МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7**

**по дисциплине  
 «МЕТОДЫ ОПТИМИЗАЦИИ»**

Выполнил студент группы МО-32/2                                 В.И.Яценко

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Курс    3

Отчет принял                                                                                 Н. Ю. Кобыльская

Краснодар

2025 г.

**Задание:** написать и отладить программу численного решения задачи (Индивидуальная задача 2) нахождения минимума функции многих переменных, используя метод Дэвидона – Флетчера – Пауэлла.  
Метод Дэвидона – Флетчера – Пауэлла (ДФП) — это один из наиболее известных и широко применяемых квазиньютоновских методов для решения задач многомерной безусловной оптимизации (поиска локального минимума функции).

Ключевая идея метода ДФП заключается в итеративном построении (или обновлении) аппроксимации обратной матрицы Гессе (матрицы вторых производных) целевой функции. В отличие от классического метода Ньютона, который требует точного вычисления и обращения матрицы Гессе на каждом шаге (что может быть вычислительно дорого или невозможно), квазиньютоновские методы используют только информацию о градиенте функции.

Направления поиска в методе ДФП задаются в виде:

**Код программы:**

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

using namespace std;

double eps1 = 0.1;

double eps2 = 0.15;

int M = 10;

int k = 0;

vector<double>x = { 1.5,0.5 };

vector<double>xnext(2, 0);

vector<double>xprev(2, 0);

vector<vector<double>>A = { {1,0},{0,1} };

vector<double>deltaGrad(2, 0);

vector<double>nextGrad(2, 0);

vector<double>deltaX(2, 0);

vector<double> prevGrad(2, 0);

double func(double x1, double x2) {

return x1 \* x1 + 8 \* x2 \* x2 + x1 \* x2 + x1;

}

void gradient(double x1, double x2, double& g1, double& g2) {

g1 = 2 \* x1 + x2 + 1;

//cout << g1 << endl;

g2 = x1 + 16 \* x2;

//cout << g2 << endl;

}

double dist\_first(double x1, double x2) {

return sqrt(pow(x1, 2) + pow(x2, 2));

}

//// Функция для квадратичной аппроксимации

double t\_value(double a, double b, double c, double t) {

return a \* t \* t + b \* t + c;

}

//// Метод золотого сечения для квадратичной функции

double find\_t(double t\_a, double t\_b, double t\_c) {

double l = 0.00001;

int k = 0;

double a = -2.0;

double b = 2.0;

double golden\_ratio = (3 - sqrt(5)) / 2;

double y\_k = a + golden\_ratio \* (b - a);

double z\_k = a + b - y\_k;

double L = 1;

while (L > l) {

double f\_y = t\_value(t\_a, t\_b, t\_c, y\_k);

double f\_z = t\_value(t\_a, t\_b, t\_c, z\_k);

if (f\_y < f\_z) {

b = z\_k;

z\_k = y\_k;

y\_k = a + b - z\_k;

}

else {

a = y\_k;

y\_k = z\_k;

z\_k = a + b - y\_k;

}

L = abs(a - b);

k++;

}

return (a + b) / 2;

}

double dist\_second(double x1, double x2, double x1next, double x2next) {

double x1res = x1next - x1;

//cout << x1res << endl;

double x2res = x2next - x2;

//cout << x2res << endl;

double res = sqrt(pow(x1res, 2) + pow(x2res, 2));

return res;

}

double dist\_third(double fx1, double fx2) {

return abs(fx2 - fx1);

}

void printMatr(vector<vector<double>>matr) {

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

cout << matr[i][j] << " ";

}

cout << endl;

}

}

void printVec(vector<double>vec) {

for (int i = 0; i < 2; i++) {

cout << vec[i] << " ";

}

cout << endl;

}

int main(){

setlocale(LC\_ALL, "Ru");

bool flag = 0;

do {

vector<double>d(2, 0);

vector<double>g(2, 0);

cout << "итерация " << k << endl;

gradient(x[0], x[1], g[0], g[1]);

cout << "3^" << k << ": gradf(x^" << k << "): (" << g[0] << "," << g[1] << ")" << endl;

cout<< "4^" << k << ": ||gradf(x^" << k << ") || = " << dist\_first(g[0], g[1]) << endl;

if (dist\_first(g[0], g[1]) <= eps1) {

cout << "выполняется критерий окончания ||gradiendf(x^" << k << ")||<=eps1 ";

xnext[0] = x[0];

xnext[1] = x[1];

cout << "x\*=x^k=(" << xnext[0] << ";" << xnext[1] << ")" << endl;

flag = 1;

break;

}

if (k >= M) {

cout << "5^" << k << ": k=" << k << ">M=" << M << endl;

xnext[0] = x[0];

xnext[1] = x[1];

flag = 1;

break;

}

if (k>=1) {

/\*cout << "g0" << g[0] << " " << g[1] << endl;

cout << "g1" << prevGrad[0] << " " << prevGrad[1] << endl;\*/

deltaGrad[0] = g[0] - prevGrad[0];

deltaGrad[1] = g[1] - prevGrad[1];

cout << "deltaGrad"<<"=(" << deltaGrad[0] << "," << deltaGrad[1]<<")" << endl;

deltaX[0] = x[0] - xprev[0];

deltaX[1] = x[1] - xprev[1];

cout << "deltaX" << "=("<< deltaX[0] << "," << deltaX[1] <<")"<< endl;

double denom1 = deltaX[0] \* deltaGrad[0] + deltaX[1] \* deltaGrad[1];

//cout << "denom1: " << denom1 << endl;

vector<vector<double>>numer1(2, vector<double>(2, 0));

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

numer1[i][j] = deltaX[i] \* deltaX[j];

numer1[i][j] /= denom1;

}

}

//printMatr(numer1);

vector<double>deltaGA(2, 0);

for (int j = 0; j < 2; j++) {

for (int i = 0; i < 2; i++) {

deltaGA[j] += deltaGrad[i] \* A[i][j];

}

}

//printVec(deltaGA);

double denom2 = deltaGA[0]\*deltaGrad[0]+ deltaGA[1] \* deltaGrad[1];

cout << "denom2: " << denom2 << endl;

vector<double>AdeltaGrad(2, 0);

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

AdeltaGrad[i] += A[i][j] \* deltaGrad[j];

}

}

vector<vector<double>>AdeltaGrad2(2, vector<double>(2, 0));

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

AdeltaGrad2[i][j] = AdeltaGrad[i] \* deltaGrad[j];

}

}

//printMatr(AdeltaGrad2);

vector<vector<double>>numer2(2, vector<double>(2, 0));

for (int i = 0; i < 2; i++) { // по строкам первой матрицы

for (int j = 0; j < 2; j++) { // по столбцам второй матрицы

for (int k = 0; k < 2; k++) { // суммирование произведений

numer2[i][j] += AdeltaGrad2[i][k] \* A[k][j];

}

}

}

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

numer2[i][j] /= denom2;

}

}

//printMatr(numer2);

vector<vector<double>>Ac(2, vector<double>(2, 0));

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

Ac[i][j] = numer1[i][j] - numer2[i][j];

}

}

cout << "матрица Ac: "<<endl;

printMatr(Ac);

vector<vector<double>>Anext(2, vector<double>(2, 0));

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

A[i][j] = A[i][j] + Ac[i][j];

}

}

cout << "матрица A^k+1: " << endl;

printMatr(A);

}

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

d[i] += -A[i][j] \* g[j];

}

}

cout <<"d = ("<< d[0] << "," << d[1]<<")" << endl;

double t\_a = 1 \* d[0] \* d[0] + 8 \* d[1] \* d[1] + d[0] \* d[1];

double t\_b = 2 \* x[0] \* d[0] + 16 \* x[1] \* d[1] + (x[0] \* d[1] + d[0] \* x[1]) + d[0];

double t\_c = func(x[0], x[1]);

double t = find\_t(t\_a, t\_b, t\_c);

cout << "t\* = " << t << endl;

xnext[0] = x[0] + t \* d[0];

xnext[1] = x[1] + t \* d[1];

cout << "x^" << k + 1 << " = (" << xnext[0] << ", " << xnext[1] << ")" << endl;

cout << "f(x^" << k + 1 << ") = " << func(xnext[0], xnext[1]) << endl;

prevGrad = g;

xprev = x;

cout << "||x^" << k + 1 << " - x^" << k << "|| = " << dist\_second(x[0], x[1], xnext[0], xnext[1]) << endl;

cout << "|f(x^" << k + 1 << ") - f(x^" << k << ")| = " << dist\_third(func(x[0],x[1]), func(xnext[0],xnext[1])) << endl;

if (dist\_second(x[0], x[1], xnext[0], xnext[1]) < eps2 && dist\_third(func(x[0], x[1]), func(xnext[0], xnext[1]) < eps2)) {

cout << "условия окончания выполнены" << endl;

flag=1;

break;

}

x = xnext;

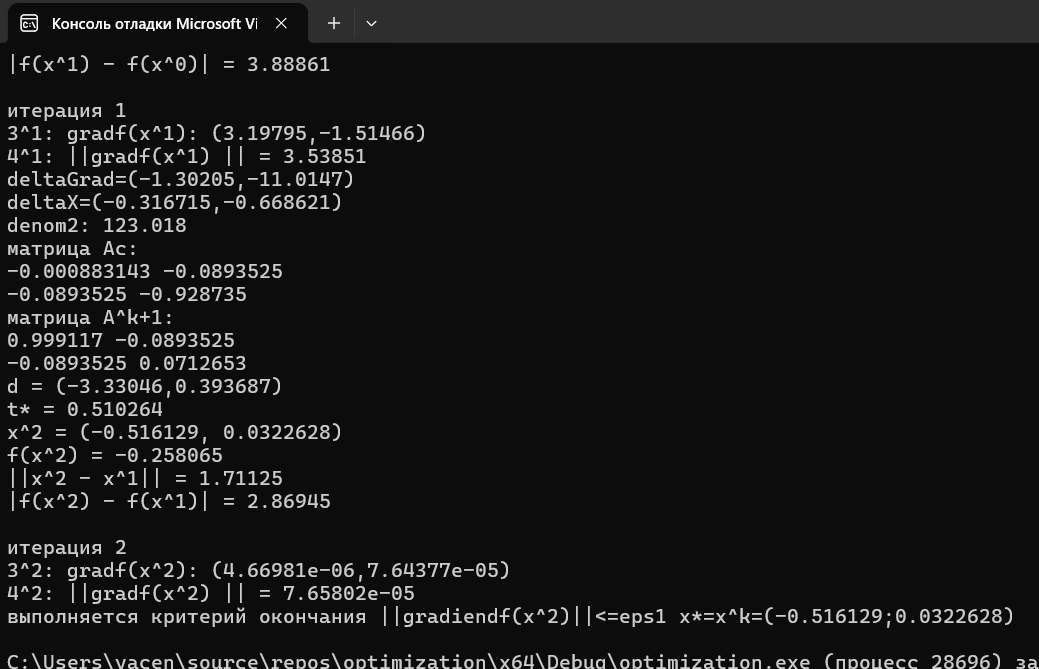
k++;

cout << endl;

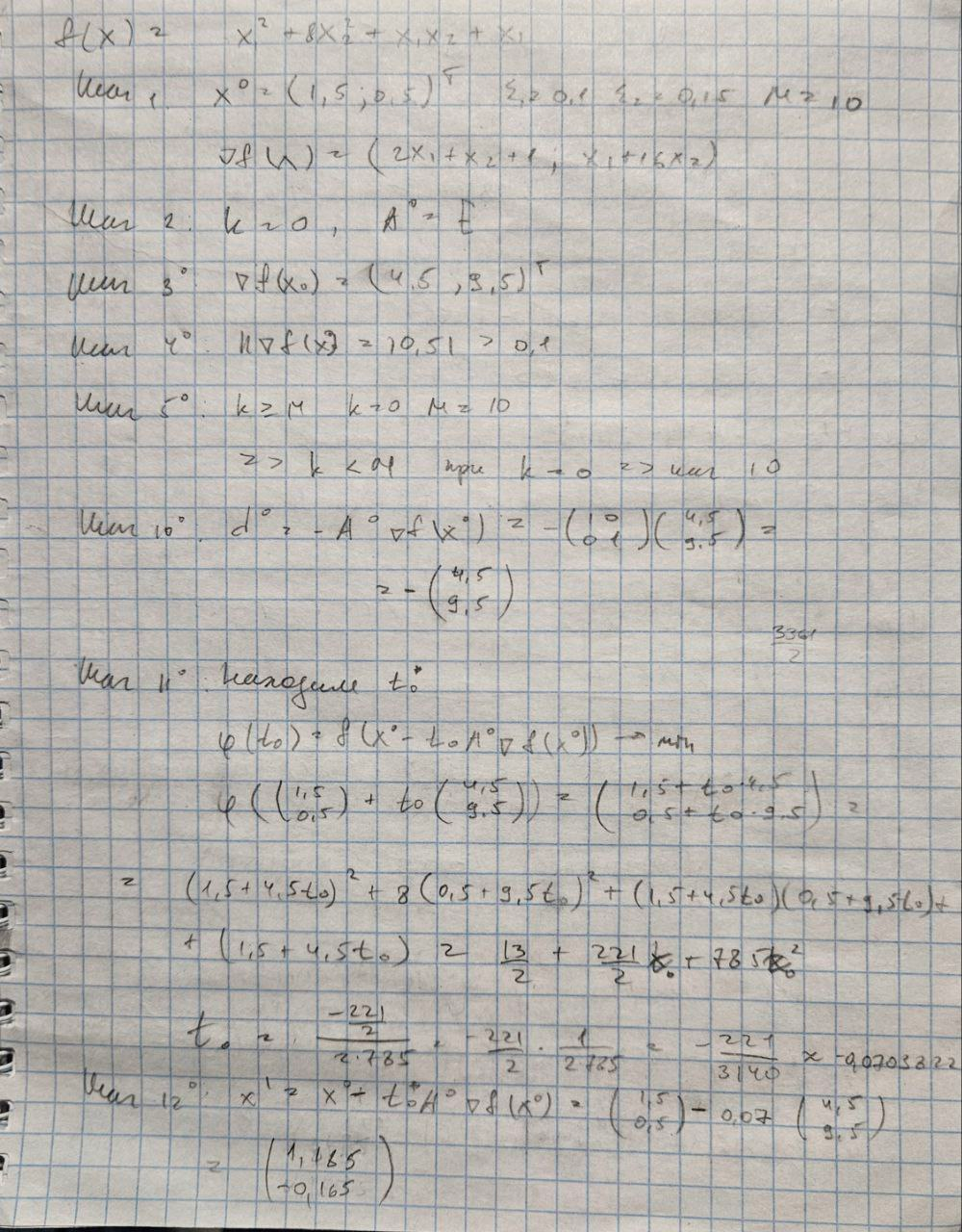
}while (k < 10);

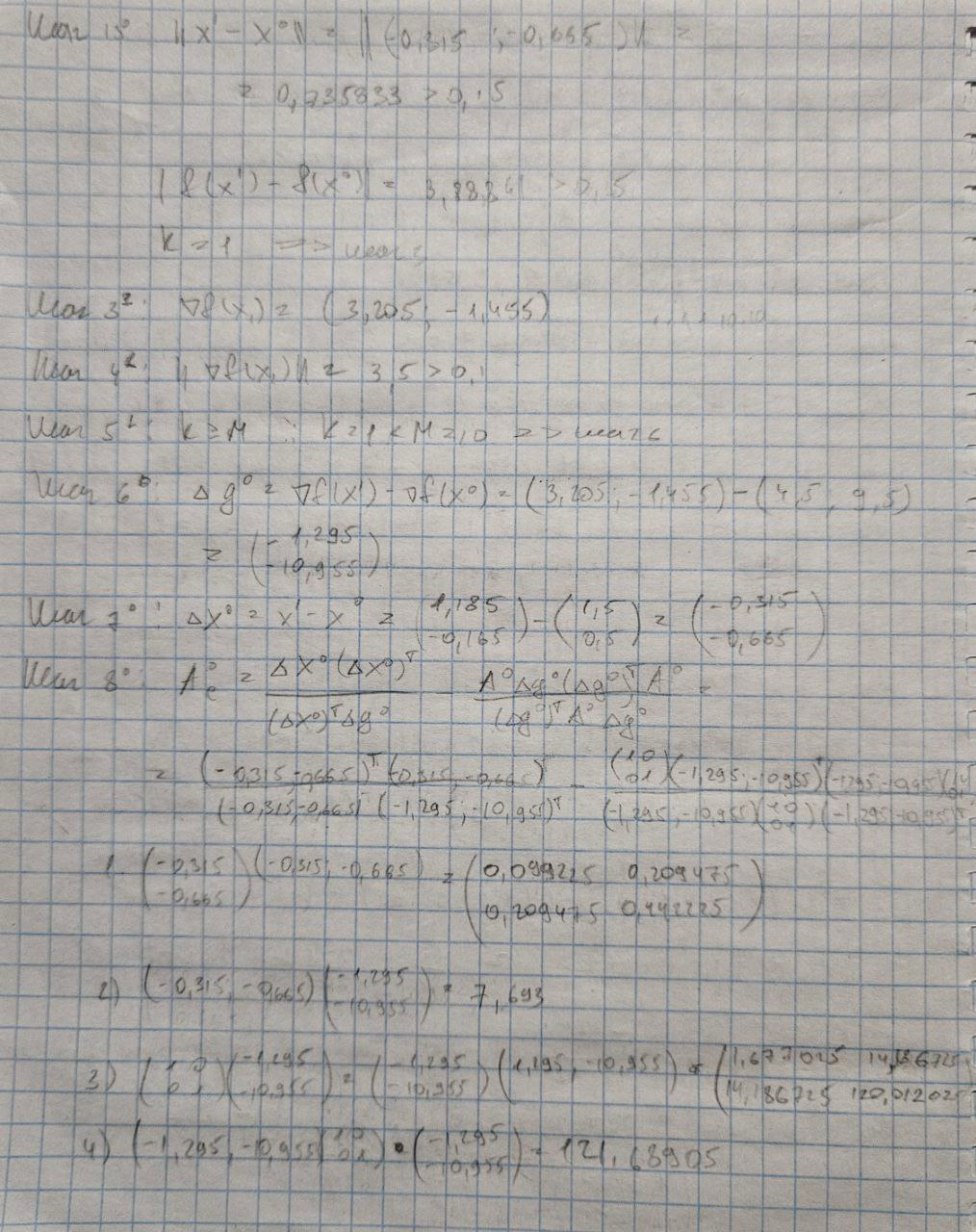
}

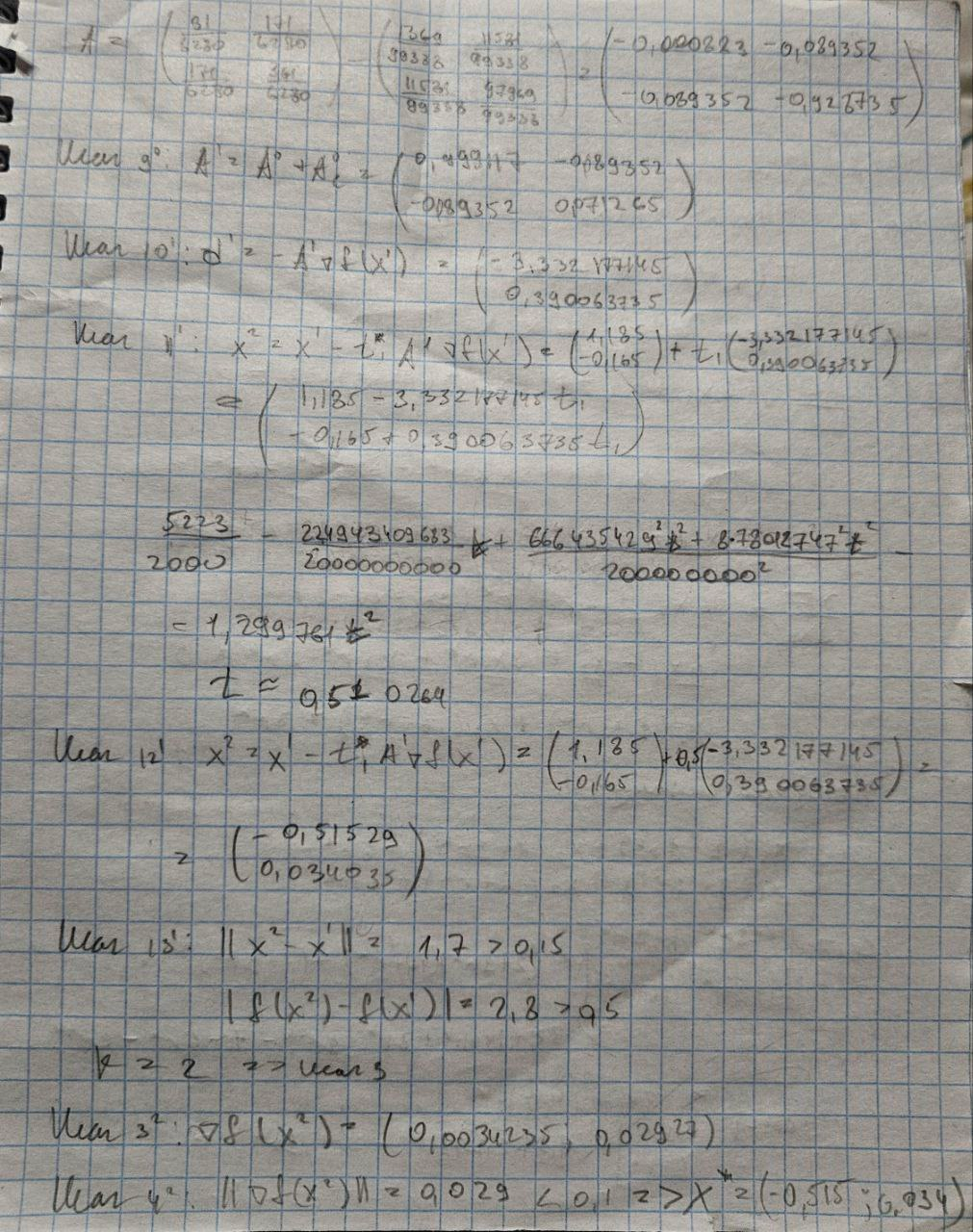
Результат в консоли:



Решение в тетради:







|  |  |
| --- | --- |
| Аналитический метод | Метод ДФП |
| (-0,5161290323 ; 0,0322580645) | (-0.516129;0.0322628) |