Государственное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

09.03.02 410008

**«Параллельные вычислительные системы»**

Семестр 7

**ОТЧЁТ**

по Лабораторной работе №8

**«Многопоточность в Python на примере генетического алгоритма»**



Преподаватель: Кашин И.В.

Студент : Шардаков В.А.

Группа : Фт-410008

Дата : 11.12.2024

Екатеринбург 2024

# Задачи

В данной работе поставлена задача нахождения 1800 минимумов заданной функции на определённом промежутке с использованием метода Генетических алгоритмов. Данная задача требует эффективного и точного подхода для идентификации минимумов функции, что может быть особенно полезно в различных областях, таких как оптимизация, машинное обучение и исследование функций. Исходная функция, на которой будут проводиться вычисления, будет илюстрирована графически (см. Рисунок 1).

# Ход работы

## Метод решения задачи

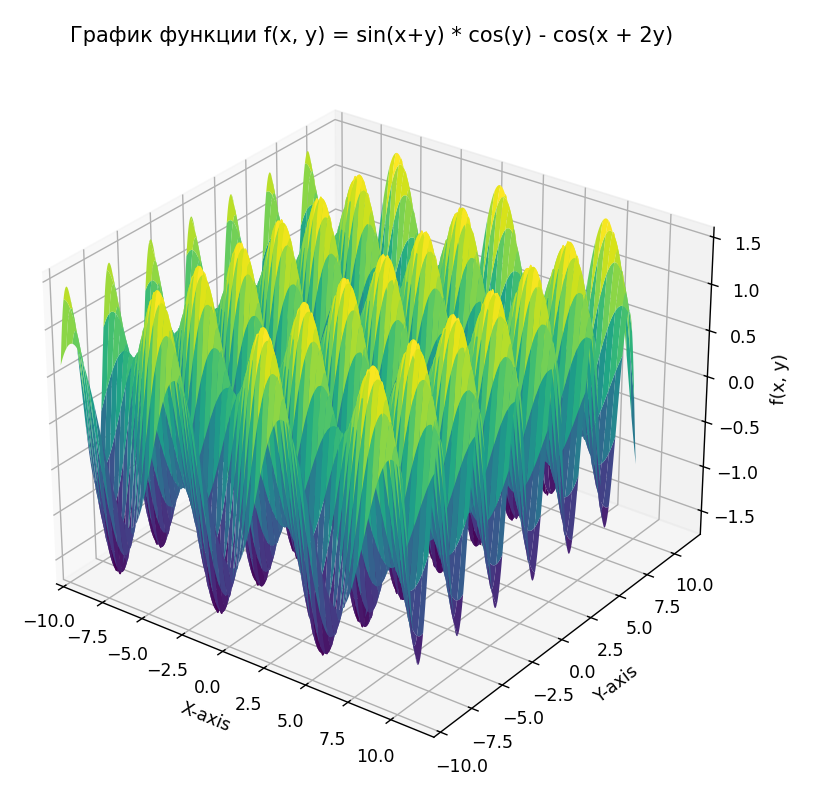
Для Решение задачи будет осуществляться с использованием алгоритма Генетических алгоритмов, как описано ниже:

1. **Инициализация**: случайным образом создаётся K точек, представляющих возможные решения задачи.
2. **Отбор**: из созданных точек производится отбор 50% лучших, основываясь на их минимальной значении.
3. **Выведение новых агентов**: для восстановления количества агентов применяется метод мутации или скрещивания двух агентов. Это позволяет создавать новые возможные решения, основанные на лучших текущих агентах.
4. **Повторение цикла**: Процесс отбора и выведения новых агентов (пункты 2–3) повторяется, позволяя алгоритму эволюционировать и улучшать решения.
5. **Определение минимумов**: Точки, которые образуются в скоплениях после нескольких итераций, принимаются за минимумы функции.
6. **Окончательная оценка**: Алгоритм повторяется до тех пор, пока не будет найдено необходимое количество минимумов (1800). При добавлении нового минимума производится оценка его близости к уже найденным минимумам; если новая точка слишком близка к старым минимумам, она отклоняется.

## Решение

***Результат работы программы (ПРИЛОЖЕНИЕ А)***

*Исходный график функции:*

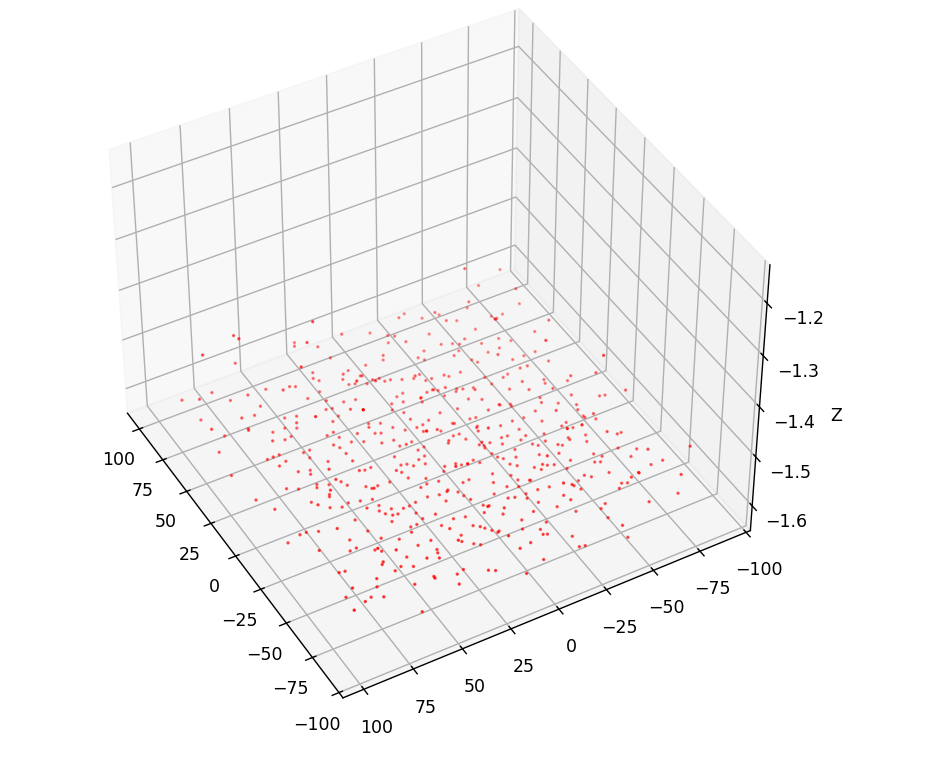
**

*Полученные минимумы:*

Изображение выглядит как текст, снимок экрана, линия

Автоматически созданное описание

*Полученные минимумы в 3D:*



*График зависимости времени от количества потоков:*

*Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание*

# Выводы

Применение Генетических алгоритмов для поиска минимумов функции демонстрирует эффективный подход к решению оптимизационных задач. Это метод позволяет гибко итерировать через множество возможных решений, улучшая результаты на каждом шаге, что делает его мощным инструментом в области вычислительной оптимизации. Метод подходит для решения задач, где традиционные подходы могут оказаться неэффективными из-за сложности зависимости функции или большого числа возможных решений. В ходе реализации алгоритма могут быть получены интересные результаты, которые потребуют дальнейшего анализа и исследования, что подчеркивает важность данной работы.

# ПРИЛОЖЕНИЕ А

import multiprocessing  
import time  
import numpy as np  
import matplotlib.pyplot as plt  
import random  
from scipy.spatial.distance import cdist  
  
X\_MIN = -100  
X\_MAX = 100  
Y\_MIN = -100  
Y\_MAX = 100  
N\_MIN = 1800  
K = 5000  
P = K // 2  
R\_N = X\_MAX // 20  
h = 10  
  
N\_ITERATIONS = 15  
  
  
def f(point):  
 x, y = point  
 return np.sin(x + y) \* np.cos(y) + np.cos(x + 2 \* y)  
  
  
def mutate(x):  
 sign\_h = 1 - np.random.randint(0, 1, size=2) \* 2  
 return x + h \* sign\_h  
  
  
def cross(x):  
 l = x.shape[0]  
 alpha = np.random.uniform(0.1, 0.9, l)  
 new\_points = np.zeros((l, 2))  
 for i in range(l):  
 ind = np.random.choice(l, size=2, replace=False)  
 x1, y1 = x[ind[0]]  
 x2, y2 = x[ind[1]]  
 new\_points[i] = np.array([x1 \* (1 - alpha[i]) + x2 \* alpha[i], y1 \* (1 - alpha[i]) + y2 \* alpha[i]])  
 return new\_points  
  
  
def find\_min(queue, points):  
 while True:  
 # Случайное задание точек  
 points = np.random.uniform(X\_MIN, X\_MAX, size=(K, 2))  
  
 # N\_ITERATIONS эволюционных шагов  
 for i in range(N\_ITERATIONS):  
 # Эволюционный шаг  
 points = np.array(sorted(points, key=lambda x: f(x)))  
  
 points = points[:P]  
  
 step\_type = random.randint(0, 1)  
 if step\_type:  
 points = np.concatenate((points, mutate(points)), axis=0)  
 else:  
 points = np.concatenate((points, cross(points)), axis=0)  
  
 min\_points = []  
 points = np.array(sorted(points, key=lambda x: f(x)))  
  
 points = points[:P]  
 z = np.array([f(point) for point in points])  
  
 # Отбор точек  
 while True:  
 for point in points:  
  
 distances = np.linalg.norm(points - point, axis=1)  
 suitable\_points = distances < 10  
 if suitable\_points.sum() < 10:  
 continue  
 min\_points.append(point)  
 points = points[~suitable\_points]  
 break  
 else:  
 break  
  
 min\_list = queue.get()  
 if len(min\_list) >= 1800:  
 queue.put(min\_list)  
 break  
  
 if len(min\_points) == 0:  
 queue.put(min\_list)  
 continue  
  
 if len(min\_list) > 0:  
 min\_points\_arr = np.array(min\_points)  
 global\_mins = np.array(min\_list)  
 distances = np.linalg.norm(min\_points\_arr[:, np.newaxis] - global\_mins, axis=-1)  
 suitable\_distance\_id = np.all(distances > 3, axis=1)  
 suitable\_points = min\_points\_arr[suitable\_distance\_id].tolist()  
 queue.put(min\_list + suitable\_points)  
 print(len(min\_list))  
 else:  
 queue.put(min\_points)  
  
  
def calculate(target\_function, procs\_num, queue):  
 # список процессов  
 processes\_list = []  
  
 # запуск процессов  
 for \_ in range(procs\_num):  
 p = multiprocessing.Process(target=target\_function, args=(queue, ""))  
 processes\_list.append(p)  
 p.start()  
 # Ждем, пока все процессы завершат работу.  
 for p in processes\_list:  
 p.join()  
  
  
def main():  
 # Создание очереди  
 m = multiprocessing.Manager()  
 queue = m.Queue()  
  
 time\_list = [0] \* 8  
 proc\_list = [i for i in range(1, 9)]  
 n\_exp = 1  
  
 for \_ in range(n\_exp):  
  
 for proc in proc\_list:  
 queue.put([])  
 start = time.time()  
 calculate(find\_min, proc, queue)  
 end = time.time()  
 t = end - start  
 print(f"t для {proc} проц: {t}")  
  
 time\_list[proc - 1] += t / n\_exp  
  
 mins = queue.get()  
 mins = np.array(mins)  
 z = np.array([f(point) for point in mins])  
 print(max(z))  
 print(np.mean(z))  
 plt.scatter(mins[:, 0], mins[:, 1], s=0.9)  
 plt.show()  
  
 fig = plt.figure()  
 ax = fig.add\_subplot(projection='3d')  
 ax.scatter(mins[:, 0], mins[:, 1], z, color="red", s=0.9)  
 ax.set\_zlabel('Z')  
 plt.show()  
  
 plt.ylabel("время, с")  
 plt.xlabel("количество потоков")  
 plt.title("Зависисмость времени от количества потоков")  
 plt.plot(range(1, 9), time\_list)  
 plt.plot(range(1, 9), [time\_list[0] / i for i in range(1, 9)], label="1/T1")  
 plt.legend()  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()