**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный УНИВЕРСИТЕТ**

**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»**

**Вариант 3 (модификация в части «коэффициент сжатия»)**

|  |  |  |
| --- | --- | --- |
| Студент гр. 23.Б16 |  | Барафанов А.А. |
| Преподаватель |  | Дик А.Г. |

**Санкт-Петербург**

**2024 г.**

**Оглавление**

[**1.** **Цель работы** 3](#_Toc182513996)

[**2.** **Задача (формализация)** 3](#_Toc182513997)

[**3.** **Теоретическая часть** 3](#_Toc182513998)

[**4.** **Описание алгоритма** 5](#_Toc182513999)

[**5.** **Описание программы** 9](#_Toc182514000)

[**5.1** **Описание классов** 9](#_Toc182514001)

[**5.2 Описание функций** 12](#_Toc182514002)

[**6.** **Рекомендации пользователю** 12](#_Toc182514003)

[**7.** **Рекомендации программисту** 13](#_Toc182514014)

[**8.** **Контрольный пример** 14](#_Toc182514021)

[**9.** **Анализ результатов и сравнение с генетическим алгоритмом.** 16](#_Toc182514022)

[**10.** **Вывод** 20](#_Toc182514023)

[**11.** **Листинг кода** 20](#_Toc182514025)

# **Цель работы**

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

# **Задача (формализация)**

Создание алгоритма роя частиц с внесенной модификацией в части коэффициента сжатия, а также создание интерфейса для ввода необходимых параметров пользователем и сравнение эффективности скорости схождения и точности результатов генетического и роевого алгоритмов на примере исследования функции двух переменных на глобальный минимум.

# **Теоретическая часть**

*Роевой интеллект (Swarm Intelligence)* — это метод оптимизации и моделирования, основанный на коллективном поведении простых агентов в больших группах, таких как стаи птиц, колонии муравьев, косяки рыб и т.д. В контексте алгоритмов роевой интеллект представляет собой подход, при котором множество агентов (например, частиц в алгоритме роя частиц или муравьев в алгоритме муравьиной колонии) взаимодействуют друг с другом и с окружающей средой, чтобы решать сложные задачи оптимизации.

*Алгоритм роя частиц (Particle Swarm Optimization, PSO)* — это метод оптимизации, основанный на моделировании коллективного поведения роя частиц в пространстве поиска. Алгоритм был разработан Джеймсом Кеннеди и Расселом Эберхартом в 1995 году и вдохновлен наблюдениями за поведением стай птиц и косяков рыб.

*Основные принципы алгоритма роя частиц* (Particle Swarm Optimization, PSO):

* *Рой частиц:* Множество частиц перемещаются в многомерном пространстве поиска, представляя возможные решения задачи оптимизации.
* *Позиция и скорость:* Каждая частица имеет текущую позицию и скорость.
* *Личное лучшее положение (pBest):* Частица запоминает свое наилучшее положение, которое она когда-либо посещала.
* *Глобальное лучшее положение (gBest):* Все частицы знают о наилучшем положении, найденном всем роем.
* *Обновление скорости и позиции:* На каждой итерации скорость и позиция каждой частицы обновляются на основе следующей формулы.

*Классическая формула обновления скорости:*

https://jenyay.net/uploads/Programming/ParticleSwarm/eqn_5.png

*Оптимизация алгоритма роя частиц (PSO) с помощью коэффициента сжатия (Constriction Factor):*

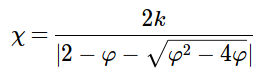
*Коэффициент сжатия (Constriction Factor)* — это метод, который позволяет улучшить сходимость алгоритма PSO и предотвратить его расхождение. Он был предложен Карпентером и Эберхартом в 1999 году и основан на идее введения нормировки коэффициентов собственного и глобального лучших значений, чтобы сходимость не так сильно зависела от их выбора. Коэффициент сжатия вводит дополнительный множитель в формулу обновления скорости, который ограничивает скорость частиц и помогает алгоритму сходиться более стабильно.

*Формула обновления скорости с коэффициентом сжатия:* (значения коэффициентов даны в описании алгоритма)

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

*Расчет коэффициента сжатия:*

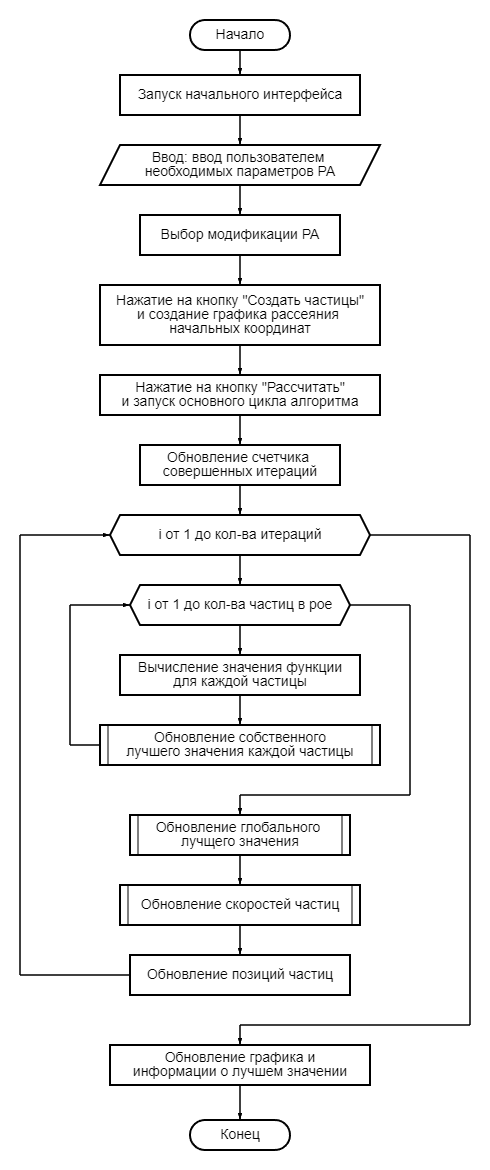
В этой лабораторной работе коэффициент сжатия рассчитывается по канонической формуле, представленной ниже:

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

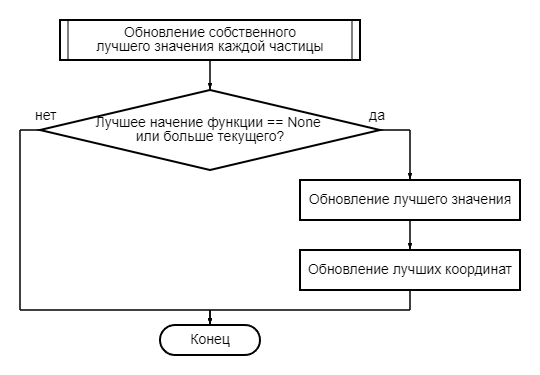
# **Описание алгоритма**

1. *Инициализация частиц:* Создание частиц роя с начальными случайными позициями и скоростями в заданном диапазоне.
2. *Оценка функции пригодности:* Вычисление значения целевой функции для текущей позиции каждой частицы.
3. *Обновление лучших значений частиц:* Сравнение текущего значения функции с лучшим значением частицы и обновление, если текущее значение лучше.
4. *Обновление глобального лучшего значения:* Поиск лучшего значения среди всех частиц и обновление глобального лучшего значения.
5. *Обновление скоростей частиц:* Расчет новых скоростей частиц на основе их текущих скоростей, лучших значений частиц и глобального лучшего значения.
6. *Обновление позиций частиц:* Перемещение частиц в соответствии с их новыми скоростями.
7. *Повторение итераций:* Повторение шагов 2-6 для заданного количества итераций.
8. *Визуализация:* Отображение текущего положения частиц на графике и обновление информации о лучшем решении.
9. *Вывод результатов:* После завершения всех итераций вывод лучшего найденного решения и соответствующего значения функции.

*Блок-схемы, иллюстрирующие шаги алгоритма показаны на рисунках 4.1-4.4.*

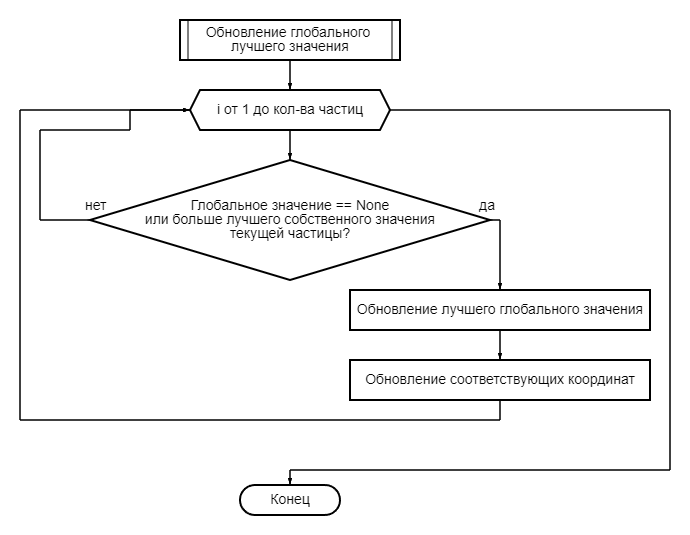
**

*Рисунок 4.1. Блок-схема основной программы.*

**

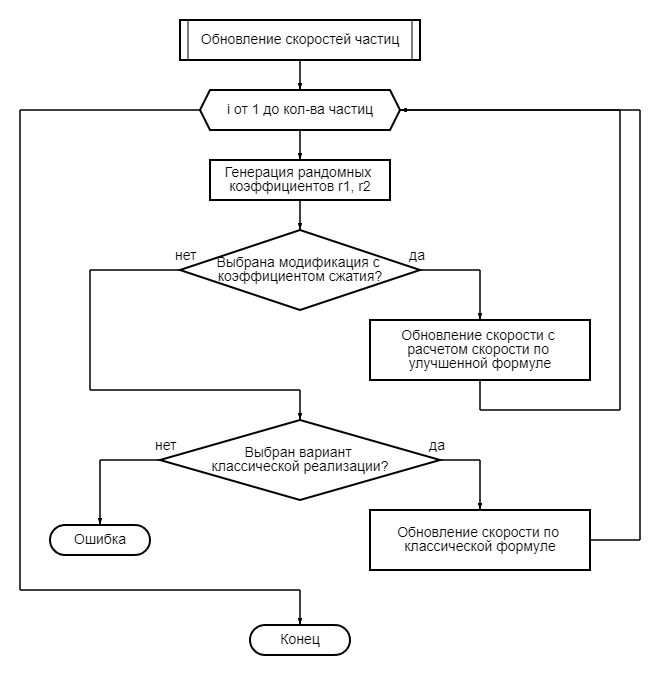
*Рисунок 4.2. Блок-схема подпрограммы*

*обновления лучшего собственного значения.*

**

*Рисунок 4.3. Блок-схема подпрограммы*

*обновления лучшего глобального значения.*

**

*Рисунок 4.4. Блок-схема подпрограммы*

*обновления скоростей.*

# **Описание программы**

## **Описание классов**

В программе используется 2 класса. В таблицах 5.1.1-5.1.2 представлено их описание.

*Таблица 5.1.1. Описание класса Particle*

|  |  |  |
| --- | --- | --- |
| Имя класса | Наследование | Описание класса |
| Particle | \_\_ | Класс, содержащий информацию и методы, касающиеся конкретной частицы |

*Таблица 5.1.2. Описание класса Swarm*

|  |  |  |
| --- | --- | --- |
| Имя класса | Наследование | Описание класса |
| Swarm | \_\_ | Класс, содержащий информацию о рое частиц |

Описание переменных класса Swarm представлено в таблице 5.1.3.

*Таблица 5.1.3. Описание переменных класса Swarm*

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Права доступа | Имя | Описание |
| float | public | global\_best\_value | Лучшее глобальное значение |
| float | public | global\_best\_x | Координата x частицы с лучшим глобальным значением |
| float | public | global\_best\_y | Координата y частицы с лучшим глобальным значением |
| str | public | mod | Выбранная модификация |
| list | public | particles | Список частиц роя |
| float | public | c1 | Коэффициент собственного лучшего значения |
| float | public | c2 | Коэффициент глобального лучшего значения |
| float | public | chi | Коэффициент сжатия |

\Описание переменных класса Particle представлено в таблице 5.1.4.

*Таблица 5.1.4. Описание переменных класса Particle*

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Права доступа | Имя | Описание |
| float | public | x | Текущая x координата частицы |
| float | public | y | Текущая y координата частицы |
| float | public | best\_value | Собственное лучшее значение частицы |
| float | public | best\_x | Координата x собственного лучшего значения |
| float | public | best\_y | Координата y собственного лучшего значения |
| float | public | vx | Скорость в направлении Ox |
| float | public | vy | Скорость в направлении Oy |

В таблице 5.1.5 представлено описание методов класса Swarm.

*Таблица 5.1.5. Описание методов класса Swarm*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Модификатор  доступа/ Возвращаемый тип | Входные данные | Описание |
| \_\_init\_\_ | public/- | self, количество частиц, диапазон изменения x, диапазон изменения y, коэффициенты для формулы скорости, модификация | Конструктор класса |
| update\_global\_best | public/- | self | Обновление глобального лучшего значения |
| update\_velocities | public/- | self | Обновление скоростей всех частиц роя |
| update\_positions | public/- | self | Обновление позиций всех частиц роя |
| run\_pso | public/- | self, целевая функция, кол-во итераций | Запуск алгоритма роя частиц |

В таблице 5.1.6 представлено описание методов класса Particle.

*Таблица 5.1.6. Описание методов класса Particle*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Модификатор  доступа/ Возвращаемый тип | Входные данные | Описание |
| \_\_init\_\_ | public/- | self, диапазон изменения x, диапазон изменения y | Конструктор класса |
| update\_best\_value | public/- | self, текущее значение скорости частицы | Обновление собственного лучшего значения частицы |

## **5.2 Описание функций**

В программе используется 4 функции, выполняющие вспомогательные процедуры программы. В таблице 5.2.1 представлено их описание.

*Таблица 5.2.1. Описание функций*

|  |  |  |
| --- | --- | --- |
| Имя функции | Входные данные | Описание функции |
| objective\_function | Координаты x и y | Расчет целевой функции |
| inserted | Поле ввода, вводимое значение | Функция для вставки значения в поле ввода |
| creating | \_\_ | Использование данных из интерфейса, создание роя и начального графика рассеяния позиций частиц |
| make\_iteration | Кол-во итераций | Запуск основного цикла алгоритма роя частиц с заданным числом итераций |

# **Рекомендации пользователю**

* Функция: Программа оптимизирует функцию -12y + 4x^2 + 4y^2 - 4xy.
* Программа запускается с помощью интерпретатора Python. Убедитесь, что у вас установлены все необходимые библиотеки: numpy, tkinter, random, matplotlib. Запустите программу, выполнив команду python имя\_файла.py.
* Установите все необходимые параметры роевого алгоритма, введя значения в соответствующие поля ввода. Также выберите модификацию для вычисления формулы обновления скоростей:
* Classic: Классический вариант алгоритма роя частиц.
* Canonical: Модификация, включающая в себя использование коэффициента сжатия
* После настройки параметров нажмите кнопку "Создать частицы". На графике рассеивания можно будет посмотреть начальные координаты частиц роя.
* Для запуска самого алгоритма нажмите «Рассчитать». Программа запустит выбранное число итераций и обновит график.
* Для получения более точного решения увеличьте количество частиц.
* Для ускорения работы алгоритма уменьшите количество частиц или количество итераций.
* Попробуйте разные модификации и формульные коэффициенты, чтобы найти наиболее эффективный для вашей задачи.

1. **Рекомендации программисту**

Для работы с программой должен быть установлен интерпретатор язык Python, а также же интегрированная среда разработки (Visual Studio). Поддерживайте актуальные версии интерпретатора и сред разработки. Установите необходимые библиотеки языка python в случае их отсутствия: numpy, tkinter, random, matplotlib.

Вы также можете настроить параметры роевого алгоритма для оптимизации конкретно под вашу задачу, а именно:

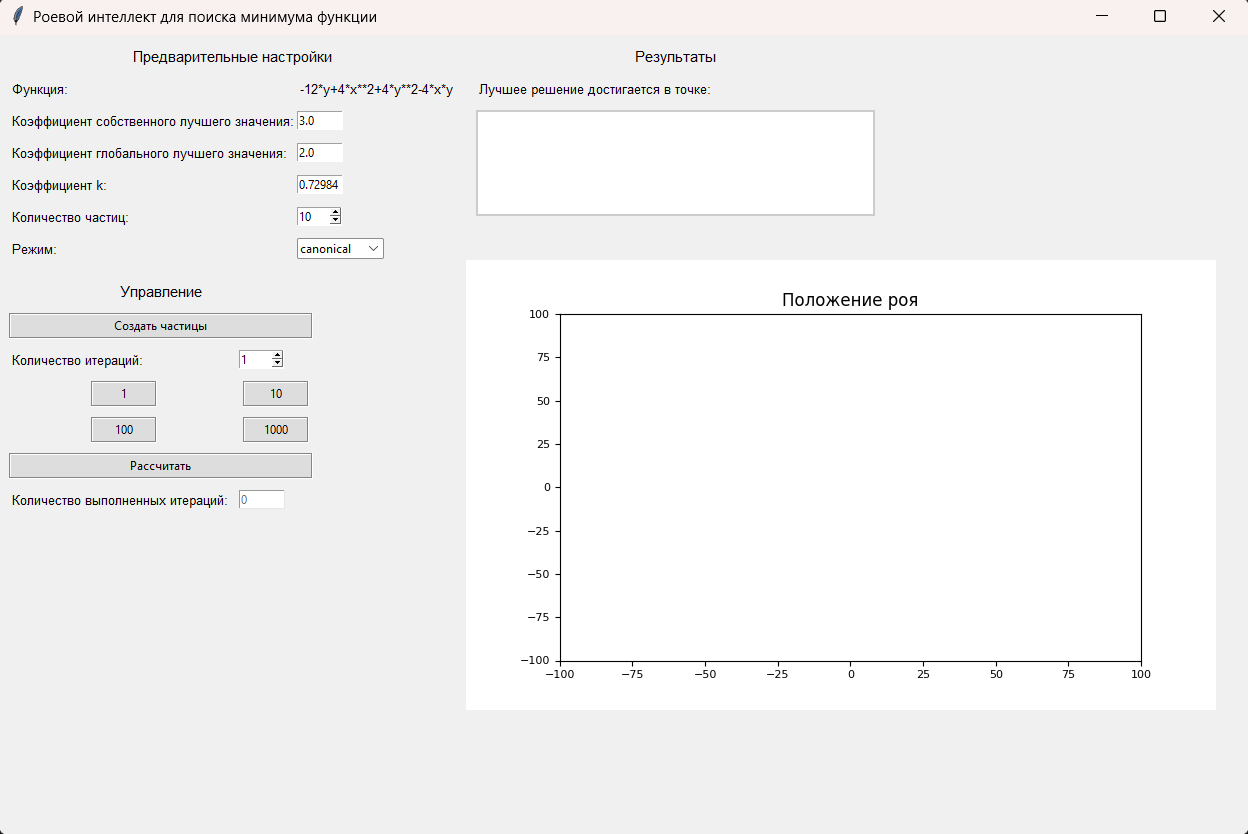
* *Коэффициент собственного лучшего решения:* Высокие показатели этого коэффициента расширят пространство поиска, но уменьшат скорость сходимости.
* *Коэффициент глобального лучшего решения:* Высокие показатели этого коэффициента обеспечат быструю сходимость, однако увеличат риск преждевременной сходимости и нахождение неоптимального решения.
* *Коэффициент k:* Выберите значение k, которое соответствует размеру вашей задачи. Слишком большое k может привести к излишней сложности, а слишком маленькое — к недостаточной сходимости.
* *Количество частиц:* Используйте больше частиц для сложных задач, чтобы увеличить вероятность нахождения оптимального решения. Однако учитывайте, что это увеличит время вычислений.

Конечно же, вы можете изменить функцию, для которой вы хотите найти минимум, однако помните, что роевой алгоритм не гарантирует нахождение глобального минимума, а только приближенное решение.

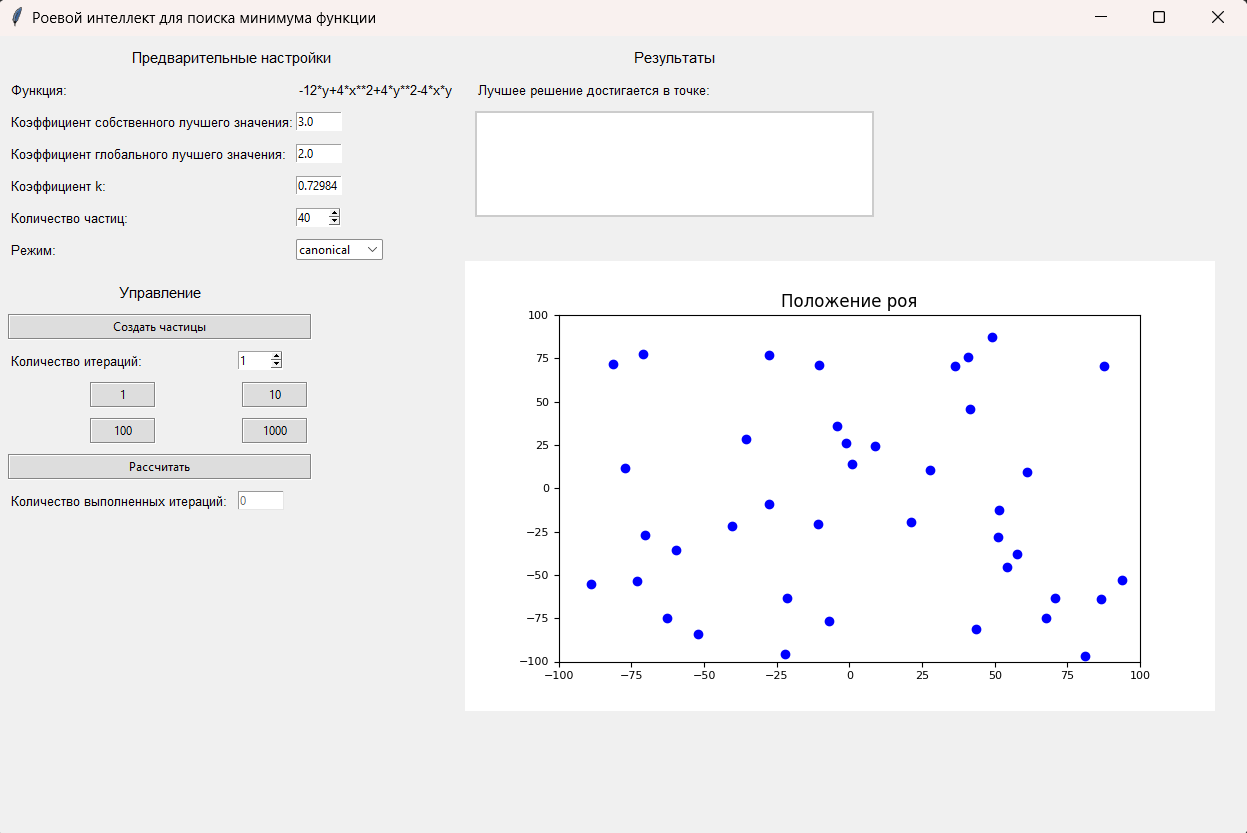
1. **Контрольный пример**

В этом разделе продемонстрирован пример запуска алгоритма роя частиц с дефолтными коэффициентами и числом частиц 40, а также результаты работы алгоритма спустя одну, две и три итерации.

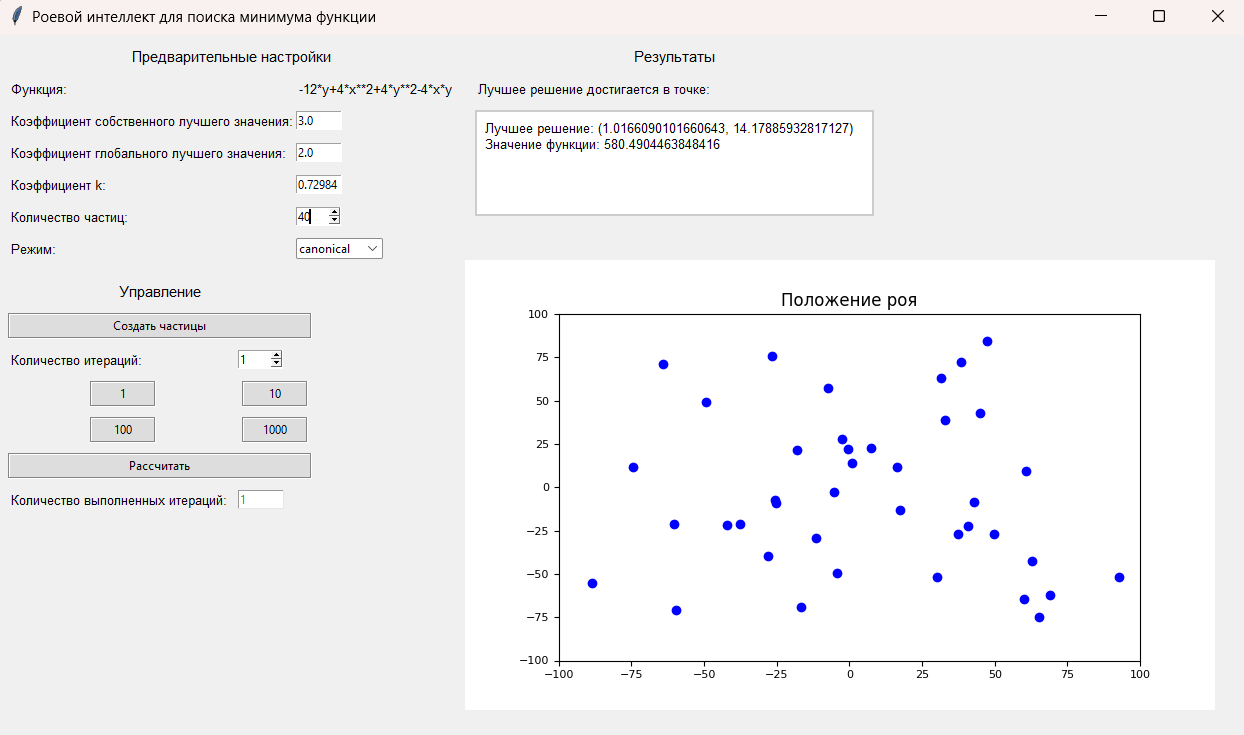
Тонкости взаимодействия с программой указаны в рекомендациях пользователю, здесь же приведены результаты работы программы.

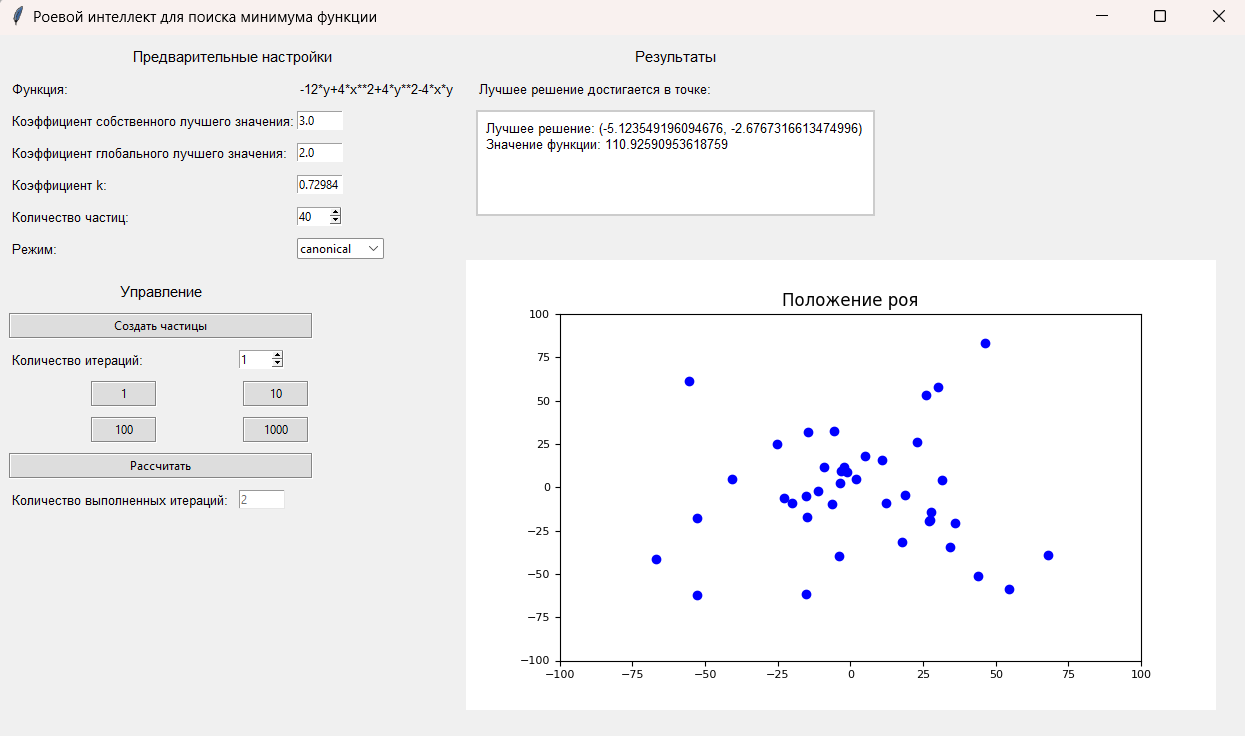


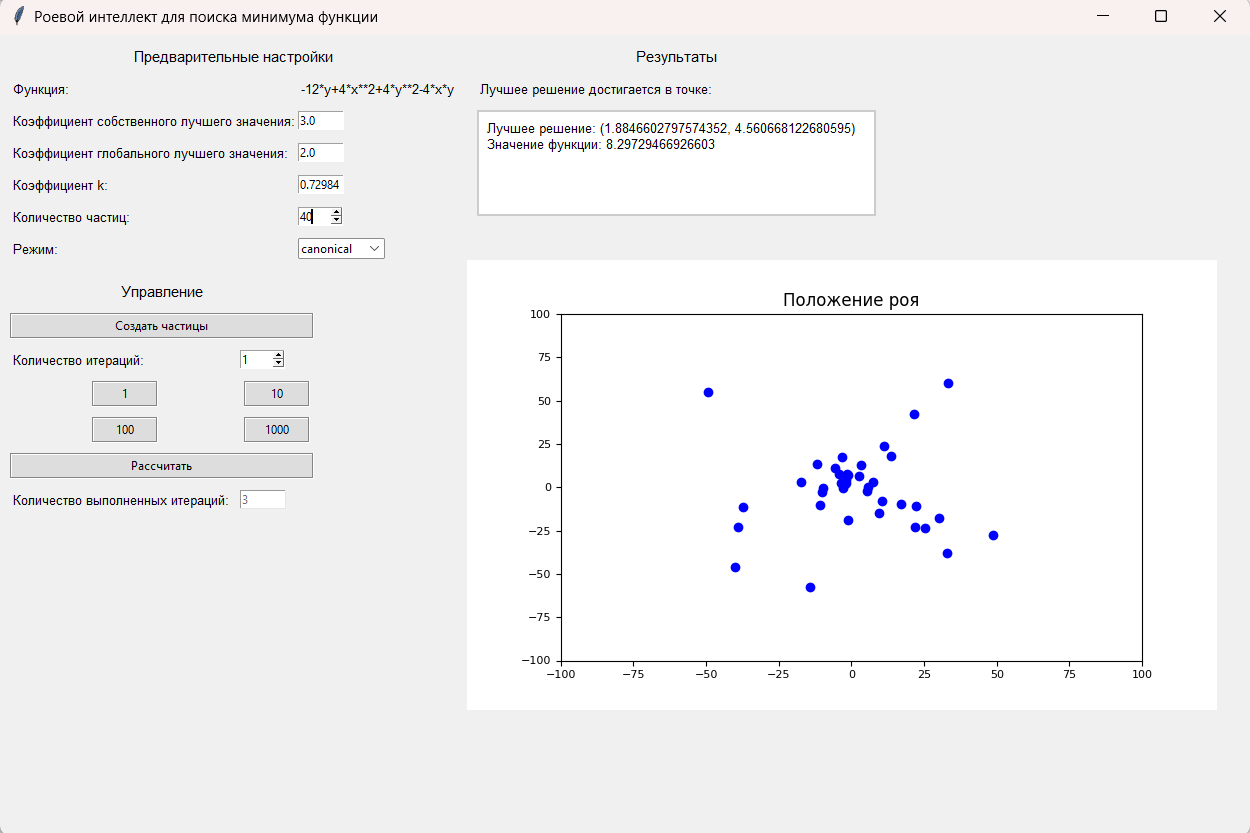
*Рисунок 8.1 Начальный интерфейс программы*



*Рисунок 8.2 Результат после нажатия на кнопку «Создать частицы»*

**

**

**

*Рисунок 8.3 Результаты алгоритма после каждой из трех первых итераций (3 нажатия на кнопку «Рассчитать»)*

# **Анализ результатов и сравнение с генетическим алгоритмом.**

Анализ результатов проведем в *три этапа*:

* Максимальное улучшение скорости сходимости РА путем подбора оптимальных коэффициентов лучших решений.
* Выяснение влияния коэффициента сжатия на результаты.
* Сравнение точности результатов РА и ГА при примерно равных начальных условиях.

1. *Подбор оптимальных коэффициентов.*

*Методика проведения эксперимента:* будем проводить исследования на 10 частицах в области поиска (-100, 100). Все остальные параметры, кроме коэффициентов одинаковы (для чистоты эксперимента). Об эффективности будем судить по темпу схождения частиц к глобальному минимуму и по погрешности результата относительно аналитического решения. Для более точного анализа следует провести несколько испытаний, так как результат сильно зависит от начального распределения точек.

*Результаты эксперимента:*

*Итоговые оптимальные значения коэффициентов:*

Коэффициент собственного лучшего значения – 3.0

Коэффициент глобального лучшего значения – 2.0

1. *Влияние коэффициента сходимости.*

В результате анализа было выяснено, что коэффициент сжатия положительным образом влияет на результаты в следующих аспектах:

* Введение коэффициента увеличивает скорость схождения РА.
* И что более важно, коэффициент делает алгоритм более стабильным. Классический вариант РА критически зависит от выбора начальных коэффициентов, соответственно приходится тратить много времени на подбор коэффициентов для схождения алгоритма. Коэффициент сжатия нормализует коэффициента, тем самым увеличивая гарантии схождения при разных коэффициентах.

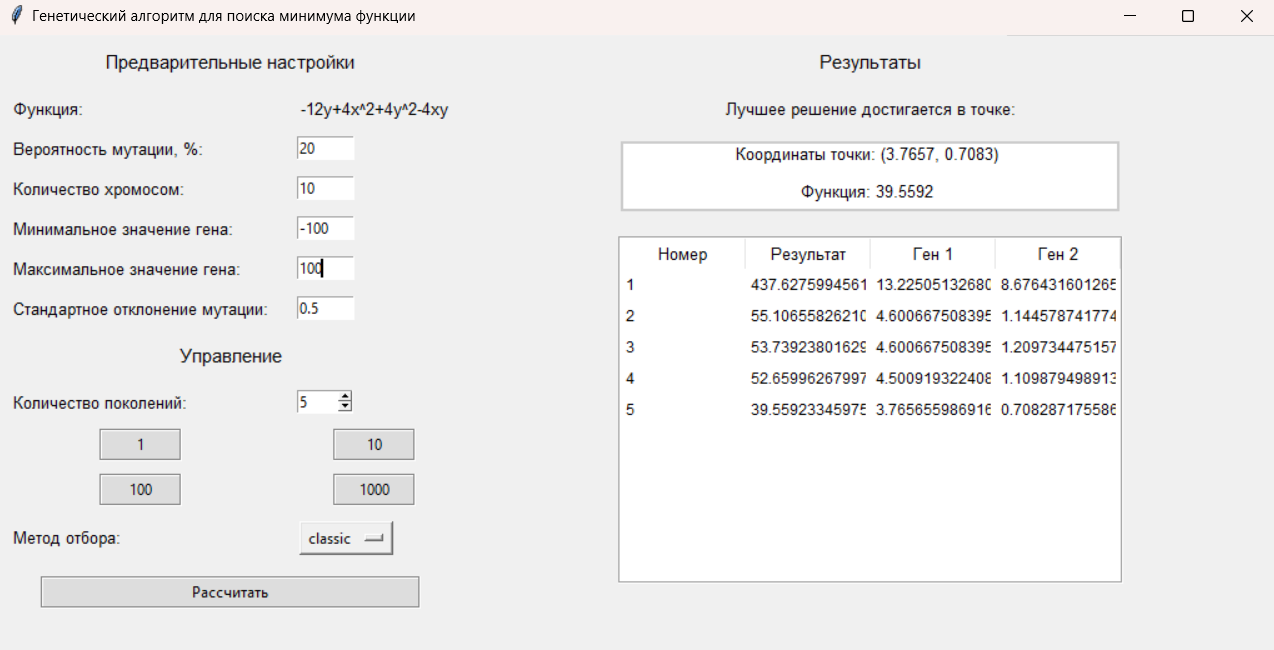
1. *Сравнение с генетическим алгоритмом.*

Оба алгоритма спустя несколько десятков поколений дают очень точные результаты, различающиеся с аналитическим минимумом на доли процентов, поэтому для сравнения эффективности посмотрим на скорость схождения алгоритмов на начальных итерациях.

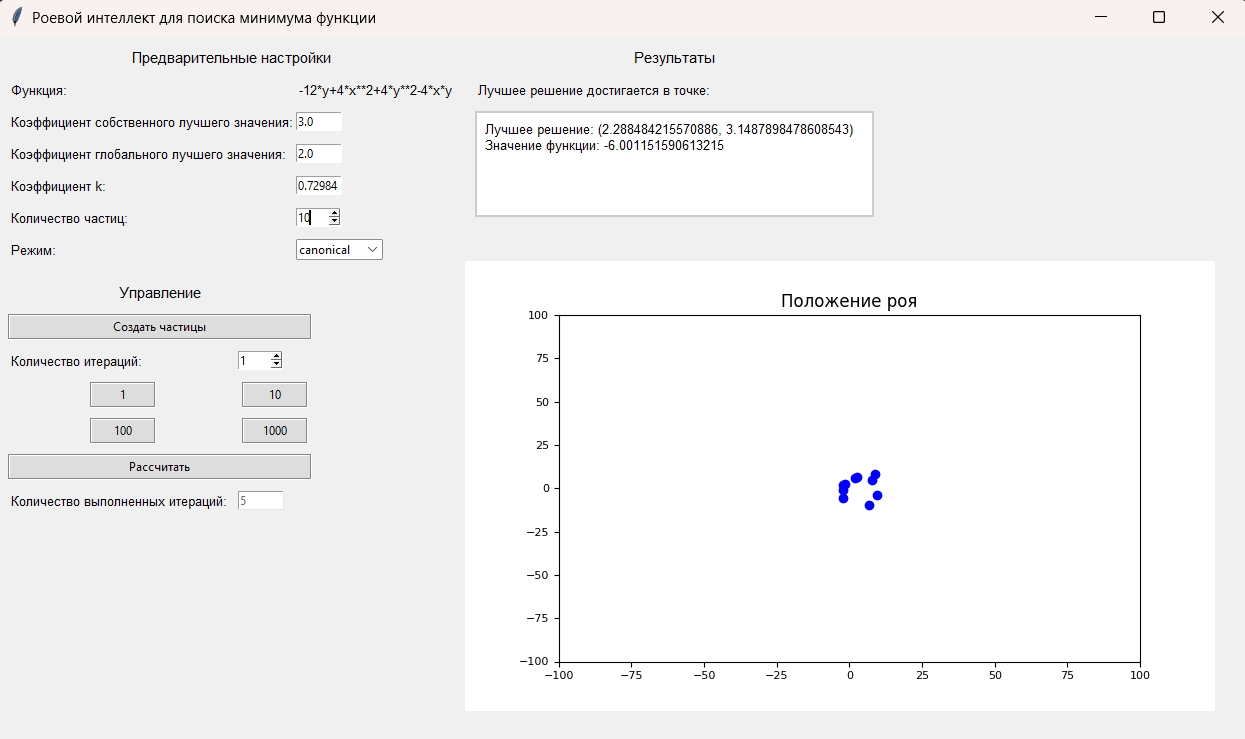
*Методика проведения эксперимента:* введем одинаковые начальные параметры:

* Алгоритм роя частиц – 10 частиц
* Генетический алгоритм – 10 хромосом
* Алгоритм роя частиц – 5 итераций
* Генетический алгоритм – 5 поколений
* Алгоритм роя частиц – (-100,100) пространство поиска
* Генетический алгоритм – (-100, 100) пространство поиска
* Одинаковая аналитическая функция (из 3-его варианта)

Теперь посмотрим какого результата достиг генетический алгоритм (рис. 9.1) и алгоритм роя частиц (рис. 9.2)

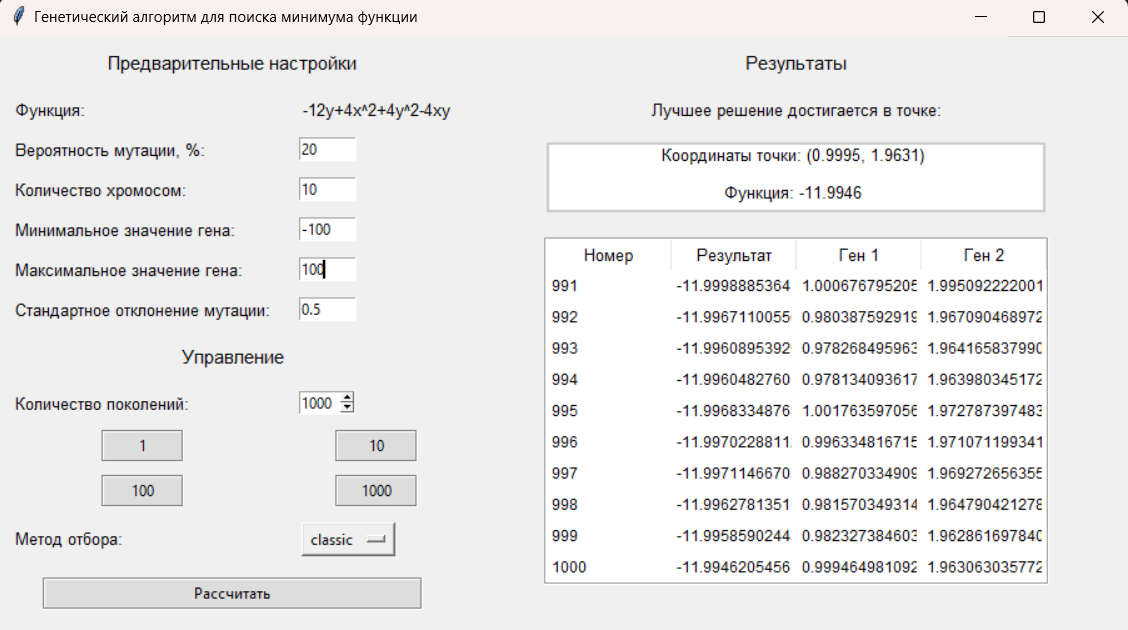


*Рисунок 9.1 ГА: 5 итераций. Лучшее значение ф-ции около 39.*

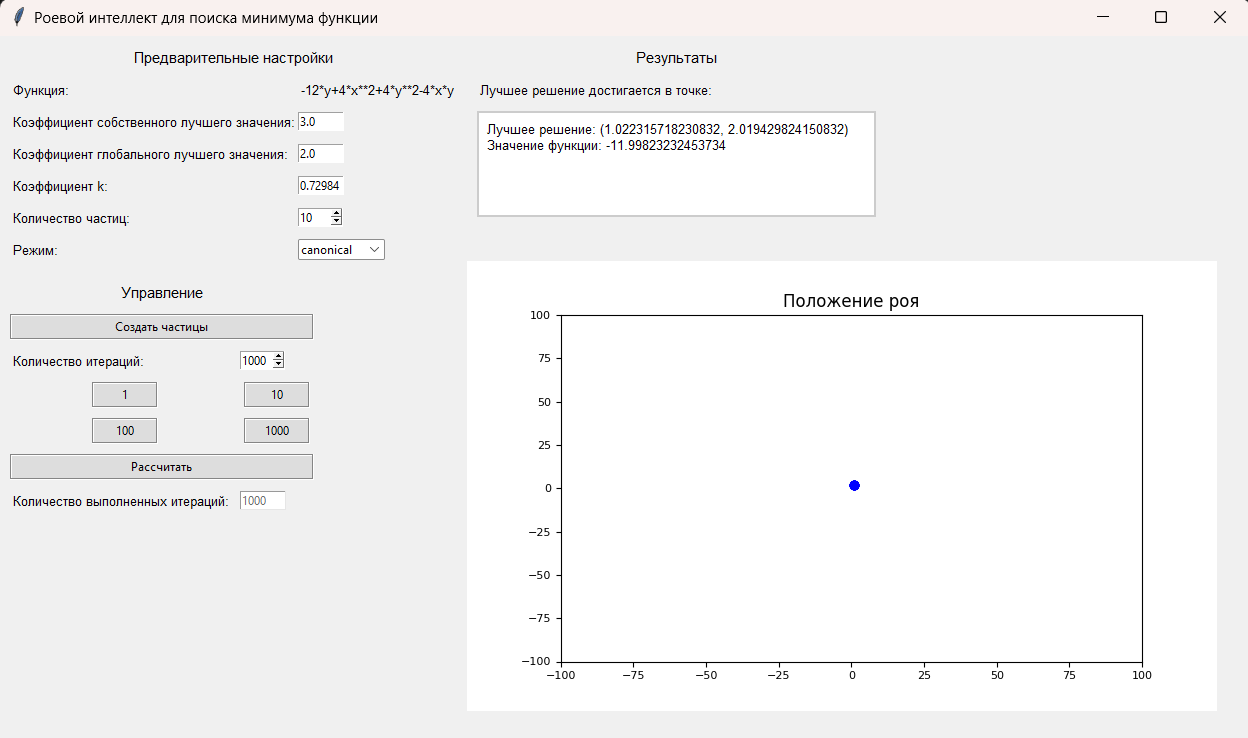


*Рисунок 9.2 РА: 5 итераций. Лучшее значение ф-ции около -6.*

Как видно алгоритм роя частиц показывает сильно лучшую скорость схождения. Однако после 1000 итераций – разница результатов минимальна. (рис. 9.3-9.4)



*Рисунок 9.3 ГА: 1000 итераций. Лучшее значение ф-ции около -11.99.*



*Рисунок 9.4 РА: 1000 итераций. Лучшее значение ф-ции около -11.99.*

Однако, стоит подчеркнуть, что полученные выводы касаются только заданной тестовой функции. Для экстраполяции этих выводов на другие функции требуется намного большее количество экспериментов.

1. **Вывод**

В ходе работы был создан алгоритм роя частиц с модификацией в виде коэффициента сжатия, а также получены базовые знания в создании пользовательского интерфейса. Были проанализированы результаты исследований, которые подтвердили теорию. В результате были определены некоторые оптимальные параметры РА для заданной тестовой функции, оценено влияние модифицированной части на результат, а также проведено сравнение эффективности классического генетического алгоритма с представленным алгоритмом роя частиц.

1. **Листинг кода**

import numpy as np

import tkinter as tk

from tkinter import ttk

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import matplotlib.pyplot as plt

import random

import math

​

class Particle:

   def \_\_init\_\_(self, x\_range, y\_range):

       self.x = random.uniform(x\_range[0], x\_range[1])

       self.y = random.uniform(y\_range[0], y\_range[1])

       self.best\_value = None

       self.best\_x = self.x

       self.best\_y = self.y

       # Инициализация скоростей

       self.vx = random.uniform(-1, 1)

       self.vy = random.uniform(-1, 1)

   def update\_best\_value(self, value):

       if self.best\_value is None or value < self.best\_value:

           self.best\_value = value

           self.best\_x = self.x

           self.best\_y = self.y

​

class Swarm:

   def \_\_init\_\_(self, num\_particles, x\_range, y\_range, c1, c2, k, mod):

       self.global\_best\_value = None

       self.global\_best\_x = None

       self.global\_best\_y = None

       self.mod = mod

       self.particles = [Particle(x\_range, y\_range) for \_ in range(num\_particles)]

       self.c1 = c1

       self.c2 = c2

       # Расчет коэффициента сжатия

       phi = c1 + c2

       if phi <= 4:

           raise ValueError("Сумма коэффициентов c1 и c2 должна быть больше 4.")

       self.chi = (2 \* k) / abs(2 - phi - math.sqrt(phi\*\*2 - 4 \* phi))

   def update\_global\_best(self):

       for particle in self.particles:

           if self.global\_best\_value is None or particle.best\_value < self.global\_best\_value:

               self.global\_best\_value = particle.best\_value

               self.global\_best\_x = particle.best\_x

               self.global\_best\_y = particle.best\_y

   def update\_velocities(self):

       for particle in self.particles:

           r1 = random.random()

           r2 = random.random()

           if self.mod == 'canonical':

               # Обновление скорости для x

               particle.vx = self.chi \* (particle.vx +

                                       self.c1 \* r1 \* (particle.best\_x - particle.x) +

                                       self.c2 \* r2 \* (self.global\_best\_x - particle.x))

               # Обновление скорости для y

               particle.vy = self.chi \* (particle.vy +

                                       self.c1 \* r1 \* (particle.best\_y - particle.y) +

                                       self.c2 \* r2 \* (self.global\_best\_y - particle.y))

​

           elif self.mod == 'classic':

               particle.vx = (particle.vx +

                                       self.c1 \* r1 \* (particle.best\_x - particle.x) +

                                       self.c2 \* r2 \* (self.global\_best\_x - particle.x))

               particle.vy = (particle.vy +

                                       self.c1 \* r1 \* (particle.best\_y - particle.y) +

                                       self.c2 \* r2 \* (self.global\_best\_y - particle.y))

​

   def update\_positions(self):

       for particle in self.particles:

           particle.x += particle.vx

           particle.y += particle.vy

   def run\_pso(self, objective\_function, num\_iterations):

       for \_ in range(num\_iterations):

           for particle in self.particles:

               # Вычисление значения функции для текущей позиции частицы

               current\_value = objective\_function(particle.x, particle.y)

               # Обновление лучшего значения частицы

               particle.update\_best\_value(current\_value)

           # Обновление глобального лучшего значения

           self.update\_global\_best()

           # Обновление скоростей частиц

           self.update\_velocities()

           # Обновление позиций частиц

           self.update\_positions()

​

def inserted(place, num):

   place.delete(0, tk.END)

   place.insert(0, str(num))

​

def creating():

   global swarm, ax, scatter, canvas\_widget

​

   # Использование данных из интерфейса

   c1 = float(c1\_entry.get())

   c2 = float(c2\_entry.get())

   k = float(k\_entry.get())

   num\_particles = int(num\_particles\_entry.get())

   mod = mod\_var.get()

​

   # Создание роя

   swarm = Swarm(num\_particles, [-100, 100], [-100, 100], c1, c2, k, mod)

​

   # Визуализация

   ax.clear()

   ax.set\_xlim(-100, 100)

   ax.set\_ylim(-100, 100)

   ax.set\_title("Положение роя")

   scatter = ax.scatter([], [], color='blue', marker='o')

   canvas\_widget.get\_tk\_widget().update\_idletasks()

   positions = np.array([[particle.x, particle.y] for particle in swarm.particles])

   scatter.set\_offsets(positions)

   canvas\_widget.draw()

​

def make\_iterations(n):

   global swarm, ax, scatter, canvas\_widget

​

   # Обновление счетчика совершенных итераций

   it = int(txt4.get())

   txt4.delete(0, tk.END)

   entry\_var.set(str(it + n))

​

   for \_ in range(n):

       for particle in swarm.particles:

           # Вычисление значения функции для текущей позиции частицы

           current\_value = objective\_function(particle.x, particle.y)

           # Обновление лучшего значения частицы

           particle.update\_best\_value(current\_value)

       # Обновление глобального лучшего значения

       swarm.update\_global\_best()

       # Обновление скоростей частиц

       swarm.update\_velocities()

       # Обновление позиций частиц

       swarm.update\_positions()

​

   # Обновление графика

   positions = np.array([[particle.x, particle.y] for particle in swarm.particles])

   scatter.set\_offsets(positions)

   canvas\_widget.draw()

   best\_position = (swarm.global\_best\_x, swarm.global\_best\_y)

   best\_fitness = swarm.global\_best\_value

​

   canvas2.delete("all")

   canvas2.create\_text(10, 10, anchor="nw", text=f"Лучшее решение: {best\_position}\nЗначение функции: {best\_fitness}",

                       font=("Arial", 10), fill="black")

​

root = tk.Tk()

root.title("Роевой интеллект для поиска минимума функции")

root.geometry('1250x800')

​

# Создание фреймов для разных разделов

left\_frame = tk.Frame(root)

left\_frame.grid(row=0, column=0, sticky="nsew")

​

left\_frame\_low = tk.Frame(root)

left\_frame\_low.grid(row=1, column=0, sticky="nsew")

​

center\_frame = tk.Frame(root)

center\_frame.grid(row=0, column=1, sticky="nsew")

​

center\_frame\_low = tk.Frame(root)

center\_frame\_low.grid(row=1, column=1, sticky="nsew")

​

right\_frame = tk.Frame(root)

right\_frame.grid(row=0, column=2, sticky="nsew")

​

# Настройка растягивания строк и столбцов

root.grid\_rowconfigure(0, weight=0)

root.grid\_columnconfigure(1, weight=1)

​

# Предварительные настройки

lbl = tk.Label(left\_frame, text="Предварительные настройки", font=("Arial", 11))

lbl.grid(row=0, column=0, columnspan=2, pady=(10, 0))

​

lblfunc = tk.Label(left\_frame, text="Функция:", font=("Arial", 10))

lblfunc.grid(row=1, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

lblfunc2 = tk.Label(left\_frame, text="-12\*y+4\*x\*\*2+4\*y\*\*2-4\*x\*y", font=("Arial", 10))

lblfunc2.grid(row=1, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

​

lblc1 = tk.Label(left\_frame, text="Коэффициент cобственного лучшего значения:", font=("Arial", 10))

lblc1.grid(row=2, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

c1\_entry = tk.Entry(left\_frame, width=7)

c1\_entry.insert(0, "3.0")

c1\_entry.grid(row=2, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

​

lblc2 = tk.Label(left\_frame, text="Коэффициент глобального лучшего значения:", font=("Arial", 10))

lblc2.grid(row=3, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

c2\_entry = tk.Entry(left\_frame, width=7)

c2\_entry.insert(0, "2.0")

c2\_entry.grid(row=3, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

​

lblk = tk.Label(left\_frame, text="Коэффициент k:", font=("Arial", 10))

lblk.grid(row=4, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

k\_entry = tk.Entry(left\_frame, width=7)

k\_entry.insert(0, "0.72984")

k\_entry.grid(row=4, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

​

lblnum\_particles = tk.Label(left\_frame, text="Количество частиц:", font=("Arial", 10))

lblnum\_particles.grid(row=5, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

num\_particles\_entry = tk.Spinbox(left\_frame, from\_=10, to=500, width=5)

num\_particles\_entry.grid(row=5, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

​

lblmod = tk.Label(left\_frame, text="Режим:", font=("Arial", 10))

lblmod.grid(row=6, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

mod\_var = tk.StringVar(value='canonical')

mod\_entry = ttk.Combobox(left\_frame, textvariable=mod\_var, values=['classic', 'canonical'], state='readonly', width=10)

mod\_entry.grid(row=6, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

​

# Управление

lbl4 = tk.Label(left\_frame\_low, text="Управление", font=("Arial", 11))

lbl4.grid(row=7, column=0, columnspan=2, pady=(20, 0))

​

but = tk.Button(left\_frame\_low, text="Создать частицы", width=42,

               command=creating,

               bg="#DDDDDD", activebackground="#CCCCCC", relief=tk.GROOVE)

but.grid(row=8, column=0, columnspan=2, padx=(10, 10), pady=(10, 0))

​

lbl5 = tk.Label(left\_frame\_low, text="Количество итераций:", font=("Arial", 10))

lbl5.grid(row=9, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

cntIt = tk.Spinbox(left\_frame\_low, from\_=1, to=5000, width=5)

cntIt.grid(row=9, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

​

but1 = tk.Button(left\_frame\_low, text="1", width=8,

                command=lambda: inserted(cntIt, 1),

                bg="#DDDDDD", activebackground="#CCCCCC", relief=tk.GROOVE)

but1.grid(row=10, column=0, padx=(10, 0), pady=(10, 0))

​

but2 = tk.Button(left\_frame\_low, text="10", width=8,

                command=lambda: inserted(cntIt, 10),

                bg="#DDDDDD", activebackground="#CCCCCC", relief=tk.GROOVE)

but2.grid(row=10, column=1, padx=(0, 10), pady=(10, 0))

​

but3 = tk.Button(left\_frame\_low, text="100", width=8,

                command=lambda: inserted(cntIt, 100),

                bg="#DDDDDD", activebackground="#CCCCCC", relief=tk.GROOVE)

but3.grid(row=11, column=0, padx=(10, 0), pady=(10, 0))

​

but4 = tk.Button(left\_frame\_low, text="1000", width=8,

                command=lambda: inserted(cntIt, 1000),

                bg="#DDDDDD", activebackground="#CCCCCC", relief=tk.GROOVE)

but4.grid(row=11, column=1, padx=(0, 10), pady=(10, 0))

​

but = tk.Button(left\_frame\_low, text="Рассчитать", width=42,

               command=lambda: make\_iterations(int(cntIt.get())),

               bg="#DDDDDD", activebackground="#CCCCCC", relief=tk.GROOVE)

but.grid(row=12, column=0, columnspan=2, padx=(10, 10), pady=(10, 0))

​

lbl5 = tk.Label(left\_frame\_low, text="Количество выполненных итераций:", font=("Arial", 10))

lbl5.grid(row=13, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

cnt = 0

entry\_var = tk.StringVar()

txt4 = tk.Entry(left\_frame\_low, width=7, textvariable=entry\_var, state='disabled', disabledbackground="white", fg="black")

txt4.grid(row=13, column=1, padx=(0, 10), pady=(10, 0), sticky="w")

entry\_var.set(str(cnt))

​

# Результаты

lbl6 = tk.Label(center\_frame, text="Результаты", font=("Arial", 11))

lbl6.grid(row=0, column=0, pady=(10, 0))

​

lbl5 = tk.Label(center\_frame, text="Лучшее решение достигается в точке:", font=("Arial", 10))

lbl5.grid(row=1, column=0, padx=(10, 0), pady=(10, 0), sticky="w")

​

canvas2 = tk.Canvas(center\_frame, width=393, height=100, bg="white", borderwidth=1, highlightbackground="#CCCCCC", highlightthickness=2)

canvas2.grid(row=2, column=0, padx=(10, 10), pady=(10, 0))

​

# График

fig, ax = plt.subplots(figsize=(7.5, 4.5))

ax.set\_facecolor('white')

ax.tick\_params(axis='both', labelsize=8)

​

ax.set\_xlim(-100, 100)

ax.set\_ylim(-100, 100)

ax.set\_title("Положение роя")

​

canvas\_widget = FigureCanvasTkAgg(fig, master=center\_frame\_low)

canvas\_widget.get\_tk\_widget().grid(row=0, column=0, sticky="nsew")

​

# Определение целевой функции

def objective\_function(x, y):

   return -12\*y + 4\*x\*\*2 + 4\*y\*\*2 - 4\*x\*y

​

root.mainloop()

​