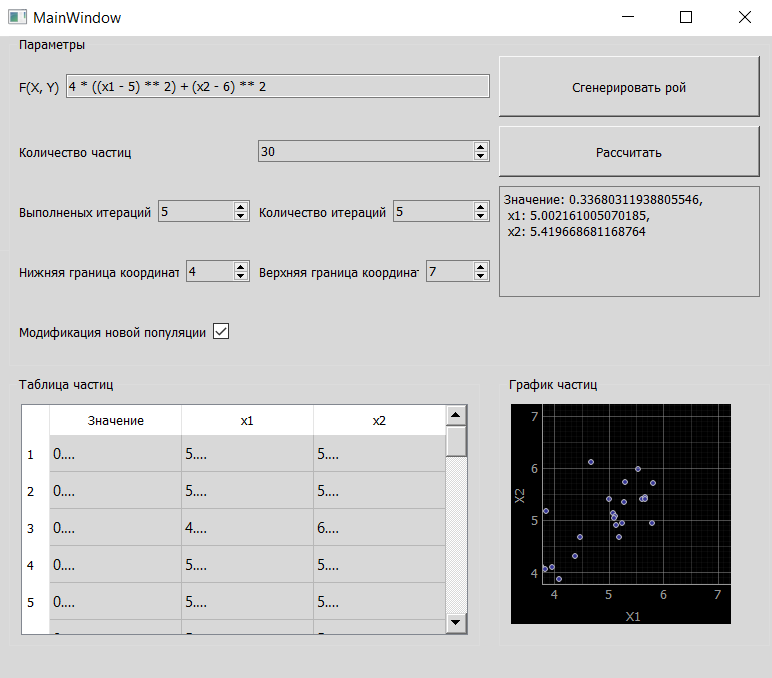


Рис 10.2 Графический интерфейс после нажатия кнопок.

3) Анализ точности вычислений в зависимости от параметров (10.3 - N частиц = 30; 10.4 – N итераций = 50)

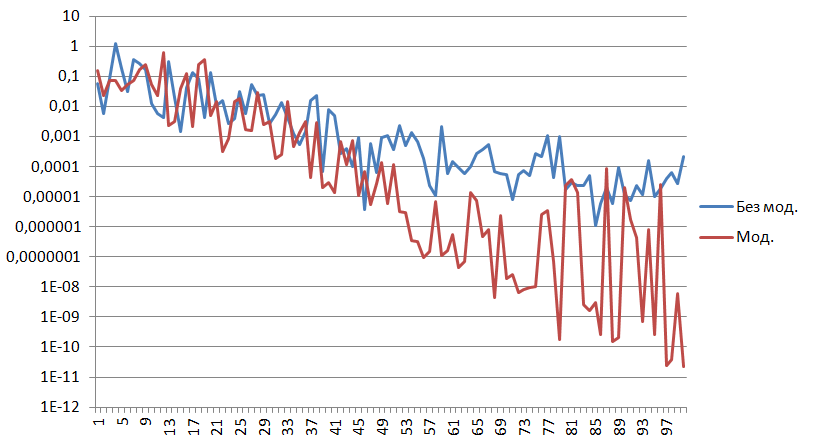


Рис 10.3 График зависимости точности от количества итераций

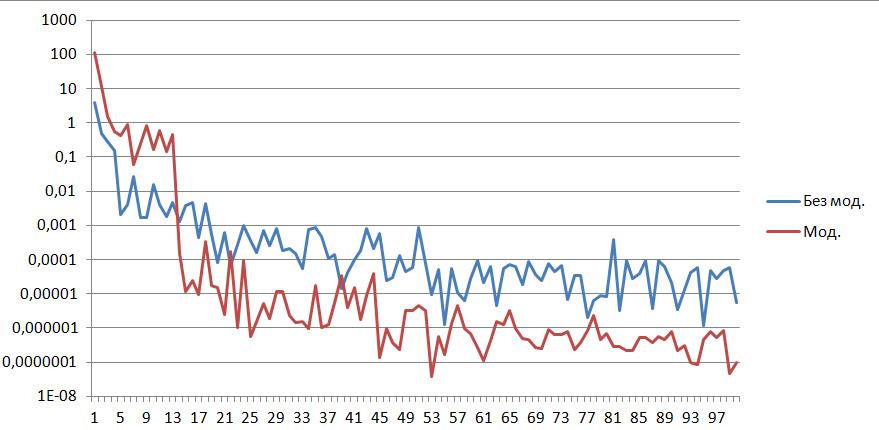


Рис 10.4 График зависимости точности от количества частиц

4) Сравнение точности ГА и РА в зависимости от параметров (10.5 – N генов/частиц = 30; 10.6 – N итераций = 50, на обоих графиках промежуток значений переменных [4; 7])

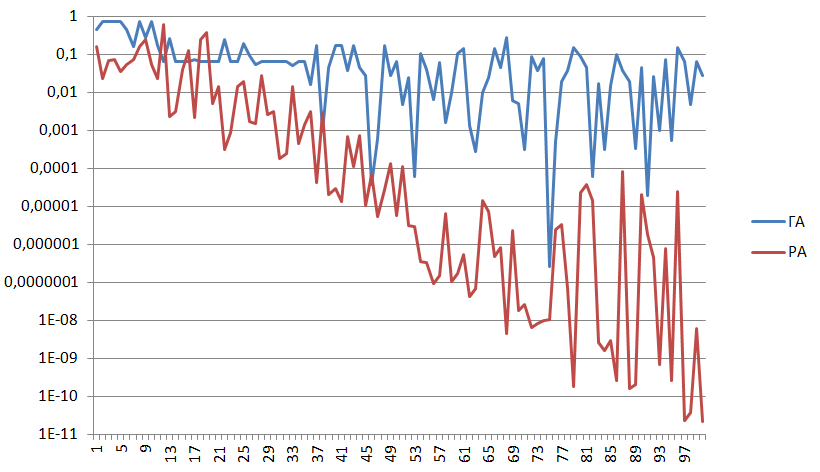


Рис 10.5 Сравнение точности ГА и РА в зависимости от количества итераций

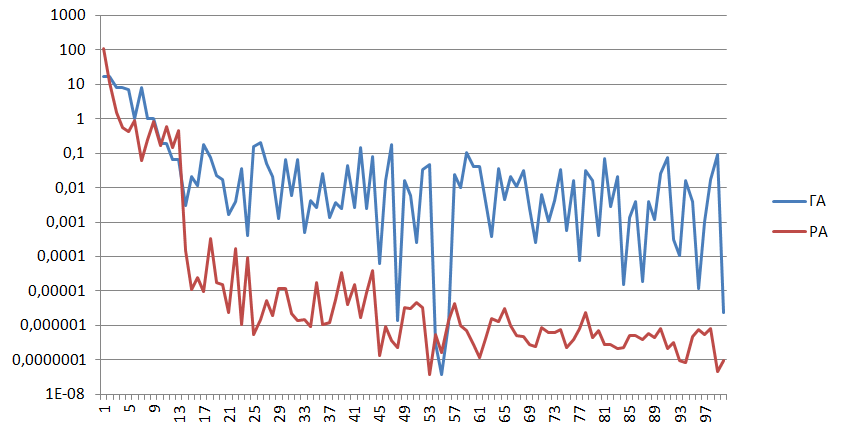


Рис 10.6 Сравнение точности ГА и РА в зависимости от количества генов/частиц

Сравнение показывает более высокую точность роевого алгоритма по сравнению с генетическим при одинаковых количествах итераций и генов/частиц, однако в некоторых случаях генетический алгоритм может сравниться или превзойти точность роевого, но в среднем роевой алгоритм показывает себя лучше ввиду итеративной реализации. Так же можно заметить более высокую стабильность роевого алгоритма ввиду меньшего разброса на графиках. Таким образом, в большинстве случаев роевой алгоритм даст более точные и стабильные результаты, хотя в некоторых задачах генетический алгоритм может превзойти роевой.

# Вывод

Были получены все нужные навыки для реализации роевых алгоритмов, а также разработана программа, реализующая роевой алгоритм с графическим интерфейсом. Проанализированы результаты работы алгоритма, выполнено сравнение параметров. Проведено сравнение точности генетического и роевого алгоритма на одинаковых или равнозначных данных.

# Источники

# struct — struct documentation // Documentation URL: <https://docs.python.org/3/library/struct.html> (дата обращения: 14.11.2024).

# copy — copy documentation // Documentation URL: <https://docs.python.org/3/library/copy.html> (дата обращения: 14.11.2024).

# typing — typing documentation // Documentation URL: <https://docs.python.org/3/library/typing.html> (дата обращения: 14.11.2024).

# PyQt5 — PyQt5 documentation // Documentation URL: <https://doc.qt.io/qtforpython-5/contents.html> (дата обращения: 14.11.2024).

# Листинг

# 1) swarm\_algorithm.py

import random

from copy import deepcopy

from typing import TypedDict

class Particle(TypedDict):

"""  
 Класс, задающий параметры частицы.  
  
 Параметры:  
 - x1: значение первой переменной.  
 - x2: значение второй переменной.

- v1: значение скорости первой координаты.

- v2: значение скорости второй координаты.

- lbest\_x1: лучшее значение первой переменной.

- lbest\_x2: лучшее значение второй переменной.  
  
 Вывод:  
 - Список частиц в формате TypedDict.  
 """

    x1: float

    x2: float

    v1: float

    v2: float

    lbest\_x1: float

    lbest\_x2: float

# Инициализация частиц со случайными значениями в указанных пределах

def initialize\_particles(bounds, n):

"""  
 Инициализация частиц со случайными значениями.  
  
 Параметры:  
 - bounds: Граничные значения.  
 - amount: Количество частиц.  
  
 Вывод:  
 - Список частиц в формате TypedDict.  
 """

    return [

        Particle(

            x1=random.uniform(bounds[0], bounds[1]),

            x2=random.uniform(bounds[0], bounds[1]),

            v1=random.uniform(0, 1),

            v2=random.uniform(0, 1),

            lbest\_x1=random.uniform(0, 1),

            lbest\_x2=random.uniform(0, 1),

        )

        for \_ in range(n)

    ]

# Вызов целевой функции

def func(fitness\_func, x1, x2):

"""  
 Вычисляет значение функции при данных значениях переменной.  
  
 Parameters:  
 - f: Строка, содержащая целевую функцию.  
 - x1, x2: Значения переменных.  
  
 Вывод:  
 - Значение функции.  
 """

    return eval(fitness\_func)

# Реализация алгоритма оптимизации на основе PSO

def algorithm(flag: bool, particles, generations, fitness\_func):

"""  
 Выполнение роевого алгоритма.  
  
 Параметры:

- flag: Флаг на включение модификации коэффициента сжатия.  
 - particles: Основной набор частиц.  
 - generations: Количество итераций.  
 - f: Строка, содержащая целевую функцию.  
 - bounds: Границы значений переменных.  
  
 Вывод:  
 - История значений частиц.  
 """

    history = [

        deepcopy(sorted(particles, key=lambda x: func(fitness\_func, x["x1"], x["x2"])))

    ]

    for \_ in range(generations):

        gbest = (history[-1][0]["x1"], history[-1][0]["x2"])

        for particle in particles:

            f\_was = func(fitness\_func, particle["x1"], particle["x2"])

            if flag == True:

                k = 0.1 #от 0 до 1

                rp, rg = random.uniform(0, 1), random.uniform(0, 1)

                fp, fg = 5, 5

                f = fp\*rp + fg\*rg # должен быть больше 4

                norm = (2\*k)/(abs(2 - f - ((f\*\*2) - (4\*f))\*\*(0.5)))

                particle["v1"] = norm \* (particle["v1"] + fp \* rp \*(particle["lbest\_x1"] - particle["x1"]) + fg \* rg \* (gbest[0] - particle["x1"]))

                particle["v2"] = norm \* (particle["v2"] + fp \* rp \*(particle["lbest\_x2"] - particle["x2"]) + fg \* rg \* (gbest[1] - particle["x2"]))

            else:

                rp, rg = random.uniform(0, 1), random.uniform(0, 1)

                fp, fg = 0.5, 0.5

                particle["v1"] = (particle["v1"] + fp \* rp \*(particle["lbest\_x1"] - particle["x1"]) + fg \* rg \* (gbest[0] - particle["x1"]))

                particle["v2"] = (particle["v2"] + fp \* rp \*(particle["lbest\_x2"] - particle["x2"]) + fg \* rg \* (gbest[1] - particle["x2"]))

            particle["x1"] += particle["v1"]

            particle["x2"] += particle["v2"]

            f\_now = func(fitness\_func, particle["x1"], particle["x2"])

            if f\_now < f\_was:

                particle["lbest\_x1"] = particle["x1"]

                particle["lbest\_x2"] = particle["x2"]

        particles = sorted(

            particles, key=lambda x: func(fitness\_func, x["x1"], x["x2"])

        )

        history.append(deepcopy(particles))

    return history

# 2) main.py

from PyQt5 import QtWidgets, uic

from swarm\_algorithm import initialize\_particles, algorithm, func

from copy import deepcopy

import pyqtgraph as pg

class GeneticAlgorithmGUI(QtWidgets.QMainWindow):

    def \_\_init\_\_(self):

        super(GeneticAlgorithmGUI, self).\_\_init\_\_()

        uic.loadUi("ui.xml", self)

        self.setStyleSheet(

            """

            background-color: #D8D8D8;

            color: #000000;

            QPushButton {

                background-color: #D8D8D8;

                color: #000000;

            }

            QPushButton:hover {

                background-color: #D8D8D8;

            }

            """

        )

        # Подключение функций к кнопкам

        self.pushButton.clicked.connect(self.calculate\_particles)

        self.pushButton\_2.clicked.connect(self.calculate)

        self.tableWidget.setColumnCount(3)

        self.tableWidget.setHorizontalHeaderLabels(["Значение", "x1", "x2"])

        self.tableWidget.horizontalHeader().setSectionResizeMode(

            QtWidgets.QHeaderView.Stretch

        )

        self.particles = []

        self.funcQLE.setText("4 \* ((x1 - 5) \*\* 2) + (x2 - 6) \*\* 2")

        self.spinBox.setMinimum(0)

        self.spinBox.setMaximum(10000)

        self.spinBox\_4.setMinimum(0)

        self.spinBox\_4.setMaximum(10000)

        self.spinBox\_5.setMinimum(0)

        self.spinBox\_5.setMaximum(100000)

        self.spinBox\_6.setMinimum(-10000)

        self.spinBox\_6.setMaximum(10000)

        self.spinBox\_7.setMinimum(-10000)

        self.spinBox\_7.setMaximum(10000)

        self.spinBox.setValue(30)

        self.spinBox\_4.setValue(0)

        self.spinBox\_5.setValue(1)

        self.spinBox\_6.setValue(4)

        self.spinBox\_7.setValue(7)

        self.checkBox\_2.setChecked(True)

        self.plotWidget = pg.PlotWidget()

        self.plotWidget.setParent(self.findChild(QtWidgets.QWidget, "plotWidget"))

        self.plotWidget.show()

    def calculate\_particles(self):

"""  
 Вычисление роя частиц.  
 """

        n = self.spinBox.value()

        lower\_bound = self.spinBox\_6.value()

        upper\_bound = self.spinBox\_7.value()

        area = (lower\_bound, upper\_bound)

        self.particles = initialize\_particles(area, n)

    def calculate(self):

"""  
 Выполнение алгоритма и вывод в интерфейс.  
 """

        if not self.particles:

            self.calculate\_particles()

        func\_str = self.funcQLE.text()

        previous\_iterations = self.spinBox\_4.value()

        total\_iterations = self.spinBox\_5.value()

        shown\_iterations = previous\_iterations + total\_iterations

        previous\_iterations = max(previous\_iterations, total\_iterations)

        self.spinBox\_4.setValue(shown\_iterations)

        if self.checkBox\_2.isChecked():

            result = algorithm(

                True, deepcopy(self.particles), shown\_iterations, func\_str

            )

        else:

            result = algorithm(

                False, deepcopy(self.particles), shown\_iterations, func\_str

            )

        self.display\_particles\_in\_table(result[-1])

        self.display\_best\_val(result[-1])

        self.particles = result[-1]

    def display\_particles\_in\_table(self, contents):

"""  
 Вывод значений в таблицу.  
 """

        self.tableWidget.setRowCount(0)

        lower\_bound = self.spinBox\_6.value()

        upper\_bound = self.spinBox\_7.value()

        self.plotWidget.clear()

        self.plotWidget.resize(220, 220)

        x\_values = []

        y\_values = []

        z\_values = []

        for \_, gene in enumerate(contents):

            func\_str = self.funcQLE.text()

            position = (gene["x1"], gene["x2"])

            value = func(func\_str, \*position)

            # Добавляем новую строку в таблицу

            row\_position = self.tableWidget.rowCount()

            self.tableWidget.insertRow(row\_position)

            # Заполняем ячейки таблицы

            self.tableWidget.setItem(

                row\_position, 1, QtWidgets.QTableWidgetItem(str(position[0]))

            )

            self.tableWidget.setItem(

                row\_position, 2, QtWidgets.QTableWidgetItem(str(position[1]))

            )

            self.tableWidget.setItem(

                row\_position, 0, QtWidgets.QTableWidgetItem(str(value))

            )

            x\_values.append(position[0])

            y\_values.append(position[1])

            z\_values.append(value)

        self.plotWidget.plot(x\_values, y\_values, pen=None, symbol='o', symbolSize=5)

        self.plotWidget.showGrid(x = True, y = True)

        self.plotWidget.addLegend()

        self.plotWidget.setXRange(lower\_bound, upper\_bound)

        self.plotWidget.setYRange(lower\_bound, upper\_bound)

        self.plotWidget.setLabel('left', 'X2')

        self.plotWidget.setLabel('bottom', 'X1')

        self.plotWidget.show()

    def display\_best\_val(self, genes):

"""  
 Вывод лучшего значения.  
 """

        best\_gene = (genes[0]["x1"], genes[0]["x2"])

        func\_str = self.funcQLE.text()

        value = func(func\_str, \*best\_gene)

        result\_text = f"Значение: {value},\n x1: {best\_gene[0]},\n x2: {best\_gene[1]}"

        self.plainTextEdit.setPlainText(result\_text)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QtWidgets.QApplication([])

    window = GeneticAlgorithmGUI()

    window.show()

    app.exec\_()