**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**Факультет прикладной математики – процессов управления**

**отчет**

**по лабораторной работе**

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

**на тему «Разработка и реализация алгоритма роевого интеллекта для решения задач глобальной оптимизации»**

**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 9](#_Toc152641683)

[7. Рекомендации программисту 9](#_Toc152641684)

[8. Контрольный пример 10](#_Toc152641685)

[9. Заключение 12](#_Toc152641686)

[10. Приложения 12](#_Toc152641687)

# **Цель работы**

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

# **Задача**

1) Изучить теоретические основы в области роевого интеллекта, методов его оптимизаций.

2) Написать программу поиска минимума функции.

3) Протестировать программу на выбранной тестовой функции.

4) Проанализировать работу алгоритма в зависимости от методов оптимизаций, параметров: коэффициента текущей скорости, коэффициента собственного лучшего значения и др.

5) Сравнить результат работы алгоритма с результатом работы генетического алгоритма.

# **Теоретическая часть**

*Роевой интеллект* (англ. Swarm Intelligence) представляет собой метод оптимизации, вдохновленный поведением коллективных систем в природе, таких как рой пчёл или стая птиц. Основная идея заключается в моделировании совместного взаимодействия и координации множества агентов для достижения коллективных целей. Агенты, или частицы, взаимодействуют между собой, обмениваются информацией и адаптируют свое поведение в соответствии с окружающей средой.

Один из методов оптимизации роевого интеллекта — это алгоритм *"Роевая оптимизация частиц"* (Particle Swarm Optimization, PSO). В данном методе частицы в пространстве поиска представляют потенциальные решения задачи оптимизации. Каждая частица имеет свою позицию и скорость, а также сохраняет информацию о лучшем решении, которое она достигла. Частицы обновляют свои позиции, основываясь на комбинации собственного опыта и опыта наилучшей частицы в рое, а также под влиянием инерции и случайных факторов. Алгоритм выполняется в течение заданного числа итераций, при этом постепенно сходясь к оптимальному решению.

Параметры PSO включают количество частиц, коэффициенты инерции, собственного и глобального лучших значений. Инерция определяет влияние предыдущей скорости на текущую, а коэффициенты собственного и глобального лучших значений регулируют вклад собственного опыта частицы и опыта наилучшей частицы в рое.

# **Описание алгоритма**

1. Получение основных характеристик через интерфейс программы: функция, количество частиц, коэффициент текущей скорости, коэффициент собственного лучшего значения, коэффициент глобального лучшего значения, количество итераций.
2. Запуск роевого интеллекта с заданными параметрами.
3. Отображение лучшего решения вместе с точкой. Итеративный вывод графика роя.

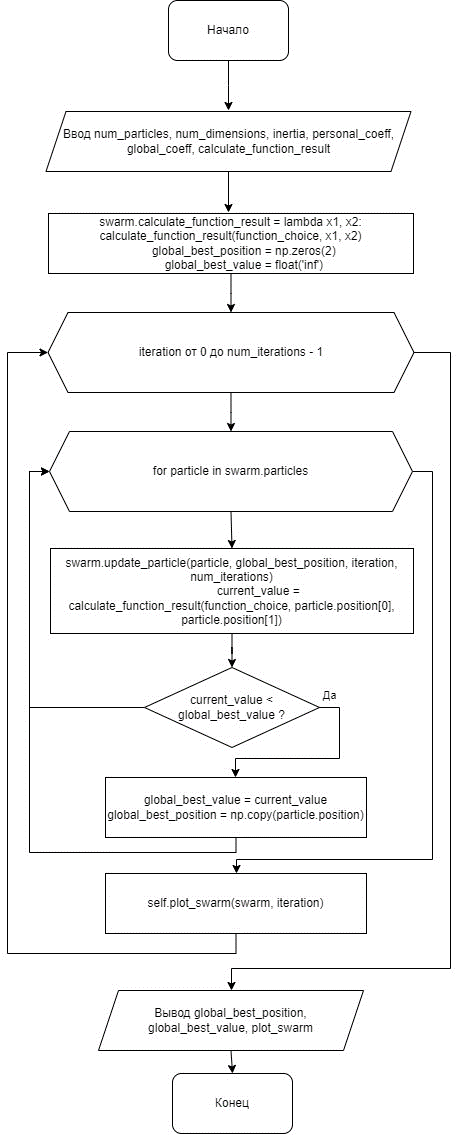


Рисунок 4.1. Блок-схема алгоритма.

# **Описание программы**

## **Описание классов**

В программе используется 3 класса: 1 связанный с интерфейсом программы, 2 связанных с роевым интеллектом. В таблице 5.1 представлено описание классов.

*Таблица 5.1. Описание классов*

|  |  |  |
| --- | --- | --- |
| Имя класса | Наследование | Описание |
| Particle | — | Класс, представляющий частицу в рое |
| Swarm | — | Класс, представляющий рой частиц |
| ParticleSwarmOptimizationApp | — | Класс, отвечающий за интерфейс |

## **5.2. Описание функций**

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

*Таблица 5.2. Описание функций класса «Swarm»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| Функция, количество частиц, коэффициент текущей скорости, коэффициент собственного лучшего значения, коэффициент глобального лучшего значения, количество итераций | \_\_init\_\_ | Инициализация класса |
| Частица, массив текущих глобальных лучших положений, номер итерации, количество итераций | update\_particle | Обновление положения и скорости частицы |

Описание функций класса *«Particle»* представлено в таблице 5.3.

*Таблица 5.3. Описание функций класса «Particle»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| Размерность | \_\_init\_\_ | Инициализация класса |

Описание функций класса *«ParticleSwarmOptimizationApp»* представлено в таблице 5.4.

*Таблица 5.4. Описание функций класса «ParticleSwarmOptimizationApp»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| Интерфейс | \_\_init\_\_ | Инициализация класса |
| — | clear\_results | Очистка результатов предыдущих вычислений |
| — | start\_swarm | Запуск роя в отдельном потоке |
| Функция, количество частиц, коэффициент текущей скорости, коэффициент собственного лучшего значения, коэффициент глобального лучшего значения, количество итераций | run\_swarm | Выполнение рассчетов и отображение результатов |
| Рой частиц, номер итерации | plot\_swarm | Отображение текущего состояния роя на графике |

## **5.3. Описание переменных**

*Таблица 5.5. Описание переменных класса «Swarm»*

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Описание |
| List[] | particles | Массив частиц |
| float | inertia | Коэффициент инерции, влияющий на сохранение текущей скорости частицы. |
| float | personal\_coeff | Коэффициент влияния на частицу её собственного лучшего положения. |
| float | global\_coeff | Коэффициент влияния на частицу глобального лучшего положения. |
| callable | calculate\_function\_result | Функция для вычисления значения целевой функции |

*Таблица 5.6. Описание переменных класса «Particle»*

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Описание |
| np.ndarray | position | Текущее положение частицы в пространстве поиска |
| np.ndarray | velocity | Текущая скорость частицы |
| np.ndarray | best\_position | Лучшее известное положение частицы в пространстве поиска |
| float | best\_value | Лучшее известное значение функции |

Полный код программы представлен в Приложении 1.

# **Рекомендации пользователю**

Поле *«Функция»* позволяет выбрать математическую функцию из выпадающего списка. Поля *«Количество частиц»*, *«Коэффициент текущей скорости»*, *«Коэффициент собственного лучшего значения»*, *«Коэффициент глобального лучшего значения»*, *«Количество итераций»* — позволяют настроить параметры роевого интеллекта.

Кнопка *«Рассчитать»* отвечает за запуск работы программы. Далее лучшее решение и значение выводится внизу интерфейса, а также происходит итеративный вывод в график роя частиц.

Для завершения работы нажмите на крестик в левом верхнем углу.

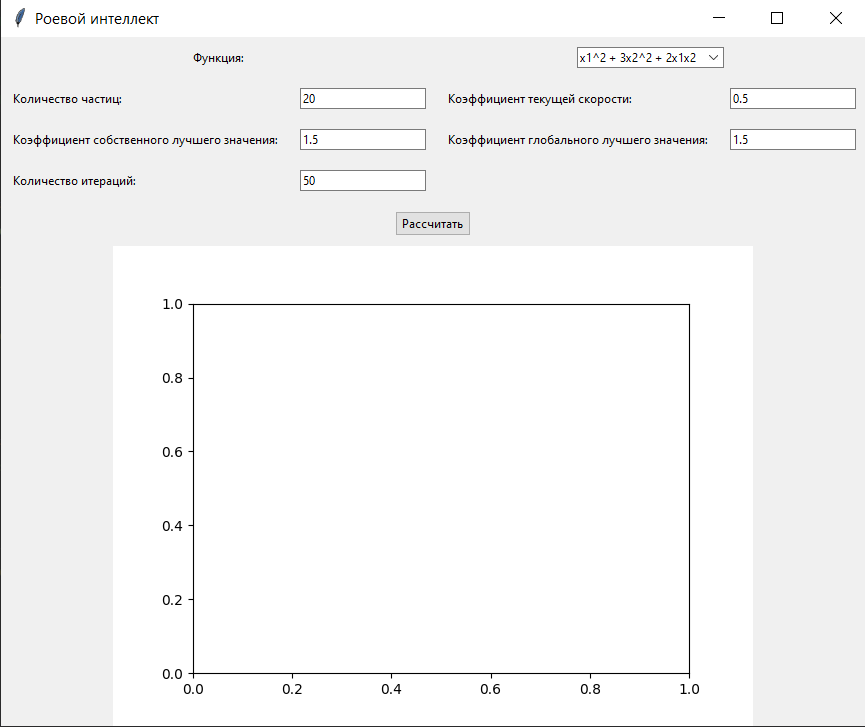
# **Рекомендации программисту**

Для запуска программы необходим Python версии не ниже 3.10.6, а также 64-битная операционная система Windows/Linux/macOS. Предварительно необходимо установить библиотеки: matplotlib версии не ниже matplotlib 3.7.1, numpy версии не ниже 1.24.2. Для работы с кодом необходим PyCharm версии не ниже 2022.2.1.

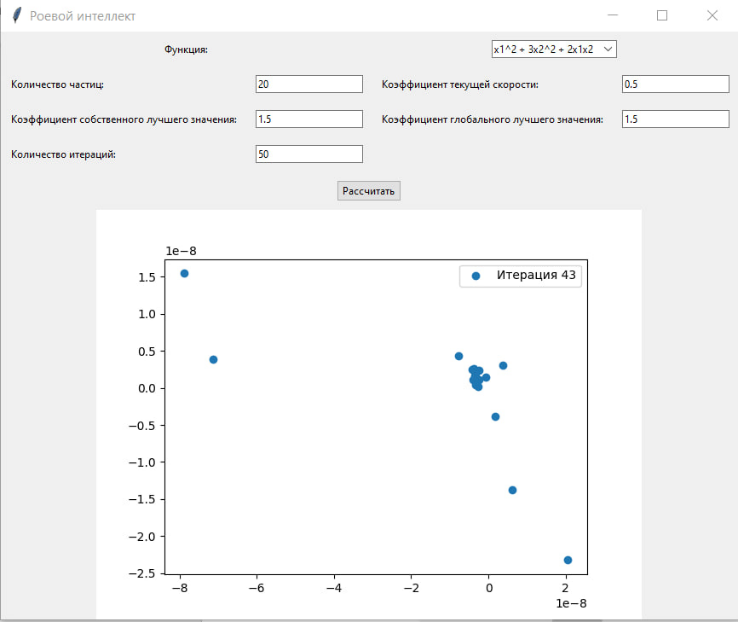
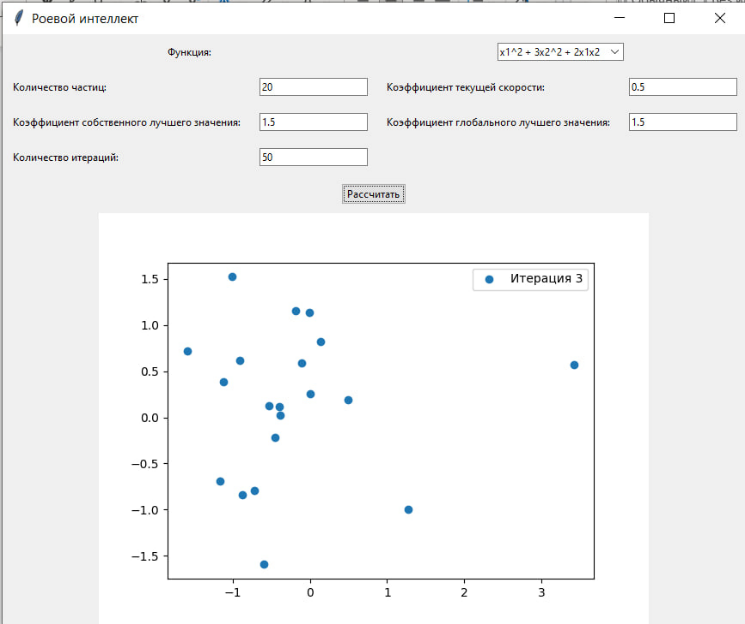
Минимальное необходимое место на диске: 0.5 МБ. Минимальное необходимое количество оперативной памяти: 100 МБ.

# **Контрольный пример**

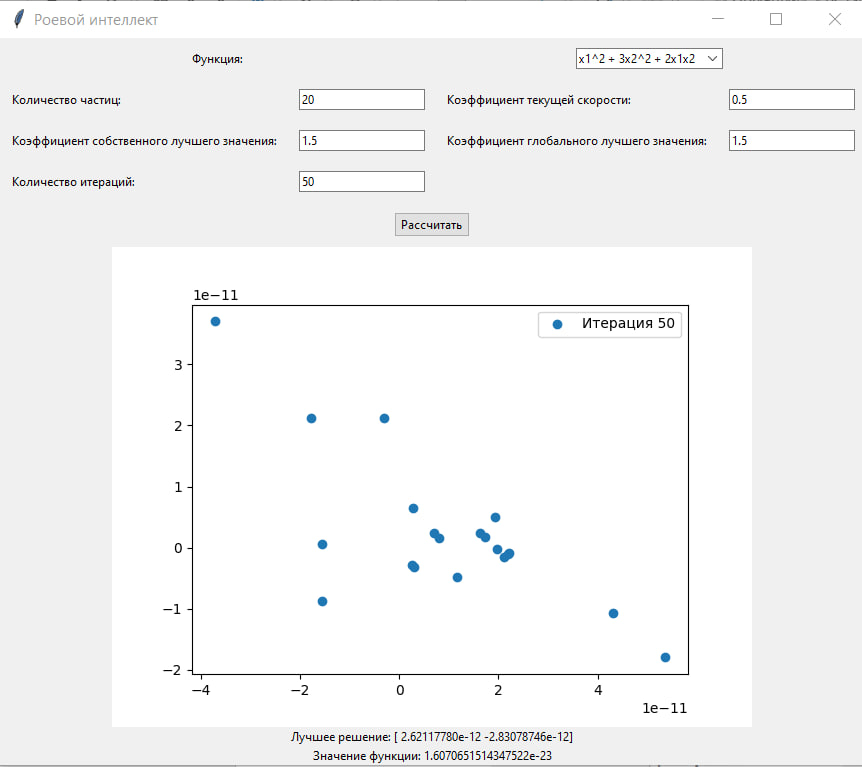
В данном разделе представлен контрольный пример, демонстрирующий способность программы искать минимальное значение функции с помощью роевого интеллекта.



*Рисунок 8.1. Интерфейс программы*



*Рисунок 8.2. Интерфейс работы программмы*



*Рисунок 8.3. Результат работы программы*

# **Заключение**

Разработанная программа, основанная на алгоритме роевого интеллекта, представляет собой эффективный инструмент для решения задач глобальной оптимизации математических функций. Результаты экспериментов на выбранных тестовых функциях подтверждают успешную работу роевого интеллекта в достижении оптимальных значений.

В ходе оптимизации особое внимание было уделено параметрам алгоритма, таким как количество частиц в рое, коэффициенты инерции, личного и глобального влияния. Эти параметры были настроены с учетом характеристик задачи оптимизации, обеспечивая гибкость и эффективность работы алгоритма.

Ключевым улучшением алгоритма стало внедрение стратегии управления скоростью и положением частиц. Это улучшение способствует лучшей адаптации роя к изменениям в пространстве поиска, обеспечивая более эффективную сходимость к оптимальным решениям.

В сравнении с генетическим алгоритмом роевой интеллект проявляет высокую точность в поиске оптимальных значений при одинаковых количествах итераций. Роевой алгоритм демонстрирует более эффективную сходимость к оптимальным решениям, обеспечивая более точные результаты на ранних этапах оптимизации.

С другой стороны, генетический алгоритм имеет преимущество в скорости сходимости на минимальных количествах итераций, показывая более высокую эффективность в начальной фазе оптимизации. Однако его точность страдает с увеличением числа поколений, и при этом увеличение точности замедляется по сравнению с роевым интеллектом.

# **Приложения**

import random  
import numpy as np  
import tkinter as tk  
from tkinter import ttk  
from threading import Thread  
import matplotlib.pyplot as plt  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
  
  
def calculate\_function\_result(function\_choice, x1, x2):  
 *""" Вычисление значения выбранной функции """* 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 Particle:  
 *""" Класс, представляющий частицу в рое """* def \_\_init\_\_(self, num\_dimensions):  
 self.position = np.array([random.uniform(-5, 5) for \_ in range(num\_dimensions)])  
 self.velocity = np.array([random.uniform(-1, 1) for \_ in range(num\_dimensions)])  
 self.best\_position = np.copy(self.position)  
 self.best\_value = float('inf')  
  
  
class Swarm:  
 *""" Класс, представляющий рой частиц """* def \_\_init\_\_(self, num\_particles, num\_dimensions, inertia, personal\_coeff, global\_coeff, calculate\_function\_result):  
 self.particles = [Particle(num\_dimensions) for \_ in range(num\_particles)]  
 self.inertia = inertia  
 self.personal\_coeff = personal\_coeff  
 self.global\_coeff = global\_coeff  
 self.calculate\_function\_result = calculate\_function\_result  
  
 def update\_particle(self, particle, global\_best\_position, iteration, max\_iterations):  
 *""" Обновление положения и скорости частицы """* current\_inertia = self.inertia \* (1 - iteration / max\_iterations)  
  
 inertia\_term = current\_inertia \* particle.velocity  
 personal\_term = self.personal\_coeff \* random.random() \* (particle.best\_position - particle.position)  
 global\_term = self.global\_coeff \* random.random() \* (global\_best\_position - particle.position)  
 particle.velocity = inertia\_term + personal\_term + global\_term  
  
 particle.position += particle.velocity  
  
 current\_value = self.calculate\_function\_result(particle.position[0], particle.position[1])  
 if current\_value < particle.best\_value:  
 particle.best\_value = current\_value  
 particle.best\_position = np.copy(particle.position)  
  
  
class ParticleSwarmOptimizationApp:  
 *""" Класс, отвечающий за интерфейс на библиотеке tkinter """* def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Роевой интеллект")  
  
 """ Переменные """  
 self.num\_particles\_var = tk.StringVar(value="20")  
 self.inertia\_var = tk.StringVar(value="0.5")  
 self.personal\_coeff\_var = tk.StringVar(value="1.5")  
 self.global\_coeff\_var = tk.StringVar(value="1.5")  
 self.num\_iterations\_var = tk.StringVar(value="50")  
 self.function\_choice\_var = tk.StringVar(value="x1^2 + 3x2^2 + 2x1x2")  
 self.result\_labels = []  
  
 """ Интерфейс """  
 ttk.Label(self.root, text="Функция:").grid(row=0, column=0, columnspan=2, pady=10)  
 self.test\_function\_combobox = ttk.Combobox(  
 self.root, values=["x1^2 + 3x2^2 + 2x1x2", "4(x1 - 5)^2+(x2 - 6)^2"],  
 textvariable=self.function\_choice\_var, state="readonly"  
 )  
 self.test\_function\_combobox.grid(row=0, column=2, columnspan=2, pady=10)  
 ttk.Label(self.root, text="Количество частиц:").grid(row=1, column=0, padx=10, pady=10, sticky="w")  
 ttk.Entry(self.root, textvariable=self.num\_particles\_var).grid(row=1, column=1, padx=10, pady=10, sticky="w")  
 ttk.Label(self.root, text="Коэффициент текущей скорости:").grid(row=1, column=2, padx=10, pady=10, sticky="w")  
 ttk.Entry(self.root, textvariable=self.inertia\_var).grid(row=1, column=3, padx=10, pady=10, sticky="w")  
 ttk.Label(self.root, text="Коэффициент собственного лучшего значения:").grid(row=2, column=0, padx=10, pady=10, sticky="w")  
 ttk.Entry(self.root, textvariable=self.personal\_coeff\_var).grid(row=2, column=1, padx=10, pady=10, sticky="w")  
 ttk.Label(self.root, text="Коэффициент глобального лучшего значения:").grid(row=2, column=2, padx=10, pady=10, sticky="w")  
 ttk.Entry(self.root, textvariable=self.global\_coeff\_var).grid(row=2, column=3, padx=10, pady=10, sticky="w")  
 ttk.Label(self.root, text="Количество итераций:").grid(row=3, column=0, padx=10, pady=10, sticky="w")  
 ttk.Entry(self.root, textvariable=self.num\_iterations\_var).grid(row=3, column=1, padx=10, pady=10, sticky="w")  
 ttk.Button(self.root, text="Рассчитать", command=self.start\_swarm).grid(row=4, column=0, columnspan=4, padx=10, pady=10)  
  
 self.figure, self.ax = plt.subplots()  
 self.canvas = FigureCanvasTkAgg(self.figure, master=self.root)  
 self.canvas\_widget = self.canvas.get\_tk\_widget()  
 self.canvas\_widget.grid(row=5, column=0, columnspan=4)  
  
 def clear\_results(self):  
 *""" Очистка результатов предыдущих вычислений """* for label in self.result\_labels:  
 label.destroy()  
 self.result\_labels = []  
  
 def start\_swarm(self):  
 *""" Запуск роя в отдельном потоке """* self.clear\_results()  
 num\_particles = int(self.num\_particles\_var.get())  
 inertia = float(self.inertia\_var.get())  
 personal\_coeff = float(self.personal\_coeff\_var.get())  
 global\_coeff = float(self.global\_coeff\_var.get())  
 num\_iterations = int(self.num\_iterations\_var.get())  
 function\_choice = self.function\_choice\_var.get()  
  
 # Start swarm in a separate thread  
 swarm\_thread = Thread(target=self.run\_swarm, args=(num\_particles, inertia, personal\_coeff, global\_coeff, num\_iterations, function\_choice))  
 swarm\_thread.start()  
  
 def run\_swarm(self, num\_particles, inertia, personal\_coeff, global\_coeff, num\_iterations, function\_choice):  
 *""" Выполнение рассчетов и отображение результатов """* self.clear\_results()  
 swarm = Swarm(num\_particles, 2, inertia, personal\_coeff, global\_coeff, calculate\_function\_result)  
 swarm.calculate\_function\_result = lambda x1, x2: calculate\_function\_result(function\_choice, x1, x2)  
 global\_best\_position = np.zeros(2)  
 global\_best\_value = float('inf')  
  
 for iteration in range(num\_iterations):  
 for particle in swarm.particles:  
 swarm.update\_particle(particle, global\_best\_position, iteration, num\_iterations)  
 current\_value = calculate\_function\_result(function\_choice, particle.position[0], particle.position[1])  
  
 if current\_value < global\_best\_value:  
 global\_best\_value = current\_value  
 global\_best\_position = np.copy(particle.position)  
  
 self.plot\_swarm(swarm, iteration)  
  
 """ Вывод результатов """  
 ttk.Label(self.root, text=f"Лучшее решение: {global\_best\_position}").grid(row=6, column=0, columnspan=4)  
 ttk.Label(self.root, text=f"Значение функции: {global\_best\_value}").grid(row=7, column=0, columnspan=4)  
  
 def plot\_swarm(self, swarm, iteration):  
 *""" Отображение текущего состояния роя на графике """* self.ax.clear()  
 positions = np.array([particle.position for particle in swarm.particles])  
 self.ax.scatter(positions[:, 0], positions[:, 1], label=f'Итерация {iteration + 1}')  
 self.ax.legend()  
 self.canvas.draw()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 window = tk.Tk()  
 ParticleSwarmOptimizationApp(window)  
 window.mainloop()

*Приложение 1. Листинг программы.*