



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"
РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА №6

по дисциплине
«Системный анализ данных в системах поддержки принятия
решений»
Алгоритм роя светлячков

Студент группы: ИКБО-04-22

Кликушин В.И.
(Ф. И.О. студента)

Преподаватель

Железняк Л.М.
(Ф.И.О. преподавателя)

Москва 2024

СОДЕРЖАНИЕ

| | |
|--|----|
| ВВЕДЕНИЕ | 3 |
| 1 АЛГОРИТМ РОЯ СВЕТЛЯЧКОВ..... | 4 |
| 1.1 Описание алгоритма | 4 |
| 1.2 Постановка задачи..... | 5 |
| 1.3 Математическая модель | 5 |
| 1.4 Ручной расчёт | 8 |
| 1.5 Программная реализация | 10 |
| ЗАКЛЮЧЕНИЕ | 13 |
| СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ | 14 |
| ПРИЛОЖЕНИЯ..... | 15 |

ВВЕДЕНИЕ

Алгоритм роя светлячков был предложен в 2007 году инженером и исследователем Синь-Шэ Янгом. Этот алгоритм относится к классу метаэвристических алгоритмов роевого интеллекта, ориентированных на оптимизацию функции.

В основе алгоритма лежит наблюдаемое в природе поведение рассматриваемых насекомых. Они излучают свет, который является механизмом коммуникации между особями: с его помощью они привлекают особей противоположного пола, сообщают о приближении хищников, привлекают добычу и так далее. Менее яркие светлячки перемещаются к более ярким, яркость одного светлячка, воспринимаемая другим, уменьшается при его удалении. Если светлячок не видит более яркого представителя роя, он перемещается хаотично.

Алгоритм роя светлячков нашел широкое применение в задачах, где требуется минимизация или максимизация сложных функций. Его успешно используют в инженерии, экономике, науке о данных и других областях, например, оптимизация маршрутов и логистики, настройка параметров нейронных сетей и машинного обучения, обработка изображений и восстановление сигналов, управление ресурсами и планирование задач.

Актуальность алгоритма обусловлена его способностью эффективно находить решения в сложных многомерных пространствах, где традиционные методы могут быть неприменимы из-за высокой вычислительной сложности или требований к непрерывности функции. Простота реализации и адаптация под широкий спектр задач делают алгоритм роя светлячков важным инструментом для оптимизации в современной науке и технике.

1 АЛГОРИТМ РОЯ СВЕТЛЯЧКОВ

1.1 Описание алгоритма

Алгоритм роя светлячков использует искусственных агентов (светлячков), которые взаимодействуют друг с другом, перемещаясь в поисковом пространстве в направлении более качественных решений. Каждый светлячок характеризуется своей позицией, которая соответствует возможному решению задачи, и уровнем «свечения» (яркости), зависящим от значения целевой функции в данной позиции.

Оптимизация начинается с инициализации популяции светлячков, которые случайным образом размещаются в пространстве решений. Затем алгоритм выполняет итерации, каждая из которых включает следующие этапы:

1. Обновление яркости. Уровень свечения каждого светлячка пересчитывается в зависимости от значения целевой функции в его текущей позиции. Это позволяет определить, насколько «привлекателен» светлячок для других (Формула 1.3.2).
2. Формирование окрестности. Для каждого светлячка определяется список соседей — агентов, находящихся в пределах заданного радиуса, которые обладают более высокой яркостью (Формула 1.3.3).
3. Выбор соседа и перемещение. Светлячок перемещается в направлении соседа, вероятность выбора которого пропорциональна его яркости. Если в окрестности нет более ярких соседей, светлячок перемещается случайным образом (Формулы 1.3.4–1.3.6).
4. Модификация радиуса окрестности. Радиус видимости светлячка корректируется в зависимости от количества соседей, чтобы адаптироваться к текущей плотности агентов в пространстве (Формула 1.3.7).
5. Обновление глобального решения. Если положение какого-либо светлячка превосходит текущее лучшее значение, обновляется

глобальное решение.

Процесс продолжается до тех пор, пока не выполнено условие остановки: либо достигнуто максимальное количество итераций, либо отсутствуют значительные улучшения в течение заданного числа шагов.

1.2 Постановка задачи

Цель работы: реализовать преобразование Коши методом роя светлячков для нахождения приближённого глобального минимума функции.

Задачи: изучить алгоритм роя светлячков, выбрать тестовую функцию для оптимизации (нахождение глобального минимума), произвести ручной расчёт итерации алгоритма, разработать программную реализацию алгоритма роя светлячков для задачи минимизации функции.

Нахождение глобального минимума функции от многих переменных состоит в поиске точки в многомерном пространстве, где значение функции будет минимальным.

Выбранная функция для оптимизации: функция Гольдшейна-Прайса (Формула 1.2.1).

$$f(x, y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]. \quad (1.2.1)$$

Глобальный минимум функции достигается в точке (0; -1) и равен 3. Функция рассматривается на области $-2 \leq x, y \leq 2$.

1.3 Математическая модель

Алгоритм имеет следующие входные параметры: β — коэффициент изменения радиуса окрестности; ρ — коэффициент уменьшения уровня люциферина; δ — коэффициент изменения позиции; r_0 — начальный радиус

окрестности; N — максимальное количество итераций алгоритма; K — размер популяции светлячков; γ — коэффициент привлекательности светлячков; x_{min}, x_{max} — минимальные и максимальные границы пространства; m — длина вектора позиции агента. Все параметры определены от нуля до единицы.

Позиция каждого светлячка инициализируется случайным образом (Формула 1.3.1).

$$x_k = (x_{k1}, x_{k2}, \dots, x_{km}), x_{kj} = x_j^{min} + (x_j^{max} - x_j^{min})rand(), \quad (1.3.1)$$

где k — номер агента.

Изначально все светлячки имеют одинаковое количество люциферина $l_k = l_0$. Радиус окрестности также инициализируется предварительно заданным значением $r_k = r_0$.

Обновление уровня люциферина зависит от позиции агента в пространстве (значения его целевой функции). Вычисление уровня люциферина осуществляется по Формуле 1.3.2.

$$l_k(t + 1) = (1 - \rho)l_k(t) + \gamma F(x_k)(t + 1), \quad (1.3.2)$$

где l — количество люциферина;

k — номер агента;

t — номер итерации.

Каждый агент выбирает того агента внутри радиуса окрестности поиска, у которого уровень люциферина выше, чем его собственный. Окрестность светлячка определяется в соответствии с Формулой 1.3.3.

$$U_k = \{m ||x_m - x_k|| < r_k, l_k < l_m, m \in \overline{1, K}\}, \quad (1.3.3)$$

где U_k — окрестность светлячка;

m – светлячок в окрестности светлячка k .

Таким образом, окрестность U_k включает светлячков m , которые находятся в пределах радиуса r_k ($\|x_m - x_k\| < r_k$), имеют уровень люциферина выше, чем у светлячка k ($l_k < l_m$).

Вычисление вероятности перемещения к соседям осуществляется по Формуле 1.3.4.

$$P_{km} = \frac{l_m - l_k}{\sum_{s \in U_k} (l_s - l_k)}, m \in 1, K, \quad (1.3.4)$$

где P_{km} – вычисленная вероятность движения светлячка k к соседу m .

Говоря иначе, P_{km} показывает относительную привлекательность соседа m по сравнению с остальными соседями.

Светлячок k выбирает номер соседа m в своей окрестности, используя метод колеса рулетки (Формула 1.3.5).

$$\text{Если } \sum_{s=1}^{c-1} P_{ks} < rand() \leq \sum_{s=1}^c P_{ks}, \text{ то } m = c, \quad (1.3.5)$$

где $rand()$ – случайное число в интервале $[0;1]$.

Обновленная позиция агента k определяется по Формуле 1.3.6.

$$x_k(t+1) = x_k(t) + \delta \frac{x_m(t) - x_k(t)}{\|x_m(t) - x_k(t)\|}, \quad (1.3.6)$$

где $x_k(t+1)$ – новая позиция агента;

В числителе второго слагаемого записан вектор смещения, направленный от текущей позиции светлячка k к соседу m . В знаменателе дроби записана длина этого вектора (евклидово расстояние). Это нормализующий фактор, чтобы перемещение происходило по направлению, но не зависело от расстояния.

Обновление радиуса окрестности r осуществляется по Формуле 1.3.7.

$$r_k = \min(r_{\max}, \max(r_{\min}, r_k + \beta(n_u - |U_k|))), \quad (1.3.7)$$

где r_k – обновленный радиус окрестности для светлячка k ;

n_u – желаемое количество соседей для светлячка k ;

$|U_k|$ – текущее количество соседей светлячка k в его окрестности.

В конце итерации определяется наименьшее значение функции, обновляется глобальный минимум, если найдено значение, меньшее текущего глобального минимума.

1.4 Ручной расчёт

Коэффициент изменения радиуса окрестности $\beta = 0,6$; коэффициент уменьшения уровня люциферина $\rho = 0,4$; коэффициент изменения позиции $\delta = 0,2$; начальный радиус окрестности $r_0 = 0,5$; размер популяции светлячков $K = 6$; коэффициент привлекательности светлячков $\gamma = 1$.

Светлячки случайным образом размещены в гиперпространстве поиска. Начальное количество люциферина для светлячков равно 0, начальный радиус равен 0,5. Ниже представлены начальные координаты светлячков:

$$x_1(0) = (-0,1979; 0,6889);$$

$$x_2(0) = (1,5653; 0,4466);$$

$$x_3(0) = (-1,9749; 0,6136);$$

$$x_4(0) = (-1,0235; -1,6855);$$

$$x_5(0) = (1,5296; -1,2012);$$

$$x_6(0) = (-1,3872; -0,3630).$$

В начале итерации обновляется количество люциферина у каждого светлячка по Формуле 1.3.2. Расчет количества люциферина представлен ниже:

$$\begin{aligned}
l_1(1) &= (1 - 0,4) * 0 + 1 * 14444,18^{-1} = 6,9232 * 10^{-5}; \\
l_2(1) &= (1 - 0,4) * 0 + 1 * 747,40^{-1} = 0,0013; \\
l_3(1) &= (1 - 0,4) * 0 + 1 * 47626,687^{-1} = 2,0996 * 10^{-5}; \\
l_4(1) &= (1 - 0,4) * 0 + 1 * 695,491^{-1} = 0,0014; \\
l_5(1) &= (1 - 0,4) * 0 + 1 * 54225,864^{-1} = 1,8441 * 10^{-5}; \\
l_6(1) &= (1 - 0,4) * 0 + 1 * 5577,416^{-1} = 0,0001;
\end{aligned}$$

Вычисляется множество соседей для каждого светлячка по Формуле 1.3.3.
 Расчет представлен ниже:

$$\begin{aligned}
U_1(1) &= \{\}; \\
U_2(1) &= \{\}; \\
U_3(1) &= \{\}; \\
U_4(1) &= \{\}; \\
U_5(1) &= \{\}; \\
U_6(1) &= \{\}.
\end{aligned}$$

На первой итерации ни у одного из светлячков нет соседей.

Изменяется радиус окрестности светлячков в соответствие с Формулой 1.3.7. Расчет радиуса представлен ниже:

$$\begin{aligned}
r_1(1) &= \min(4, \max(0,1, r_1(0) + 0,6(5 - 0))) = 3,5; \\
r_2(1) &= \min(4, \max(0,1, r_2(0) + 0,6(5 - 0))) = 3,5; \\
r_3(1) &= \min(4, \max(0,1, r_3(0) + 0,6(5 - 0))) = 3,5; \\
r_4(1) &= \min(4, \max(0,1, r_4(0) + 0,6(5 - 0))) = 3,5; \\
r_5(1) &= \min(4, \max(0,1, r_5(0) + 0,6(5 - 0))) = 3,5; \\
r_6(1) &= \min(4, \max(0,1, r_6(0) + 0,6(5 - 0))) = 3,5;
\end{aligned}$$

Лучшее глобальное значение функции на первой итерации равно 695,491. Точка, в которой достигается минимальное значение: $(-1,0235; -1,6855)$.

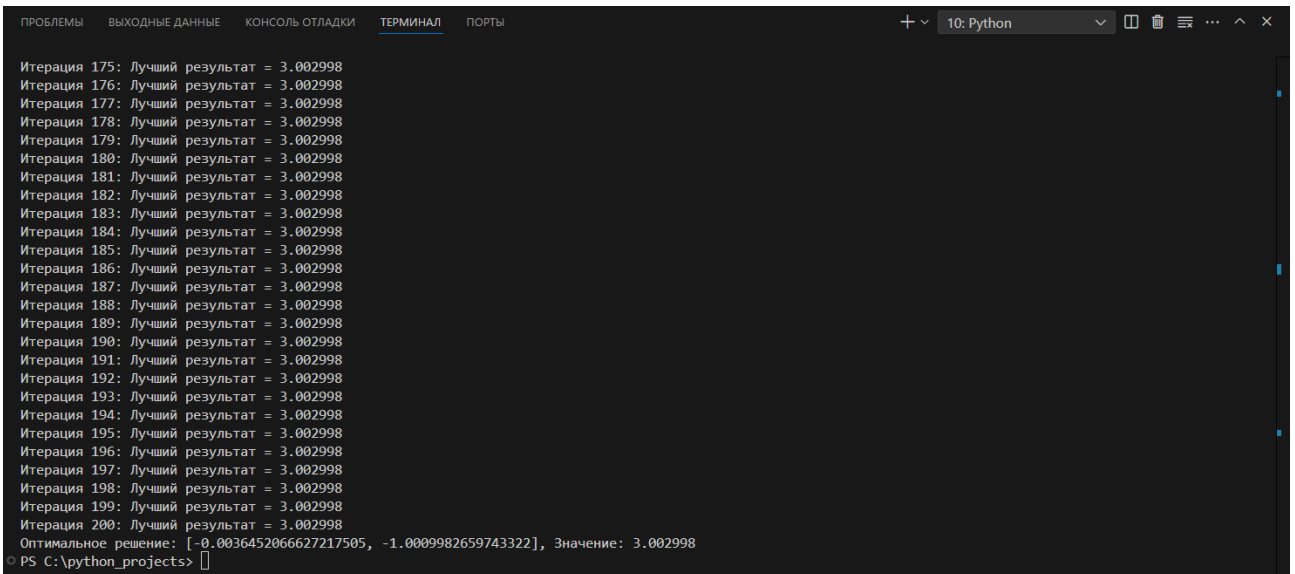
1.5 Программная реализация

Разработаны классы Firefly и FireflySwarm, которые реализуют отдельного светлячка и рой светлячков.

Код алгоритма роя светлячков для задачи поиска глобального минимума функции представлен в Приложении А.

Коэффициент изменения радиуса окрестности $\beta = 0,6$; коэффициент уменьшения уровня люциферина $\rho = 0,4$; коэффициент изменения позиции $\delta = 0,2$; начальный радиус окрестности $r_0 = 0,5$; размер популяции светлячков $K = 100$; коэффициент привлекательности светлячков $\gamma = 1$; максимальное количество итераций $N = 200$.

Результат работы алгоритма роя светлячков представлен на Рисунке 1.5.1.



```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ
+ ~ 10: Python
Итерация 175: Лучший результат = 3.002998
Итерация 176: Лучший результат = 3.002998
Итерация 177: Лучший результат = 3.002998
Итерация 178: Лучший результат = 3.002998
Итерация 179: Лучший результат = 3.002998
Итерация 180: Лучший результат = 3.002998
Итерация 181: Лучший результат = 3.002998
Итерация 182: Лучший результат = 3.002998
Итерация 183: Лучший результат = 3.002998
Итерация 184: Лучший результат = 3.002998
Итерация 185: Лучший результат = 3.002998
Итерация 186: Лучший результат = 3.002998
Итерация 187: Лучший результат = 3.002998
Итерация 188: Лучший результат = 3.002998
Итерация 189: Лучший результат = 3.002998
Итерация 190: Лучший результат = 3.002998
Итерация 191: Лучший результат = 3.002998
Итерация 192: Лучший результат = 3.002998
Итерация 193: Лучший результат = 3.002998
Итерация 194: Лучший результат = 3.002998
Итерация 195: Лучший результат = 3.002998
Итерация 196: Лучший результат = 3.002998
Итерация 197: Лучший результат = 3.002998
Итерация 198: Лучший результат = 3.002998
Итерация 199: Лучший результат = 3.002998
Итерация 200: Лучший результат = 3.002998
Оптимальное решение: [-0.0036452066627217505, -1.0009982659743322], Значение: 3.002998
PS C:\python_projects>
```

Рисунок 1.5.1 – Результат работы алгоритма роя светлячков для задачи поиска глобального минимума функции

Для каждой итерации выводится её номер и глобальный текущий минимум функции.

Визуализация процесса поиска представлена на Рисунках 1.5.2–1.5.3.

В верхней части графика выводится номер итерации, зелёные точки соответствуют светлячкам в гиперпространстве поиска решений. Светлячки с

более высоким уровнем люциферина светятся ярче. На вертикальной оси - значение координаты по оси y , и на горизонтальной оси – значение координаты по оси x .

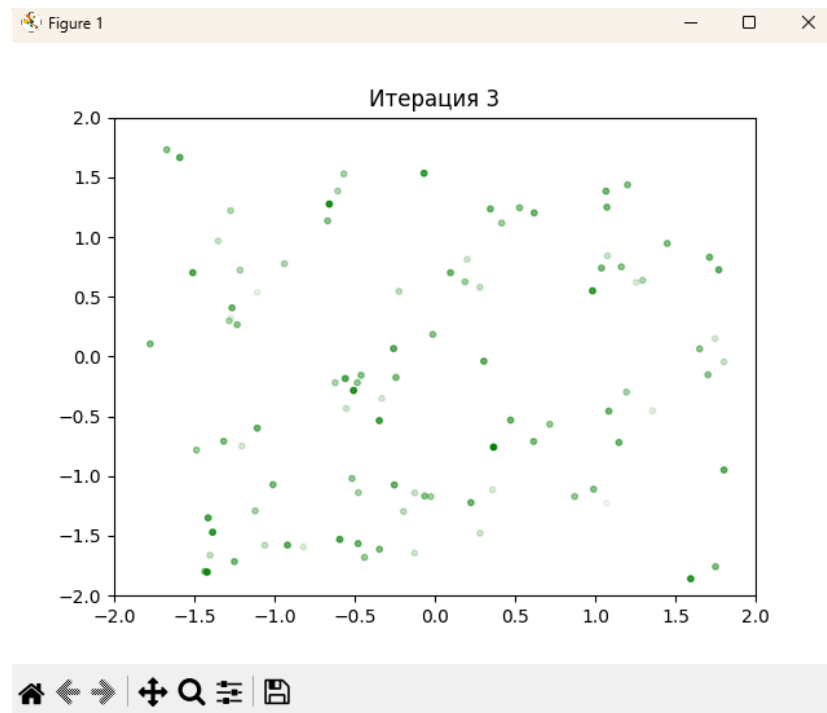


Рисунок 1.5.2 – Визуализация работы алгоритма роя светлячков на начальных итерациях

На Рисунке 1.5.2 видно, что светлячки распределены по всему гиперпространству поиска на начальных итерациях работы алгоритма.

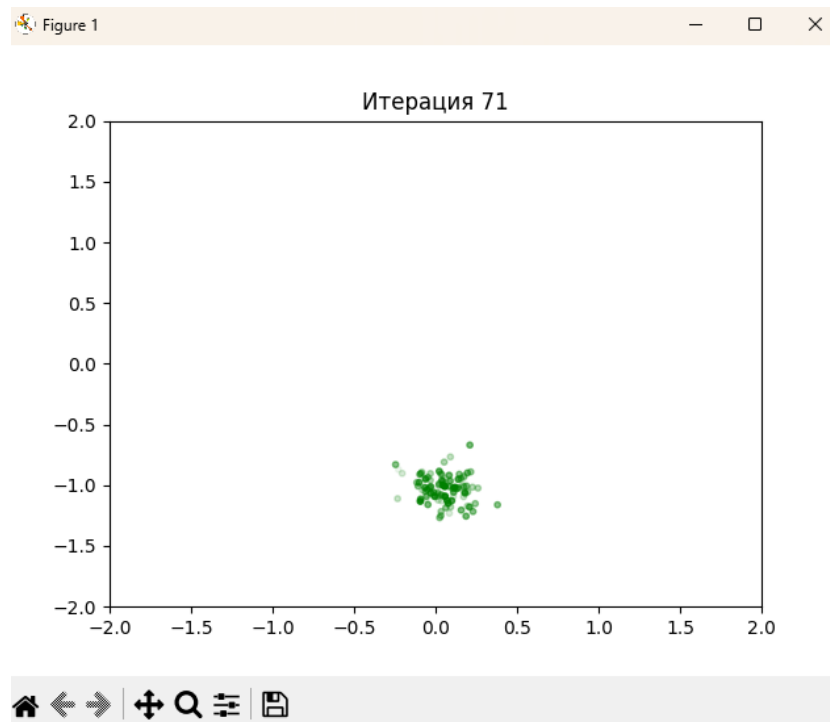


Рисунок 1.5.3 – Сходимость светлячков к глобальному минимуму

На Рисунке 1.5.3 светлячки сходятся к одной точке – глобальному минимуму рассматриваемой функции оптимизации.

Объединение светлячков в гиперпространстве поиска в один кучный рой говорит о сходимости алгоритма.

График сходимости алгоритма роя светлячков представлен на Рисунке 1.5.4.

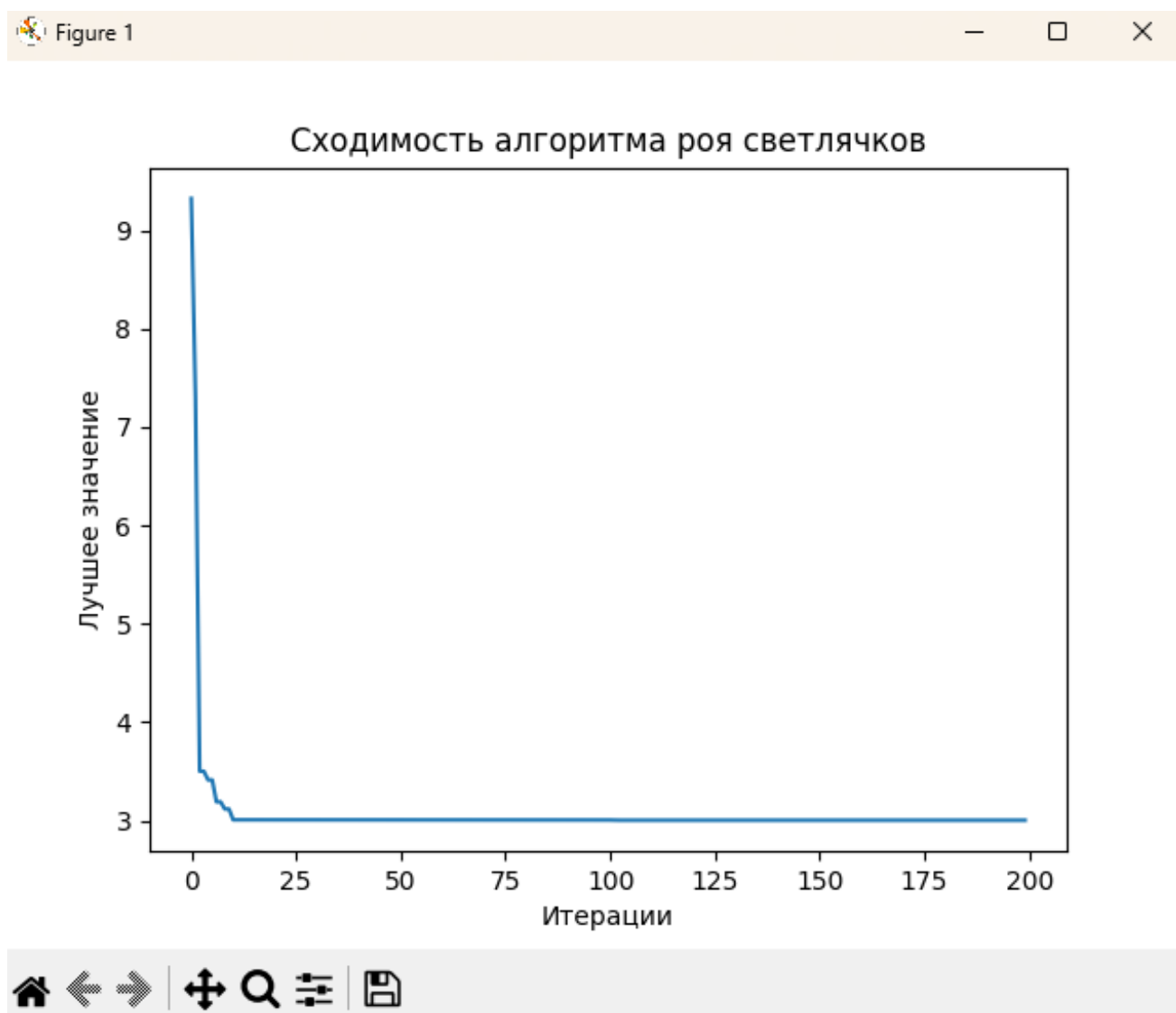


Рисунок 1.5.4 – График сходимости алгоритма роя светлячков

На горизонтальной оси отложен номер итерации, на вертикальной оси - глобальный текущий минимум функции.

По графику видно, что алгоритм находит приближённый глобальный минимум функции Гольдшейна-Прайса примерно спустя десять итераций работы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы изучен алгоритм роя светлячков, проведён его ручной расчёт одной итерации, а также разработана программа на языке Python для оптимизации функции Голдштейна-Прайса.

Основное преимущество алгоритма роя светлячков заключается в его способности эффективно находить решения сложных многомерных задач оптимизации. Это достигается благодаря принципу взаимодействия агентов (светлячков), которые движутся в направлении более перспективных областей пространства решений, ориентируясь на значение целевой функции. Такой подход позволяет комбинировать исследование новых областей пространства с углубленным поиском вокруг текущих лучших решений.

Реализованный алгоритм демонстрирует, как параметры, такие как радиус окрестности, коэффициент уменьшения люциферина, количество итераций и другие, влияют на сходимость и качество найденного решения. Визуализация работы алгоритма показала, как светлячки перемещаются в пространстве, группируясь вокруг глобального экстремума, что подтверждает его способность находить оптимальные решения в сложных ландшафтах целевой функции.

Таким образом, алгоритм роя светлячков является универсальным инструментом для решения задач оптимизации. Благодаря своей простоте, адаптивности и способности избегать локальных экстремумов, он нашел широкое применение в инженерии, науке о данных и других областях, требующих эффективного поиска решений в многомерных пространствах.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс] : Учебно-методическое пособие / А. Б. Сорокин – Москва: Московский технологический университет (МИРЭА), 2019.
2. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
3. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
4. Firefly algorithm. [Электронный ресурс]: Википедия. – URL: https://en.wikipedia.org/wiki/Firefly_algorithm (Дата обращения: 30.11.2024).

ПРИЛОЖЕНИЯ

Приложение А — Код реализации алгоритма роя светлячков.

Приложение А

Код реализации алгоритма роя светлячков

Листинг А – Реализация алгоритма роя светлячков

```
import random
import math
from typing import List, Tuple, Callable
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def goldstein_price(x: float, y: float) -> float:
    """
    Функция Голдштейна-Прайса для оптимизации.
    Параметры:
        x (float): Координата по оси x.
        y (float): Координата по оси y.
    Возвращает:
        float: Значение функции Голдштейна-Прайса.
    """
    term1 = (1 + (x + y + 1) ** 2 * (19 - 14 * x + 3 *
        x ** 2 - 14 * y + 6 * x * y + 3 * y ** 2))
    term2 = (30 + (2 * x - 3 * y) ** 2 * (18 - 32 * x + 12 *
        x ** 2 + 48 * y - 36 * x * y + 27 * y ** 2))
    return term1 * term2

class Firefly:
    def __init__(self, position: List[float], luciferin: float = 0.0, radius:
float = 1.0):
        """
        Инициализирует светлячка.
        Параметры:
            position (List[float]): Начальная позиция светлячка.
            luciferin (float): Уровень люциферина.
            radius (float): Радиус окрестности светлячка.
        """
        self.position = position
        self.luciferin = luciferin
        self.radius = radius

    def update_luciferin(self, function_value: float, rho: float, gamma: float):
        """
        Обновляет уровень люциферина светлячка.
        Параметры:
            function_value (float): Значение целевой функции в текущей позиции.
            rho (float): Коэффициент уменьшения люциферина.
            gamma (float): Коэффициент привлекательности светлячка.
        """
        self.luciferin = (1 - rho) * self.luciferin + \
            gamma * (1 / function_value)

    def move_towards(self, other: 'Firefly', delta: float, bounds:
List[Tuple[float, float]]):
        """
        Перемещает светлячка в направлении другого более яркого светлячка.
        Параметры:
            other (Firefly): Другой светлячок, к которому перемещается текущий.
            delta (float): Коэффициент изменения позиции.
            bounds (List[Tuple[float, float]]): Границы пространства поиска.
        """
        direction = [other.position[i] - self.position[i]
```



```

        for i in range(len(self.position))
    distance = math.sqrt(sum(d ** 2 for d in direction))
    if distance > 0:
        normalized_direction = [d / distance for d in direction]
        self.position = [
            min(max(
                self.position[i] + delta * normalized_direction[i],
bounds[i][0]), bounds[i][1])
            for i in range(len(self.position))
        ]

class FireflySwarm:
    def __init__(self, fitness_function: Callable[..., float], bounds:
List[Tuple[float, float]],
        num_fireflies: int, max_iterations: int, beta: float, rho:
float,
        delta: float, gamma: float, initial_radius: float):
        '''
        Инициализирует алгоритм роя светлячков.
        Параметры:
            fitness_function (Callable[..., float]): Целевая функция для
оптимизации.
            bounds (List[Tuple[float, float]]): Границы пространства поиска
[(xmin, xmax), (ymin, ymax)].
            num_fireflies (int): Количество светлячков.
            max_iterations (int): Максимальное количество итераций.
            beta (float): Коэффициент изменения радиуса окрестности.
            rho (float): Коэффициент уменьшения уровня люциферина.
            delta (float): Коэффициент изменения позиции.
            gamma (float): Коэффициент увеличения люциферина.
            initial_radius (float): Начальный радиус окрестности.
        '''
        self.fitness_function = fitness_function
        self.bounds = bounds
        self.num_fireflies = num_fireflies
        self.max_iterations = max_iterations
        self.beta = beta
        self.rho = rho
        self.delta = delta
        self.gamma = gamma
        self.initial_radius = initial_radius
        self.fireflies = []
        self.min_radius = 0.1
        self.max_radius = max(
            bounds[0][1] - bounds[0][0], bounds[1][1] - bounds[1][0])
        self.best_values = []
        self.positions_history = []

    def initialize_fireflies(self):
        '''Инициализирует популяцию светлячков в случайных позициях.'''
        self.fireflies = [
            Firefly(
                position=[random.uniform(bounds[0], bounds[1])
                        for bounds in self.bounds],
                radius=self.initial_radius
            )
            for _ in range(self.num_fireflies)
        ]

    def calculate_neighbors(self, firefly: Firefly) -> List[Firefly]:
        '''

```

Продолжение Листинга А

```
        Вычисляет множество соседей светлячка.
        Параметры:
            firefly (Firefly): Текущий светлячок.
        Возвращает:
            List[Firefly]: Список соседей светлячка.
    '''
    return [
        other for other in self.fireflies
        if other is not firefly
        and math.dist(firefly.position, other.position) < firefly.radius
        and firefly.luciferin < other.luciferin
    ]

    def calculate_probabilities(self, firefly: Firefly, neighbors:
List[Firefly]) -> List[float]:
    '''
        Вычисляет вероятность перемещения к соседям.
        Параметры:
            firefly (Firefly): Текущий светлячок.
            neighbors (List[Firefly]): Список соседей.
        Возвращает:
            List[float]: Список вероятностей перемещения к каждому соседу.
    '''
    total_difference = sum(
        other.luciferin - firefly.luciferin for other in neighbors)
    if total_difference == 0:
        return [1 / len(neighbors)] * len(neighbors)
    probabilities = [(other.luciferin - firefly.luciferin) /
        total_difference for other in neighbors]
    return probabilities

    def select_neighbor(self, neighbors: List[Firefly], probabilities:
List[float]) -> Firefly:
    '''
        Выбирает соседа на основе вероятностей методом рулетки.
        Параметры:
            neighbors (List[Firefly]): Список соседей.
            probabilities (List[float]): Список вероятностей.
        Возвращает:
            Firefly: Выбранный сосед.
    '''
    cumulative_probabilities = [
        sum(probabilities[:i + 1]) for i in range(len(probabilities))]
    rand = random.random()
    for i, prob in enumerate(cumulative_probabilities):
        if rand <= prob:
            return neighbors[i]

    def adjust_radius(self, firefly: Firefly, desired_neighbors: int):
    '''
        Корректирует радиус окрестности светлячка.
        Параметры:
            firefly (Firefly): Текущий светлячок.
            desired_neighbors (int): Целевое количество соседей.
    '''
    current_neighbors = len(self.calculate_neighbors(firefly))
    new_radius = firefly.radius + self.beta * \
        (desired_neighbors - current_neighbors)
    firefly.radius = min(self.max_radius, max(self.min_radius, new_radius))

    def optimize(self) -> Tuple[List[float], float]:
```

```

'''
Запускает процесс оптимизации.
Возвращает:
    Tuple[List[float], float]: Лучшая позиция и значение целевой
функции.
'''
self.initialize_fireflies()
best_position = None
best_value = float('inf')
desired_neighbors = 5

for iteration in range(self.max_iterations):
    for firefly in self.fireflies:
        function_value = self.fitness_function(*firefly.position)
        firefly.update_luciferin(function_value, self.rho, self.gamma)

    iteration_positions = []
    for firefly in self.fireflies:
        neighbors = self.calculate_neighbors(firefly)
        if neighbors:
            probabilities = self.calculate_probabilities(
                firefly, neighbors)
            selected_neighbor = self.select_neighbor(
                neighbors, probabilities)
            firefly.move_towards(
                selected_neighbor, self.delta, self.bounds)
        iteration_positions.append(firefly.position)
    self.positions_history.append(iteration_positions)

    for firefly in self.fireflies:
        self.adjust_radius(firefly, desired_neighbors)

    for firefly in self.fireflies:
        value = self.fitness_function(*firefly.position)
        if value < best_value:
            best_value = value
            best_position = firefly.position

    self.best_values.append(best_value)
    print(f"Итерация {iteration +
        1}: Лучший результат = {best_value:.6f}")

    return best_position, best_value

def plot_history(self):
    '''Отображает график сходимости.'''
    plt.plot(self.best_values)
    plt.title("Сходимость алгоритма роя светлячков")
    plt.xlabel("Итерации")
    plt.ylabel("Лучшее значение")
    plt.show()

def visualize(self):
    '''Анимация перемещения светлячков с учётом яркости.'''
    fig, ax = plt.subplots()
    x_min, x_max = self.bounds[0]
    y_min, y_max = self.bounds[1]
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)

    def update(frame):

```

Окончание Листинга А

```
        ax.clear()
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.set_title(f"Итерация {frame + 1}")
        positions = self.positions_history[frame]
        luciferin_values = [firefly.luciferin for firefly in self.fireflies]
        max_luciferin = max(luciferin_values)
        min_luciferin = min(luciferin_values)
        normalized_brightness = [
            (1 - min_luciferin) / (max_luciferin - min_luciferin + 1e-9)
            for l in luciferin_values
        ]
        x_coords, y_coords = zip(*positions)
        ax.scatter(
            x_coords, y_coords,
            c="green",
            s=10,
            alpha=normalized_brightness
        )

        anim = FuncAnimation(
            fig, update, frames=len(self.positions_history), blit=False,
            interval=500, repeat=False
        )
        plt.show()

if __name__ == "__main__":
    swarm = FireflySwarm(
        fitness_function=goldstein_price,
        bounds=[(-2, 2), (-2, 2)],
        num_fireflies=100,
        max_iterations=200,
        beta=0.6,
        rho=0.4,
        delta=0.25,
        gamma=1.0,
        initial_radius=0.5
    )
    best_position, best_value = swarm.optimize()
    print(f"Оптимальное решение: {best_position}, Значение: {best_value:.6f}")
    swarm.plot_history()
    swarm.visualize()
```