**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ФИЗИКИ

ЛАБОРАТОРНАЯ РАБОТА

**МЕТОД ГРАДИЕНТНОГО СПУСКА**

Работу выполнил:

магистр 1-го курса магистратуры

Галимзянов Рафиль

Группа 06-129

Казань-2022

Архитектура системы:

Ноутбук: Acer Swift 3

Процессор, частота: AMD Ryzen 3 3200U, 2.60 GHz

Количество потоков: 4

Операционная система: Windows 11

Python:3.9.2

Метод градиентного спуска

**Градиентный спуск** — метод нахождения локального экстремума (минимума или максимума) функции с помощью движения вдоль градиента

Изображение выглядит как текст

Автоматически созданное описание

Из формулы видно, что для точки x0 производная отражает функцию в точке x0. Мгновенная скорость изменения в точке x0. Распространяясь на многомерные функции, существует концепция градиента, которая представляет собой комбинацию векторов, отражающих направление максимальной скорости изменения в многомерной графике.

Основная идея алгоритма состоит в варьировании некоторой начальной точки x0 в сторону наискорейшего уменьшения:

Где λ – это шаг сходимости алгоритма.

Аналогично данный алгоритм распространяется на функции нескольких переменных, при заданном шаге, будет происходить изменение каждой переменной ФНП независимо друг от друга.

|  |
| --- |
|  |
| Рис. 1. Окно программы во время выполнения скрипта в 1 потоке. |

|  |
| --- |
|  |
| Рис. 2. Окно программы во время выполнения в многопоточном режиме (4 потока). |

|  |
| --- |
|  |
| Рис. 3. Окно программы во время выполнения в многопоточном режиме (2 потока). |

**Программа:**

from threading import Thread  
import time  
import numpy as np  
import matplotlib.pyplot as plt  
  
N = 1000000 # число итераций   
lmd = 0.0000001 # шаг сходимости  
  
  
def f(x1, x2, x3, x4):  
 return x1 \*\* 3 + 2 \* x1 \* x2 + x2 \*\* 3 + 2 \* x2 \* x3 + x3 \*\* 2 + x4 \*\* 2  
  
  
def df1(x1, x2, x3, x4):  
 return 3 \* x1 \*\* 2 + 2 \* x2  
  
  
def df2(x1, x2, x3, x4):  
 return 2 \* x1 + 3 \* x2 \*\* 2 + 2 \* x3  
  
  
def df3(x1, x2, x3, x4):  
 return 2 \* x2 + 2 \* x3  
  
  
def df4(x1, x2, x3, x4):  
 return 2 \* x4  
  
  
start = time.time()  
  
  
def grad\_min\_x1(N, lmd):  
 x0, y0, z0, k0 = 1, 1, 1, 1  
  
 for i in range(N):  
 x0 = x0 - lmd \* df1(x0, y0, z0, k0) #изменение аргумента на текущей итерации  
 print(f'x = {x0} \n')  
 return x0  
  
  
def grad\_min\_x2(N, lmd):  
 x0, y0, z0, k0 = 1, 1, 1, 1  
 for i in range(N):  
 y0 = y0 - lmd \* df2(x0, y0, z0, k0)  
 print(f'y = {y0} \n')  
 return y0  
  
  
def grad\_min\_x3(N, lmd):  
 x0, y0, z0, k0 = 1, 1, 1, 1  
 for i in range(N):  
 z0 = z0 - lmd \* df3(x0, y0, z0, k0)  
 print(f'z = {z0} \n')  
 return z0  
  
  
def grad\_min\_x4(N, lmd):  
 x0, y0, z0, k0 = 1, 1, 1, 1  
 for i in range(N):  
 k0 = k0 - lmd \* df4(x0, y0, z0, k0)  
 print(f'k = {k0} \n')  
 return k0  
  
  
x0 = grad\_min\_x1(N, lmd)  
y0 = grad\_min\_x2(N, lmd)  
z0 = grad\_min\_x3(N, lmd)  
k0 = grad\_min\_x4(N, lmd)  
  
print(f'Затраченное время: {time.time() - start} \n')

### МНОГОПОТОЧНОЕ ВЫЧИСЛЕНИЕ

start = time.time()   
  
my\_thread1 = Thread(target=grad\_min\_x1, args=(N, lmd))  
my\_thread2 = Thread(target=grad\_min\_x2, args=(N, lmd))  
my\_thread3 = Thread(target=grad\_min\_x3, args=(N, lmd))  
my\_thread4 = Thread(target=grad\_min\_x4, args=(N, lmd))  
my\_thread1.start()  
my\_thread2.start()  
my\_thread3.start()  
my\_thread4.start()  
  
print(f'Затраченное время: {time.time()-start}')

**Заключение:**

При последовательном выполнении метода градиентного спуска общее время, затраченное на выполнение, оказалось равным t1 = 2.028 с.

При многопоточном ускорении программы удалось сократить время примерно в 4 раза, t4 = 0.444 с. (4 потока) и t2 = 1.947 с. (2 потока)