**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный университет**

**Факультет прикладной математики-процессов управления**

**Кафедра “фундаментальная информатика и информационные технологии”**

**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 – 3**

|  |  |  |
| --- | --- | --- |
| Студент гр. 23Б16-пу |  | Сулимов А.С. |
| Преподаватель |  | Дик А.Г. |

**2024 г**

**Оглавление**

[**Цель работы** 3](#_Toc181951402)

[**Задачи** 3](#_Toc181951403)

[**Теоретическая часть** 3](#_Toc181951404)

[**Описание алгоритма** 5](#_Toc181951405)

[**Описание программы** 7](#_Toc181951406)

[**Рекомендации для пользователя** 9](#_Toc181951407)

[**Рекомендации для программиста** 11](#_Toc181951408)

[**Контрольный пример** 13](#_Toc181951409)

[**Анализ результатов работы алгоритма** 15](#_Toc181951410)

[**Сравнение генетического алгоритма и алгоритма роя частиц** 23](#_Toc181951411)

[**Вывод** 25](#_Toc181951412)

# **Цель работы**

Разработать программу, которая находит глобальный минимум функции при помощи алгоритма роя частиц.

# **Задачи**

1. Изучить особенности алгоритма роевого интеллекта
2. Разработать алгоритм поиск минимума функции при помощи алгоритма роя частиц
3. Написать программу, которая находит глобальный минимум заданной функции и найти минимум заданной функции
4. Проанализировать результаты работы алгоритма
5. Сравнить результат работы роевого интеллекта с результатами работы генетического алгоритма.

# **Теоретическая часть**

Алгоритм роя частиц (PSO) представляет собой итеративный метод оптимизации, который направлен на нахождение оптимальных значений функции путем моделирования поведения группы частиц, имитирующих естественные процессы. В данном алгоритме каждая частица является представителем потенциального решения и перемещается по пространству поиска с заданной скоростью. Каждая частица фиксирует свою наилучшую позицию, а также лучшую позицию, достигнутую всей группой.

В процессе работы алгоритма на каждой итерации обновляются как положение, так и скорость каждой частицы, основываясь на её текущем положении, локально лучшей позиции (pbest) и глобально лучшей позиции (gbest) группы. Частицы взаимодействуют друг с другом, обмениваясь информацией о своих достижениях, что позволяет им совместно искать оптимальное решение задачи.

Для расчета скорости частицы используется формула:

, где:

– вектор скорости частицы

– Случайно значение в промежутке (0, 1)

– постоянная локального ускорения

– постоянная глобального ускорения

– точка локального оптимального значения

– точка глобального оптимального значения

– координаты точки частицы

Новая координата точке расчитывается по формуле:

, где:

– координаты точки частицы

– вектор скорости частицы

Алгоритм продолжается до тех пор, пока не будет достигнут заданный критерий остановки, например, по количеству итераций или при достижении заданного значения целевой функции.

**velocity limit**

Одной из модификаций алгоритма роя частиц является ограничение скорости частицы (velocity limit). Этот подход заключается во введении максимального значения для скорости перемещения частиц в пространстве поиска. Ограничение скорости позволяет предотвратить слишком резкие изменения положения частиц, которые могут привести к «проскоку» оптимальных областей и нестабильной сходимости алгоритма.

Основная цель данной модификации — повысить стабильность поиска и контролировать область поиска, ограничивая перемещения частиц в пределах допустимого диапазона. Это особенно важно при решении задач с узкими областями оптимума или при оптимизации функций с несколькими локальными минимумами, где высокие скорости могут снизить точность поиска.

Преимущества метода velocity limit включают:

1. Повышение устойчивости алгоритма, так как частицы меньше склонны покидать границы интересующих областей поиска.

2. Снижение вероятности попадания в состояние «хаотичного» движения, что может происходить при высоких скоростях, усложняя процесс поиска глобального минимума.

3. Повышение точности и скорости сходимости к оптимальному решению, так как частицы двигаются более управляемо и не перескакивают через оптимальные значения.

# **Описание алгоритма**

1. Инициализация роя частиц со случайными параметрами в заданных пределах.
2. Определение наилучшего глобального значения для всего роя и локального значения для каждой частицы.
3. Вычисление вектора скорости для каждой частицы на основе глобального значения роя.
4. Обновление координат частиц с учетом их текущих координат и векторов скорости.
5. Повторение шагов 2-4 до достижения заданного числа итераций.
6. Вывод глобального минимума функции и координат точки, в которой он достигается.

Такой подход позволяет алгоритму эффективно исследовать пространство решений и адаптироваться к изменениям, что делает его мощным инструментом для решения задач глобальной оптимизации.

**Блок-схема программы**

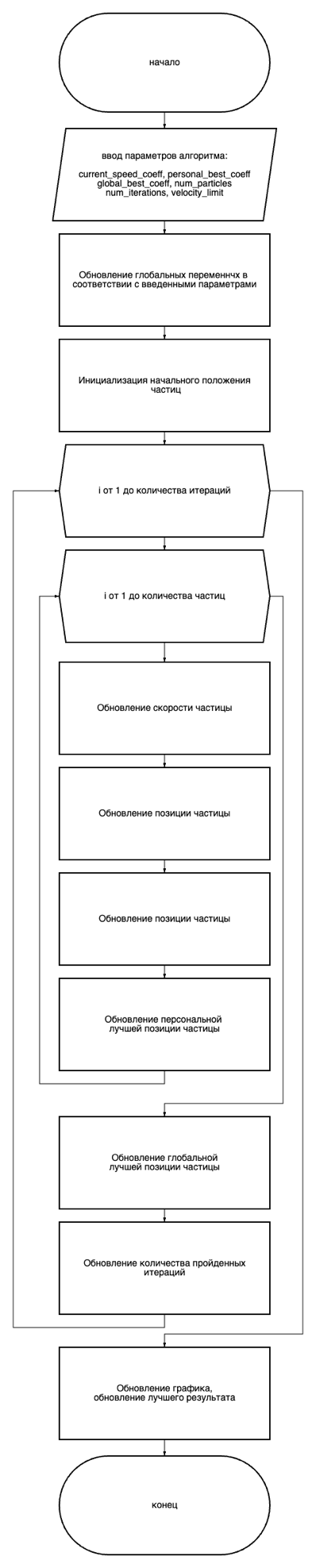


Рисунок 1. Блок схема программы

# **Описание программы**

Программа реализована при помощи языка python 3.11.4, с использованием библиотек: os, tkinter, sys, numpy, random, matplotlib. В программе использовались 2 класса, 15 функций и 4 структур данных. В таблице 1 представлено описание методов класса Particle.

Таблица 1. Описание методов класса Particle

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Тип возвращаемого значения** |
| update\_velocity | обновляет скорость частицы на основе текущей скорости, личного и глобального лучших положений | Float |
| update\_position |  обновляет положение частицы, ограничивая его в заданных границах. | Float |
| update\_personal\_best | обновляет личное лучшее положение и значение функции, если текущее положение лучше | Float |

В таблице 2 представлено описание методов класса GUI

Таблица 2. Описание методов класса GUI

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Тип возвращаемого значения** |
| create\_gui | Создает основной интерфейс приложения с полями для ввода параметров, кнопками для управления алгоритмом и местом для отображения графиков и результатов | None |
| draw\_swarm\_plot | строит график текущих позиций частиц и глобального лучшего положения | None |
| display\_best\_solution | обновляет метки на интерфейсе с лучшим найденным решением и значением функции | None |
| show | Получение количества поколений, которые необходимо вывести, запись параметров в таблицу | None |

В таблице 3 представлено описание функций

Таблица 3. Описание функций

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Тип возвращаемого значения** |
| initialize\_particles | инициализирует частицы и сбрасывает параметры | Particle |
| run\_iteration | выполняет одну итерацию алгоритма, обновляя положение и скорость всех частиц, а также глобальное лучшее решение | Particle |
| run\_iterations | Расчет вектора скорости для каждой частицы | Particle |
| fitness\_function | Определяет целевую функцию, которую необходимо минимизировать | Float |
| main | Основная функция | None |

В таблице 4 представлено описание структур данных.

Таблица 4. Описание структур данных

|  |  |  |
| --- | --- | --- |
| **Имя структуры** | **Тип структуры** | **Хранимые данные** |
| num\_particles | int | Число частиц, используемых в алгоритме |
| num\_iterations | int | Общее количество итераций, которое алгоритм будет выполнять |
| executed\_iterations | int | Счетчик для отслеживания текущего числа выполненных итераций |
| velocity\_limit | float | Максимальная скорость частиц |
| current\_speed\_coeff, personal\_best\_coeff, global\_best\_coeff | float | Параметры работы алгоритма |

# **Рекомендации для пользователя**

Для запуска программы необходимо наличие устройства с установленной операционной системой Linux, macOS или Windows, а также среды разработки, поддерживающей запуск python версии 3.12

После запуска приложения появится главное окно программы.

Вверху находится меню программы с кнопками (рисунок 2).

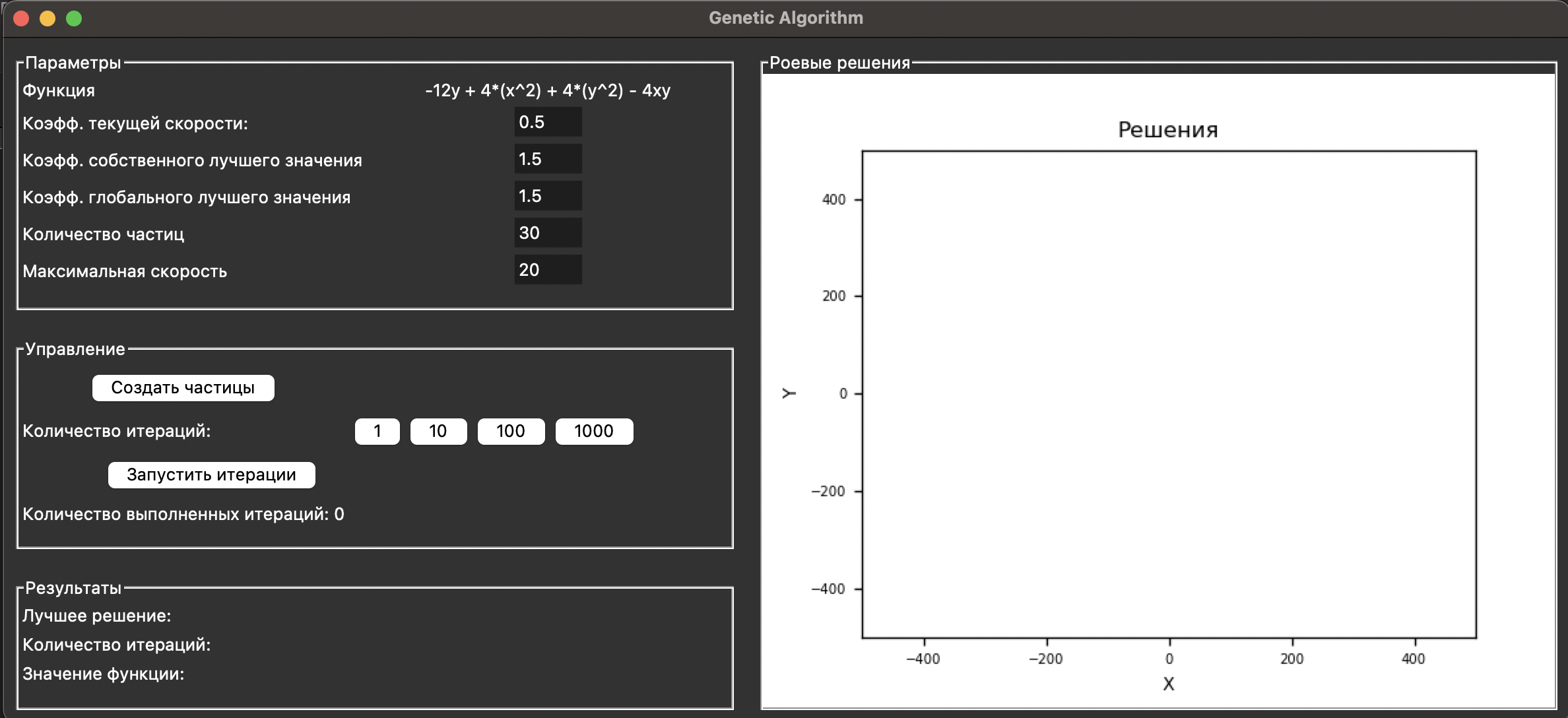


Рисунок 2. Основное окно программы

Параметры вводятся в окно ввода параметров (рис 3).

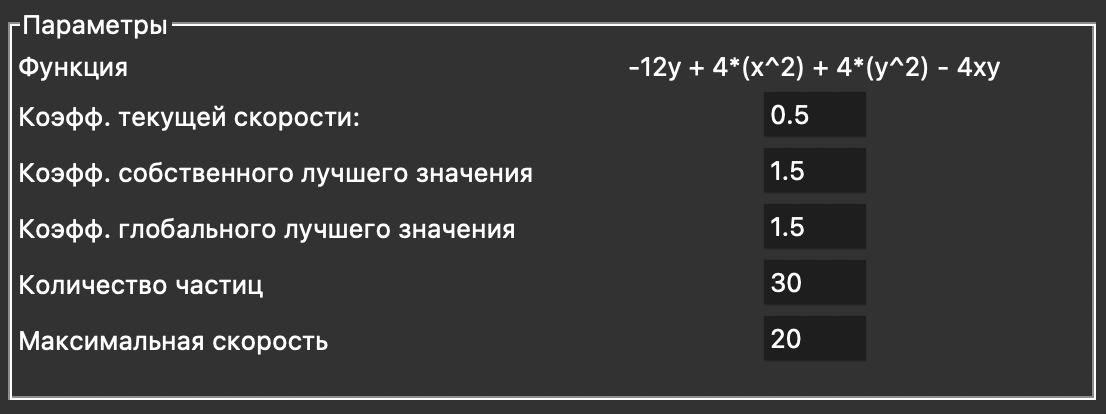


Рисунок 3. Окно ввода параметров

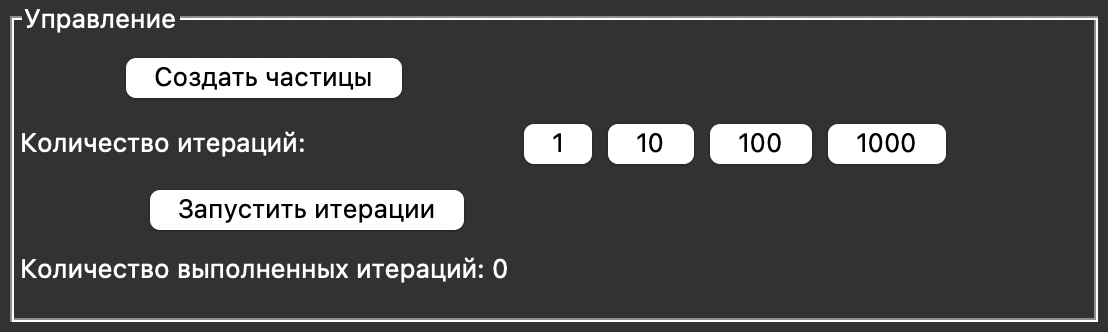
Управление частицами осуществляется с помощью кнопок (рис 4).

Рисунок 4. Окно управления частицами

Наблюдать за инициализацией частиц можно в специальном окне (рис 5).

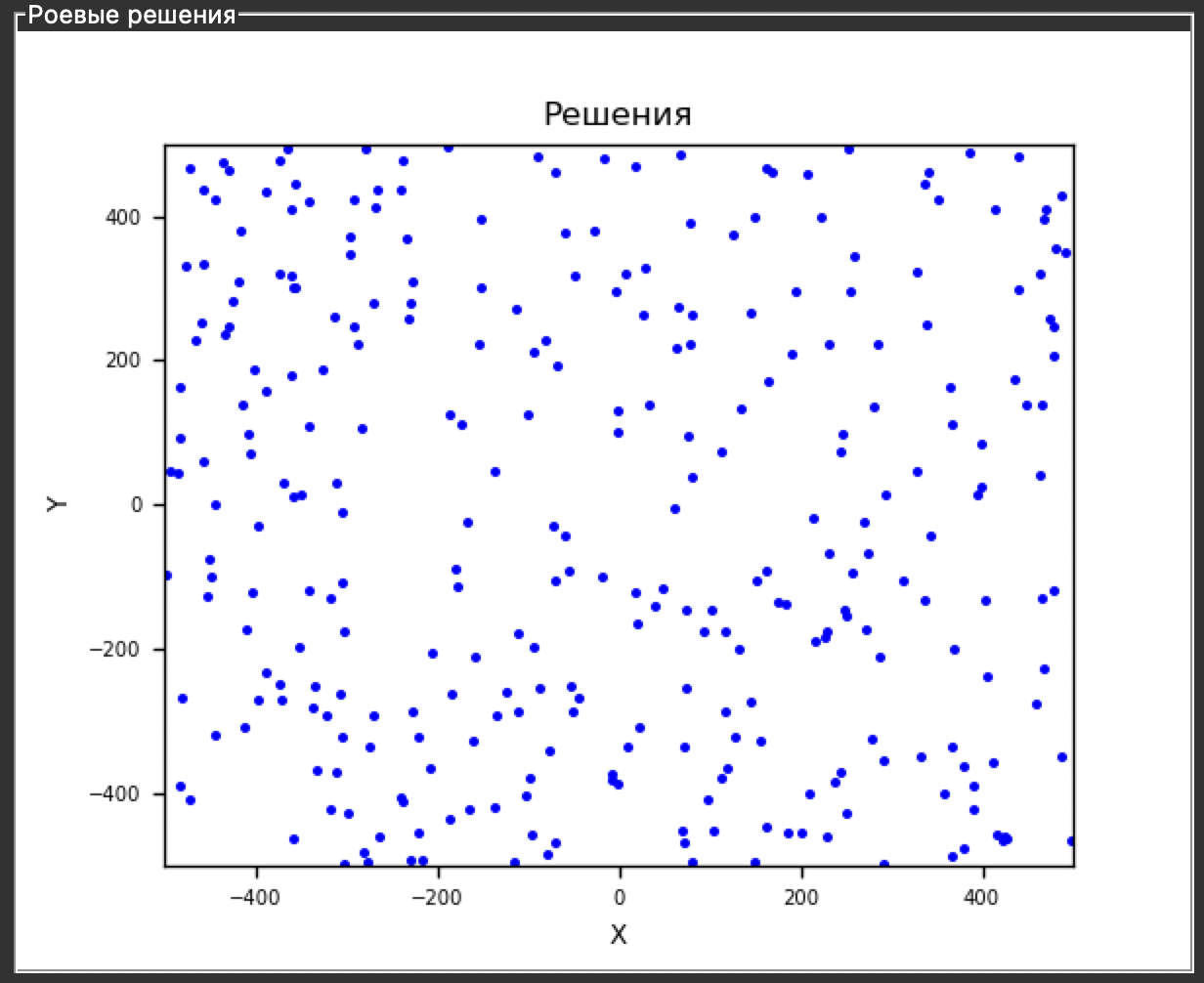


Рисунок 5. Окно показа частиц

Результаты обновляются каждый цикл и показываются в окне результатов (рис 6).

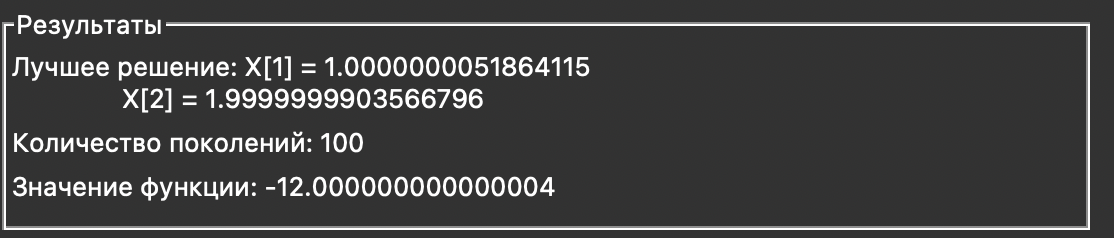


Рисунок 6. Окно результатов

# **Рекомендации для программиста**

Для внесения изменений в исходный код программы необходимо наличие устройство с установленное операционной системой Linux, macOS или Windows, а также среды разработки, поддерживающей запуск python версии 3.11.4. Так же убедитесь что установлены библиотеки tkinter, matplotlib, numpy.

**1. Организация и структура кода**

* **Модульность**: Подумайте о разделении кода на отдельные модули. Например, вы можете выделить классы и функции в отдельные файлы (например, particle.py, gui.py) для улучшения читаемости и удобства обслуживания.
* **Документация**: Добавьте краткие комментарии и документацию для классов и функций, чтобы пояснить их назначение. Это поможет в дальнейшем сопровождении кода.

**2. Оптимизация производительности**

* **Скорость выполнения**: Возможно, стоит использовать библиотеку numba или другие инструменты для ускорения вычислений, особенно если вы работаете с большим количеством частиц.

**3. Улучшение читаемости кода**

* **Константы**: Переменные, такие как пределы области поиска (-500, 500), лучше вынести в отдельные константы, чтобы их было легко изменить и они не терялись в коде.

**4. Интерфейс пользователя (UI)**

* **Масштабируемость интерфейса**: Если GUI используется на устройствах с разными разрешениями экрана, возможно, стоит использовать гибкую компоновку с динамической настройкой размеров элементов.
* **Улучшение интерфейса**: Рассмотрите возможность использования matplotlib с более интерактивными возможностями (например, с библиотекой mplcursors для выбора точек на графике).

**5. Проверка и отладка**

* **Логирование**: Добавьте логирование для отслеживания процесса выполнения и выявления потенциальных проблем. Библиотека logging поможет лучше понять поведение программы в режиме отладки.
* **Тестирование**: Напишите тесты для ваших функций, особенно для алгоритма обновления скорости и положения частиц. Используйте библиотеки unittest или pytest для автоматизированного тестирования.

**6. Оптимизация алгоритма**

* **Параллельные вычисления**: Для ускорения работы алгоритма рассмотрите возможность распараллеливания некоторых вычислений с использованием multiprocessing или concurrent.futures.

**7. Визуализация**

* **Улучшение графиков**: Вы можете добавить цветовую шкалу для отображения областей с разными значениями функции пригодности, что поможет лучше понимать распределение частиц.

Эти рекомендации помогут улучшить качество и производительность проекта, а также сделают его более читаемым и удобным в использовании.

# **Контрольный пример**

В этом разделе представлен пример работы программы.

На рисунке 8 представлены параметры, установленные до начала работы программы.

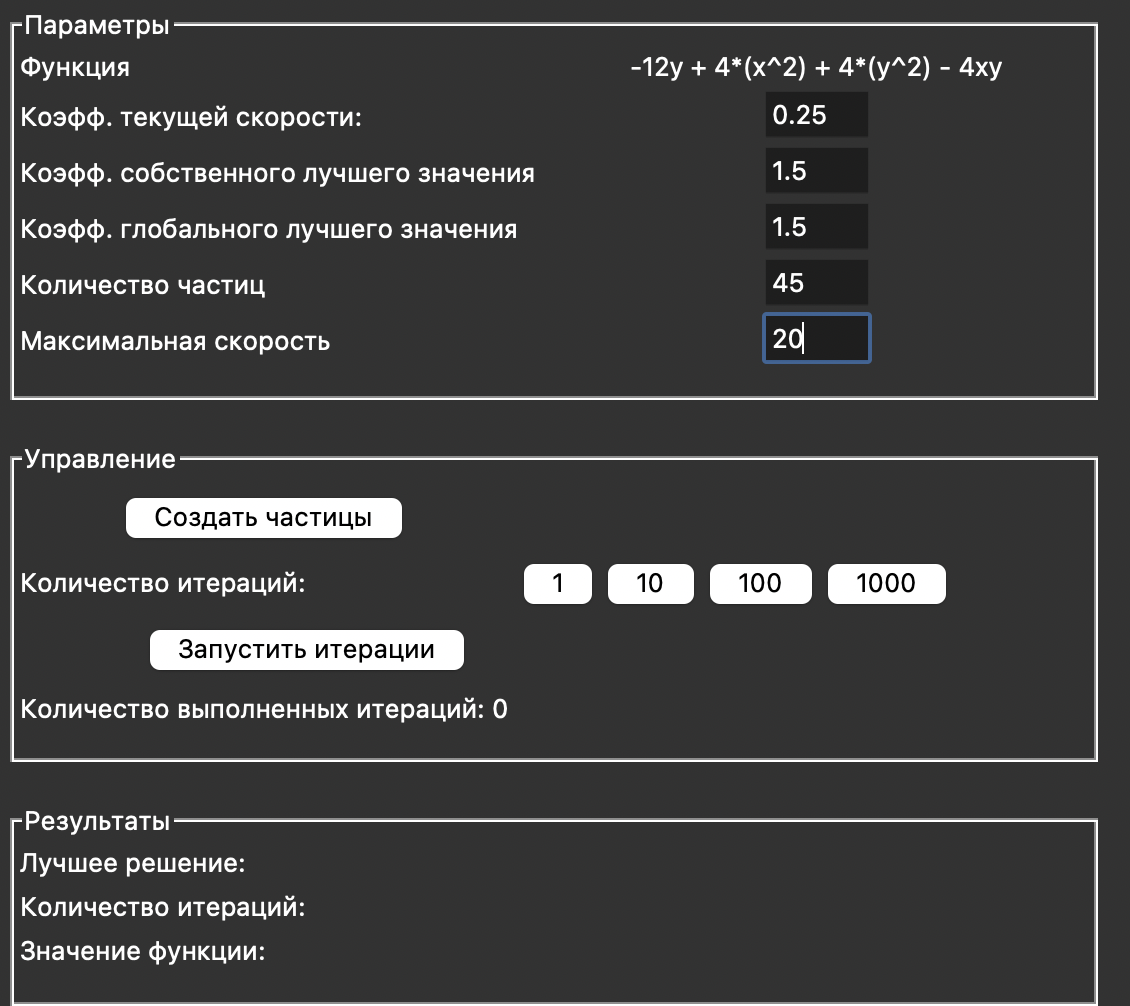


Рисунок 7. Параметры до запуска программы

На рисунке 9 представлен результат инициализации частиц.

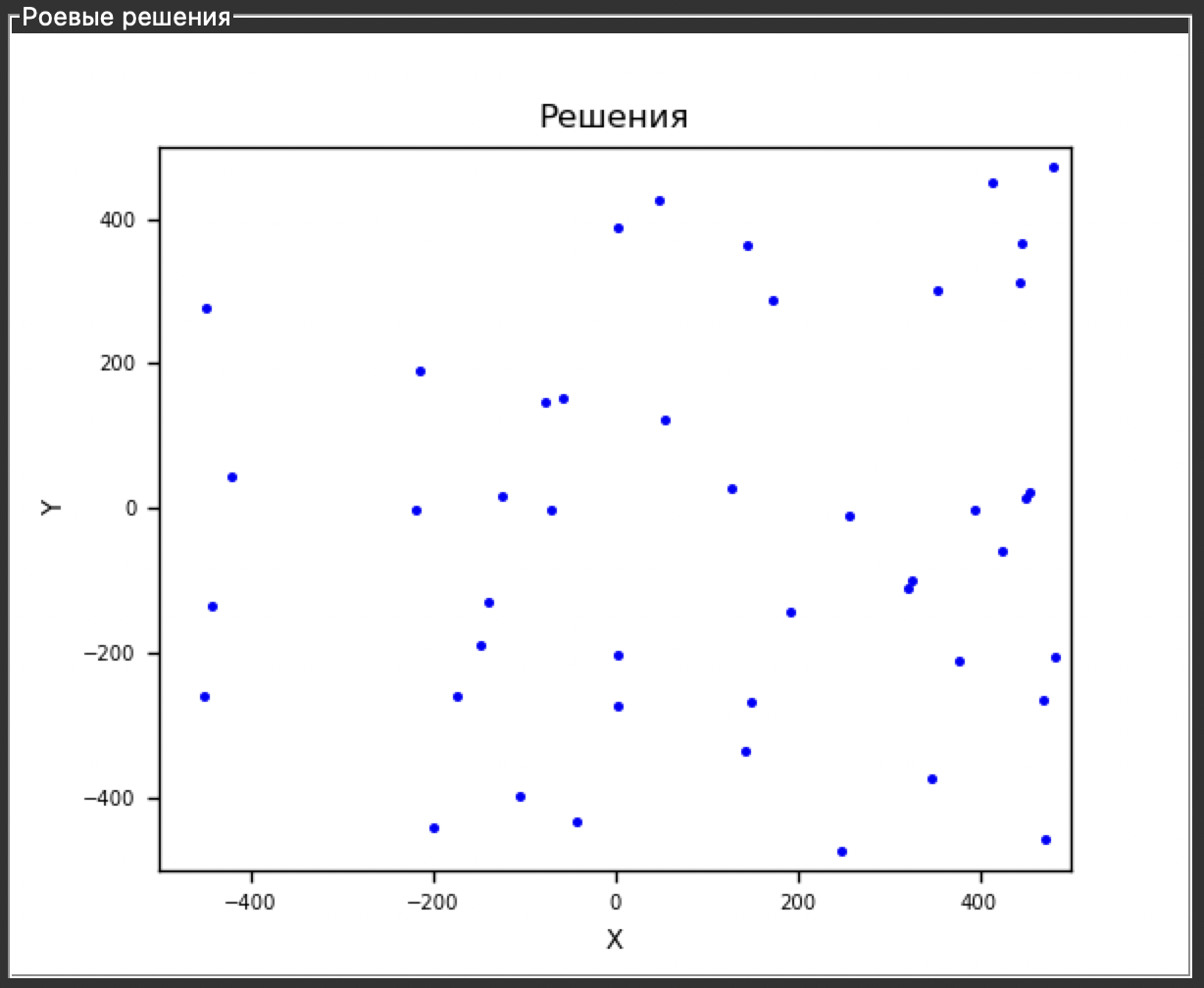


Рисунок 8. Результат инициализации частиц

На рисунке 9 представлен результат работы программы за 10 итераций.

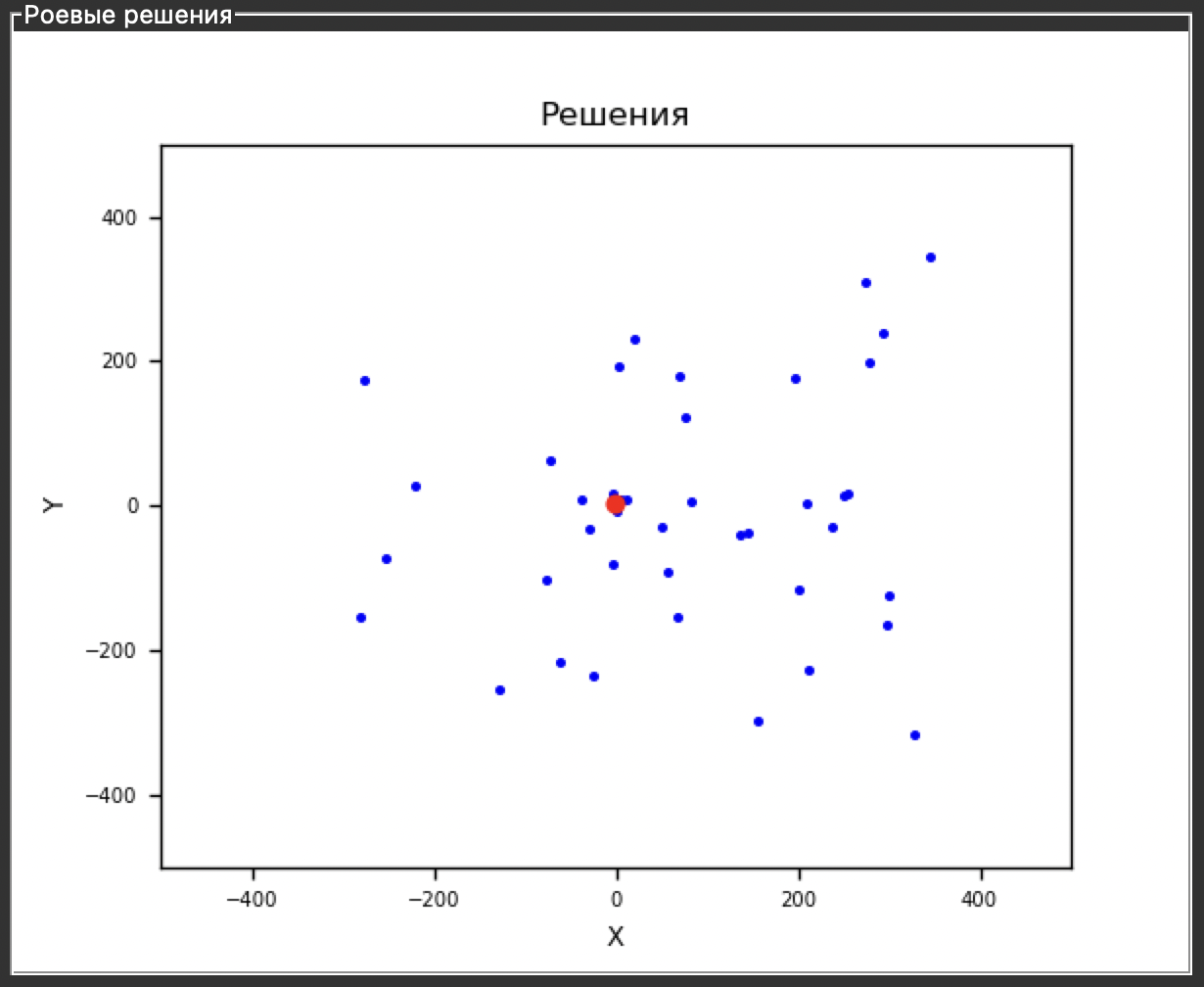


Рисунок 9. Результат 10 итераций

# **Анализ результатов работы алгоритма**

**Исследуем теперь среднеквадратичную ошибку алгоритма в зависимости от количества частиц в одной итерации**

**Первый график (5-30 частиц)**

1. **Резкое уменьшение ошибки**: На участке с количеством частиц от 5 до 10 наблюдается резкое падение средней квадратичной ошибки. Это указывает на значительное улучшение качества оптимизации при добавлении частиц.
2. **Стабилизация**: Начиная с 10 частиц и далее, ошибка стабилизируется и почти не изменяется, что указывает на достижение определенного уровня насыщенности. Увеличение количества частиц выше этого порога не приводит к значительному улучшению качества решения.

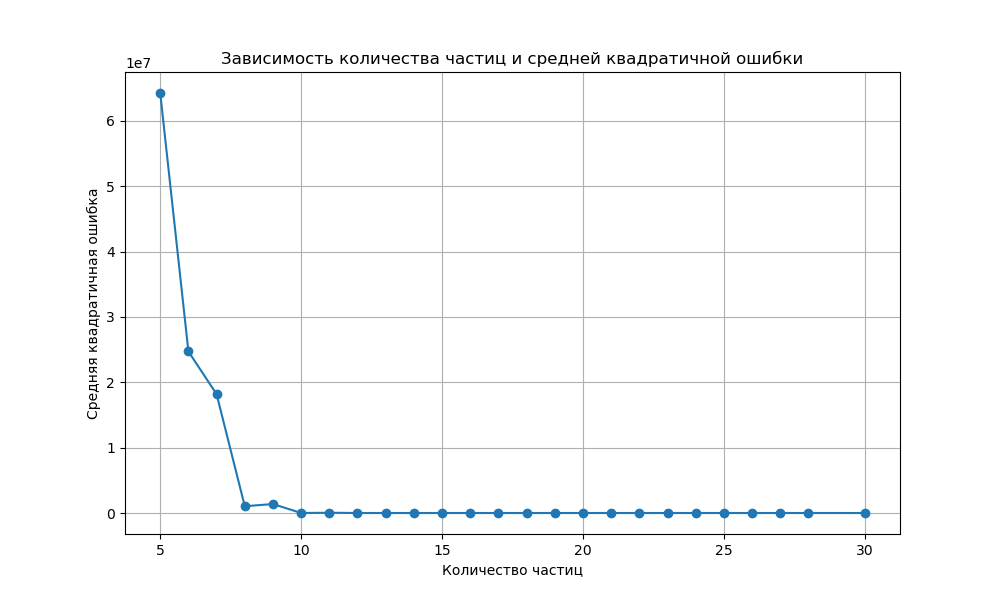


Рисунок 10. 5-30 частиц

**Второй график (20-40 частиц)**

1. **Небольшие колебания**: На участке от 20 до 25 частиц наблюдаются некоторые колебания средней квадратичной ошибки, что может быть связано с неоптимальными начальными положениями частиц. Однако в целом средняя ошибка остаётся достаточно низкой.
2. **Устойчивое улучшение**: Начиная примерно с 30 частиц, ошибка стремится к минимальному значению и стабилизируется. Это может указывать на то, что добавление большого количества частиц улучшает вероятность нахождения глобального оптимума, но эффект от увеличения становится менее значимым.

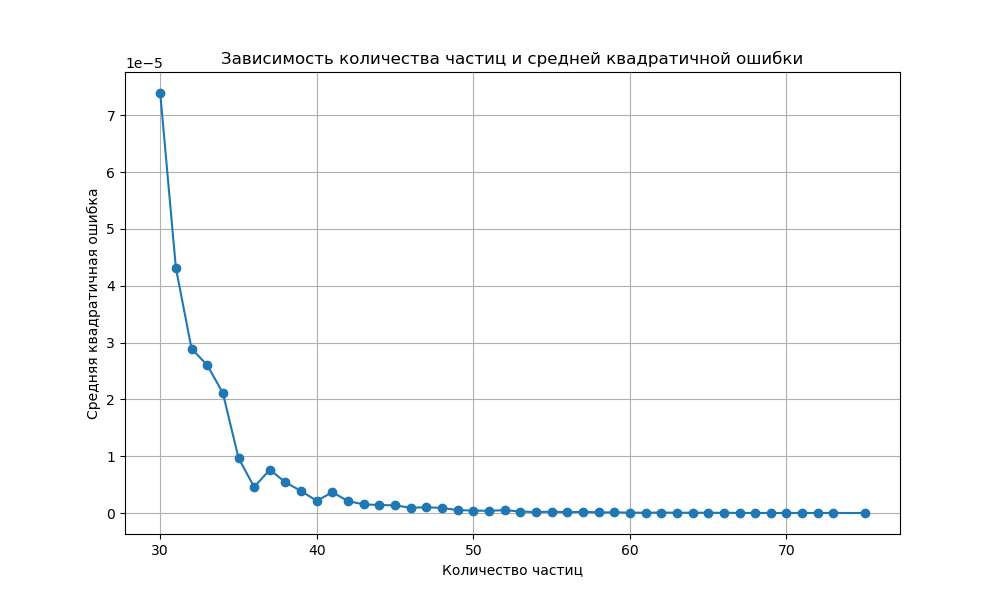


Рисунок 11. 20-40 частиц

**Третий график (75-100 частиц)**

1. **Небольшие колебания**: На участке от 75 до 85 частиц наблюдаются некоторые колебания средней квадратичной ошибки, что может быть связано с рандомными начальными положениями частиц. Однако в целом средняя ошибка остаётся достаточно низкой.
2. **Устойчивое улучшение**: Начиная примерно с 85 частиц, ошибка стремится к минимальному значению и продолжает падать.

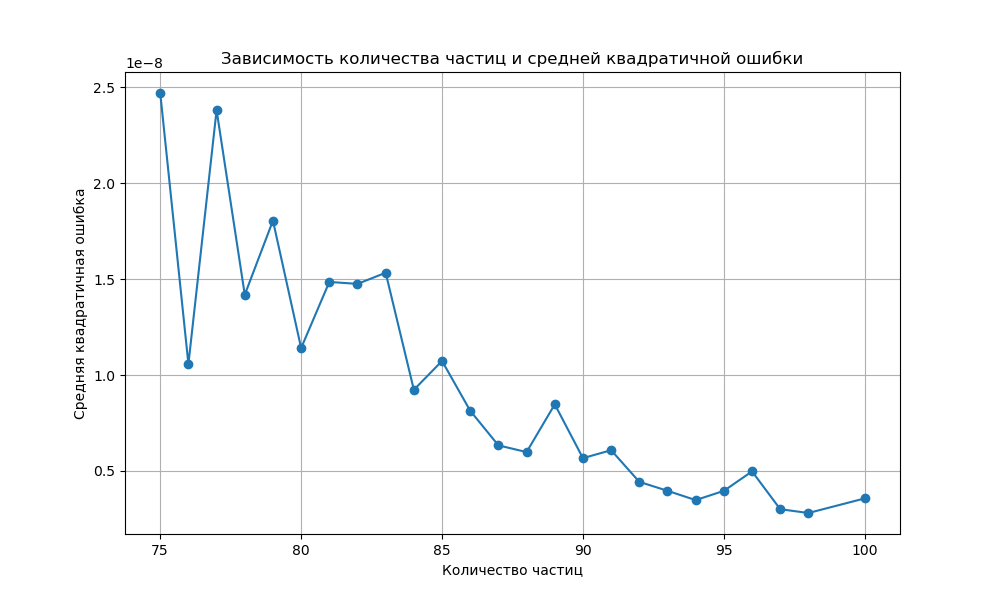


Рисунок 12. 75-100 частиц

**Исследуем зависимость среднеквадратичной ошибки от количества итераций**

Количество частиц фиксировано (40)

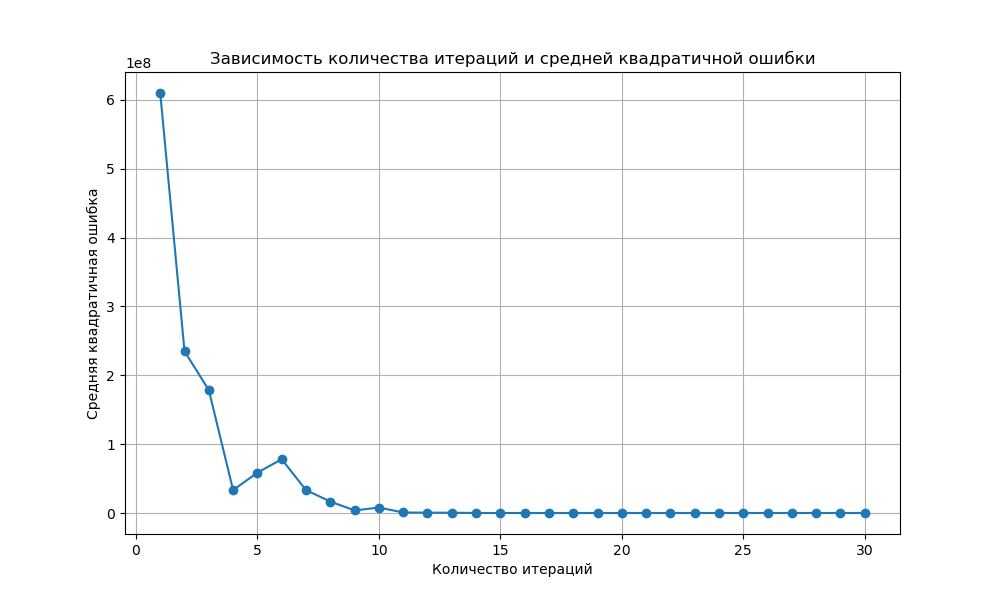


Рисунок 13. 0-30 итераций

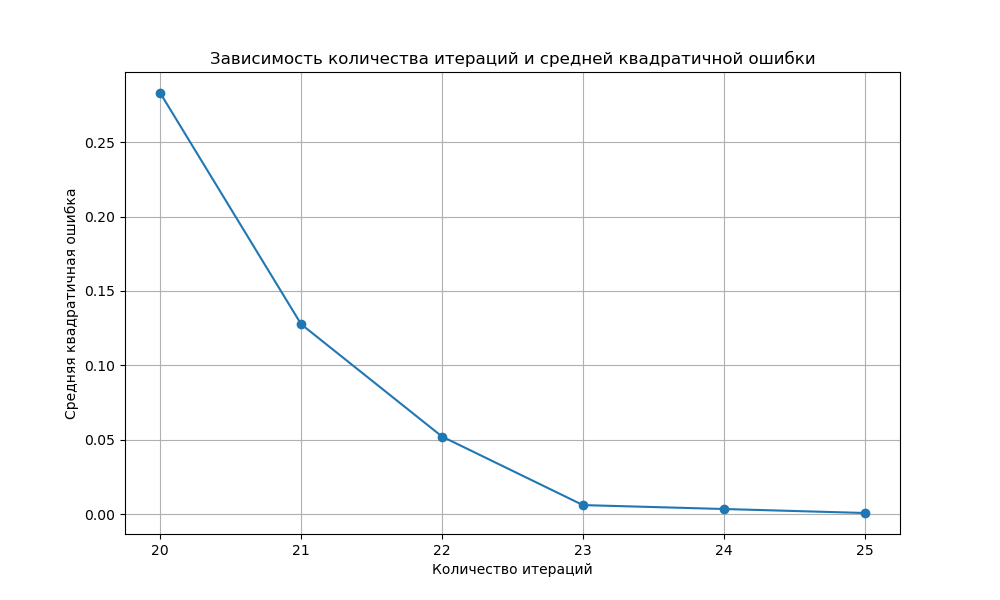


Рисунок 14. 20-25 итераций

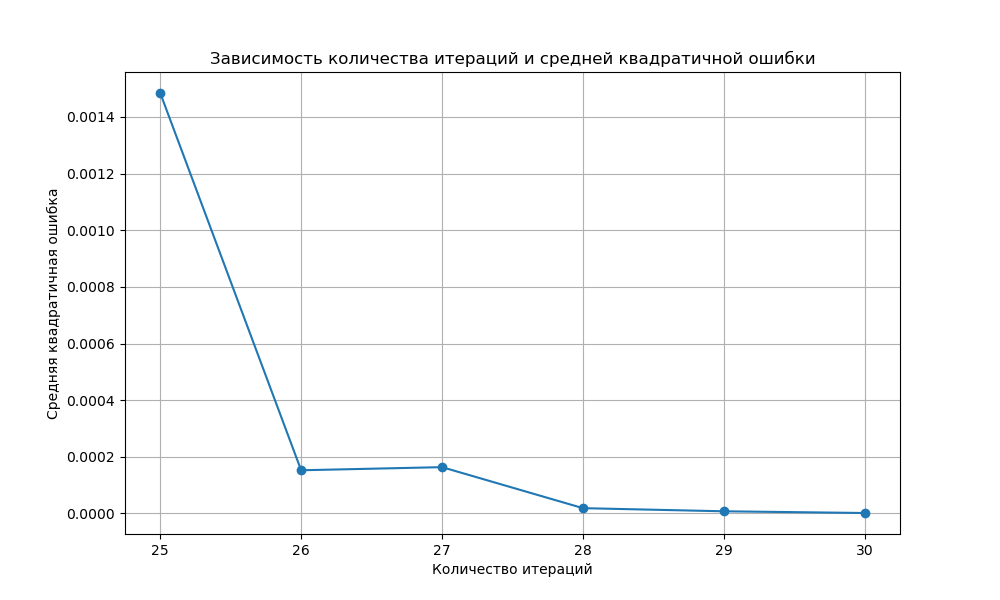


Рисунок 15. 25-30 итераций

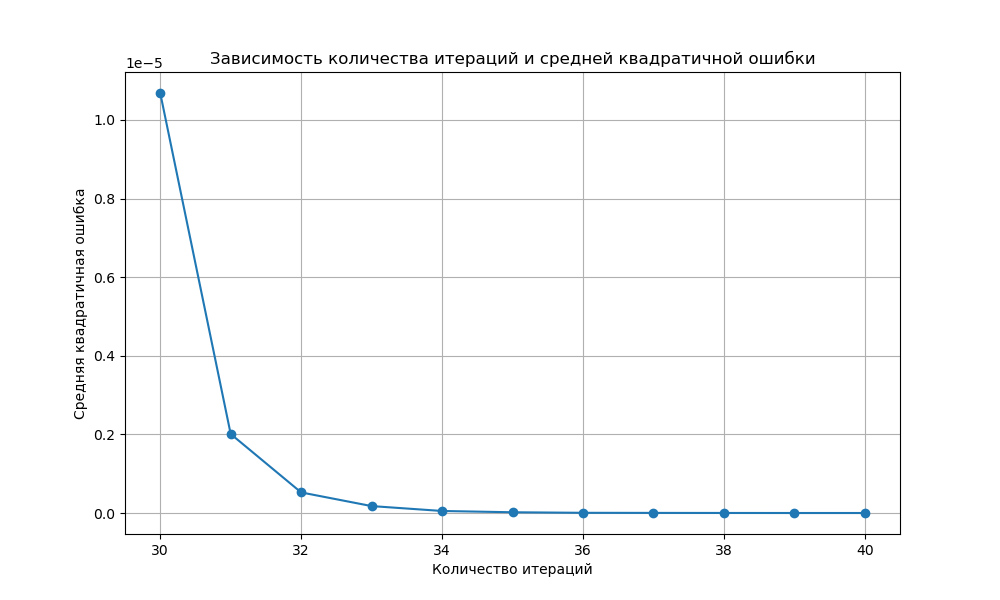


Рисунок 16. 30-40 итераций

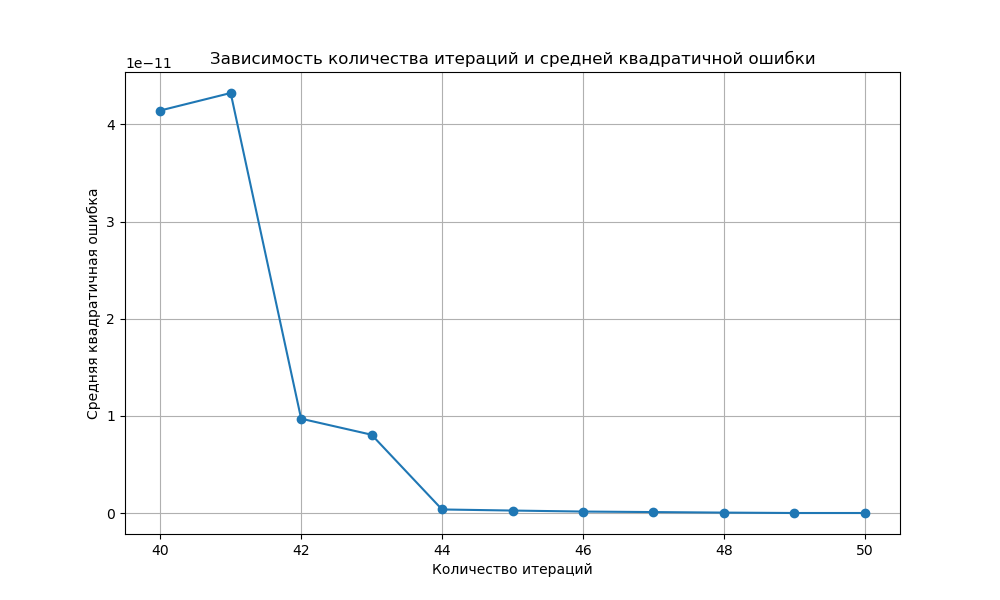


Рисунок 17. 40-50 итераций

Как видно из графиков, увеличение количества итераций сильно влияет на точность алгоритма, поэтому оптимально использовать хотя бы 35-40 итераций.

**Исследуем зависимость среднеквадратичной ошибки от velocity limit**

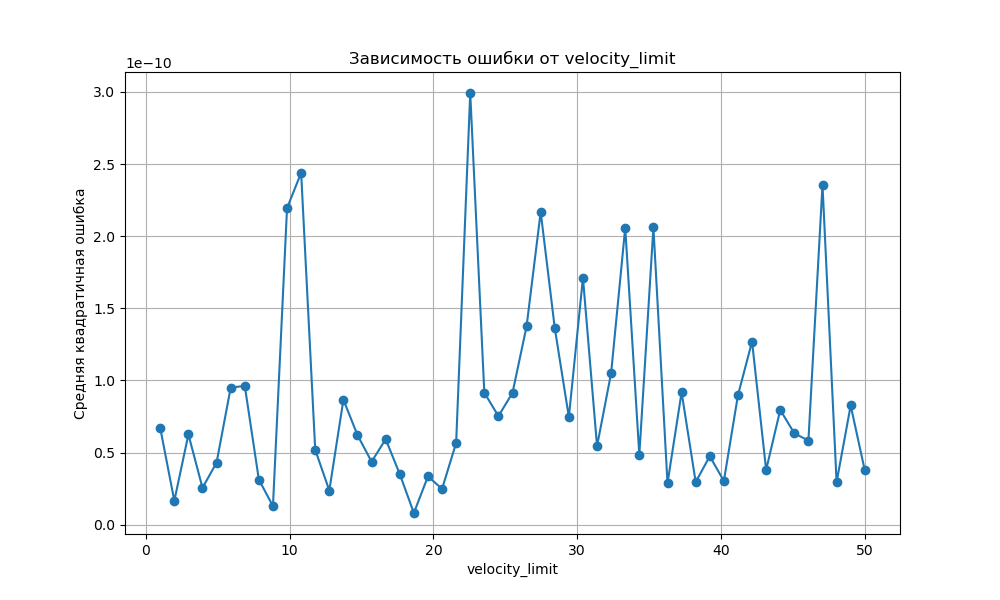


Рисунок 18. Зависимость ошибки от velocity limit

На графике представлена зависимость среднеквадратичной ошибки от параметра velocity limit.

Можно заметить, что при низких значениях velocity limit среднеквадратичная ошибка имеет относительно стабильные и низкие значения, однако с увеличением этого параметра наблюдается значительное колебание ошибки. Это указывает на то, что при высоких значениях скорости частицы могут выходить за оптимальные зоны, приводя к повышенной ошибке.

Оптимальные результаты, судя по графику, достигаются при средних значениях velocity limit, где ошибка минимальна и более устойчива. Таким образом, правильно подобранный velocity limit помогает избежать хаотичных движений частиц и обеспечивает более точную сходимость алгоритма.

**Оптимальный параметр – 19-20.**

**Исследуем зависимость среднеквадратичной ошибки от коэффициента текущей скорости.**

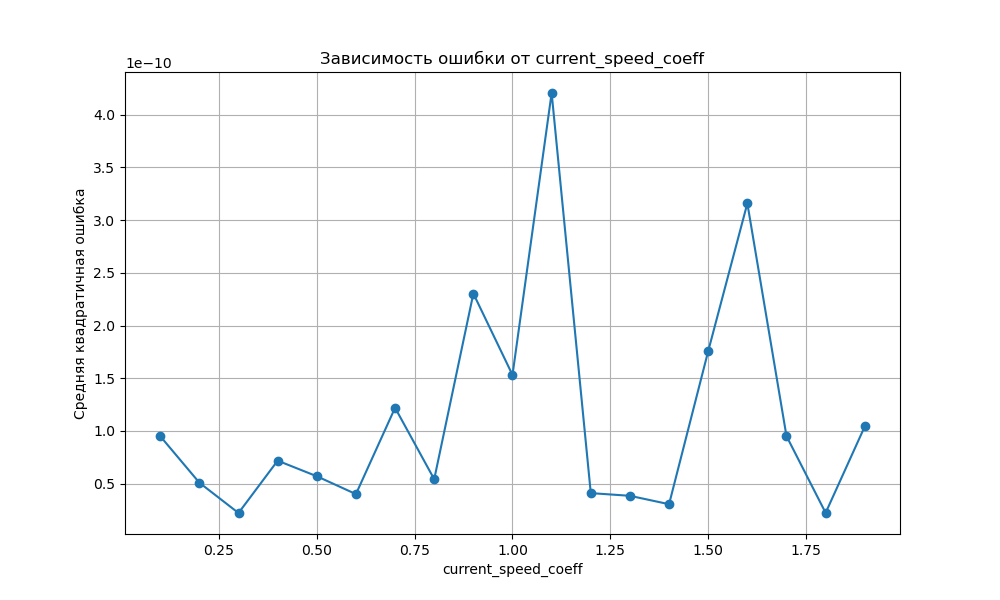
****

Рисунок 19. График среднеквадратичной ошибки от current speed

Как видно из графика оптимальные параметры достигаются при значениях в диапазоне **1.22-1.35**

**Исследуем зависимость среднеквадратичной ошибки от коэффициента глобального оптимума.**

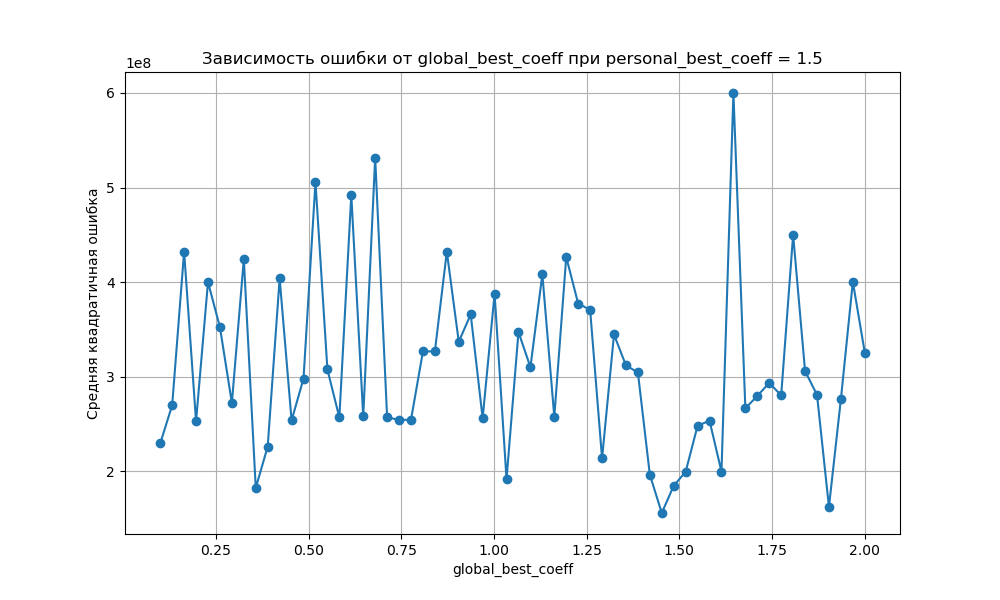


Рисунок 20. График среднеквадратичной ошибки от коэффициента глобального оптимума

Как видно из графика, оптимальные значения достигаются при балансе персонального и глобального коэффициентов (1.35 – 1.5)

**Исследуем оптимальные параметры глобального и персонального коэффициентов.**

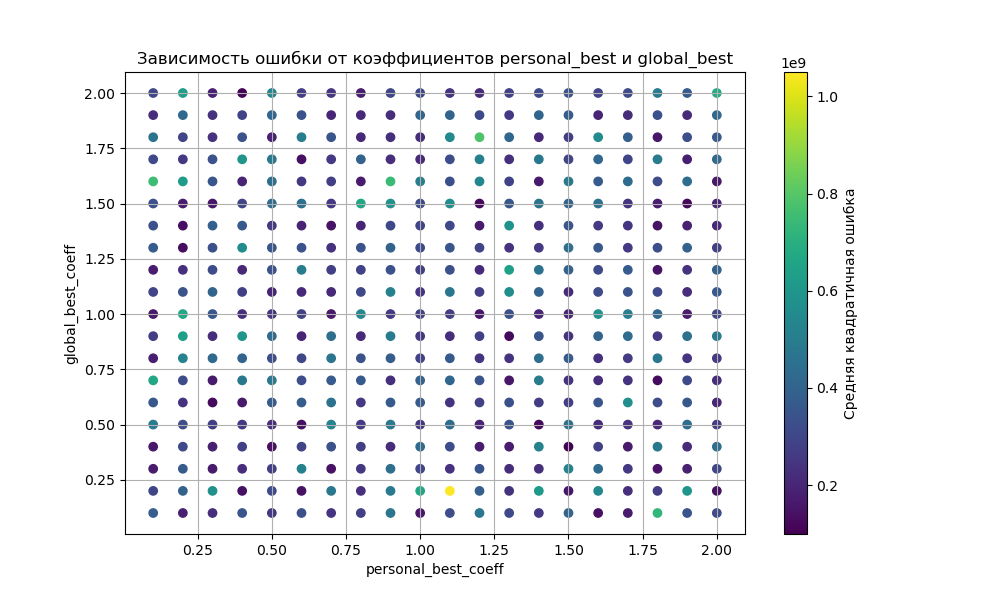
****

Рисунок 21. График ошибки от различных комбинаций параметров

**Optimal personal\_best\_coeff = 1.5, Optimal global\_best\_coeff = 1.4, with error = 87494748.07067955e-11**

**Общие выводы**

1. **Эффективность увеличения количества частиц**: Результаты показывают, что увеличение количества частиц до определенного уровня (около 10-15) приводит к значительному улучшению результатов оптимизации. После этого добавление частиц улучшает результат, но с меньшей отдачей.
2. **Снижение прироста**: После достижения примерно 30 частиц дальнейшее увеличение частиц практически не улучшает среднюю ошибку, что указывает на снижение эффективности по сравнению с затратами вычислительных ресурсов.
3. **Выбор оптимального количества частиц**: Оптимальным решением будет выбор количества частиц в диапазоне от 10 до 30, в зависимости от доступных ресурсов и требований к точности.

В общем, графики подчеркивают важность баланса между количеством частиц и вычислительными затратами, показывая, что есть точки насыщения, после которых прирост эффективности минимален.

# **Сравнение генетического алгоритма и алгоритма роя частиц**

Сравним оригинальный алгоритм роя частиц и генетический алгоритм с вещественным кодированием

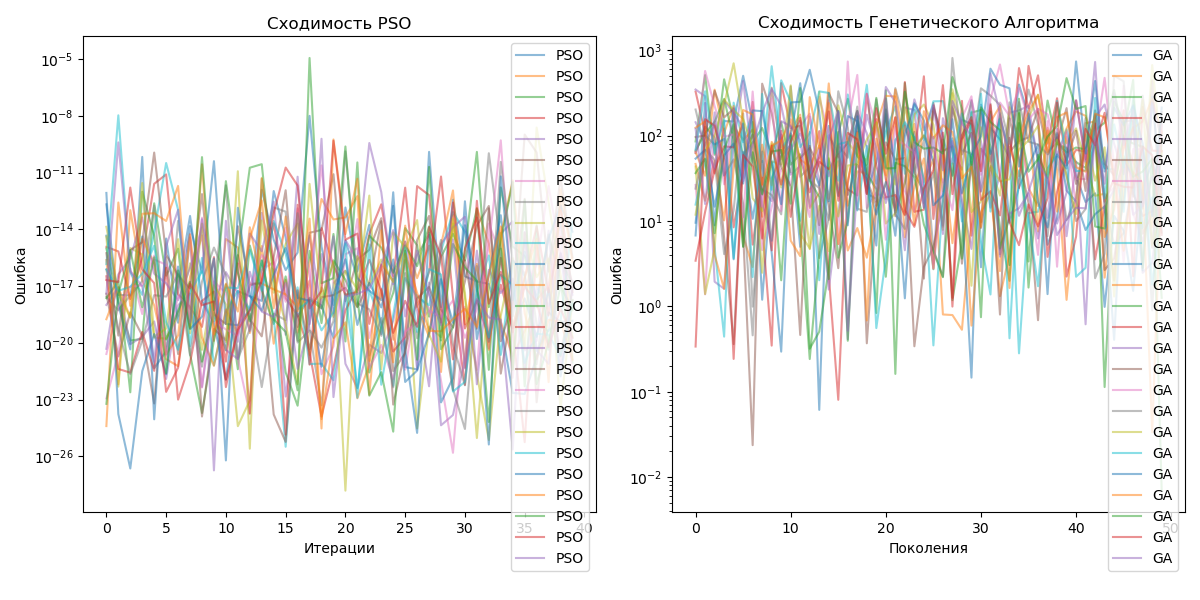


Рисунок 22. Графики сходимости 30 запусков PSO и GA алгоримтов

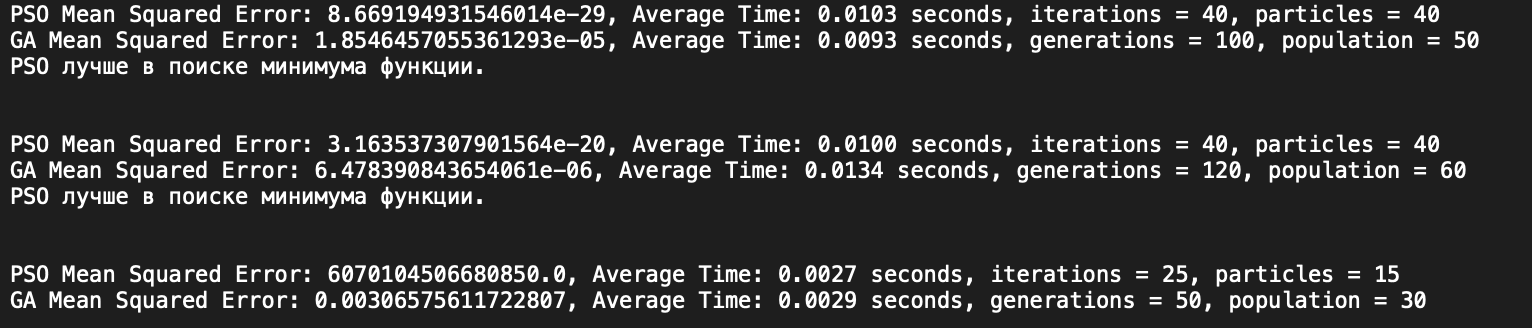


Рисунок 23. Результаты тестов GA и PSO

На основе предоставленных данных можно провести сравнение генетического алгоритма (GA) и алгоритма роя частиц (PSO) по нескольким ключевым характеристикам для задачи поиска минимума функции:

Точность (Mean Squared Error, MSE): В первом и втором случаях PSO показывает значительно меньшую среднеквадратичную ошибку по сравнению с GA, что указывает на его лучшую точность в достижении минимума функции. Однако в третьем примере PSO демонстрирует высокую ошибку, тогда как GA достигает значимо меньшей ошибки. Это может указывать на то, что эффективность PSO в точности сильно зависит от конфигурации алгоритма и особенностей функции.

Время работы: В среднем время работы для PSO и GA оказывается сопоставимым: в первом и втором случаях время выполнения PSO немного меньше, а в третьем случае — практически такое же. Это говорит о том, что по времени работы оба алгоритма могут быть эффективными, но их производительность зависит от числа итераций, количества частиц (для PSO) и размера популяции (для GA).

Скорость сходимости: PSO демонстрирует более быстрое достижение хорошей точности за меньшее количество итераций (40 итераций против 100–120 поколений в GA в первых двух примерах). Это указывает на высокую скорость сходимости PSO по сравнению с GA, что может быть преимуществом для задач, где важна быстрая оптимизация.

Поиск глобального минимума: В предоставленных примерах PSO чаще достигает более низкой ошибки, что говорит о лучшей способности находить глобальный минимум. GA, хотя и может иногда подходить к оптимальному значению, склонен к преждевременной сходимости к локальным минимумам, особенно если размер популяции или число поколений недостаточны. PSO, благодаря механизму коллективного поиска и обмена информацией между частицами, более устойчив к застреванию в локальных минимумах.

Зависимость от параметров: Как PSO, так и GA чувствительны к настройке параметров (число частиц и итераций для PSO, размер популяции и поколений для GA). Для достижения хороших результатов PSO требует относительно небольшого числа частиц и итераций, в то время как GA для аналогичной точности может требовать более крупной популяции и большего числа поколений.

Устойчивость к шуму и сложным ландшафтам функции: PSO обычно лучше работает на гладких функциях с менее выраженными локальными минимумами, где он быстро достигает глобального минимума. GA может показать лучшие результаты в условиях многомодальных функций с выраженными локальными минимумами, так как генетический механизм (мутации и кроссоверы) позволяет лучше исследовать пространство поиска и избегать застревания.

# **Вывод**

В ходе выполнения данной работы были изучены особенности роевого интеллекта. Разработан алгоритм поиска минимума функции при помощи алгоритма роя частиц. Написана программа, которая находит глобальный минимум функции при помощи алгоритма роя частиц. Найден глобальный минимум функции:

**равный -12 и достигается в точке (2, 1)**

Было проведено сравнение результатов работы алгоритмов Роя частиц и генетического алгоритма. PSO в большинстве случаев оказывается быстрее и точнее для поиска минимума функции, достигая высокой точности за меньшее число итераций. Однако, GA может быть более устойчивым в ситуациях с многомодальными функциями и сложным пространством поиска, где он лучше избегает преждевременной сходимости. Выбор алгоритма зависит от специфики задачи: для задач с относительно гладким ландшафтом функций и строгими требованиями по времени PSO предпочтительнее, тогда как для задач с множеством локальных минимумов GA может быть более подходящим.