Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №4

по методам оптимизации

Факультет: ПМИ

Группа: ПМ-01

Студент: Конев А.М., Ряховский М.И.

Вариант: 1

Преподаватель: Черникова О.С., Чимитова Е.В.

Новосибирск

2013

# Цель работы

Ознакомится с методами штрафных функций при решении задач нелинейного программирования. Изучить типы штрафных и барьерных функций, их особенности, способы и области применения, влияние штрафных функций на сходимость алгоритмов, зависимость точности решения задачи нелинейного программирования от величины коэффициента штрафа.

# Задания варианта

Решить задачу нелинейного программирования при ограничениях: .

# Нахождение локального минимума, стратегия изменения прямоугольников

Изначально прямоугольник выбирается так, чтобы в него входили точки целевой области, в противном случае прямоугольник будет увеличиваться, пока данное условие не будет выполнено.

В процессе поиска локального минимума на очередной итерации выполняется поиск минимума в прямоугольнике, новый минимум становится центром следующего прямоугольника а расстояние от центра до сторон изменится по следующему соотношению:

, где .

Вычисление останавливается, когда . Так же, чтобы метод был более эффективен для слабо меняющихся функций, при  длина шага меняется по следующему соотношению: 

# Распараллеливание алгоритма поиска минимума в прямоугольнике

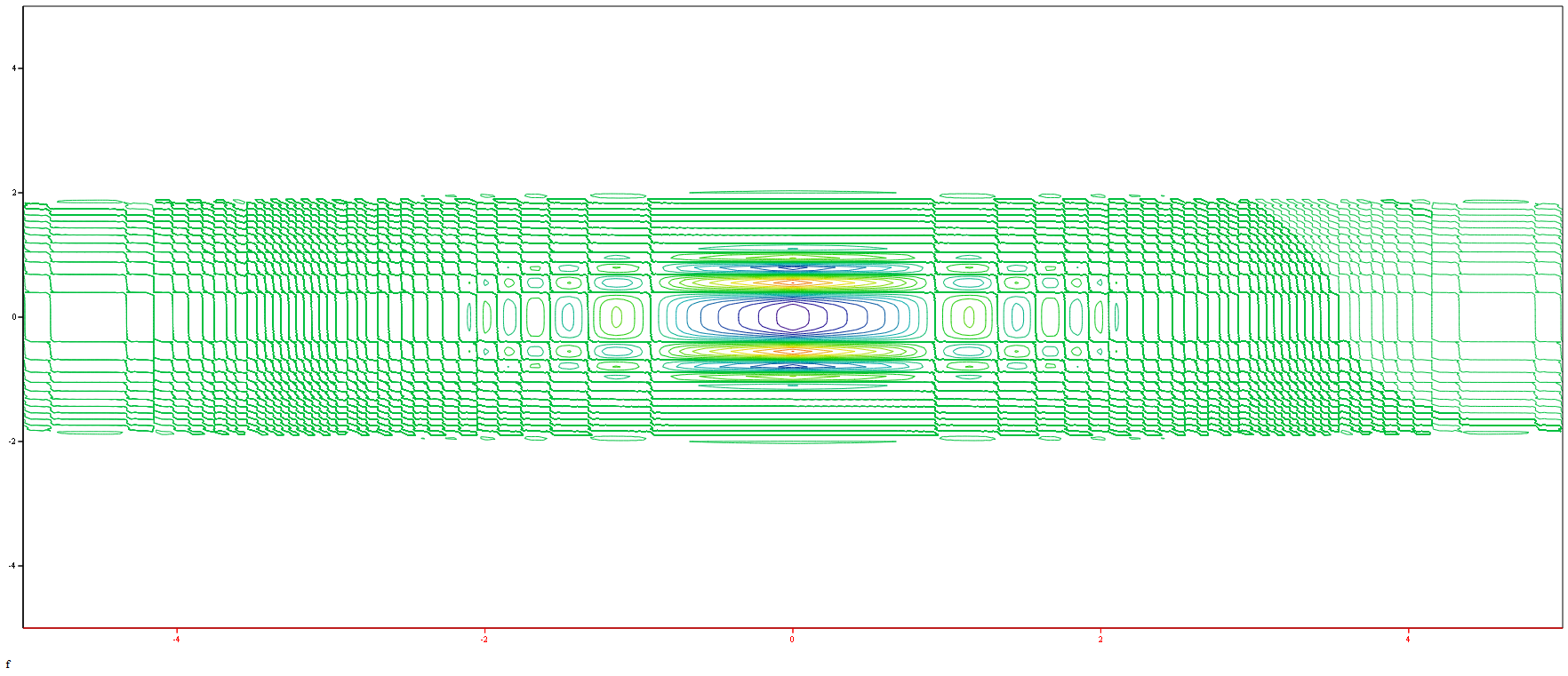
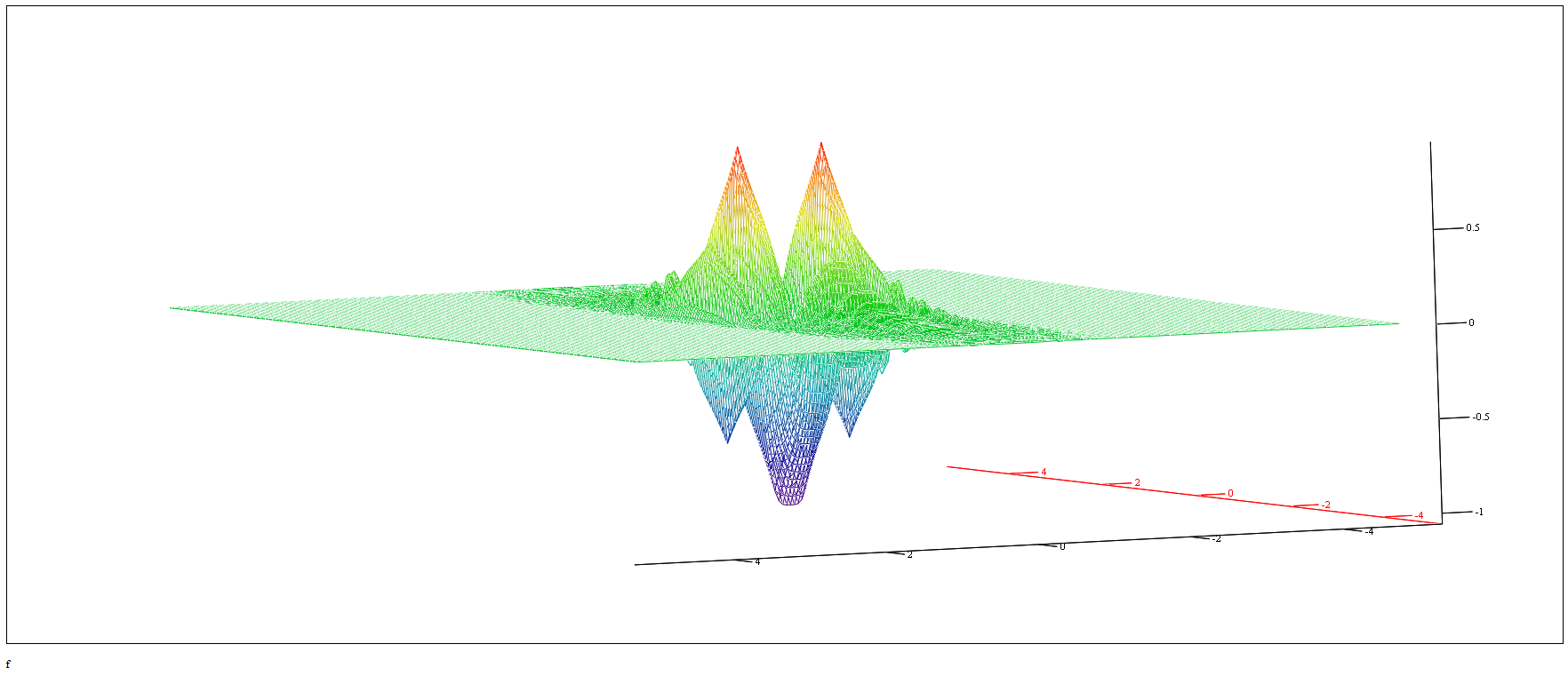
Поскольку генерация точек в одном прямоугольнике не зависима, от её можно распараллелить. Пусть всего надо сгенерировать  точек и есть  потоков. Каждому потоку выделим  точек и разделим их на группы по  точек. В каждом потоке будем генерировать точек, затем искать минимум, выбирая его за текущий. После завершение выберем минимум из числе - результатов по потокам.

# Нахождение глобального минимума

Тестовая функция: 

Минимум: 

Её график и линии уровня (красным цветом выделена ось ):



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер теста | Центр прямоугольника |  |  |  |  | Количество проб |  |  |  |
| 1 |  | 5 | 5 |  |  | 402358 | -0.000015 | -0.037522 | -0.999883 |
| 2 |  | 5 | 5 |  |  | 40235947 | -0.000015 | 0.002915 | -0.999985 |
| 3 |  | 5 | 5 |  |  | 575645 | -0.000015 | -0.019517 | -0.999977 |
| 4 |  | 5 | 5 |  |  | 57564626 | -0.000015 | -0.037523 | -0.999884 |
| 5 |  | 5 | 5 |  |  | 1151290 | -0.000015 | -0.019517 | -0.999977 |
| 6 |  | 5 | 5 |  |  | 115129252 | -0.000015 | -0.001053 | -0.999985 |
| 7 |  | 2 | 3 |  |  | 276307 | 0.000305 | 0.019608 | -0.999687 |
| 8 |  | 2 | 3 |  |  | 27631018 | 0.000061 | -0.001816 | -0.999939 |
| 9 |  | 2 | 3 |  |  | 276307 | 0.000296 | 0.032020 | -0.999650 |
| 10 |  | 2 | 3 |  |  | 27631018 | -0.000009 | -0.005335 | -0.999991 |
| 11 |  | 2 | 3 |  |  | 18420678 | 0.000006 | 0.000140 | -0.999994 |
| 12 |  | 2 | 3 |  |  | 36841356 | 0.000006 | 0.000140 | -0.999994 |

# Нахождение локального минимума

Функция: 

Локальный минимум: 

Под числом итераций понимается число прямоугольников, построенных в ходе выполнения алгоритма.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер теста | Центр начального прямоугольника |  |  |  |  | Общее количество проб |  |  |  | Число итераций |
| 1 |  | 10 | 10 |  |  | 61348 | 0.000278 | -0.993180 | 25.002828 | 8 |
| 2 |  | 10 | 10 |  |  | 6115019 | 0.000128 | -0.982947 | 25.001570 | 8 |
| 3 |  | 10 | 10 |  |  | 611229409 | 0.000008 | -0.999984 | 25.000080 | 8 |
| 4 |  | 10 | 10 |  |  | 56323 | 0.001253 | -0.976115 | 25.013101 | 2 |
| 5 |  | 10 | 10 |  |  | 5630307 | 0.000076 | -0.981336 | 25.001107 | 2 |
| 6 |  | 10 | 10 |  |  | 665509010 | 0.000059 | -0.999924 | 25.000587 | 3 |
| 7 |  | 1 | 1 |  |  | 1545 | 0.000218 | -0.999003 | 25.002178 | 7 |
| 8 |  | 1 | 1 |  |  | 156357 | 0.000104 | -0.999851 | 25.001045 | 7 |
| 9 |  | 1 | 1 |  |  | 15628458 | 0.000003 | -0.993785 | 25.000066 | 7 |
| 10 |  | 0.1 | 0.1 |  |  | 133266957 | 0.000000 | -0.999288 | 25.000004 | 60 |

# Выводы

Случайные методы поиска, для достижения нужной точности(особенно не направленные), могут требовать больше вычислительных затрат, чем обычные методы. Однако, их можно оптимизировать (приведённые методы хорошо параллелятся) и получить выигрыш по времени. Так же они позволяют находить как локальный, так и глобальный минимум, функции, когда все обычные методы ориентированы на нахождение локального минимума.

При решении задачи с ограничениями у случайных методы перед методами барьерных и штрафных функций есть как преимущества, так и недостатки. Преимущества – мы ищем минимум исходной функции, не искажая её добавками и так же точки мы можем генерировать только в исходной области, что может ускорить сходимость. Недостатки – требуется большее число раз вычислять функцию и сходимость имеет вероятностный характер, поэтому сильно зависит от правильности и надежности генератора, а так же его близости к нужному виду распределения(в нашем случае – к равномерному).

# Код программы

## Файл «best\_sample\_search.h»

#pragma once

#include <stdio.h>

#include <omp.h>

#include <vector>

#include <algorithm>

#include "random\_generator.h"

using namespace std;

//Класс, реализующий метод наилучшей пробы с направляющим гиперквадратом, в области

//x - y <= 1

//0 <= x <= 4

class best\_sample\_search{

public:

void init\_rectangle(vect\_2d s\_x0, double s\_hx, double s\_hy); //Установка центральной точки прямоугольника и начальных шагов по сеткам

void init\_tests(double s\_eps, double s\_P); //Установка числа генерируемых точек(оно вычиялется каждый раз через точность, вероятность и площадь прямоугольника)

void init\_func(func\_2d s\_func); //Установка искомой функции

void init\_max\_iters(int s\_max\_iters = 100); //Установка максимального числа итераций

void init\_eps\_grow(double s\_eps\_grow = 1E-1); //Установить значеине разности функций, при котором следует увеличить квадрат

vect\_2d search(); //Поиск минимума

private:

vect\_2d x0; //текущее приближение

func\_2d min\_f; //минимизиуемая функци

double min\_f\_val; //текущее значение минимума

int max\_iters; //Максимальное число смен прямоугольников

vect\_2d left\_bottom, right\_top; //Вершины прямоугольника

double hx, hy; //Длины старон прямоугольника

void change\_rect(); //Сменить прямоугольник

double eps; //Точность (минимальная длина стороны прямоугольника)

double P; //вероятность, нахождение минимума с нужной точностью

double eps\_grow; //Значеине разности функций, при котором следует увеличить квадрат

int tests\_N; //количество генерируеммых точек(в одном прямоуголльнике)

random\_generator rand\_gen; //генератор случайных чисел

bool check\_bounds(vect\_2d ch\_p); //Проверяет принадлежит ли точка ch\_p указанным ограничениям, если точка в области - true, иначе - false

};

## Файл «easy\_random\_search.h»

#pragma once

#include <stdio.h>

#include <omp.h> //пареллель!

#include <vector>

#include <algorithm>

using namespace std;

#include "random\_generator.h"

//Класс, реализующий простейщий случайный

class easy\_random\_search{

public:

void init\_rectangle(vect\_2d s\_x0, double s\_hx, double s\_hy); //Установка центра прямоугольника и растояний от центра до края

void init\_tests(double eps, double P); //Установка числа генерируемых точек (устанавливается точность и требуемая вероятность нахождения, число итераций высчитывается)

void init\_func(func\_2d s\_func); //Установка искомой функции

vect\_2d search(); //Поиск минимума

private:

vect\_2d x0; //текущее приближение

func\_2d min\_f; //минимизиуемая функци

double min\_f\_val; //текущее значение минимума

vect\_2d left\_bottom, right\_top; //Вершины прямоугольника

long int tests\_N; //количество генерируеммых точек

random\_generator rand\_gen; //генератор случайных чисел

};

## Файл «random\_generator.h»

#pragma once

#include <stdlib.h>

#include "extra.h"

typedef double(\*func\_2d)(vect\_2d); //функция двух переменных

//Класс, реализующий генератор случайных двумерных точек в прямоугольнике

class random\_generator{

public:

//Конструктуор по умолчанию - стандартное семя 0

random\_generator(){

srand(0);

}

//Конструктор с некоторым другим семенем

random\_generator(unsigned int core){

srand(core);

}

//Функция, генерирующая случайную точку в указанном прямоугольнике

//left\_bottom - нижняя левая вершина прямоугольника

//right\_top - правая верхняя вершина прямоугольника

vect\_2d generate(vect\_2d left\_bottom, vect\_2d right\_top){

//Два случайных числа в отрезке [0,1]

double c\_x = (rand()\*1.0 + 1.0)/ RAND\_MAX;

double c\_y = (rand()\*1.0 + 1.0)/ RAND\_MAX;

//Предвиним числа в наш прямоугольник

double h\_x = fabs(right\_top.x - left\_bottom.x); //длина прямоугольника по оси x

double h\_y = fabs(right\_top.y - left\_bottom.y); //длина прямоугольника по оси y

double x = c\_x \* h\_x + left\_bottom.x; //координата x генерируемой точки

double y = c\_y \* h\_y + left\_bottom.y; //координата y генерируемой точки

return vect\_2d(x, y);

}

};

## Файл «extra.h»

#pragma once

#include <math.h>

#define N\_dim 2 //размерность пространства

//Двухмерный вектор и матрица

struct vect\_2d{

double x, y;

vect\_2d(){

}

vect\_2d(double s\_x, double s\_y){

x = s\_x; y = s\_y;

}

vect\_2d operator+ (vect\_2d op2){

return vect\_2d(x+op2.x, y+op2.y);

}

vect\_2d operator- (vect\_2d op2){

return vect\_2d(x-op2.x, y-op2.y);

}

double operator\* (vect\_2d op2){

return x\*op2.x + y\*op2.y;

}

vect\_2d operator\* (double op2){

return vect\_2d(op2\*x, op2\*y);

}

double norm(){

return sqrt(x\*x + y\*y);

}

double& operator[] (int i){

if(i==0) return x;

if(i==1) return y;

}

};

## Файл «best\_sample\_search.cpp»

#include "best\_sample\_search.h"

//Константы для параллелизма

const int ch\_n = 4; //количество потоков

void best\_sample\_search::init\_func(func\_2d s\_func){

min\_f = s\_func;

}

void best\_sample\_search::init\_rectangle(vect\_2d s\_x0, double s\_hx, double s\_hy){

x0 = s\_x0;

hx = s\_hx;

hy = s\_hy;

change\_rect();

}

void best\_sample\_search::init\_tests(double s\_eps, double s\_P){

eps = s\_eps;

P = s\_P;

}

void best\_sample\_search::init\_max\_iters(int s\_max\_iters){

max\_iters = s\_max\_iters;

}

void best\_sample\_search::init\_eps\_grow(double s\_eps\_grow){

eps\_grow = s\_eps\_grow;

}

void best\_sample\_search::change\_rect(){

left\_bottom = x0 - vect\_2d(hx/2, hy/2);

right\_top = x0 + vect\_2d(hx/2, hy/2);

}

bool best\_sample\_search::check\_bounds(vect\_2d ch\_p){

if(ch\_p.x >= 0 && ch\_p.x <= 4 && ch\_p.x - ch\_p.y <= 1)

return true;

else

return false;

}

vect\_2d best\_sample\_search::search(){

// vect\_2d xk; //Очередное приближение

// double fk; //Значение в приближении

int teaste = 0; //общее количество проб

min\_f\_val = min\_f(x0);

//Проверяем содержит ли квадрат точки области, если не содержит - увеличиваем квадрат, пока не начнёт содержать

if(!check\_bounds(x0)){

while(!(check\_bounds(left\_bottom) || check\_bounds(right\_top) || check\_bounds(x0+vect\_2d(hx/2,0)) || check\_bounds(x0+vect\_2d(-hx/2,0)) || check\_bounds(x0+vect\_2d(0,hy/2)) || check\_bounds(x0+vect\_2d(0,-hy/2)))){

hx \*= 2.0;

hy \*= 2.0;

change\_rect();

}

}

FILE \*out\_f = fopen("iters\_info.txt", "w");

//Собственно поиск

bool cycle\_continue = true; //дополнительные условия выхода из цикла

for(int iter = 0; iter < max\_iters && cycle\_continue; iter++){

printf("%d\t%.7lf\t%.7lf\t%.15lf\t%.7lf\t%.7lf\n", iter, x0.x, x0.y, min\_f\_val, hx, hy);

fprintf(out\_f, "%d\t%.7lf\t%.7lf\t%.15lf\t%.7lf\t%.7lf\n", iter, x0.x, x0.y, min\_f\_val, hx, hy);

//Поиск минимума в текущем прямоугольнике

double min\_val\_st = min\_f\_val; //минимальное занчение с предыдущей итерации

double P\_eps = eps\*eps / (hx \* hy);

tests\_N = log(1 - P) / log(1 - P\_eps);

printf("%d\t%d\n", iter, tests\_N);

teaste += tests\_N;

//============ Начало паралелльной версии =============

int p\_iter; //интераро по потокам

int it;

long int thread\_n = tests\_N / ch\_n + 1; //количество чисел на поток

vector<vect\_2d> points; //генерируемые точки

vector<double> vals; //значения функции

vector<vect\_2d> rez\_p;

vector<double> rez\_f;

rez\_p.resize(ch\_n);

rez\_f.resize(ch\_n);

for(int i = 0 ; i < ch\_n; i++){

rez\_p[i] = x0;

rez\_f[i] = min\_f\_val;

}

int num; //номер текущего потока

#pragma omp parallel default(shared)

{

#pragma omp for private(points, vals, num)

for(p\_iter = 0; p\_iter < ch\_n; p\_iter++){

num = omp\_get\_thread\_num(); //узнаём номер потока

const int N = 10000; //количесво чисел, которые мы генерируем за раз

points.resize(N);

vals.resize(N);

#pragma omp parallel default(shared)

{

#pragma omp for private(iter)

for(iter = 0; iter < thread\_n / N + 1; iter++){

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

do{

points[i] = rand\_gen.generate(left\_bottom, right\_top);

}while(!check\_bounds(points[i]));

vals[i] = min\_f(points[i]);

}

int min\_n = distance(vals.begin(), min\_element(vals.begin(), vals.end())); //номер минимального элемента

if(vals[min\_n] < rez\_f[num]){

rez\_p[num] = points[min\_n];

rez\_f[num] = vals[min\_n];

}

}

}

}

}

int glob\_min\_n = distance(rez\_f.begin(), min\_element(rez\_f.begin(), rez\_f.end()));

x0 = rez\_p[glob\_min\_n];

min\_f\_val = rez\_f[glob\_min\_n];

//Последовательная версия

/\*

xk = rand\_gen.generate(left\_bottom, right\_top);

while(!check\_bounds(xk)){

xk = rand\_gen.generate(left\_bottom, right\_top);

}

//printf("%.8lf\t%.8lf\n", xk.x, xk.y);

fk = min\_f(xk);

if(fk < min\_f\_val){

x0 = xk;

min\_f\_val = fk;

}

}\*/

double alpha = min\_val\_st / min\_f\_val; //коэффциент, на который уменьшается прямоугольник

//Двигаем прямоугольник

double diff = fabs(min\_val\_st - min\_f\_val);

if(diff > eps\_grow){

hx /= alpha;

hy /= alpha;

}

else{

hx \*= alpha;

hy \*= alpha;

}

change\_rect();

cycle\_continue = (hx > eps || hy > eps) && diff > eps; //пока длина сторон большАя и значения функции меняются - продолжаем считать

}

fprintf(out\_f, "Min is:\t%.6lf\t%.6lf\t%.6lf\n%d", x0.x, x0.y, min\_f\_val, teaste);

fclose(out\_f);

return x0;

}

## Файл «easy\_random\_search.cpp»

#include "easy\_random\_search.h"

//Константы для параллелизма

const int ch\_n = 4; //количество потоков

void easy\_random\_search::init\_func(func\_2d s\_func){

min\_f = s\_func;

}

void easy\_random\_search::init\_rectangle(vect\_2d s\_x0, double s\_hx, double s\_hy){

x0 = s\_x0;

left\_bottom = x0 - vect\_2d(s\_hx/2, s\_hy/2);

right\_top = x0 + vect\_2d(s\_hx/2, s\_hy/2);

}

void easy\_random\_search::init\_tests(double eps, double P){

double lx = right\_top.x - left\_bottom.x;

double ly = right\_top.y - left\_bottom.y;

double P\_eps = eps \* eps / (lx \* ly);

tests\_N = log(1 - P) / log(1 - P\_eps);

printf("Total iters:\t%d\n", tests\_N);

}

vect\_2d easy\_random\_search::search(){

// vect\_2d xk; //генерируемая точка

// double fk; //значение функции в точек xk

min\_f\_val = min\_f(x0); //находим минимальное значение

long int iter = 0;

long int p\_iter = 0;

long int thread\_n = tests\_N / ch\_n + 1; //количество чисел на поток

vector<vect\_2d> points; //генерируемые точки

vector<double> vals; //значения функции

vector<vect\_2d> rez\_p;

vector<double> rez\_f;

rez\_p.resize(ch\_n);

rez\_f.resize(ch\_n);

for(int i = 0 ; i < ch\_n; i++){

rez\_p[i] = x0;

rez\_f[i] = min\_f\_val;

}

int num; //номер текущего потока

#pragma omp parallel default(shared)

{

#pragma omp for private(points, vals, num)

for(p\_iter = 0; p\_iter < ch\_n; p\_iter++){

num = omp\_get\_thread\_num(); //узнаём номер потока

const int N = 10000; //количесво чисел, которые мы генерируем за раз

points.resize(N);

vals.resize(N);

#pragma omp parallel default(shared)

{

#pragma omp for private(iter)

for(iter = 0; iter < thread\_n / N + 1; iter++){

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

points[i] = rand\_gen.generate(left\_bottom, right\_top);

vals[i] = min\_f(points[i]);

}

int min\_n = distance(vals.begin(), min\_element(vals.begin(), vals.end())); //номер минимального элемента

if(vals[min\_n] < rez\_f[num]){

rez\_p[num] = points[min\_n];

rez\_f[num] = vals[min\_n];

printf("Min change in %d\n", num);

printf("%d\t%.15lf\t%.15lf\t%.15lf\n", iter, rez\_p[num].x, rez\_p[num].y, rez\_f[num]);

}

}

}

}

}

int glob\_min\_n = distance(rez\_f.begin(), min\_element(rez\_f.begin(), rez\_f.end()));

x0 = rez\_p[glob\_min\_n];

min\_f\_val = rez\_f[glob\_min\_n];

/\*

for(iter = 0; iter < tests\_N; iter++){

xk = rand\_gen.generate(left\_bottom, right\_top);

fk = min\_f(xk);

fprintf(out\_f,"%d\t%.7lf\t%.7lf\t%.15lf\n", iter, xk.x, xk.y, fk);

if(fk < min\_f\_val){

x0 = xk;

min\_f\_val = fk;

printf("Min change\n");

printf("%d\t%.7lf\t%.7lf\t%.15lf\n", iter, xk.x, xk.y, fk);

}

}

\*/

FILE \*out\_f = fopen("easy\_iters.txt", "w");

fprintf(out\_f, "Min is:\t%.6lf\t%.6lf\t%.6lf", x0.x, x0.y, min\_f\_val);

fclose(out\_f);

return x0;

}

## Файл «launch.cpp»

#include <windows.h>

#include "easy\_random\_search.h"

#include "best\_sample\_search.h"

double func1(vect\_2d arg){

return (arg.x+5) \* (arg.x+5) + (arg.y+1)\*(arg.y+1);

}

double func2(vect\_2d arg){

double x = arg.x;

double y = arg.y;

return -exp(-abs(x)-y\*y\*y\*y)\*cos(2\*x\*x\*x)\*cos(10\*y\*y);

}

int main(){

/\*easy\_random\_search our\_rs;

our\_rs.init\_func(func2);

our\_rs.init\_rectangle(vect\_2d(0.1,0.1), 0.2, 0.2);

our\_rs.init\_tests(1E-4, 0.9999);

vect\_2d min = our\_rs.search();

printf("Minimun in:\t %.7lf\t%.7lf\nValue is:\t%.15lf\n", min.x, min.y, func2(min));

system("pause");\*/

best\_sample\_search our\_bss;

our\_bss.init\_func(func1);

our\_bss.init\_rectangle(vect\_2d(2,1), 1E-1, 1E-1);

our\_bss.init\_max\_iters(500000);

our\_bss.init\_tests(1E-4, 0.99);

our\_bss.init\_eps\_grow(1E-2);

vect\_2d min = our\_bss.search();

printf("Minimun in:\t %.7lf\t%.7lf\nValue is:\t%.15lf\n", min.x, min.y, func1(min));

system("pause");

return 0;

}