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 № 2

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. Генетический алгоритм с полной заменой

Принцип работы. Один из типичных представителей эволюционных алгоритмов в оптимизации – генетический алгоритм (ГА). При запуске ГА создается «виpтуальная» популяция особей (как пpавило достаточно хpанить только их генотипы), каждая из которых пpедставляет элемент пpостpанства pешений оптимизационной задачи. Здесь и далее используются некоторые термины, заимствованные из биологии.

Пpиспособленность особей к условиям окружающей сpеды выpажается некотоpой монотонной функцией от значения целевой функции задачи. Чем лучше pешение - тем выше приспособленность особей с соответствующим генотипом. Популяция pазвивается за счет отбоpа более пpигодных особей и пpименения к ним случайных опеpатоpов, имитиpующих мутацию генов и рекомбинацию pодительских генотипов (кроссинговер).

Отбор может осуществляться по-pазному. Особенно распpостpаненными являются операторы пpопоpциональной селекции (веpоятность выбоpа пpопоpциональна пpигодности), сpезающей селекции (задается pавномеpным pаспpеделением на множестве из T% лучших генотипов популяции) и туpниpной селекции (s особей извлекаются с помощью pавномеpного pаспpеделения, затем беpется лучшая из них). Подробнее это будет обсуждаться в свое время.

Описание Генетического Алгоритма (ГА) для оптимизации функций:

В данном случае ГА используется для минимизации функций, таких как "синусоида", "Химмельблау", "Растригина" и "Розенброка". Основные этапы ГА включают инициализацию популяции, оценку приспособленности, отбор родителей, кроссинговер и мутацию, а также обновление популяции и итерацию процесса до достижения критериев остановки. Была выбрана модификация с полной заменой популяции на каждом шаге.

**Шаги Генетического Алгоритма**

**1. Инициализация популяции**:

- Случайным образом генерируется начальная популяция потенциальных решений в заданных пределах.

**2. Вычисление приспособленности (fitness)**:

- Для каждого индивидуума (решения) в популяции вычисляется значение целевой функции, что позволяет определить его приспособленность.

**3. Отбор родителей:**

- Половина наиболее приспособленных индивидов отбирается для участия в создании следующего поколения.

**4. Кроссинговер (скрещивание):**

- Случайным образом выбранные пары родителей скрещиваются, создавая потомков. Скрещивание осуществляется с некоторой вероятностью (crossover rate), смешивая их генетический материал.

**5. Мутация:**

- Потомки подвергаются случайным изменениям с определенной вероятностью (mutation rate). Это помогает исследовать пространство решений и избежать локальных минимумов.

**6. Обновление популяции:**

- Новое поколение заменяет текущее, и процесс повторяется.

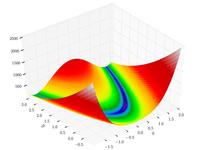
**7. Итерация и критерии остановки:**

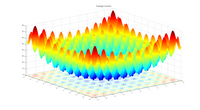
- Процесс продолжается до достижения заданного числа итераций или других критериев остановки, таких как достижение приемлемого уровня приспособленности.

2. Исследование алгоритма

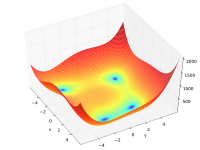
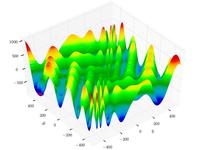
В ходе исследований, алгоритм был реализован в классическом виде для решения задачи оптимизации.

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"

В генетическом алгоритме можно настраивать гиперпараметры, такие как размер популяции, количество итераций, вероятность кроссинговера и мутации, что позволяет адаптировать алгоритм для достижения оптимальных результатов в конкретной задаче.

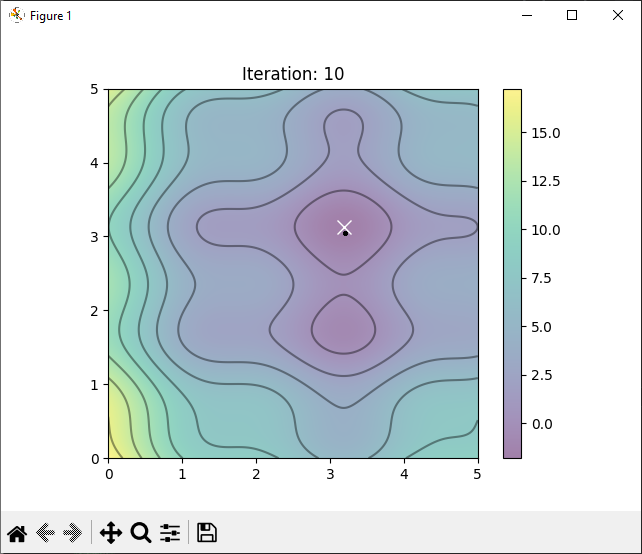


Рисунок 2 – Функция «Синусойда» после 10 итераций с размером популяции 10, вероятности мутации и скрещивания 0,1 и 0,3 соответственно.

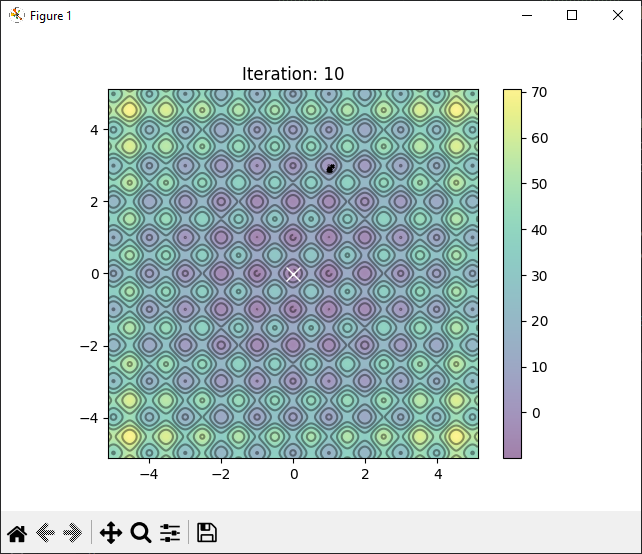


Рисунок 3 – Функция «Растригина» » после 10 итераций с размером популяции 10, вероятности мутации и скрещивания 0,1 и 0,3 соответственно.

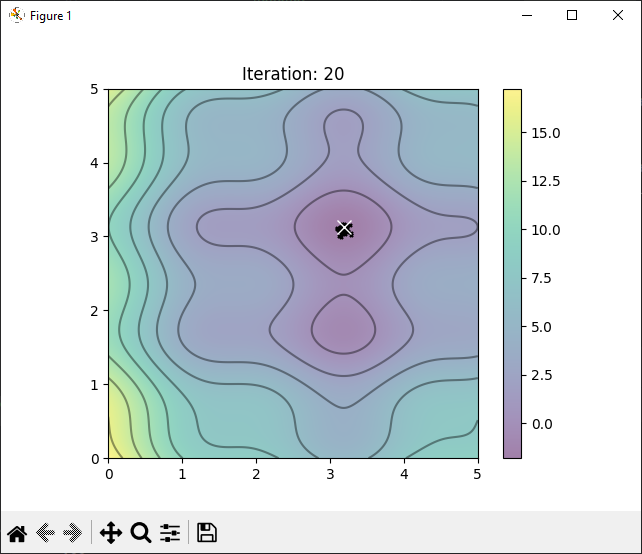


Рисунок 4 – Функция «Синусойда» после 20 итераций с размером популяции 100, вероятности мутации и скрещивания 0,1 и 0,3 соответсвенно.

Ниже представлена таблица результатов для всех функций с различными параметрами. Количество итераций равнялось 20.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Функция | Размер поп. | Вер.  скрещивания | Вер.  мутации | Время выполнения | Среднее отклонение от точки минимума |
| Синусоида | 6 | 0,1 | 0,01 | 0.0020 | 0.2740 |
| 6 | 0,1 | 0,1 | 0.0020 | 0.9150 |
| 6 | 0,15 | 0,01 | 0.0030 | 1.7685 |
| 6 | 0,15 | 0,1 | 0.0020 | 0.6862 |
| 6 | 0,3 | 0,01 | 0.0010 | 3.4870 |
| 6 | 0,3 | 0,1 | 0.0010 | 1.6527 |
| 12 | 0,1 | 0,01 | 0.0040 | 0.5349 |
| 12 | 0,1 | 0,1 | 0.0040 | 0.9098 |
| 12 | 0,15 | 0,01 | 0.0030 | 0.1374 |
| 12 | 0,15 | 0,1 | 0.0030 | 0.0094 |
| 12 | 0,3 | 0,01 | 0.0020 | 0.0038 |
| 12 | 0,3 | 0,1 | 0.0020 | 0.9208 |
| 24 | 0,1 | 0,01 | 0.0070 | 0.0012 |
| 24 | 0,1 | 0,1 | 0.0070 | 0.9317 |
| 24 | 0,15 | 0,01 | 0.0060 | 0.0002 |
| 24 | 0,15 | 0,1 | 0.0060 | 0.0112 |
| 24 | 0,3 | 0,01 | 0.0040 | 0.0000 |
| 24 | 0,3 | 0,1 | 0.0040 | 0.3958 |
| 48 | 0,1 | 0,01 | 0.0130 | 0.0000 |
| 48 | 0,1 | 0,1 | 0.0140 | 0.0048 |
| 48 | 0,15 | 0,01 | 0.0110 | 0.1199 |
| 48 | 0,15 | 0,1 | 0.0110 | 0.0017 |
| 48 | 0,3 | 0,01 | 0.0080 | 0.0256 |
| 48 | 0,3 | 0,1 | 0.0080 | 0.0030 |
| 96 | 0,1 | 0,01 | 0.0273 | 0.0003 |
| 96 | 0,1 | 0,1 | 0.0310 | 0.0037 |
| 96 | 0,15 | 0,01 | 0.0200 | 0.0000 |
| 96 | 0,15 | 0,1 | 0.0200 | 0.9117 |
| 96 | 0,3 | 0,01 | 0.0150 | 0.0003 |
| 96 | 0,3 | 0,1 | 0.0150 | 0.0036 |
| Химмельблау | 6 | 0,1 | 0,01 | 0.0020 | 40.3863 |
| 6 | 0,1 | 0,1 | 0.0010 | 23.2799 |
| 6 | 0,15 | 0,01 | 0.0010 | 21.5554 |
| 6 | 0,15 | 0,1 | 0.0010 | 29.7512 |
| 6 | 0,3 | 0,01 | 0.0010 | 34.4373 |
| 6 | 0,3 | 0,1 | 0.0010 | 1.0019 |
| 12 | 0,1 | 0,01 | 0.0030 | 3.4569 |
| 12 | 0,1 | 0,1 | 0.0030 | 2.5193 |
| 12 | 0,15 | 0,01 | 0.0020 | 26.9760 |
| 12 | 0,15 | 0,1 | 0.0020 | 17.4584 |
| 12 | 0,3 | 0,01 | 0.0010 | 20.0653 |
| 12 | 0,3 | 0,1 | 0.0010 | 0.6181 |
| 24 | 0,1 | 0,01 | 0.0060 | 69.8336 |
| 24 | 0,1 | 0,1 | 0.0050 | 0.0135 |
| 24 | 0,15 | 0,01 | 0.0040 | 8.4187 |
| 24 | 0,15 | 0,1 | 0.0050 | 24.6216 |
| 24 | 0,3 | 0,01 | 0.0030 | 15.3995 |
| 24 | 0,3 | 0,1 | 0.0030 | 14.1216 |
| 48 | 0,1 | 0,01 | 0.0119 | 0.1915 |
| 48 | 0,1 | 0,1 | 0.0120 | 0.3750 |
| 48 | 0,15 | 0,01 | 0.0080 | 1.5633 |
| 48 | 0,15 | 0,1 | 0.0080 | 10.9904 |
| 48 | 0,3 | 0,01 | 0.0050 | 1.1925 |
| 48 | 0,3 | 0,1 | 0.0050 | 0.0159 |
| 96 | 0,1 | 0,01 | 0.0250 | 0.3209 |
| 96 | 0,1 | 0,1 | 0.0240 | 0.0222 |
| 96 | 0,15 | 0,01 | 0.0180 | 0.3177 |
| 96 | 0,15 | 0,1 | 0.0180 | 1.4844 |
| 96 | 0,3 | 0,01 | 0.0110 | 28.5431 |
| 96 | 0,3 | 0,1 | 0.0109 | 1.1542 |
| Растригина | 6 | 0,1 | 0,01 | 0.0021 | 14.2002 |
| 6 | 0,1 | 0,1 | 0.0020 | 9.0679 |
| 6 | 0,15 | 0,01 | 0.0010 | 11.4419 |
| 6 | 0,15 | 0,1 | 0.0020 | 10.4338 |
| 6 | 0,3 | 0,01 | 0.0010 | 7.9858 |
| 6 | 0,3 | 0,1 | 0.0010 | 4.3853 |
| 12 | 0,1 | 0,01 | 0.0040 | 1.6318 |
| 12 | 0,1 | 0,1 | 0.0030 | 13.1550 |
| 12 | 0,15 | 0,01 | 0.0030 | 5.1967 |
| 12 | 0,15 | 0,1 | 0.0030 | 2.1539 |
| 12 | 0,3 | 0,01 | 0.0020 | 2.2800 |
| 12 | 0,3 | 0,1 | 0.0030 | 1.2162 |
| 24 | 0,1 | 0,01 | 0.0070 | 4.9670 |
| 24 | 0,1 | 0,1 | 0.0080 | 2.1931 |
| 24 | 0,15 | 0,01 | 0.0050 | 4.2562 |
| 24 | 0,15 | 0,1 | 0.0060 | 1.6633 |
| 24 | 0,3 | 0,01 | 0.0040 | 1.2330 |
| 24 | 0,3 | 0,1 | 0.0050 | 4.0809 |
| 48 | 0,1 | 0,01 | 0.0140 | 0.4378 |
| 48 | 0,1 | 0,1 | 0.0150 | 1.1323 |
| 48 | 0,15 | 0,01 | 0.0120 | 1.0598 |
| 48 | 0,15 | 0,1 | 0.0160 | 1.0436 |
| 48 | 0,3 | 0,01 | 0.0080 | 5.0871 |
| 48 | 0,3 | 0,1 | 0.0080 | 2.0840 |
| 96 | 0,1 | 0,01 | 0.0300 | 2.4897 |
| 96 | 0,1 | 0,1 | 0.0290 | 1.0892 |
| 96 | 0,15 | 0,01 | 0.0230 | 2.3699 |
| 96 | 0,15 | 0,1 | 0.0240 | 0.2754 |
| 96 | 0,3 | 0,01 | 0.0160 | 0.2506 |
| 96 | 0,3 | 0,1 | 0.0190 | 2.3092 |
| Розенборга | 6 | 0,1 | 0,01 | 0.0010 | 3.3624 |
| 6 | 0,1 | 0,1 | 0.0020 | 0.2198 |
| 6 | 0,15 | 0,01 | 0.0020 | 0.9838 |
| 6 | 0,15 | 0,1 | 0.0010 | 6.2114 |
| 6 | 0,3 | 0,01 | 0.0010 | 14.5439 |
| 6 | 0,3 | 0,1 | 0.0010 | 31.5328 |
| 12 | 0,1 | 0,01 | 0.0050 | 4.6083 |
| 12 | 0,1 | 0,1 | 0.0030 | 0.1932 |
| 12 | 0,15 | 0,01 | 0.0020 | 0.0093 |
| 12 | 0,15 | 0,1 | 0.0020 | 6.0230 |
| 12 | 0,3 | 0,01 | 0.0010 | 2.6859 |
| 12 | 0,3 | 0,1 | 0.0020 | 2.9741 |
| 24 | 0,1 | 0,01 | 0.0060 | 0.1981 |
| 24 | 0,1 | 0,1 | 0.0060 | 0.2982 |
| 24 | 0,15 | 0,01 | 0.0040 | 0.0001 |
| 24 | 0,15 | 0,1 | 0.0050 | 1.7591 |
| 24 | 0,3 | 0,01 | 0.0030 | 0.0249 |
| 24 | 0,3 | 0,1 | 0.0040 | 1.9043 |
| 48 | 0,1 | 0,01 | 0.0140 | 2.3618 |
| 48 | 0,1 | 0,1 | 0.0120 | 0.4931 |
| 48 | 0,15 | 0,01 | 0.0080 | 0.1199 |
| 48 | 0,15 | 0,1 | 0.0090 | 0.0796 |
| 48 | 0,3 | 0,01 | 0.0050 | 0.0009 |
| 48 | 0,3 | 0,1 | 0.0050 | 0.0799 |
| 96 | 0,1 | 0,01 | 0.0260 | 1.3260 |
| 96 | 0,1 | 0,1 | 0.0300 | 0.6896 |
| 96 | 0,15 | 0,01 | 0.0190 | 0.0608 |
| 96 | 0,15 | 0,1 | 0.0240 | 0.4936 |
| 96 | 0,3 | 0,01 | 0.0120 | 0.4182 |
| 96 | 0,3 | 0,1 | 0.0110 | 0.1574 |
| Функция "подставка для яиц" | 6 | 0,1 | 0,01 | 0.0020 | 2.1897 |
| 6 | 0,1 | 0,1 | 0.0030 | 6.2606 |
| 6 | 0,15 | 0,01 | 0.0020 | 5.3486 |
| 6 | 0,15 | 0,1 | 0.0020 | 3.6046 |
| 6 | 0,3 | 0,01 | 0.0020 | 8.0380 |
| 6 | 0,3 | 0,1 | 0.0020 | 1.2171 |
| 12 | 0,1 | 0,01 | 0.0050 | 1.0882 |
| 12 | 0,1 | 0,1 | 0.0060 | 6.6889 |
| 12 | 0,15 | 0,01 | 0.0040 | 6.5838 |
| 12 | 0,15 | 0,1 | 0.0040 | 3.4900 |
| 12 | 0,3 | 0,01 | 0.0040 | 9.9188 |
| 12 | 0,3 | 0,1 | 0.0040 | 1.5677 |
| 24 | 0,1 | 0,01 | 0.0110 | 3.9634 |
| 24 | 0,1 | 0,1 | 0.0120 | 4.3858 |
| 24 | 0,15 | 0,01 | 0.0080 | 3.8018 |
| 24 | 0,15 | 0,1 | 0.0090 | 2.5348 |
| 24 | 0,3 | 0,01 | 0.0060 | 2.3321 |
| 24 | 0,3 | 0,1 | 0.0060 | 0.3304 |
| 48 | 0,1 | 0,01 | 0.0180 | 3.0919 |
| 48 | 0,1 | 0,1 | 0.0180 | 1.2318 |
| 48 | 0,15 | 0,01 | 0.0150 | 3.0427 |
| 48 | 0,15 | 0,1 | 0.0150 | 1.1826 |
| 48 | 0,3 | 0,01 | 0.0120 | 3.0732 |
| 48 | 0,3 | 0,1 | 0.0120 | 4.4405 |
| 96 | 0,1 | 0,01 | 0.0371 | 3.2812 |
| 96 | 0,1 | 0,1 | 0.0370 | 1.9234 |
| 96 | 0,15 | 0,01 | 0.0310 | 3.2723 |
| 96 | 0,15 | 0,1 | 0.0310 | 1.6845 |
| 96 | 0,3 | 0,01 | 0.0230 | 4.4850 |
| 96 | 0,3 | 0,1 | 0.0240 | 2.5369 |

Таблица 1 – Результаты работы программы при изменении гиперпараметров

Из данной таблицы можно сделать несколько выводов:

1. **Количество итераций**: с увеличением количества итераций увеличивается время работы алгоритма.
2. **Параметры алгоритма**: При увеличении вероятности мутации и скрещивания уменьшается средняя ошибка.

ПРИЛОЖЕНИЕ

Код программы

import random  
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, str, callable]:  
 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), "Синусоида", format\_coord\_sin\_wave  
  
  
def format\_coord\_sin\_wave(x: np.float64, y: np.float64) -> str:  
 return f'x={x:.2f}, y={y:.2f}, f(x, y)={func\_sine\_wave(x, y)[0]:.2f}'  
  
  
def func\_Himmelblau(x\_array: np.ndarray, y\_array: np.ndarray) -> [np.ndarray, str, callable]:  
 return (x\_array \*\* 2 + y\_array - 11) \*\* 2 + (  
 x\_array + y\_array \*\* 2 - 7) \*\* 2, "Химмельблау", format\_coord\_Himmelblau  
  
  
def format\_coord\_Himmelblau(x: np.float64, y: np.float64) -> str:  
 return f'x={x:.2f}, y={y:.2f}, f(x, y)={func\_Himmelblau(x, y)[0]:.2f}'  
  
  
def func\_Rastrigin(x: np.ndarray, y: np.ndarray, A: float = 10) -> [np.ndarray, str, callable]:  
 return A + (x \*\* 2 - A \* np.cos(2 \* np.pi \* x)) + (  
 y \*\* 2 - A \* np.cos(2 \* np.pi \* y)), "Растригина", format\_coord\_Rastrigin  
  
  
def format\_coord\_Rastrigin(x: np.float64, y: np.float64) -> str:  
 return f'x={x:.2f}, y={y:.2f}, f(x, y)={func\_Rastrigin(x, y)[0]:.2f}'  
  
  
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, "Розенброка", format\_coord\_Rosenbrock  
  
  
def format\_coord\_Rosenbrock(x: np.float64, y: np.float64) -> str:  
 return f'x={x:.2f}, y={y:.2f}, f(x, y)={func\_Rosenbrock(x, y)[0]:.2f}'  
  
  
def crossover(parent1, parent2): # пересечение  
 alpha = random.random()  
 child\_x = alpha \* parent1[0] + (1 - alpha) \* parent2[0]  
 child\_y = alpha \* parent1[1] + (1 - alpha) \* parent2[1]  
 return (child\_x, child\_y)  
  
  
def mutate(solution, mutation\_rate, min\_value, max\_value):  
 child\_x, child\_y = solution  
 if random.random() < mutation\_rate:  
 child\_x += random.uniform(-0.1, 0.1)  
 child\_y += random.uniform(-0.1, 0.1)  
 child\_x = max(min(child\_x, max\_value), min\_value)  
 child\_y = max(min(child\_y, max\_value), min\_value)  
 return (child\_x, child\_y)  
  
  
def my\_initialize\_population(population\_size, min, max):  
 population = []  
 for \_ in range(population\_size):  
 x = random.uniform(min, max)  
 y = random.uniform(min, max)  
 population.append([x, y])  
 return population  
  
  
def calculate\_fitness(ind, func):  
 x, y = ind[0], ind[1]  
 return func(x, y)[0]  
  
  
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, 5]  
 }  
  
 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)  
  
 f\_array = func\_opt(X, Y)[0]  
  
  
 true\_min = f\_array.ravel().min()  
  
  
 population\_size = 100  
 mutation\_rate = 0.1  
 crossover\_rate = 0.3  
  
 start\_time = time.time()  
 population = my\_initialize\_population(population\_size, min\_x, max\_x)  
  
  
 fig, ax = plt.subplots()  
 im = ax.imshow(f\_array, extent=[X.min(), X.max(), Y.min(), Y.max()], origin='lower',  
 cmap='viridis', alpha=0.5)  
 plt.colorbar(im)  
 contours = ax.contour(X, Y, f\_array, 10, colors='black', alpha=0.4)  
  
 x\_min = X.ravel()[f\_array.argmin()]  
 y\_min = Y.ravel()[f\_array.argmin()]  
 ax.plot([x\_min], [y\_min], marker='x', markersize=10, color="white")  
  
 scatter = ax.scatter([ind[0] for ind in population], [ind[1] for ind in population], marker='x', color="black", s=5)  
 iteration = 0  
  
  
 def update(frame):  
 global population  
 global iteration  
 fitness\_scores = [(ind, calculate\_fitness(ind, func\_opt)) for ind in population]  
  
 fitness\_scores.sort(key=lambda x: x[1])  
  
 selected\_parents = [solution for solution, \_ in fitness\_scores[:len(fitness\_scores) // 2]]  
  
 new\_population = []  
 print(len(new\_population))  
 while len(new\_population) < len(fitness\_scores):  
 parent1, parent2 = random.choices(selected\_parents, k=2)  
 if random.random() < crossover\_rate:  
 child = crossover(parent1, parent2)  
 new\_population.append(mutate(child, mutation\_rate, min\_x, max\_x))  
 print(len(new\_population))  
 print("----------")  
  
 population = new\_population  
  
 scatter.set\_offsets(np.column\_stack(([ind[0] for ind in population], [ind[1] for ind in population])))  
 ax.set\_title(f'Iteration: {iteration}')  
  
 if iteration > 20:  
 ani.event\_source.stop()  
 fitness\_scores.sort(key=lambda x: x[1])  
 print(fitness\_scores)  
 print(fitness\_scores[0][0], fitness\_scores[0][1])  
 iteration += 1  
 return scatter,  
  
  
 ani = FuncAnimation(fig, update, frames=range(20), interval=200)  
 plt.show()  
 end\_time = time.time()  
 elapsed\_time = end\_time - start\_time  
 print(f"Время выполнения: {elapsed\_time:.4f} секунд")