Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине «Теоретические интеллектуальные и информационные технологии»

Тема: «Подключение внешних модулей и применение параллельного подхода»

Выполнил:

Студент 1 курса

Группы ИИ-21(1)

Кирилович А.А.

Проверил:

Анфилец С. В.

Брест 2022

**Цель:** написать алгоритм случайного поиска с использованием параллельных вычислений.

**Ход работы:**

Изменим текст кода из лабораторной работы №3 для параллельных вычислений:

#include <iostream>

#include <cstdlib>

#include <math.h>

#include <omp.h>

#include <random>

#include <fstream>

using namespace std;

float y(float x, float b) {

return cos(x) + \

1 / b \* cos(9 \* x + 1) + \

1 / pow(b, 2) \* cos(pow(9, 2) \* x + 2) + \

1 / pow(b, 3) \* cos(pow(9, 3) \* x + 3) + \

1 / pow(b, 4) \* cos(pow(9, 4) \* x + 4);

}

int main() {

setlocale(LC\_CTYPE, "rus");

float b, alfaMin = 0.00001, c = 0.8;

const int Nthread = 1000;

float arrX[Nthread];

int iterator = 0, Nmax, n;

cout << "Начальная точка выберется рандомом.\n";

cout << "Начальный шаг alfa равен 2.\n";

cout << "Введите значение b: ";

cin >> b;

cout << "Введите количество итераций Nmax: ";

cin >> Nmax;

cout << "Введите параметр терпения n: ";

cin >> n;

ofstream file;

file.open("Results.txt");

random\_device randX0;

mt19937 genX0(randX0());

uniform\_int\_distribution<> distX0(-750000, 750000);

random\_device rand;

mt19937 gen(rand());

uniform\_int\_distribution<> dist(-100000, 100000);

#pragma omp parallel num\_threads(Nthread)

{

float x0 = float(distX0(genX0)) / 100000;

float x1, alfa = 2;

int j = n;

float rand;

for (int i = 0; i < Nmax; i++) {

rand = float(dist(gen)) / 100000;

x1 = x0 + alfa \* rand;

if (y(x1, b) < y(x0, b)) {

x0 = x1;

j = n;

}

else --j;

alfa \*= c;

if (alfa <= alfaMin) break;

if (j <= 0) break;

}

#pragma omp critical

{

arrX[iterator] = x0;

iterator++;

}

}

for (int i = 0; i < Nthread; i++) file << arrX[i] << endl;

file.close();

double buffer = arrX[0];

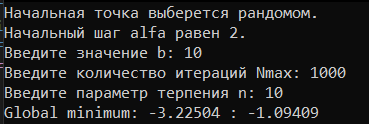
for (int i = 1; i < Nthread; i++) {

if (y(arrX[i], b) < buffer) buffer = arrX[i];

}

cout << "Global minimum: " << buffer << " : " << y(buffer, b);

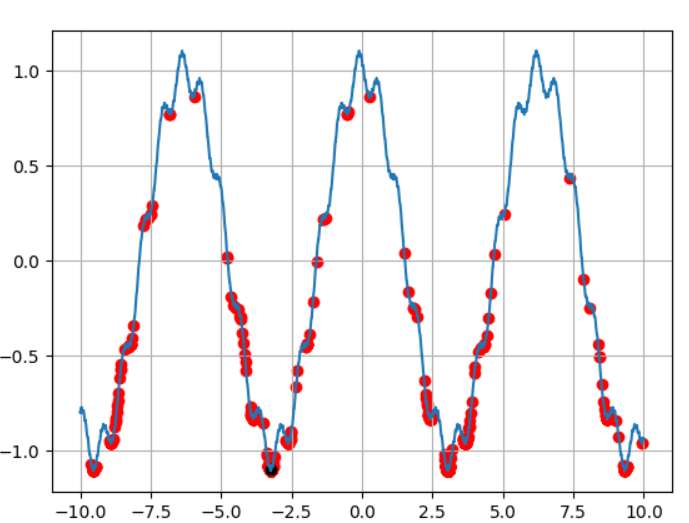
}



Для наглядности нарисуем график:

(красные точки – все точки, которые вычислили 1000 потоков, локальные экстремумы)

(черная точка – глобальный экстремум)



**Вывод:** с увеличением количества точек, увеличивается вероятность найти глобальный экстремум.