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

**Вариант – 5**

**Студент гр. 23Б16-пу**

**Горыцнцев Р.Н.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

Оглавление

1. [Цель работы 3](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013329)
2. [Описание задачи (формализация задачи) 3](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013330)
3. [Теоретическая часть 3](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013331)
4. [Основные шаги программы 6](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013332)
5. [Блок схема программы 7](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013333)
6. [Описание программы 8](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013334)
7. [Рекомендации пользователя 9](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013335)
8. [Контрольный пример 10](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013338)
9. [Анализ результатов работы 11](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013339)
10. [Сравнение ГА с алгоритмом роевого интеллекта 15](#_Toc181658801)
11. [Вывод 16](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013340)
12. [Листинг 17](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013340)

# **Цель работы**

Цель данной лабораторной работы — исследование особенностей алгоритмов роевого интеллекта для решения задач глобальной оптимизации и сравнение с генетическим алгоритмом.

# **Описание задачи (формализация задачи)**

1. Изучение особенностей кодирования роевого интеллекта

* Изучить принципы работы роевого алгоритма.
* Ознакомиться с основными параметрами: коэффициенты инерции, когнитивный и социальный компоненты, ограничение скорости.

2. Написание программы поиска минимума функции

* Разработать программу на Python для поиска минимума функции.
  1. Класс для представления частиц.
  2. Функция для реализации алгоритма.
  3. Графический интерфейс

3. Тестирование программы

* Запустить программу с различными параметрами.
* Проанализировать сходимость, точность и время выполнения.

# **Теоретическая часть**

Роевой алгоритм — метод оптимизации, основанный на коллективном поведении группы частиц, аналогичном поведению косяков рыб или стай птиц. Этот метод относится к эволюционным алгоритмам и применяется для решения задач оптимизации в сложных многомерных пространствах. Основная идея заключается в том, что группа частиц (потенциальных решений) исследует пространство поиска, стремясь найти глобально оптимальное решение за счёт взаимодействия между отдельными частицами.

**Основные принципы работы роевого алгоритма**

Каждая частица в роевом алгоритме представлена как потенциальное решение задачи и характеризуется определённой позицией и скоростью. Позиция частицы в пространстве решений определяет текущее значение целевой функции, которая оценивает качество решения. Частицы обновляются путём изменения положения и скорости, что позволяет рою постепенно сходиться к оптимальному решению.

Алгоритм работает следующим образом:

* Инициализация — частицы случайным образом распределяются в пространстве решений.
* Оценка приспособленности — каждая частица вычисляет значение целевой функции в своей текущей позиции. Это значение используется для определения личного и глобального наилучшего решения.
* Обновление скорости и положения — для каждой частицы обновляются параметры с учетом личного и глобального наилучшего положения, а также случайных факторов.

**Параметры роевого алгоритма**

* Коэффициент инерции (w) — влияет на баланс между глобальным и локальным поиском. Высокие значения увеличивают диапазон поиска, а низкие — способствуют сходимости.
* Коэффициенты социального и когнитивного ускорения (c1 и c2) — управляют скоростью, с которой частица стремится к личному и глобальному лучшему положению.
* Количество частиц и итераций — параметры, определяющие продолжительность и детализацию процесса поиска оптимума.

**Применение роевого алгоритма**

Роевой алгоритм применяется для решения различных задач, таких как:

* Оптимизация сложных функций;
* Обучение моделей машинного обучения;
* Комбинаторные задачи, включая планирование и маршрутизацию;
* Управление ресурсами и процессы распределения в сетевых системах.

**Модификация для ограничения скорости**

В этом алгоритме используется модификация, ограничивающая скорость каждой частицы, чтобы она не превышала определённое максимальное значение. Это сделано для контроля перемещения частиц и предотвращения слишком резких изменений их положения, которые могут привести к непредсказуемому поведению и ухудшению поиска. Если скорость частицы на следующем шаге превышает допустимый предел, её значение снижается до максимально допустимого уровня, при этом сохраняется направление движения частицы. Благодаря такому ограничению частицы перемещаются более стабильно и плавно, что улучшает сходимость алгоритма, помогая избежать слишком большого разброса частиц и способствуя эффективному поиску оптимума.

# **Основные шаги программы**

Генерация начальной популяции:

* Случайная генерация начальных позиций и скоростей частиц в заданном диапазоне.

Определение целевой функции:

* Вычисление значения тестовой функции для каждой частицы, что определяет её пригодность.

Обновление лучшего положения:

* Обновление личного лучшего положения частицы и глобального лучшего положения всего роя, если текущая позиция улучшает результат.

Обновление скорости частиц:

* Скорость каждой частицы обновляется с учётом её личного и глобального наилучшего положения, а также случайного фактора для улучшения поиска.

Обновление положения частиц:

* Положение каждой частицы обновляется на основе новой скорости.

Итерационный процесс:

* Повторение шагов обновления скоростей и позиций до достижения заданного числа итераций или условия сходимости.

Визуализация процесса:

* Отображение движения частиц на графике с помощью matplotlib, обновляемого на каждой итерации.

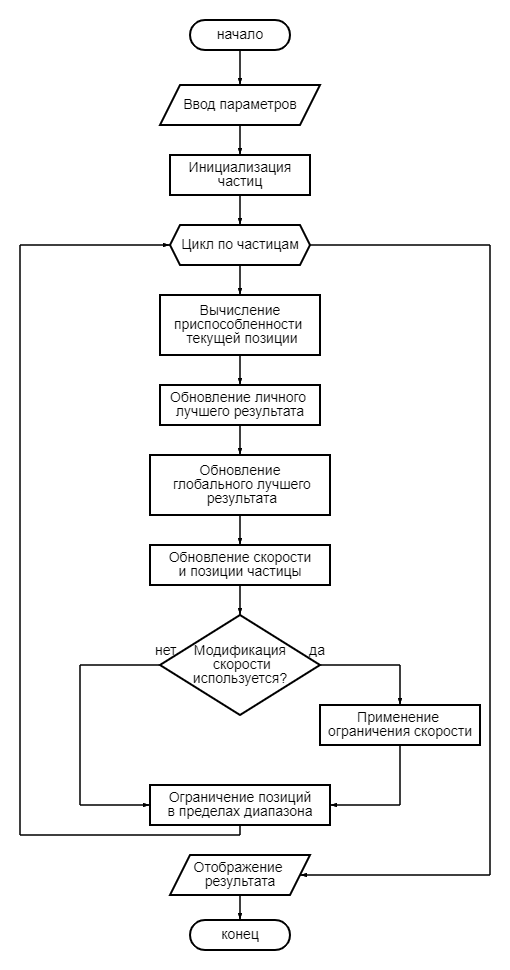
Ввод параметров пользователем:

* Пользователь вводит параметры алгоритма (число частиц, итерации, коэффициенты).

Отображение результатов:

* Вывод конечного положения частиц и оптимального значения целевой функции.

# **Блок схема программы**



*Рис 1: Блок схема программы*

# **Описание программы**

Программа реализована на языке Python 3.12 с использованием следующих библиотек: tkinter, numpy, matplotlib. В процессе разработки программы использовались следующие функции и классы:

*Таблица 1. lab\_5.py*

|  |  |  |
| --- | --- | --- |
| Функция/Класс | Описание | Возвращаемое значение |
| test\_function | Тестовая функция, которая будет оптимизирована. | float |
| Particle | Класс, представляющий частицу в алгоритме PSO. Содержит информацию о позиции, скорости, лучшей позиции и лучшем значении функции. | None |
| pso | Основная функция алгоритма PSO, которая выполняет оптимизацию. | tuple |
| GUI | Класс, реализующий графический интерфейс пользователя (GUI). | None |

*Таблица 2. Класс Particle*

|  |  |
| --- | --- |
| Метод | Описание |
| \_\_init\_\_ | Инициализация объекта частицы с начальной позицией и скоростью. |
| position | Текущая позиция частицы в пространстве поиска. |
| velocity | Текущая скорость частицы. |
| best\_position | Лучшая позиция, которую частица нашла на данный момент. |
| best\_value | Значение функции в лучшей позиции частицы. |

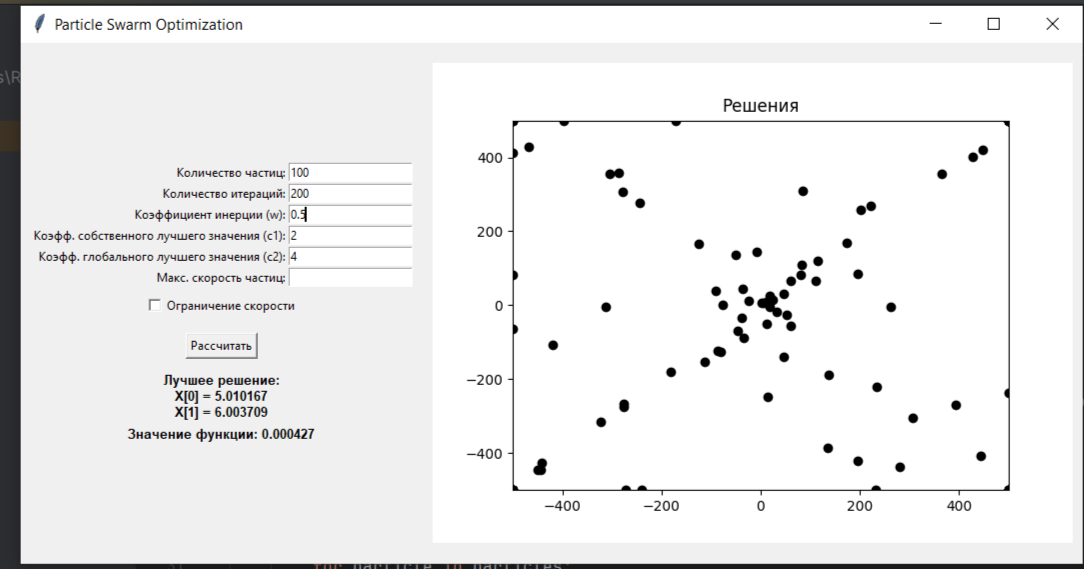
*Таблица 3. Класс GUI*

|  |  |
| --- | --- |
| Метод | Описание |
| \_\_init\_\_ | Инициализирует графический интерфейс пользователя (GUI). Создает окно приложения, размещает элементы управления для ввода параметров PSO, кнопку для запуска расчета, а также область для отображения графика. |
| start\_pso | Запускает алгоритм PSO с параметрами, введенными пользователем. После завершения расчета обновляет отображение результатов и строит график расположения частиц. Если введены некорректные данные, выводит сообщение об ошибке. |

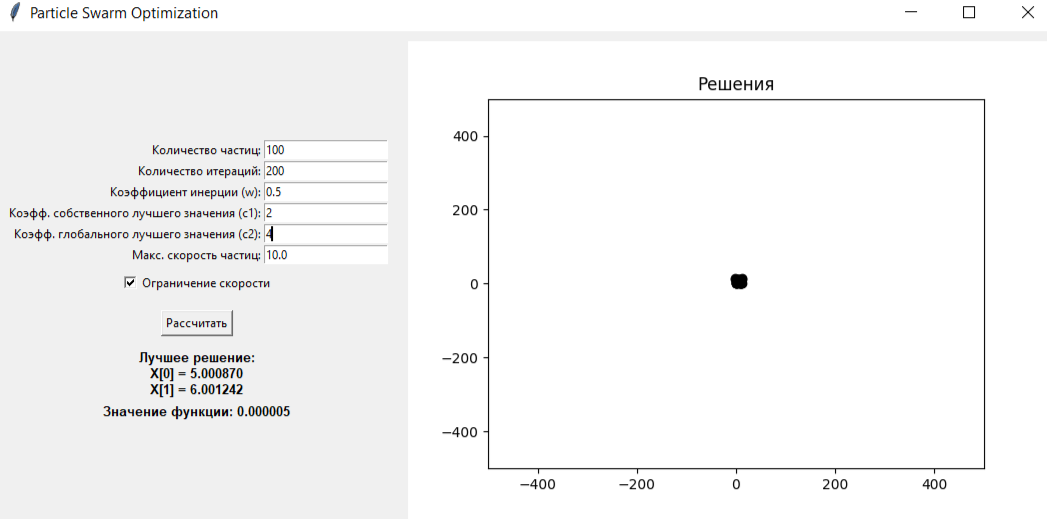
# **Рекомендации пользователя**

Установите библиотеки tkinter, numpy, matplotlib, random. Перед запуском программы задайте параметры алгоритма: количество частиц, итераций, коэффициенты инерции и ускорения, диапазоны начальных положений и скоростей. Запустите программу, введите параметры через интерфейс и проверьте визуализацию процесса. После завершения убедитесь, что поля «Лучшее решение» и «Значение функции» отображаются корректно. При необходимости измените параметры и повторите запуск для улучшения результата.

# **Контрольный пример**



*Рис 2: пример окна программы без модификации*



*Рис 3: пример окна программы с модификацией*

# **Анализ результатов работы:**

*Таблица 4: тесты без ограничения скорости*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество частиц | Количество итераций | Коэффициент инерции | Коэф. собств. лучшего значения | Коэф. глобал. лучшего значения | Лучшее решение | Значение в лучшей точке |
| 50 | 100 | 0.3 | 5 | 5 | (5.129501; 5.276797) | 0.590105 |
| 100 | 200 | 0.7 | 2 | 2.5 | (5.000009; 5.999999) | 0.000001 |
| 10 | 50 | 0.9 | 3 | 1 | (6.046262; 7.338538) | 6.170338 |
| 300 | 1000 | 0.4 | 2.2 | 3.5 | (5.000007; 6.000028) | 0.000000 |
| 50 | 500 | 0.1 | 3 | 1 | (5.000000; 6.000000) | 0.000000 |
| 100 | 1000 | 0.6 | 1.5 | 2.5 | (5.000000; 6.000000) | 0.000000 |
| 200 | 200 | 0.8 | 1.8 | 3.5 | (5.131727; 6.437415) | 0.260740 |
| 300 | 500 | 0.5 | 2.2 | 2 | (5.000000; 6.000000) | 0.000000 |
| 50 | 1000 | 0.7 | 1.5 | 3 | (5.000000; 6.000000) | 0.000000 |
| 100 | 500 | 0.4 | 4 | 2.5 | (4.998515; 6.020009) | 0.000409 |

*Таблица 5: тесты с ограничением скорости*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Количество частиц | Количество итераций | Коэффициент инерции | Коэф. собств. лучшего значения | Коэф. глобал. лучшего значения | Макс.  скорость частиц | Лучшее решение | Значение в лучшей точке |
| 50 | 100 | 0.3 | 5 | 5 | 1 | (-0.081233; 27.687657) | 573.630198 |
| 100 | 200 | 0.7 | 2 | 2.5 | 2 | (4.999994; 6.000021) | 0.000000 |
| 10 | 50 | 0.9 | 3 | 1 | 0.5 | (-35.970726; -0.349906) | 6754.722821 |
| 300 | 1000 | 0.4 | 2.2 | 3.5 | 5 | (5.000000; 6.000000) | 0.000000 |
| 50 | 500 | 0.1 | 3 | 1 | 10 | (5.000000; 6.000000) | 0.000000 |
| 100 | 1000 | 0.6 | 1.5 | 2.5 | 3 | (5.000000; 6.000000) | 0.000000 |
| 200 | 200 | 0.8 | 1.8 | 3.5 | 4 | (5.00297; 5.998678) | 0.000036 |
| 300 | 500 | 0.5 | 2.2 | 2 | 6 | (5.000000; 6.000000) | 0.000000 |
| 50 | 1000 | 0.7 | 1.5 | 3 | 15 | (5.000000; 6.000000) | 0.000000 |
| 100 | 500 | 0.4 | 4 | 2.5 | 8 | (4.999978; 5.999819) | 0.000000 |

**1. Влияние ограничения скорости частиц:**

* Без ограничения скорости:
  + В некоторых случаях (например, при 10 частицах и 50 итерациях) алгоритм находит далекие от оптимального решения, что может свидетельствовать о том, что частицы слишком быстро "улетают" от оптимальной области.
  + При большом количестве частиц и итераций (например, 300 частиц и 1000 итераций) алгоритм находит оптимальное решение, что говорит о том, что большое количество частиц и итераций помогает стабилизировать поиск.
* С ограничением скорости:
  + Ограничение скорости частиц значительно улучшает стабильность поиска. В большинстве случаев (например, при 100 частицах и 200 итерациях, 300 частицах и 1000 итерациях) алгоритм находит оптимальное решение.
  + При низком ограничении скорости (например, 0.5 для 10 частиц и 50 итераций) алгоритм находит далекое от оптимального решение, что может свидетельствовать о том, что ограничение слишком сильное и замедляет поиск.
  + При высоком ограничении скорости (например, 15 для 50 частиц и 1000 итераций) алгоритм также находит оптимальное решение, что говорит о том, что умеренное ограничение скорости помогает стабилизировать поиск.

**2. Влияние количества частиц и итераций:**

Количество частиц:

* + При малом количестве частиц (например, 10) алгоритм часто находит далекие от оптимального решения, что может свидетельствовать о недостаточном разнообразии поиска.
  + При большом количестве частиц (например, 300) алгоритм находит оптимальное решение, что говорит о том, что большое количество частиц увеличивает вероятность нахождения глобального минимума.

Количество итераций:

* При малом количестве итераций (например, 50) алгоритм часто находит далекие от оптимального решения, что может свидетельствовать о недостаточном времени для поиска.
* При большом количестве итераций (например, 1000) алгоритм находит оптимальное решение, что говорит о том, что большое количество итераций увеличивает вероятность нахождения глобального минимума.

**3. Влияние коэффициентов инерции, собственного и глобального лучшего значения:**

1. Коэффициент инерции (w):

* Низкие значения (например, 0.1) могут замедлить поиск, так как частицы меньше "доверяют" своим предыдущим скоростям.
* Высокие значения (например, 0.9) могут привести к быстрому поиску, но также к колебаниям вокруг оптимального решения.
* Оптимальные значения обычно находятся в диапазоне 0.4-0.7.

1. Коэффициент собственного лучшего значения (c1):

* Низкие значения (например, 1.5) могут привести к недостаточному учету собственного опыта частицы.
* Высокие значения (например, 5) могут привести замедлению поиска глобального минимума.
* Оптимальные значения обычно находятся в диапазоне 1.5-2.5.

1. Коэффициент глобального лучшего значения (c2):

* Низкие значения (например, 1) могут привести к недостаточному учету глобального опыта.
* Высокие значения (например, 5) могут привести замедлению поиска глобального минимума.
* Оптимальные значения обычно находятся в диапазоне 2.0-3.5.

# **Сравнение ГА с алгоритмом роевого интеллекта:**

Для роевого алгоритма использовались следующие дефолтные данные:

* Количество частиц: 20
* Коэффициент инерции: 0.7
* Коэффициент собственного лучшего значения: 1.5
* Коэффициент глобального лучшего значения: 1.5

Для генетического алгоритма использовались следующие дефолтные данные:

* Кодировка: вещественная
* Кроссинговер: одноточечный
* Вероятность мутации: 0.01
* Количество хромосом: 50
* Минимальное значение гена: -10
* Максимальное значение гена: -10

*Таблица 6: сравнение алгоритмов*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество итераций/поколений | Лучшее решение ГА | Значение в лучшей точке в ГА | Лучшее решение РА | Значение в лучшей точке в РА |
| 50 | (4.930904710901;  4.650016580043) | 1.841551870060 | (4.997391; 6.003037) | 0.000036 |
| 100 | (4.730351837268;  5.862955544799) | 0.309621709360 | (4.999993; 5.999998) | 0.000000 |
| 200 | (4.878805618333;  6.715166428568) | 0.570215333141 | (5.000000; 6.000000) | 0.000000 |
| 500 | (5.046807125629;  6.087112900180) | 0.016352285416 | (5.000000; 6.000000) | 0.000000 |

Сравнение результатов показывает, что алгоритм роевого интеллекта в большинстве случаев находит более точное значение функции в оптимальной точке, чем генетический алгоритм, особенно при увеличении числа итераций/поколений. При меньшем количестве итераций (50 и 100) ГА выдает близкие к оптимальным решениям, но при увеличении до 200 и 500 итераций РА находит абсолютный минимум (значение функции в лучшей точке достигает 0.000000), в то время как ГА все еще немного отклоняется от него.

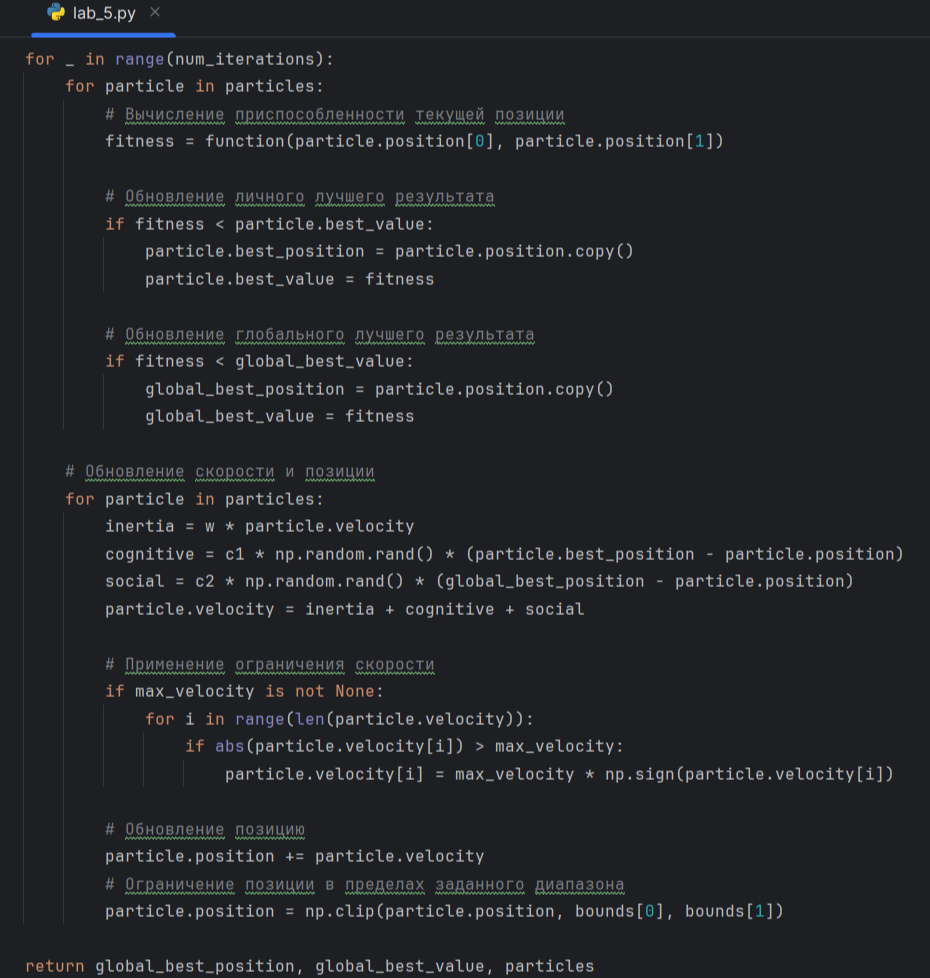
# **Вывод**

В ходе работы были исследованы алгоритмы роевого интеллекта для задач глобальной оптимизации и проведено их сравнение с генетическим алгоритмом. Алгоритмы роевого интеллекта показали более высокую эффективность, быстро сходясь к глобальному минимуму и лучше избегая локальных минимумов, что делает их предпочтительными для сложных задач. Генетический алгоритм, хотя и продемонстрировал хорошие результаты при ограниченном числе поколений, оказался менее устойчивым к локальным минимумам и сходился медленнее. Таким образом, алгоритмы роевого интеллекта более эффективны для задач глобальной оптимизации, хотя генетический алгоритм может быть полезен при многокритериальной оптимизации и поиске альтернативных решений.

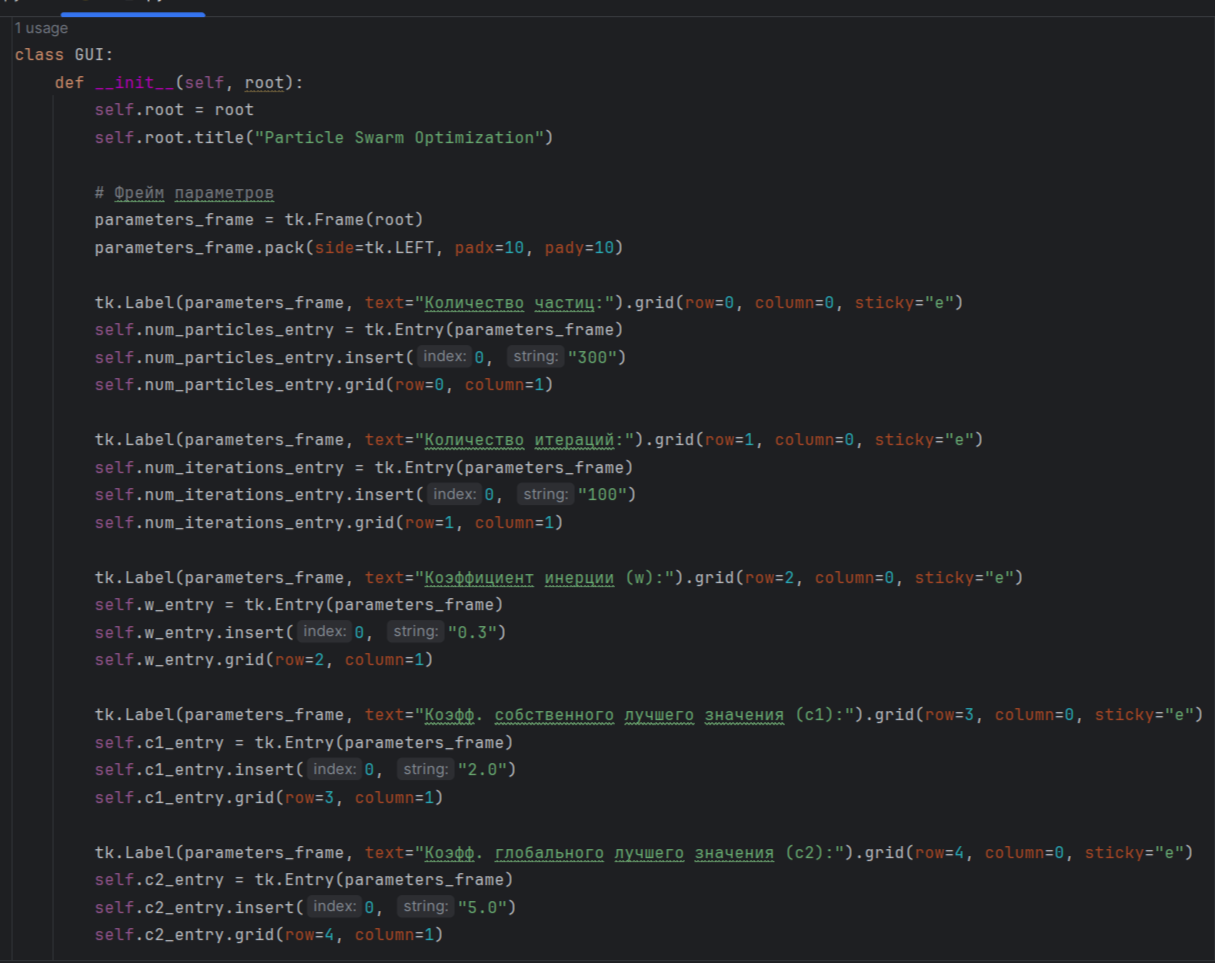
# **Листинг**



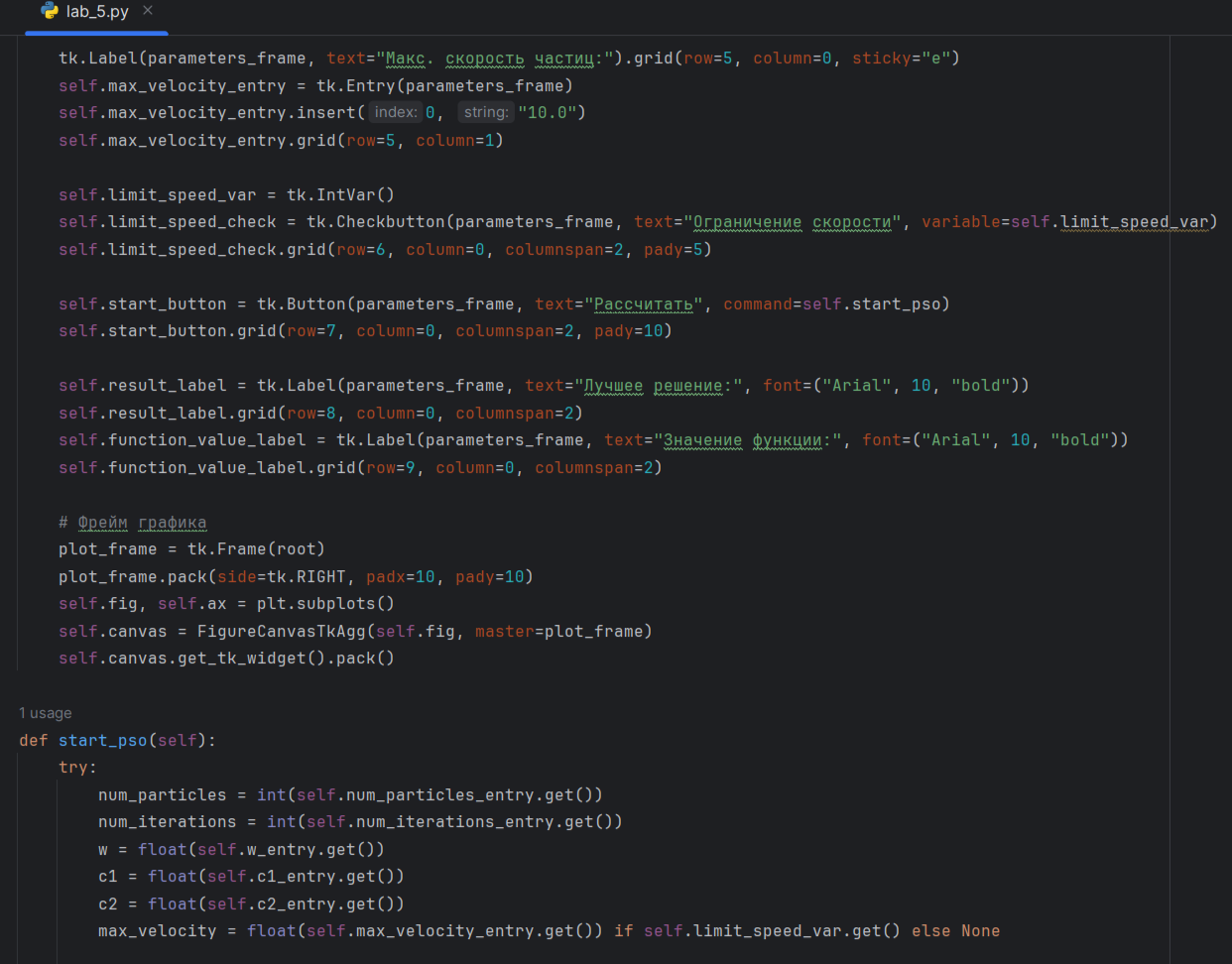
*Рис 4: Листинг*



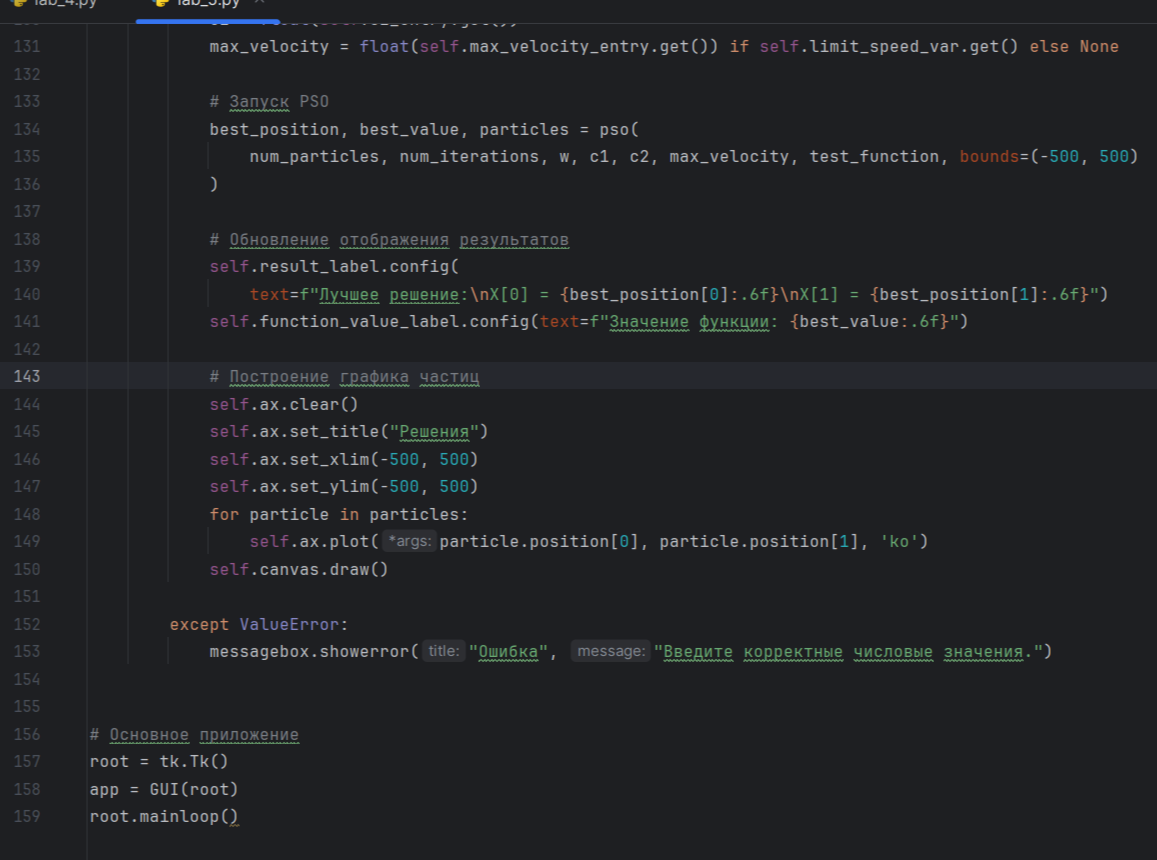
*Рис 5: Листинг*



*Рис 6: Листинг*



*Рис 7: Листинг*



*Рис 8: Листинг*