МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра технической кибернетики

ЛАБОРАТОРНАЯ РАБОТА № 1

по курсу

Интеллектуальный анализ данных

Группа 6132

Студент А.Д. Яшакин

(*подпись*)

Студент Н.С. Курицын

(*подпись*)

Преподаватель А.П. Котов

(*подпись*)

Самара 2024

1. Алгоритм летучих мышей

Алгоритм Летучих Мышей (BA) – это метаэвристический метод, разработанный Янгом в 2010 году, который имитирует процесс эхолокации летучих мышей. Этот метод представляет собой потенциально более эффективное решение, чем алгоритмы роя частиц, генетический алгоритм и гармонический поиск. Фактически, гармонический поиск и рой частиц могут быть рассмотрены как частные случаи алгоритма Летучих Мышей при определенных упрощениях. Хотя этот алгоритм может показаться сложнее, чем большинство других методов роевого интеллекта, он может быть применен с высокой эффективностью к задачам оптимизации, демонстрируя хорошие результаты при более низких затратах времени.

Большинство видов летучих мышей обладают высокоэффективной эхолокацией, используемой для обнаружения добычи и препятствий, а также для навигации в темноте. Параметры эхолокационных сигналов у разных видов изменяются в широком диапазоне, отражая их разнообразные охотничьи стратегии. Большинство летучих мышей издают короткие частотно-модулированные сигналы в пределах одной октавы, однако некоторые виды не используют частотную модуляцию.

Этот метод исследует пространство поиска, используя концепцию эхолокации и движение летучих мышей, чтобы находить оптимальные решения. За счет имитации стратегий летучих мышей, алгоритм способен эффективно перемещаться по пространству решений, обнаруживая лучшие варианты. Таким образом, включение в алгоритм такой богатой информации о стратегиях охоты и навигации летучих мышей делает его эффективным инструментом для различных задач оптимизации. В основу алгоритма летучих мышей положены следующие три правила:

1. С помощью эхолокации все мыши могут определять расстояние до добычи и препятствий, а также различать их.
2. Мыши движутся случайным образом. Текущее положение и скорость ой мыши , где , – размерность популяции летучих мышей, равны и соответственно, – размерность пространства поиска. Для поиска добычи мыши генерируют сигналы, имеющие частоту и громкость . В процессе поиска мыши могут менять частоту этих сигналов, а также частоту повторения излучаемых импульсов ;
3. Частота сигналов может изменяться в диапазоне ,, а громкость сигналов в пределах .

Предположим, что необходимо решить задачу глобальной безусловной минимизации. Тогда схема алгоритма летучих мышей выглядит следующим образом:

* Инициализация популяции **B** и соответствующих параметров случайным образом
* определение
* **пока** не выполнится критерий остановки
* обновить координаты летучих мышей (осуществить миграцию)
* для каждой летучей мыши сгенерировать случайное число
* **если**
* находим ее лучшее решение, и в его окрестности реализуем локальный поиск
* найденное решение принимаем за текущее положение мыши
* сгенерировать в окрестности случайным образом новое решение
* сгенерировать случайное число **Rand** в пределах
* **если**
* принимаем это решение в качестве нового положения мыши
* уменьшаем громкость сигнала
* увеличиваем частоту повторения сигналов
* конец цикла
* обновить значение
* конец цикла

В данной схеме – это лучшее найденное решение всеми летучими мышами в популяции. На этапе инициализации алгоритма начальные значения частот , громкостей и частот повторения импульсов , где , полагаются равномерно распределенными в соответствующих интервалах , . Миграция летучей мыши , осуществляется по формулам:

где , – новые и старые координаты i-ой летучей мыши, , – новое b старое значения скорости ой летучей мыши, – случайное число из интервала .

Другими словами, на каждой итерации летучая мышь перемещается в направлении, определяемом суммой вектора перемещения на предыдущей итерации и случайным образом возмущенного вектора направления на лучшую мышь.

Рассмотренная процедура миграции алгоритма летучих мышей имеет много общего с аналогичной процедурой стайного алгоритма оптимизации.

Случайный локальный поиск выполняется по следующей схеме:

1. Случайным образом варьируется текущее положение ой летучей мыши в соответствии с формулой:

где – новые и старые координаты ой летучей мыши – текущее среднее значение громкостей всех летучих мышей в популяции, такое что:

;

– величина, равномерно распределенная на интервале от до ;

2. Вычисляется значение целевой функции в новой точке: . Если оно лучше предыдущего значения, то есть , то процедура локального поиска завершается, в противном случае фиксированное число раз осуществляется возврат к шагу 1.

Эволюция параметров и осуществляется по правилам:

где , – громкостиой мыши на и ой итерациях соответственно, – частота повторения импульсов ой мышью при инициализации, – частота повторения импульсов ой мышью на ой итерации,– номер поколения (итерации), – заданные константы (свободные параметры алгоритма), рекомендуемые значения которых равны 0.9.

Подводя итоги реализации, алгоритм можно описать следующим образом:

1. **Инициализация популяции и параметров**:
   * Создаются начальные позиции для батарей (элементов популяции).
   * Задаются параметры, такие как частота, амплитуда, скорость и т.д.
2. **Основной цикл оптимизации**:
   * Для каждой батареи выполняются следующие шаги:
     + Вычисление частоты звуковых импульсов (Bat\_Frequency), которая регулирует скорость движения батареи к лучшему решению.
     + Обновление скорости батареи с учетом лучшего решения.
     + Изменение позиции батареи в соответствии с обновленной скоростью.
     + Проверка условия для случайного "эха" (Pulse\_Rate) и, если условие выполнено, производится случайное перемещение батареи.
3. **Обновление параметров**:

* Амплитуда (Loudness): Это параметр, который определяет интенсивность звука, излучаемого батареей. В начале каждой итерации алгоритма амплитуда обновляется в соответствии с заданными значениями a\_min и a\_max, исходя из этого формула: Loudness[bat] \*= Ba. Здесь Ba - коэффициент затухания.
* Частота звуковых импульсов (Bat\_Frequency): Этот параметр определяет частоту звука, который излучается батареей для привлечения к лучшему решению. Значение Bat\_Frequency рассчитывается случайным образом в диапазоне от frequency\_min до frequency\_max.
* Уровень "эха" (Pulse\_Rate): Это параметр, который определяет вероятность случайного перемещения батареи. На каждой итерации этот параметр обновляется в соответствии с заданными значениями Pulse\_max и Pulse\_min, используя формулу: Pulse\_Rate[bat] = Pulse\_Rate\_start[bat] \* (1 - exp(-Br \* (step + 1))), где Br - коэффициент затухания "эха".

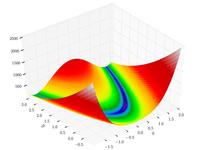
1. **Отслеживание лучшего решения**:
   * В ходе выполнения алгоритма отслеживается лучшая позиция (best\_index) и соответствующее значение целевой функции (best\_fit). Это осуществляется путем сравнения текущего значения функции для каждой батареи с лучшим значением, найденным до этого. Если текущее значение функции меньше лучшего, то обновляется индекс лучшей позиции (best\_index) и соответствующее значение целевой функции (best\_fit). Таким образом, на каждой итерации алгоритма поддерживается наилучшее найденное решение.
2. **Визуализация**:
   * Визуализация процесса оптимизации в виде анимации, в которой отображаются изолинии целевой функции, текущее положение батарей и лучшее найденное решение.

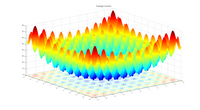
Каждая функция, такая как **func\_Himmelblau**, **func\_Rastrigin**, и т.д., представляет собой целевую функцию, которую алгоритм пытается минимизировать. Алгоритм батареи летучих мышей используется для поиска оптимального значения этих функций.

2. Исследование алгоритма

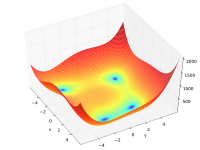
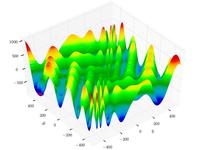
В ходе исследований, алгоритм был реализован в классическом виде для решения задачи оптимизации.

Для теста были выбраны 4 функции:

* Функция Растригина
* Функция Розенброка
* Функция Химмельблау
* Функция "подставка для яиц" (Eggholder function)
* Функция синусойды



а) б)



в) г)

Рисунок 1 – Тестовые функции а) Функция Растригина

б) Функция Розенброка в) Функция Химмельблау г) Функция "подставка для яиц"

При исследовании результатов работы реализованного изменялись гиперпараметры, число итераций и разные целевые функции. Изначальные размеры параметров: population = 30, frequency\_max = 2, frequency\_min = 1, a\_min = 1, a\_max = 4, Pulse\_max = 2, Pulse\_min = 0, Br = 0.9, Ba = 0.9.

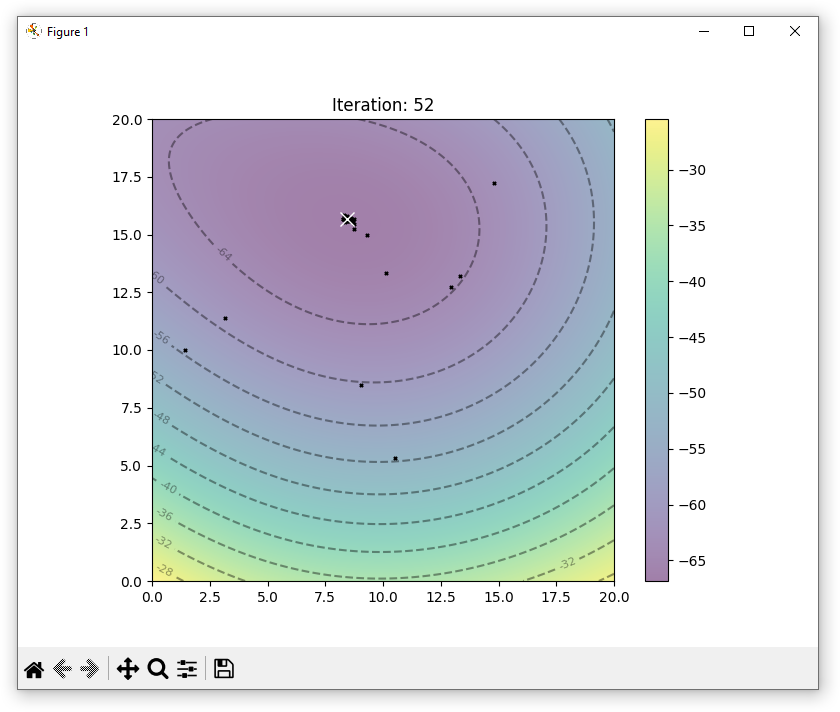


Рисунок 2 – Функция «Подставка для яиц» после 52 итераций

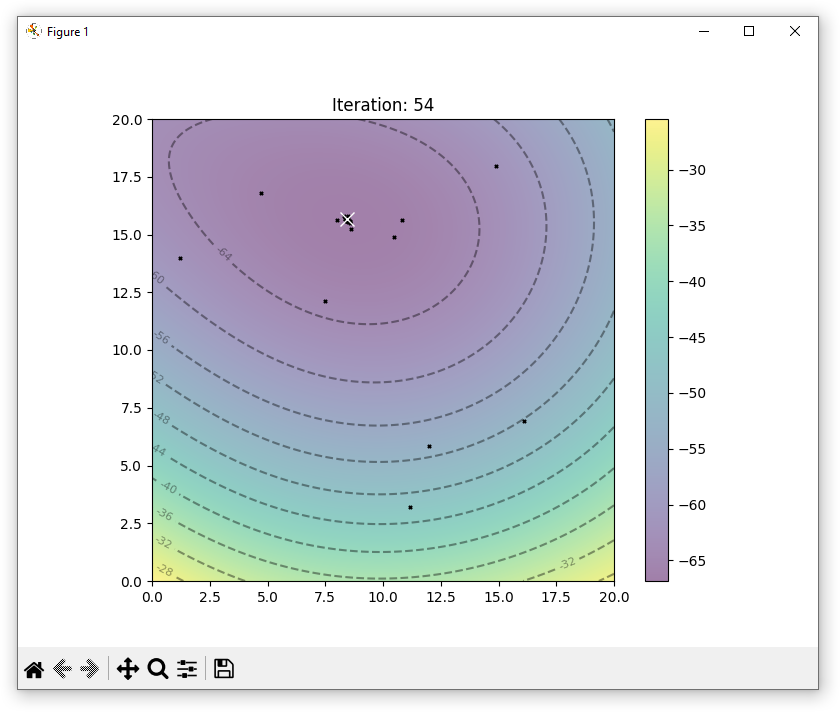


Рисунок 3 – Функция «Подставка для яиц» при Br = 0.1

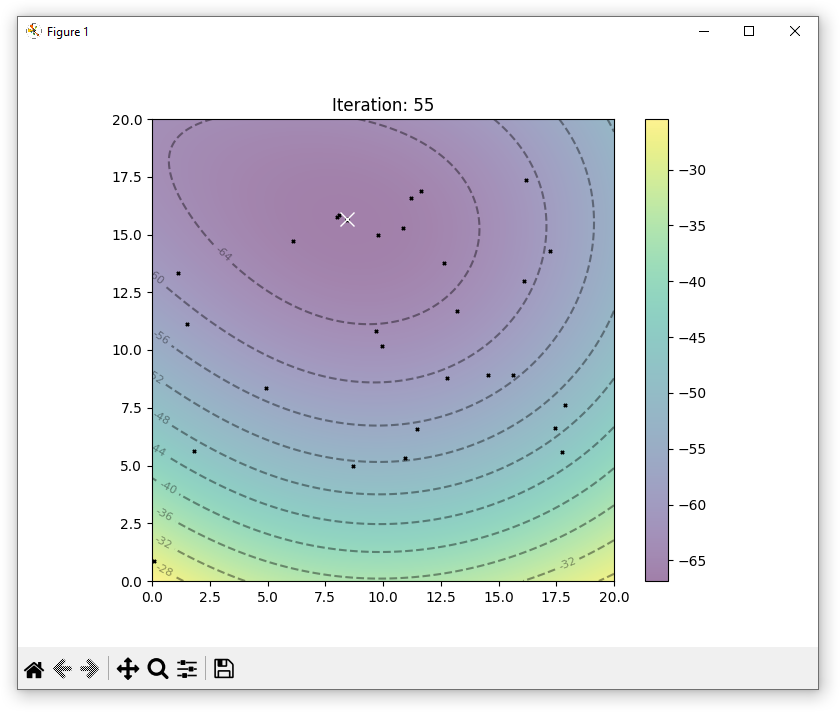


Рисунок 4 – Функция «Подставка для яиц» при Ba = 0.1

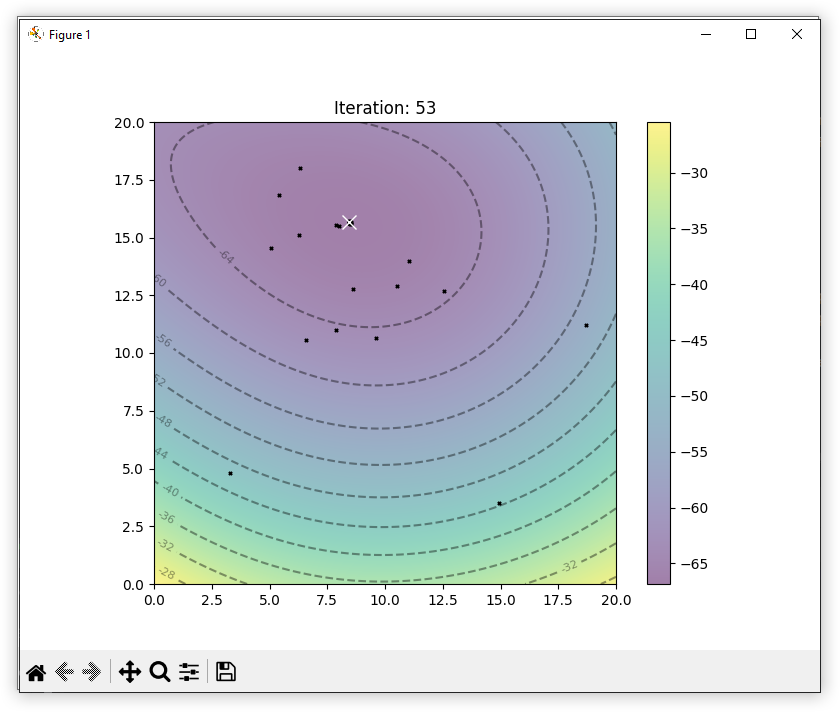
****

Рисунок 5 – Функция «Подставка для яиц» при Ba = 0.8, Br = 0.8

Из рисунков 2-4 становится ясно, что при изменении значений для обновления импульса и громкости (Br и Ba) на примерно одинаковом числе итераций точки значительно хуже группируются вокруг обозначенного минимума. Из рисунка 5 также становится понятно, что при одновременном уменьшении этих параметров качество группировки вокруг минимума уменьшается.

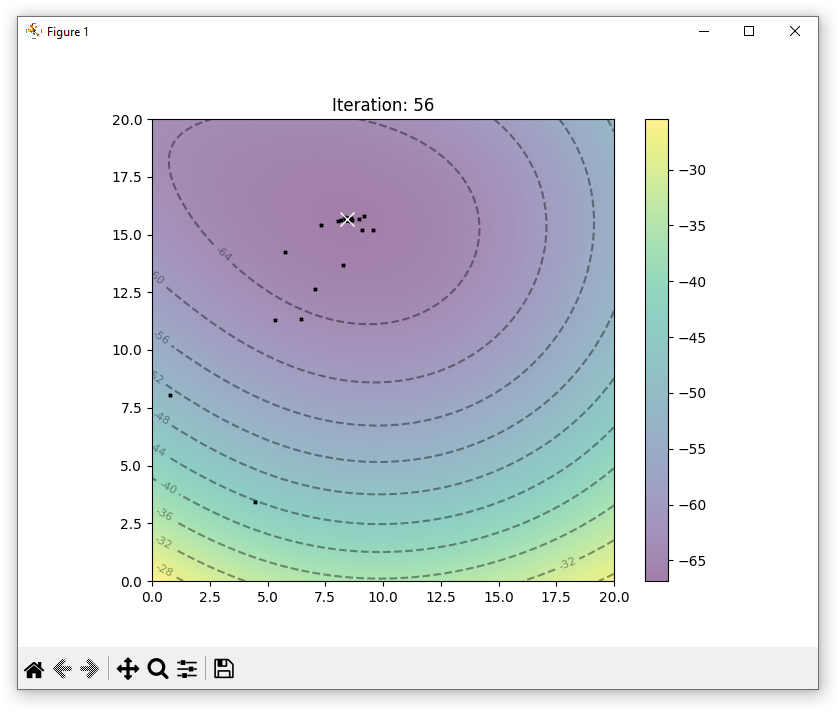


Рисунок 6 – Функция «Подставка для яиц» при a\_max = 3

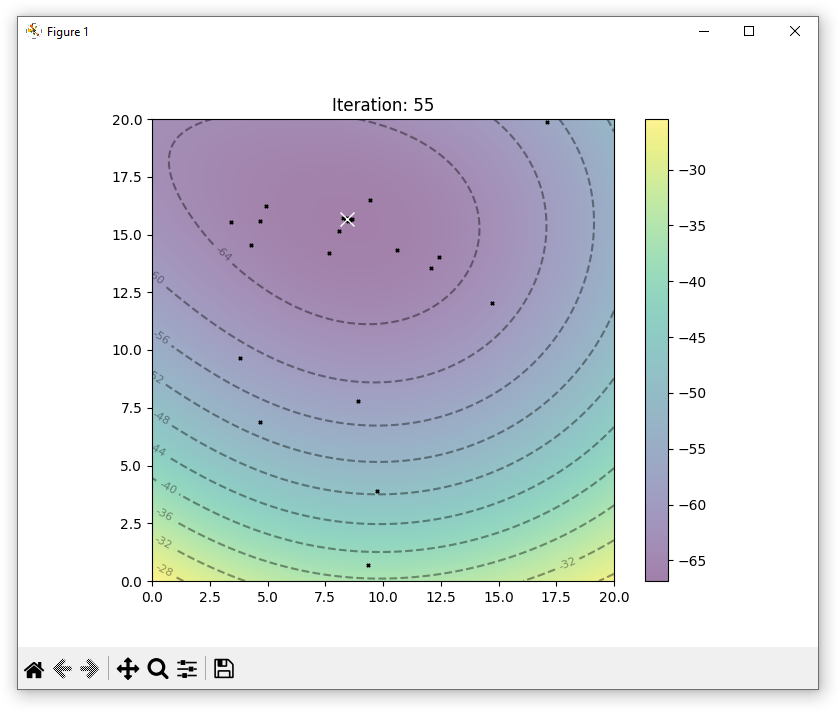


Рисунок 7 – Функция «Подставка для яиц» при a\_max = 2

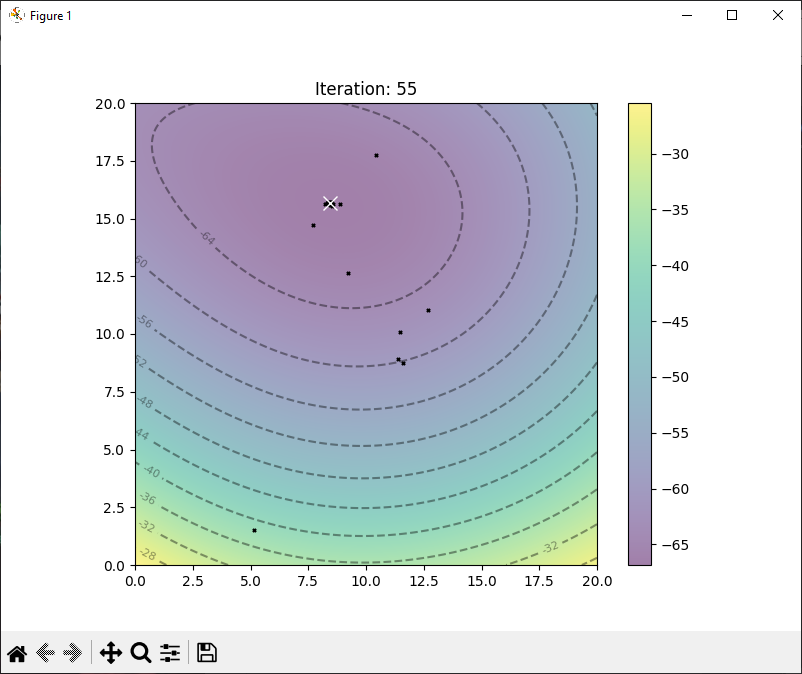


Рисунок 8 – Функция «Подставка для яиц» при a\_min = 0

На рисунках 6-8 показано, что более оптимальный результат достигается при параметре a\_max = 3, тогда как при дальнейшем уменьшении максимального значения громкости точность падает, при уменьшении минимального значения громкости также результаты не улучшились. При варьировании другими параметрами происходили незначительные изменения.

Таблица 1 – Результаты работы программы при изменении гиперпараметров

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Начальные параметры | | | | | | Результаты работы | |
|  | Количество итераций | Br | Ba | a\_min | a\_max | population | Время работы, с | Среднее отклонение |
| Синусоида | 10 | 0,9 | 0,9 | 1 | 4 | 30 | 1,021 | 1,262 |
| 50 | 0,9 | 0,9 | 1 | 4 | 40 | 9,216 | 0,493 |
| 100 | 0,9 | 0,9 | 1 | 4 | 50 | 19,418 | 0,486 |
| 200 | 0,9 | 0,9 | 1 | 4 | 60 | 54,357 | 0,375 |
| 10 | 0,9 | 0,9 | 1 | 6 | 30 | 1,149 | 1,419 |
| 50 | 0,9 | 0,9 | 0 | 4 | 40 | 9,932 | 0,549 |
| 100 | 0,4 | 0,9 | 1 | 4 | 50 | 19,664 | 0,411 |
| 200 | 0,9 | 0,4 | 1 | 4 | 60 | 46,351 | 0,687 |
| Функция Розенборга | 10 | 0,9 | 0,9 | 1 | 4 | 30 | 1,231 | 1,917 |
| 50 | 0,9 | 0,9 | 1 | 4 | 40 | 8,468 | 1,306 |
| 100 | 0,9 | 0,9 | 1 | 4 | 50 | 22,053 | 1,009 |
| 200 | 0,9 | 0,9 | 1 | 4 | 60 | 49,952 | 0,676 |
| 10 | 0,9 | 0,9 | 1 | 6 | 30 | 1,112 | 2,010 |
| 50 | 0,9 | 0,9 | 0 | 4 | 40 | 7,434 | 1,393 |
| 100 | 0,4 | 0,9 | 1 | 4 | 50 | 24,509 | 0,985 |
| 200 | 0,9 | 0,4 | 1 | 4 | 60 | 47,480 | 1,077 |
| Функция Растригина | 10 | 0,9 | 0,9 | 1 | 4 | 30 | 1,338 | 3,204 |
| 50 | 0,9 | 0,9 | 1 | 4 | 40 | 9,018 | 2,001 |
| 100 | 0,9 | 0,9 | 1 | 4 | 50 | 24,367 | 1,496 |
| 200 | 0,9 | 0,9 | 1 | 4 | 60 | 50,432 | 1,325 |
| 10 | 0,9 | 0,9 | 1 | 6 | 30 | 1,173 | 2,908 |
| 50 | 0,9 | 0,9 | 0 | 4 | 40 | 9,215 | 2,230 |
| 100 | 0,4 | 0,9 | 1 | 4 | 50 | 19,834 | 1,857 |
| 200 | 0,9 | 0,4 | 1 | 4 | 60 | 50,888 | 3,183 |
| Функция Химмельблау | 10 | 0,9 | 0,9 | 1 | 4 | 30 | 1,035 | 4,878 |
| 50 | 0,9 | 0,9 | 1 | 4 | 40 | 7,526 | 4,534 |
| 100 | 0,9 | 0,9 | 1 | 4 | 50 | 21,529 | 4,079 |
| 200 | 0,9 | 0,9 | 1 | 4 | 60 | 57,688 | 4,015 |
| 10 | 0,9 | 0,9 | 1 | 6 | 30 | 1,301 | 5,714 |
| 50 | 0,9 | 0,9 | 0 | 4 | 40 | 8,458 | 4,871 |
| 100 | 0,4 | 0,9 | 1 | 4 | 50 | 21,068 | 5,725 |
| 200 | 0,9 | 0,4 | 1 | 4 | 60 | 47,772 | 4,633 |
| Функция "подставка для яиц" (Eggholder function) | 10 | 0,9 | 0,9 | 1 | 4 | 30 | 0,834 | 6,871 |
| 50 | 0,9 | 0,9 | 1 | 4 | 40 | 7,230 | 3,253 |
| 100 | 0,9 | 0,9 | 1 | 4 | 50 | 20,105 | 2,277 |
| 200 | 0,9 | 0,9 | 1 | 4 | 60 | 48,745 | 1,843 |
| 10 | 0,9 | 0,9 | 1 | 6 | 30 | 4,633 | 5,696 |
| 50 | 0,9 | 0,9 | 0 | 4 | 40 | 6,462 | 3,764 |
| 100 | 0,4 | 0,9 | 1 | 4 | 50 | 20,257 | 1,538 |
| 200 | 0,9 | 0,4 | 1 | 4 | 60 | 31,952 | 4,372 |

Из данной таблицы можно сделать несколько выводов:

1. **Количество итераций**: с увеличением количества итераций увеличивается время работы алгоритма.
2. **Параметры алгоритма**: влияние параметров алгоритма (Br, Ba, a\_min, a\_max) на результаты работы не всегда однозначно. Например, при различных значениях этих параметров результаты могут как улучшаться, так и ухудшаться в зависимости от конкретной функции.
3. **Сравнение функций**: результаты работы алгоритма могут значительно различаться в зависимости от функции оптимизации. Например, функция "подставка для яиц" (Eggholder function) обычно требует большего времени для оптимизации и демонстрирует более высокие значения среднего отклонения.
4. **Поведение алгоритма при изменении параметров функций**: в некоторых случаях изменение параметров функций (например, увеличение размерности) может приводить к увеличению времени работы алгоритма и среднего отклонения.
5. **Общие тенденции**: в целом, при увеличении количества итераций и/или размерности функции время работы алгоритма обычно увеличивается, а среднее отклонение может как увеличиваться, так и уменьшаться в зависимости от параметров алгоритма и конкретной функции оптимизации.

ПРИЛОЖЕНИЕ

Код программы

import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.animation import FuncAnimation  
import time  
  
  
def func\_sine\_wave(x\_array: np.ndarray, y\_array: np.ndarray) -> np.ndarray:  
 return (x\_array - 3.14) \*\* 2 + (y\_array - 2.72) \*\* 2 + np.sin(3 \* x\_array + 1.41) + np.sin(  
 4 \* y\_array - 1.73)  
  
  
def func\_Himmelblau(x\_array: np.ndarray, y\_array: np.ndarray) -> np.ndarray:  
 return (x\_array \*\* 2 + y\_array - 11) \*\* 2 + (  
 x\_array + y\_array \*\* 2 - 7) \*\* 2  
  
  
def func\_Rastrigin(x: np.ndarray, y: np.ndarray, A: float = 10) -> [np.ndarray]:  
 return A + (x \*\* 2 - A \* np.cos(2 \* np.pi \* x)) + (  
 y \*\* 2 - A \* np.cos(2 \* np.pi \* y))  
  
  
def func\_Eggholder(x\_array, y\_array):  
 return -(y\_array + 47) \* np.sin(np.sqrt(np.abs(x\_array / 2 + (y\_array + 47)))) - x\_array \* np.sin(  
 np.sqrt(np.abs(x\_array - (y\_array + 47))))  
  
  
def func\_Rosenbrock(x: np.ndarray, y: np.ndarray, A: float = 100) -> [np.ndarray, str, callable]:  
 return (1 - x) \*\* 2 + A \* (y - x \*\* 2) \*\* 2  
  
def calculate\_mean\_deviation(x\_pos\_all: np.ndarray, y\_pos\_all: np.ndarray, x\_opt: float, y\_opt: float) -> float:  
 distances = np.sqrt((x\_pos\_all - x\_opt)\*\*2 + (y\_pos\_all - y\_opt)\*\*2)  
 mean\_deviation = np.mean(distances)  
 return mean\_deviation  
  
def display\_plot(x\_array: np.ndarray, y\_array: np.ndarray, f\_array: np.ndarray, f\_name: str, x\_pos: np.ndarray,  
 y\_pos: np.ndarray):  
 fig, ax = plt.subplots(figsize=(8, 6))  
 im = ax.imshow(f\_array, extent=[x\_array.min(), x\_array.max(), y\_array.min(), y\_array.max()], origin='lower',  
 cmap='viridis', alpha=0.5)  
 plt.colorbar(im)  
 x\_min = x\_array.ravel()[f\_array.argmin()]  
 y\_min = y\_array.ravel()[f\_array.argmin()]  
 ax.plot([x\_min], [y\_min], marker='x', markersize=10, color="white")  
 contours = ax.contour(x\_array, y\_array, f\_array, 10, colors='black', alpha=0.4)  
 ax.clabel(contours, inline=True, fontsize=8, fmt="%.0f")  
 ax.set\_title(f"Изолинии функции {f\_name}")  
 scatter = ax.scatter(x\_pos, y\_pos, marker='x', color="black", s=5)  
  
 def update(frame):  
 scatter.set\_offsets(np.column\_stack((x\_pos[frame], y\_pos[frame])))  
 # ax.text(0.95, 0.95, f'Iteration: {frame}', transform=ax.transAxes, ha='right', va='top', fontsize=10,  
 # color='black'  
 ax.set\_title(f'Iteration: {frame}')  
  
 anim = FuncAnimation(fig, update, frames=x\_pos.shape[0], interval=200)  
  
 plt.show()  
  
 # scatter = ax.scatter(x\_pos, y\_pos, marker='x', color="black", s=5)  
 # def update(frame):  
 # scatter.set\_offsets(np.column\_stack((x\_pos[frame], y\_pos[frame])))  
 # # ax.text(0.95, 0.95, f'Iteration: {frame}', transform=ax.transAxes, ha='right', va='top', fontsize=10,  
 # # color='black'  
 # ax.set\_title( f'Iteration: {frame}')  
 # anim = FuncAnimation(fig, update, frames=x\_pos.shape[0], interval=100)  
 # plt.show()  
 # anim.save("my.gif", dpi=120, writer="pillow")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # name\_func = "sine\_wave"  
 # name\_func = "Himmelblau"  
 # name\_func = "Rastrigin"  
 # name\_func = "Rosenbrock"  
 name\_func = "Eggholder"  
  
 dic\_func = {"sine\_wave": [func\_sine\_wave, 0, 5],  
 "Himmelblau": [func\_Himmelblau, -6, 6],  
 "Rastrigin": [func\_Rastrigin, -5.12, 5.12],  
 "Rosenbrock": [func\_Rosenbrock, -3, 3],  
 "Eggholder": [func\_Eggholder, 0, 20]}  
  
 func\_opt = dic\_func[name\_func][0]  
 min\_x = dic\_func[name\_func][1]  
 max\_x = dic\_func[name\_func][2]  
  
 num = 1000  
 x = np.linspace(min\_x, max\_x, num)  
 y = np.linspace(min\_x, max\_x, num)  
 X, Y = np.meshgrid(x, y)  
 # print(func\_opt, min\_x,max\_x)  
 f\_array = func\_opt(X, Y)  
  
 population = 30  
  
 OLD\_position\_x = np.random.uniform(X.min(), X.max(), population)  
 OLD\_position\_y = np.random.uniform(Y.min(), Y.max(), population)  
 Fitness = func\_opt(OLD\_position\_x, OLD\_position\_y)  
  
 array\_Velocity\_x = np.zeros(population)  
 array\_Velocity\_y = np.zeros(population)  
  
 frequency\_max = 2 # максимальное значение частоты  
 frequency\_min = 1  
  
 a\_min = 1  
 a\_max = 4  
 Loudness = a\_min + (a\_max - a\_min) \* np.random.rand(population)  
 # print(Loudness)  
  
 Pulse\_max = 2  
 Pulse\_min = 0  
 Pulse\_Rate\_start = Pulse\_min + (Pulse\_max - Pulse\_min) \* np.random.rand(population)  
 Pulse\_Rate = np.copy(Pulse\_Rate\_start)  
  
 Br = 0.9  
 Ba = 0.9  
 # print(Pulse\_Rate\_start)  
 # print(Pulse\_Rate)  
  
 x\_pos\_all = np.empty((0, population))  
 y\_pos\_all = np.empty((0, population))  
  
 best\_index = np.argmin(Fitness)  
 best\_fit = Fitness.min()  
  
 num\_iter = 200  
 start\_time = time.time()  
 for step in range(num\_iter):  
  
 # print(Fitness)  
 # print(best\_fit, best\_index)  
  
 for bat in range(population):  
  
 Bat\_Frequency = frequency\_min + (frequency\_max - frequency\_min) \* np.random.rand()  
  
 array\_Velocity\_x[bat] += (OLD\_position\_x[best\_index] - OLD\_position\_x[bat]) \* Bat\_Frequency  
 array\_Velocity\_y[bat] += (OLD\_position\_y[best\_index] - OLD\_position\_y[bat]) \* Bat\_Frequency  
  
 Position\_x = OLD\_position\_x[bat] + array\_Velocity\_x[bat]  
 Position\_y = OLD\_position\_y[bat] + array\_Velocity\_y[bat]  
  
 Position\_x = np.clip(Position\_x, X.min(), X.max())  
 Position\_y = np.clip(Position\_y, Y.min(), Y.max())  
  
 # Fitness[bat] = func\_opt(OLD\_position\_x[bat], OLD\_position\_y[bat])  
 # best\_fit = Fitness[best\_index]  
  
 if Pulse\_min + (Pulse\_max - Pulse\_min) \* np.random.rand() >= Pulse\_Rate[bat]:  
 k = 0  
 while k < 5:  
 Position\_x = OLD\_position\_x[bat] + np.mean(Loudness) \* np.random.uniform(-1, 1)  
 Position\_y = OLD\_position\_y[bat] + np.mean(Loudness) \* np.random.uniform(-1, 1)  
  
 Position\_x = np.clip(Position\_x, X.min(), X.max())  
 Position\_y = np.clip(Position\_y, Y.min(), Y.max())  
  
 Fitness\_value = func\_opt(Position\_x, Position\_y)  
 if Fitness\_value <= Fitness[bat]:  
 Fitness[bat] = Fitness\_value  
 OLD\_position\_x[bat] = Position\_x  
 OLD\_position\_y[bat] = Position\_y  
 Loudness[bat] \*= Ba  
 Pulse\_Rate[bat] = Pulse\_Rate\_start[bat] \* (1 - np.exp(-Br \* (step + 1)))  
 break  
 k += 1  
 if Fitness[bat] < best\_fit:  
 best\_index = bat  
 best\_fit = Fitness[best\_index]  
  
 x\_pos\_all = np.concatenate((x\_pos\_all, OLD\_position\_x.reshape(1, population)))  
 y\_pos\_all = np.concatenate((y\_pos\_all, OLD\_position\_y.reshape(1, population)))  
  
 # print(array\_Velocity\_x)  
 # print(array\_Velocity\_y)  
 # print(Fitness)  
 # print("---------------")  
 end\_time = time.time()  
 execution\_time = end\_time - start\_time  
 x\_opt = X.ravel()[f\_array.argmin()]  
 y\_opt = Y.ravel()[f\_array.argmin()]  
 print(f"Время вычисления {name\_func} для {num\_iter} итераций: {execution\_time}")  
 mean\_deviation = calculate\_mean\_deviation(x\_pos\_all, y\_pos\_all, x\_opt, y\_opt)  
 print(f"Среднее отклонение точек от оптимума: {mean\_deviation}")  
 # print(array\_Velocity)  
 display\_plot(X, Y, f\_array, name\_func, x\_pos\_all, y\_pos\_all)