**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЁТ**

**по лабораторной работе №5**

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

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

**Вариант - 6**

**Студентка гр. 23Б15-пу**

**Беляева А.П.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

**Оглавление**

Цель работы..................................................................................................................3

Описание задачи...........................................................................................................3

Теоретическая часть.....................................................................................................4

Основные шаги программы.........................................................................................5

Блок-схема программы................................................................................................6

Описание программы...................................................................................................7

Рекомендации пользователя........................................................................................8

Рекомендации для программиста................................................................................8

Исходный код программы............................................................................................8

Контрольный пример....................................................................................................9

Анализ............................................................................................................................11

Вывод.............................................................................................................................13

Источники......................................................................................................................14

**Цель работы**

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

**Описание задачи (формализация задачи)**

Необходимо исследовать принципы работы алгоритма оптимизации роя частиц (PSO) и его применение для нахождения глобальных экстремумов функций. Изучить, как параметры алгоритма (размер популяции, коэффициенты инерции и ускорения) влияют на эффективность поиска. Особое внимание уделить модификации "Инерция веса" и её роли в улучшении сходимости и повышении устойчивости алгоритма при нахождении глобального минимума. Сравнить результаты с генетическим алгоритмом.

**Теоретическая часть**

*Алгоритм роевого интеллекта (PSO, Particle Swarm Optimization)* представляет собой оптимизационный метод, который моделирует коллективное поведение группы агентов (или частиц), стремящихся к нахождению глобального экстремума целевой функции. Каждая частица в алгоритме имеет свое положение в поисковом пространстве и скорость, которая обновляется на основе индивидуального опыта частицы (лучшее найденное решение) и опыта всей популяции (глобальное лучшее решение).

*Основные этапы алгоритма PSO:*

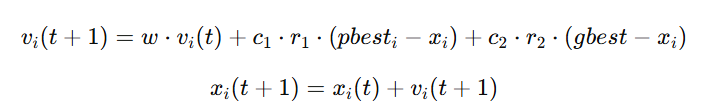
1. *Инициализация:* Каждая частица инициализируется случайным образом с определенным положением и скоростью в поисковом пространстве.
2. *Обновление скорости и положения:* На каждой итерации скорость и положение каждой частицы обновляются в зависимости от следующего выражения:

Рис 1.Формулы обновления позиции и скорости частицы

*Модификация инерция веса:*

В стандартной версии PSO вес инерции является фиксированным. Однако в модификации с инерцией веса параметр w может изменяться с каждой итерацией, что помогает улучшить сходимость алгоритма:

1. На начальных этапах поиска значение w может быть большим, что позволяет частицам исследовать пространство поиска более широко.
2. По мере приближения к глобальному минимуму значение w уменьшается, что помогает частицам сосредоточиться на точной настройке найденных решений и ускоряет сходимость к оптимальному значению.

Обычно w изменяется линейно от максимального значения (на первых итерациях) к минимальному (на последних итерациях):

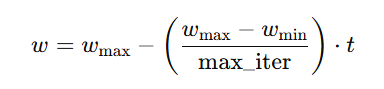


Рис 2. Формула инерции веса.

**Основные шаги программы**

1. Запуск программы.
2. Настройка параметров.
3. Инициализация популяции агентов случайными позициями и скоростями.
4. Вычисление текущих значений целевой функции для каждого агента.
5. Определение глобального и локального лидера на текущей итерации.
6. Обновление скоростей и позиций агентов с учетом лучших значений функции.
7. Проверка условий остановки (например, достижение точности или максимальное число итераций).
8. Сохранение лучшего найденного значения и позиций.

# Блок схема программы

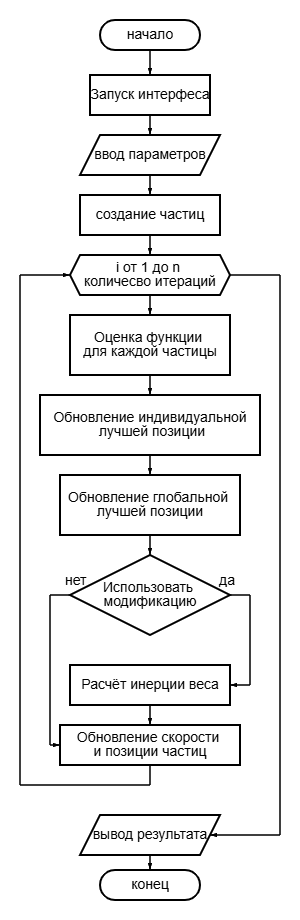


Рис 3. Блок-схема программы.

# Описание программы

Программа разработана на языке Python с использованием библиотек tkinter, numpy и matplotlib для вычислений и создания интерфейса.

Пользователю предоставляется возможность выбора параметров, после чего программа автоматически применяет выбранные действия и высчитываются результаты. Программа включает три класса: Particle - для создания частиц, PSO - для нахождения лучшего значения и PSOApp - для интерфейса.

Таблица 1. main.py

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| objective\_function | Определение целевой функции | int |
| \_\_init\_\_ | Инициализация частицы | None |
| update\_velocity | Обновление скорости частицы | None |
| update\_position | Обновление позиции частицы | None |
| optimize | Обновление скорости и позиции каждой частицы, проверка лучшего глобального значения | None |
| create\_particles | Создание роя частиц с заданными параметрами | None |
| run\_optimization | Запуск оптимизации | None |
| plot\_particles | Построение графика | None |

# Рекомендации пользователя

1. Введите значения для коэффициентов и количества частиц в соответствующие поля.
2. Установите галочку "Использовать инерцию веса" для использования инерции в расчете скорости частиц.
3. Нажмите кнопку "Создать частицы" для инициализации роя частиц.
4. Введите количество итераций и нажмите "Рассчитать" для запуска процесса оптимизации.
5. Наблюдайте за движением частиц на графике и следите за текущим лучшим решением.

# Рекомендации программиста

Убедитесь, что у вас установлен Python версии 3.12.0 и выше, установлены все необходимые библиотеки tkinter, numpy и matplotlib. Убедитесь, что графический интерфейс открывается корректно, и результаты отображаются, как ожидается. Если интерфейс не запускается, проверьте совместимость установленных библиотек и версии Python.

# Исходный код программы

[GitHub](https://github.com/hysterria/Swarm_Alg.git)

# Контрольный пример

# 1. Запуск программы

# Для запуска программы используйте файл **main.py**.

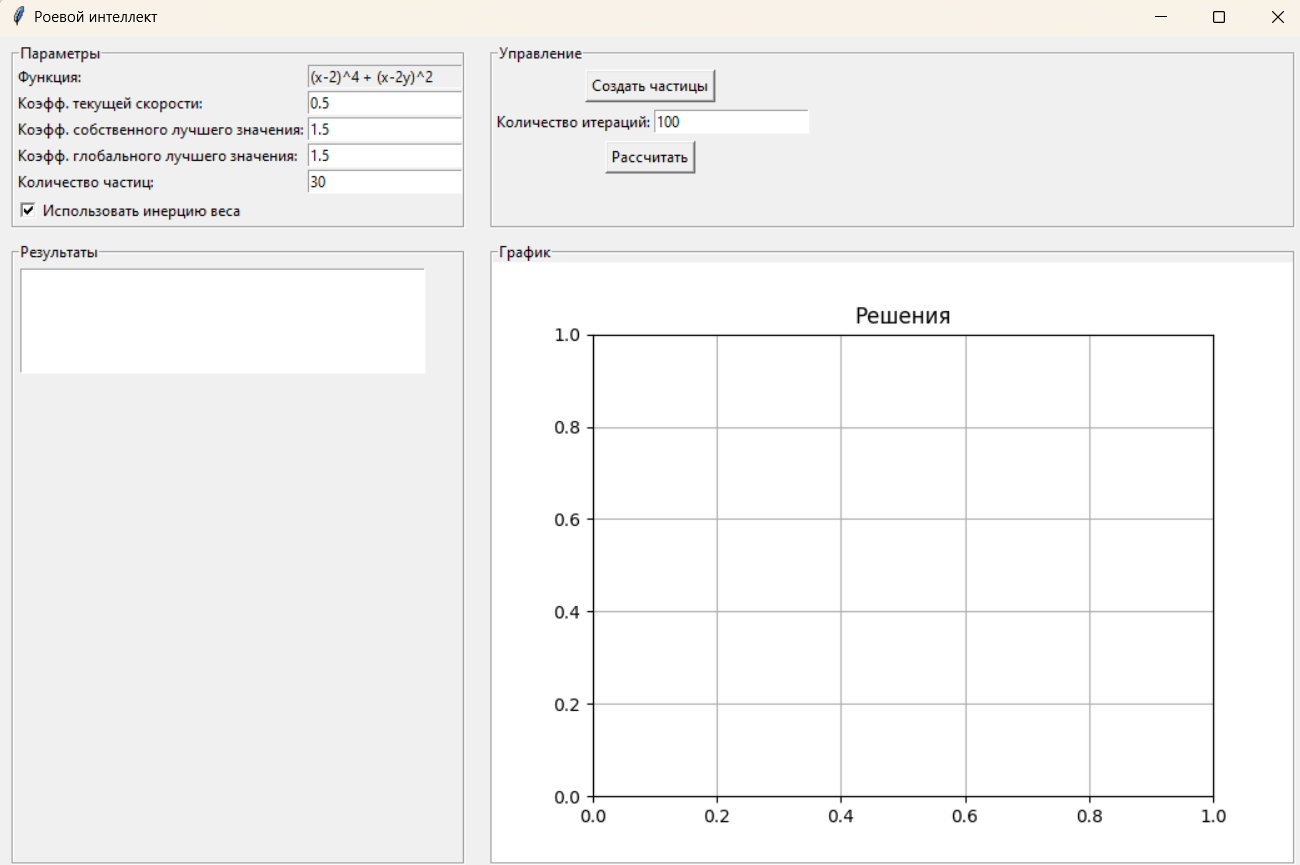


Рис 4. Запуск программы.

2. Ввод параметров алгоритма(коэффициент текущей скорости, коэффициент собственного лучшего значения, количество частиц, коэффициент глобального лучшего значения, флаг использования модификации).

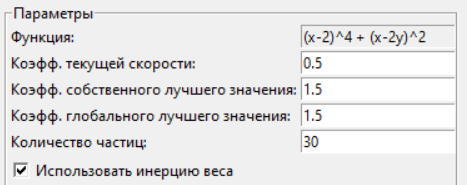


Рис 5. Ввод параметров.

1. Создание частиц и выбор числа итераций.

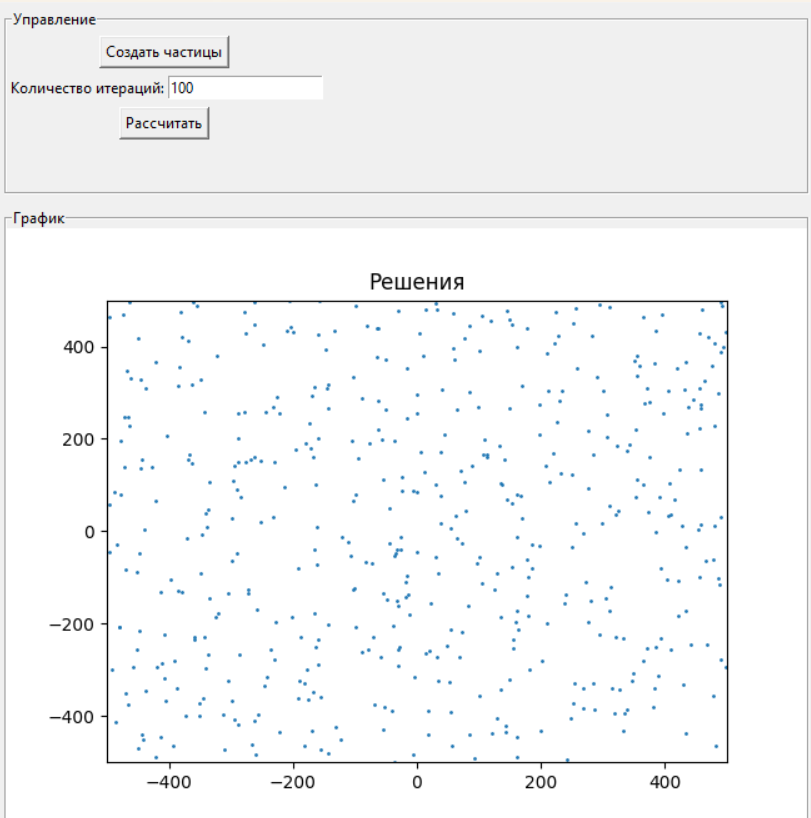


Рис 6. Создание частиц.

1. Расчёт значения.

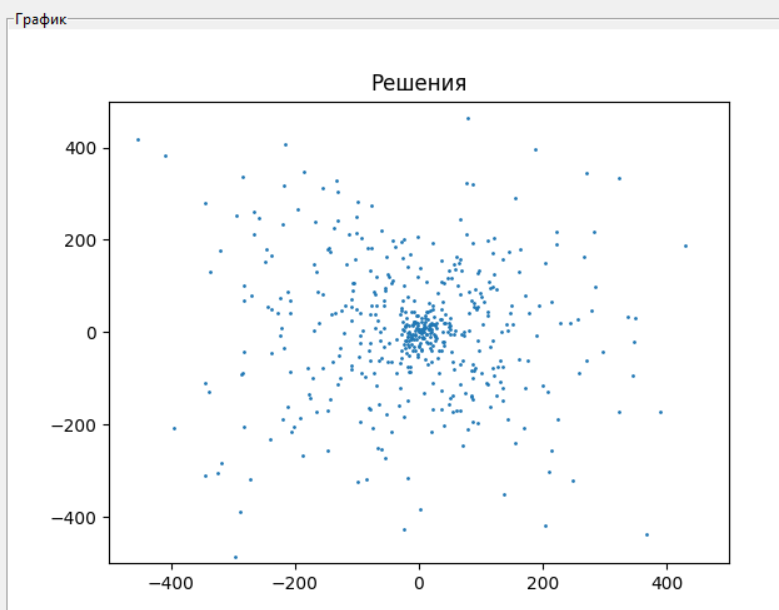
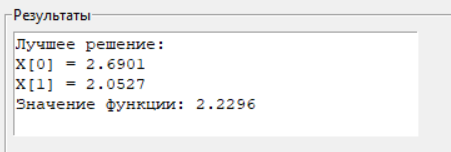


Рис 7. График значений. Рис 8. Окно значений.

# Анализ

Тесты проводились с коэффициентом текущей скорости 0.5, собственного лучшего решения 1.5 и глобального лучшего решения 1.5.

Таблица 2. Тест алгоритма с модификацие и без с кол-во итераций 10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во частиц | Решение с модификацией | Значение в решение с модификацией | Решение без модификации | Значение в решении без модификации |
| 10 | X[0] = -0.4069  X[1] = 1.5811 | 46.2996 | X[0] = 11.0153  X[1] = 432.5779 | 736161.6862 |
| 20 | X[0] = 1.1928  X[1] = 0.6947 | 0.4632 | X[0] = -1.8291  X[1] = -1.2994 | 215.5632 |
| 30 | X[0] = 1.9753  X[1] = 0.9788 | 0.0003 | X[0] = 2.1996  X[1] = 1.0064 | 0.0365 |
| 40 | X[0] = 2.0008  X[1] = 0.9997 | 0.0000 | X[0] = 1.6891  X[1] = 0.7614 | 0.0370 |

Таблица 3. Тест алгоритма с модификацие и без с кол-во частиц 50.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во итераций | Решение с модификацией | Значение в решение с модификацией | Решение без модификации | Значение в решении без модификации |
| 10 | X[0] = 2.3766  X[1] = 0.9594 | 0.0097 | X[0] = 1.9885  X[1] = 0.9942 | 0.0003 |
| 20 | X[0] = 2.2411  X[1] = 1.1225 | 0.0034 | X[0] = 2.0002  X[1] = 1.0001 | 0.0020 |
| 30 | X[0] = 1.9813  X[1] = 0.9830 | 00.0002 | X[0] = 2.0000  X[1] = 1.0000 | 0.0000 |
| 40 | X[0] = 2.0071  X[1] = 1.0036 | 0.0000 | X[0] = 2.0000  X[1] = 1.0000 | 0.0000 |

Таблица 4. Сравнение.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество итераций | Генетический | Генетический с модификацией | Роевой | Роевой с модификацией |
| 10 | 0.23548992 | 0.03654897 | 0.00356234 | 0.00000001 |
| 20 | 0.04879323 | 0.01235458 | 0.00004235 | 0.00000000 |
| 30 | 0.00112951 | 0.04270814 | 0.00000004 | 0.00000000 |
| 40 | 0.00112951 | 0.01971530 | 0.00000000 | 0.00000000 |

Как мы видим результаты работы алгоритма напрямую зависят от количества итераций и частиц, а именно, чем их больше, тем точнее результат.

*Преимущества модификации с инерцией веса:*

1. Лучшее исследование пространства: На первых этапах поиска высокое значение инерции способствует более широкому исследованию поискового пространства. Это помогает избежать застревания в локальных минимумах.
2. Быстрая сходимость: Когда алгоритм приближается к решению, инерция уменьшается, что помогает частицам более точно настраивать свои позиции, увеличивая точность нахождения глобального минимума.

*Недостатки*

1. Может замедлить сходимость: Если инерция слишком велика, частицы могут долго "блуждать", не приближаясь к решению

*Сравнение Генетического и Роевого алгоритма:*

1. Скорость сходимости: Из таблицы видно, что роевой алгоритм (как с модификацией, так и без неё) достигает более низких значений ошибки за меньшее количество итераций по сравнению с генетическим алгоритмом. Например, при 10 итерациях роевой алгоритм с модификацией уже достигает практически нулевого значения ошибки (0.00000001), тогда как генетический алгоритм даёт значительно большее значение ошибки (0.23548992 без модификации и 0.03654897 с модификацией).
2. Эффективность модификации: Модификация инерционного веса в роевом алгоритме даёт заметное улучшение точности на малом количестве итераций (например, 0.00000001 против 0.00356234 при 10 итерациях), тогда как модификация генетического алгоритма даёт менее значительный эффект. При увеличении числа итераций эффект модификации в роевом алгоритме становится менее заметным, так как оба варианта уже достигают минимальных значений.
3. Сходимость на больших итерациях: На 40 итерациях оба варианта роя достигают минимальных значений (0.00000000), тогда как генетический алгоритм (с модификацией и без неё) всё ещё показывает остаточную ошибку.

Заключение:

Роевой алгоритм показывает более быструю и точную сходимость по сравнению с генетическим. Внедрение модификации инерционного веса особенно эффективно на начальных этапах, ускоряя достижение точного результата*.*

# Вывод

Реализация алгоритма роевого интеллекта позволила эффективно найти минимум тестовой функции. В ходе работы были изучены особенности и параметры алгоритма, а также проведено сравнение с генетическим алгоритмом, что позволило оценить их особенности в задачах глобальной оптимизации.

# Источники

1. numpy // Numpy URL: [https://numpy.org/](https://numpy.org/" \t "_new) (дата обращения: 09.11.2024).
2. tkinter — Python interface to Tcl/Tk // Tkinter URL: [https://docs.python.org/3/library/tkinter.html](https://docs.python.org/3/library/tkinter.html" \t "_new) (дата обращения: 09.11.2024).
3. matplotlib: Visualization with Python // Matplotlib URL: <https://matplotlib.org/> (дата обращения: 10.11.2024).