|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
|  | | |
| Практическое задание № 2 | | |
| по дисциплине «Методы оптимизации» | | |
| МЕТОДЫ СПУСКА (0-го, 1-го и 2-го ПОРЯДКА И ПЕРЕМЕННОЙ МЕТРИКИ) | | |
|  | | |
|  | Вариант 8 | Барсукова наталья |
| Группа ПМ-91 | Грибова Александра |
|  | Затолоцкая юлия |
|  |
|  |
|  |
| Преподаватели | Филипова елена владимировна |
|  |  |
| Новосибирск, 2022 | | |

# Цель работы

Ознакомиться с методами поиска минимума функции n переменных в оптимизационных задачах без ограничений.

# Постановка задачи

* Реализовать два метода поиска экстремума функции: метод Бройдена и метод деформируемого многогранника. Включить в реализуемый алгоритм собственную процедуру, реализующую одномерный поиск по направлению. Методы поиска для самостоятельной реализации выбираются студентом в зависимости от уровня сложности. Выбранные методы должны иметь разный порядок.
* С использованием разработанного программного обеспечения исследовать алгоритмы на квадратичной функции , функции Розенброка и на заданной в соответствии с вариантом тестовой функции, осуществляя спуск из различных исходных точек (не менее двух). Исследовать сходимость алгоритма, фиксируя точность определения минимума/максимума, количество итераций метода и количество вычислений функции в зависимости от задаваемой точности поиска. Результатом выполнения данного пункта должны быть выводы об объеме вычислений в зависимости от задаваемой точности и начального приближения

Тестовые функции:

# Результаты исследований

**Метод деформируемого многогранника**

Функция:

Точный минимум достигается в точке

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Точность | ( | Кол-во вычислений | Число итераций | Найденная точка | Значение функции |
| 10-3 | (0,0) | 161 | 76 | (1.00031 ; 1.0006) | 1.10478e-07 |
|  | (-1,1) | 242 | 116 | (0.999752 ; 0.999469) | 1.8839e-07 |
|  | (1.5, -1.5) | 190 | 67 | (0.999635 ; 0.999157) | 1.44088e-06 |
| 10-6 | (0,0) | 230 | 107 | (0.999999 ; 0.999999) | 3.87831e-13 |
|  | (-1,1) | 320 | 151 | (1.00000002 ; 1.0000000948) | 3.87366e-15 |
|  | (1.5, -1.5) | 266 | 121 | (1.0000002614 ; 1.0000004008) | 1.5548105663e-12 |

Функция:

Точный минимум достигается в точке

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Точность | ( | Кол-во вычислений | Число итераций | Найденная точка | Значение функции |
| 10-3 | (0,0) | 112 | 50 | (0.99975324991; 0.99974508729) | 6.75484474e-08 |
|  | (-1,1) | 211 | 98 | (0.99772642581 ; 0.99778824484) | 5.5512988625e-06 |
|  | (1.5, -1.5) | 135 | 62 | (0.99881489783 ; 0.99896606867) | 3.6897292847e-06 |
| 10-6 | (0,0) | 168 | 75 | (0.99999853034 ; 0.99999868364) | 4.5100097908e-12 |
|  | (-1,1) | 360 | 167 | (0.99999978017 ; 0.99999973648) | 2.3928282947e-13 |
|  | (1.5, -1.5) | 232 | 107 | (1.0000009702 ; 1.0000007726) | 4.8478934885e-12 |

Функция:

Точный максимум достигается в точке

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Точность | ( | Кол-во вычислений | Число итераций | Найденная точка | Значение функции |
| 10-3 | (0,0) | 84 | 37 | (1.9996542931 ; 2.7564129829) | 4.5125569661 |
|  | (-1,1) | 149 | 70 | (2.0001434425 ; 2.7567301587) | 4.5125569945 |
|  | (1.5, -1.5) | 134 | 61 | (2.756516221 ; -4.5125552372) | 4.5125552372 |
| 10-6 | (0,0) | 173 | 78 | (1.9999992145 2.7563666809) | 4.512557268 |
|  | (-1,1) | 237 | 111 | (1.9999998471 ; 2.7563643097) | 4.512557268 |
|  | (1.5, -1.5) | 183 | 83 | (1.9999994491 ; 2.7563637139) | 4.512557268 |

**Метод Бройдена**

Функция:

Точный минимум достигается в точке

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Точность | ( | Кол-во вычислений | Число итераций | Найденная точка | Значение функции |
| 10-3 | (0,0) | 1361 | 14 | (0.99996545493 0.9999323438) | 1.3986400165e-09 |
|  | (-1,1) | 1701 | 17 | (0.99999702521 0.99999424421) | 1.2604261698e-11 |
|  | (1.5, -1.5) | 3873 | 37 | (0.99999957972 0.99999921588) | 4.9518989678e-13 |
| 10-6 | (0,0) | 1545 | 16 | (0.99999999333 0.99999998662) | 4.4632343889e-17 |
|  | (-1,1) | 1797 | 18 | ( 0.99999999817 0.99999999607) | 1.1145955785e-17 |
|  | (1.5, -1.5) | 3957 | 38 | ( 0.99999999989 0.99999999977) | 1.874233637e-20 |

Функция:

Точный минимум достигается в точке

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Точность | ( | Кол-во вычислений | Число итераций | Найденная точка | Значение функции |
| 10-3 | (0,0) | 172 | 2 | (0.99999844248 0.99999991069) | 2.1798907375e-10 |
|  | (-1,1) | 231 | 3 | (0.99999986139 0.99999987347) | 3.382299863e-14 |
|  | (1.5, -1.5) | 228 | 3 | (1 1) | 1.1831315593e-23 |
| 10-6 | (0,0) | 238 | 3 | (0.99999999983 0.99999999999) | 3.382299863e-14 |
|  | (-1,1) | 231 | 3 | (0.99999986139 0.99999987347) | 1.1145955785e-17 |
|  | (1.5, -1.5) | 228 | 3 | (1.0000000702 ; 1.0000000726) | 1.1831315593e-23 |

Функция:

Точный максимум достигается в точке

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Точность | ( | Кол-во вычислений | Число итераций | Найденная точка | Значение функции |
| 10-3 | (0,0) | 482 | 5 | (1.9999999519 2.7563607428) | 4.512557268 |
|  | (-1,1) | 811 | 8 | (1.9999964261 2.7563778991) | 4.5125569945 |
|  | (1.5, -1.5) | 694 | 7 | (1.9999493997 2.7563914577) | 4.5125552372 |
| 10-6 | (0,0) | 509 | 8 | (1.9999999992 2.7563653334) | 4.512557268 |
|  | (-1,1) | 829 | 10 | (2.0000000021 2.7563653893) | 4.512557268 |
|  | (1.5, -1.5) | 710 | 9 | (1.99999996 2.7563654325) | 4.512557268 |

**Таблицы с данными**

Функция:

**Метод: деформируемого многогранника**

Начальная точка: (0,0)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1 | (0, 0) | 1 | (0, 0, 0)  (0.70710678119, 0.24148145657, 65.127869314)  (0.24148145657, 0.70710678119, 64.734059845) |
| 2 | (0, 0) | 1 | (0, 0, 0)  (0.35355339059, 0.12074072829, 15.928413938)  (0.12074072829, 0.35355339059, 16.062774233) |
| 7 | (0.02964338243, 0.02964338243) | 0.94159196526 | (0, 0, 0)  (0.0037731477589, 0.011048543456, 0.023244722491)  (0.0072753956971, 0.0072753956971, 0.049556996487) |
| 8 | (0.048102771493, 0.040827375796) | 0.91140147189 | (0.018459389064, 0.011183993366, 0.030190493369)  (0.0072753956971, 0.0072753956971, 0.0071060276821)  (0.011048543456, 0.0037731477589, 0.0055239450108) |
| 15 | (0.25196875065, 0.25196875065) | 0.55955075 | (0, 0 , 0)  (0, 0, 0)  (0.12948662327, 0.10766043617, 0.19272331915) |
| 24 | (0.96340992897, 0.96340992897) | 0.0013388332977 | (0, 0, 0)  (0.057467915936, 0.061105613784, 0.0096720275255)  (0.061105613784, 0.057467915936, 0.010699824302) |
| 25 | (0.96340992897, 0.96340992897) | 0.0013388332977 | (0, 0, 0)  (0.028733957968, 0.030552806892, 0.0013666293185)  (0.030552806892, 0.028733957968, 0.0015570267013) |

**Метод Бройдена**

Функция:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | angle | H |
| 0 | ( 0.000000, 0.000000) | 1.000000 |  |  |  |  |  |
| 1 | ( 0.010486, 0.000000) | 0.990134 | (-2.000000, 0.000000) | 0.005243 | 0.010486 | 1.000000 | [ 0.498759 0.498747] |
|  |  |  |  |  | 0.000000 |  | [ 0.498747 0.503734] |
|  |  |  |  |  | 0.009866 |  |  |
| 2 | ( 0.999998, 1.000000) | 0.000000 | (-0.987033, -0.997494) | 1.002512 | 0.989513 | 0.999986 | [ 0.500000 0.500000] |
|  |  |  |  |  | 1.000000 |  | [ 0.500000 0.505000] |
|  |  |  |  |  | 0.990134 |  |  |

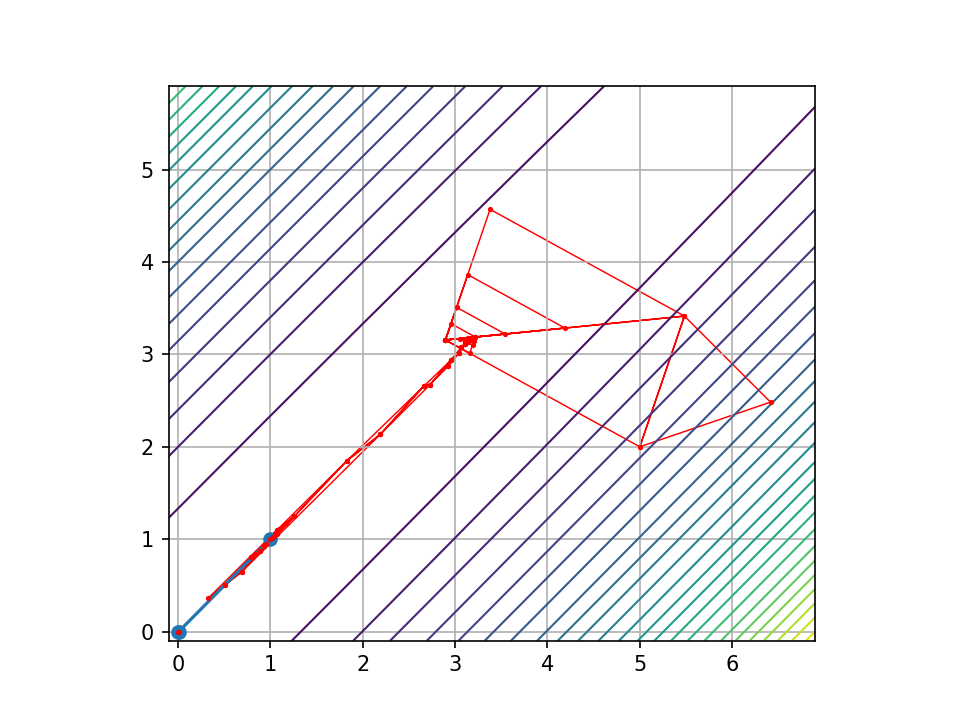
Функция:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | angle | H |
| 0 | ( 0.000000, 0.000000) | 1.000000 |  |  |  |  |  |
| 1 | ( 0.167772, 0.000000) | 0.771831 | (-2.000000, 0.000000) | 0.083886 | 0.167772 | 1.000000 | [ 0.883361 0.319257] |
|  |  |  |  |  | 0.000000 |  | [ 0.319257 0.126154] |
|  |  |  |  |  | 0.228169 |  |  |
| 2 | ( 0.296208, 0.051289) | 0.628187 | (-1.598949, -0.638513) | 0.080325 | 0.128435 | 0.978347 | [ 0.070144 0.036143] |
|  |  |  |  |  | 0.051289 |  | [ 0.036143 0.027591] |
|  |  |  |  |  | 0.143644 |  |  |
| 3 | ( 0.340756, 0.123364) | 0.439858 | (-0.059286, -0.095920) | 0.751414 | 0.044548 | 0.783923 | [ 0.042430 0.030428] |
|  |  |  |  |  | 0.072075 |  | [ 0.030428 0.026412] |
|  |  |  |  |  | 0.188329 |  |  |
| 4 | ( 0.457901, 0.192864) | 0.322125 | (-0.053752, -0.031890) | 2.179364 | 0.117145 | 0.990654 | [ 0.244908 0.194576] |
|  |  |  |  |  | 0.069501 |  | [ 0.194576 0.159487] |
|  |  |  |  |  | 0.117733 |  |  |
| 5 | ( 0.538762, 0.265146) | 0.275834 | (-0.165649, -0.148073) | 0.488146 | 0.080861 | 0.963210 | [ 0.086289 0.080947] |
|  |  |  |  |  | 0.072281 |  | [ 0.080947 0.078087] |
|  |  |  |  |  | 0.046291 |  |  |
| 6 | ( 0.959597, 0.897222) | 0.057349 | (-0.019161, -0.028779) | 21.962982 | 0.420836 | 0.973306 | [ 0.086994 0.100182] |
|  |  |  |  |  | 0.632077 |  | [ 0.100182 0.602296] |
|  |  |  |  |  | 0.218485 |  |  |
| 7 | (-7.464400, 54.023815) | 358.423337 | ( 0.308214, -1.943773) | 27.331691 | 8.423997 | 0.999801 | [ 0.003988 -0.035466] |
|  |  |  |  |  | 53.126592 |  | [-0.035466 0.380621] |
|  |  |  |  |  | 358.365989 |  |  |
| 8 | ( 1.060565, 1.124872) | 0.003669 | (-8.220757, 51.011273) | 1.037005 | 8.524965 | -0.609185 | [ 0.004091 -0.036102] |
|  |  |  |  |  | 52.898943 |  | [-0.036102 0.384573] |
|  |  |  |  |  | 358.419669 |  |  |
| 9 | ( 1.060518, 1.125580) | 0.003740 | (-0.000154, 0.002320) | -0.305172 | 0.000047 | -0.680681 | [ 0.000500 0.000768] |
|  |  |  |  |  | 0.000708 |  | [ 0.000768 0.006012] |
|  |  |  |  |  | 0.000071 |  |  |
| 10 | ( 1.060509, 1.124672) | 0.003661 | ( 0.000009, 0.000865) | 1.049518 | 0.000010 | -0.734786 | [ 0.478320 1.014537] |
|  |  |  |  |  | 0.000907 |  | [ 1.014537 2.156874] |
|  |  |  |  |  | 0.000079 |  |  |
| 11 | ( 1.019180, 1.037018) | 0.000661 | ( 0.057885, 0.122770) | 0.713974 | 0.041328 | -0.944034 | [ 0.405201 0.848805] |
|  |  |  |  |  | 0.087655 |  | [ 0.848805 1.781225] |
|  |  |  |  |  | 0.003001 |  |  |
| 12 | ( 0.998336, 0.996283) | 0.000018 | ( 0.007722, 0.015091) | 2.699352 | 0.020844 | -0.951272 | [ 0.497427 1.019699] |
|  |  |  |  |  | 0.040735 |  | [ 1.019699 2.097889] |
|  |  |  |  |  | 0.000642 |  |  |
| 13 | ( 0.998392, 0.996406) | 0.000017 | (-0.003734, -0.008249) | 0.014997 | 0.000056 | 0.935447 | [ 0.379870 0.762776] |
|  |  |  |  |  | 0.000124 |  | [ 0.762776 1.536385] |
|  |  |  |  |  | 0.000001 |  |  |
| 14 | ( 0.999965, 0.999932) | 0.000000 | (-0.001546, -0.003464) | 1.017978 | 0.001573 | 0.933871 | [ 0.384550 0.772625] |
|  |  |  |  |  | 0.003526 |  | [ 0.772625 1.557111] |
|  |  |  |  |  | 0.000017 |  |  |

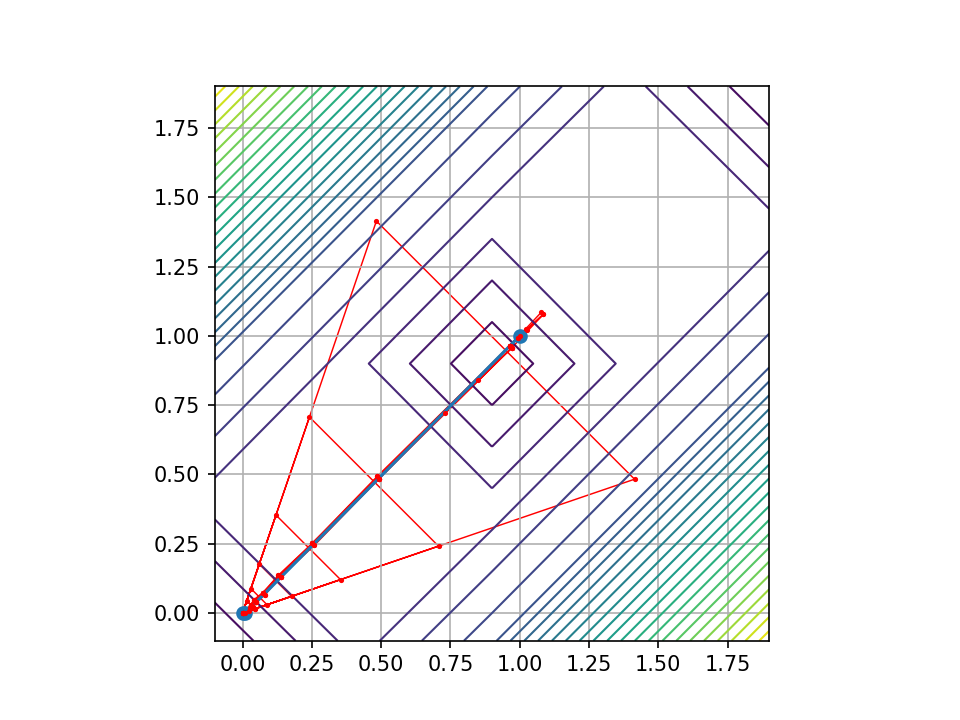
# Визуализация

Изображение показывает траекторию сходимости,

Функция:



Функция:



Красные линии – метод деформируемого многогранника.

Синии линии – метод Бройдена.

Вывод

Об объеме вычислений в зависимости от точности:

Для метода Бройдена на всех трех функциях количество итераций практически не зависит от точности. Увеличивается количество вычислений функции, что связано с одномерным поиском.

Для метода деформируемого многогранника количество итераций растет с увеличением точности на всех трех функция. Также расчет количество вычислений функции.

Об объеме вычислений в зависимости от начального приближения:

Чем ближе к точке минимума, тем быстрее сходимость.

В случае метода Бройдена – метод сходится за меньшее число итераций, но существенно проигрывает по числу вычислений функции методу многогранников.

Код

Mo\_2.cpp

#include"Method\_0.cpp"

#include"Method\_variable\_metric.cpp"

#include"lib.cpp"

int main()

{

point point;

std::cout << "Enter x0: ";

std::cin >> point.x ;

std::cout << "Enter y0: ";

std::cin >> point.y;

std::vector<test> Tests = {

{f1, "table1\_1e-3.txt",1e-3},

{f2, "table2\_1e-3.txt",1e-3},

{f3, "table3\_1e-3.txt",1e-3},

{f1, "table1\_1e-6.txt",1e-6},

{f2, "table2\_1e-6.txt",1e-6},

{f3, "table3\_1e-6.txt",1e-6},

};

for (test t : Tests) {

std::cout << "Broyden's\n";

method\_variable\_metric create\_metod\_variable(point, t.eps, t.func, t.fname);

std::cout << "Simplex\n";

method\_0 create\_method0 (point, t.eps, t.func, t.fname);

}

std::cout << std::endl << count\_f << std::endl;

}

Method\_0.cpp

#pragma once

#include <functional>

#include <iostream>

#include <iomanip>

#include <cmath>

#include <vector>

#include <string>

double eps\_f = 1e-3;

int count\_f;

struct point

{

double x;//координата х

double y;//координата y

double f;//значение функции в точке

};

typedef std::function<double(point)> OneDimensionFunction;

struct test

{

OneDimensionFunction func;

std::string fname;

double eps;

};

double f1(point p) {

double x = p.x;

double y = p.y;

return (100 \* (y - x) \* (y - x) + (1 - x) \* (1 - x));

}

double f2(point p) {

double x = p.x;

double y = p.y;

return 100 \* (y - x \* x) \* (y - x \* x) + (1 - x) \* (1 - x);

}

double f3(point p) {

double x = p.x;

double y = p.y;

return -1 \* (3 / (1 + pow(x - 2, 2) + pow((y - 2) / 2, 2)) + 2 / (1 + pow((x - 2) / 3, 2) + pow(y - 3, 2)));

}

point operator-(point x, point y) {

x.x = x.x - y.x;

x.y = x.y - y.y;

return x;

}

point operator+(point x, point y) {

x.x = x.x + y.x;

x.y = x.y + y.y;

return x;

}

double operator\*(point x, point y) {

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

}

point operator\*(double a, point y) {

y.x \*= a;

y.y \*= a;

return y;

}

point operator\*(double (&H)[2][2], point y) {

point m;

m.x = H[0][0] \* y.x + H[0][1] \* y.y;

m.y = H[1][0] \* y.x + H[1][1] \* y.y;

return m;

}

double norm (point x) {

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

}

lib.cpp

#pragma once

#include "lib.cpp"

class method\_0

{

int n = 2;

OneDimensionFunction function;

void sort(point\* point\_array)

{

point tmp;

for (int i = 0; i < 3; i++) {

for (int j = 2; j >= (i + 1); j--) {

if (point\_array[j].f < point\_array[j - 1].f) {

tmp = point\_array[j];

point\_array[j] = point\_array[j - 1];

point\_array[j - 1] = tmp;

}

}

}

}

double d1(double t)

{

return (t / (n \* sqrt(2)) \* (sqrt(n + 1) + n - 1));

}

double d2(double t)

{

return (t / (n \* sqrt(2) \* (sqrt(n + 1) - 1)));

}

point create\_point(point point0, point vector)//ńîçäŕíčĺ äîď 2 ňî÷ĺę

{

point0.x = point0.x + 0.05 \* vector.x;

point0.y = point0.y + 0.05 \* vector.y;

return point0;

}

point elongation(point new\_point, point mid, double gamma)//đŕńň˙ćĺíčĺ

{

new\_point.x = mid.x + gamma \* (new\_point.x - mid.x);

new\_point.y = mid.y + gamma \* (new\_point.y - mid.y);

new\_point.f = function(new\_point);

return new\_point;

}

void reduction(point\* point\_array)

{

for (int i = 1; i < 3; i++)

{

point\_array[i].x = point\_array[0].x + 0.5 \* (point\_array[i].x - point\_array[0].x);

point\_array[i].y = point\_array[0].y + 0.5 \* (point\_array[i].y - point\_array[0].y);

}

point\_array[1].f = function(point\_array[1]);

point\_array[2].f = function(point\_array[2]);

}

point compression(point worst, point mid, double betta)

{

worst.x = mid.x + betta \* (worst.x - mid.x);

worst.y = mid.y + betta \* (worst.y - mid.y);

worst.f = function(worst);

count\_f++;

return worst;

}

void reflection(double alpha, double betta, double gamma, point mid, point\* point\_array) //îňđŕćĺíčĺ őóäřĺé ňî÷ęč îňíîńčňĺëüíî mid

{

point new\_point, point;//íŕäî ëč point âîîáůĺ?

new\_point.x = mid.x + alpha \* (mid.x - point\_array[2].x);

new\_point.y = mid.y + alpha \* (mid.y - point\_array[2].y);

new\_point.f = function(new\_point);

count\_f++;

while (true)

{

if (new\_point.f >= point\_array[2].f)

{

reduction(point\_array);//đĺäóęöč˙

break;

}

if (new\_point.f <= point\_array[0].f)//ĺńëč ěĺíüřĺ ěčíčěŕëüíîăî íŕ k-ýňŕďĺ

{

point = elongation(new\_point, mid, gamma);//đŕńň˙ćĺíčĺ

if (point.f < point\_array[0].f)

{

point\_array[2] = point\_array[1];

point\_array[1] = point\_array[0];

point\_array[0] = point;

}

else

{

point\_array[2] = point\_array[1];

point\_array[1] = point\_array[0];

point\_array[0] = new\_point;

}

break;

}

if (new\_point.f > point\_array[1].f && new\_point.f <= point\_array[2].f)//ěĺćäó äâóě˙ őóäřčěč ňî÷ęŕěč

{

point\_array[2] = compression(point\_array[2], mid, betta);//ńćŕňčĺ íŕäî ëč âîçâđŕůňü âîîáůĺ??

break;

}

if (new\_point.f > point\_array[0].f && new\_point.f <= point\_array[1].f)

{

point\_array[2] = new\_point;

break;

}

}

}

public:

method\_0(point point1, double eps, OneDimensionFunction function, std::string fname)

{

this->function = function;

point point\_array[3], mid, point\_prew\_array[3];

int iter;

double flag1;

double alpha = 1;

double betta = 0.5;

double gamma = 2;

bool flag\_i = true, flag\_f = true;

int k = 0;

double t = 0.1;

point\_array[0] = point1;

point\_array[1] = { point\_array[0].x + d1(t),point\_array[0].y + d2(t) };

point\_array[2] = { point\_array[0].x + d2(t),point\_array[0].y + d1(t) };

point\_array[0].f = function(point\_array[0]);

point\_array[1].f = function(point\_array[1]);

point\_array[2].f = function(point\_array[2]);

count\_f = 3;

for (iter = 1; flag\_f; iter++)

{

sort(point\_array);

for (int i = 0; i < 3; i++) point\_prew\_array[i] = point\_array[i];

mid = { (point\_array[0].x + point\_array[1].x) / 2, (point\_array[0].y + point\_array[1].y) / 2 };

mid.f = function(mid);

count\_f++;

reflection(alpha, betta, gamma, mid, point\_array);

flag1 = sqrt(pow(point\_array[0].f, 2) - pow(mid.f, 2)) / 3;

if (flag1 < eps\_f) flag\_f = false;

std::cout << iter << ": Direction: (" << mid.x - point\_array[2].x << " , " << mid.x - point\_array[2].x << ") " << std::endl;

for (int i = 0; i < 3; i++) std::cout << std::setprecision(11) << " (" << abs(point\_array[i].x - point\_prew\_array[i].x) << ", " << abs(point\_array[i].y - point\_prew\_array[i].y) << " , " << abs(point\_array[i].f - point\_prew\_array[i].f) << ") " << std::endl;

std::cout << std::setprecision(11) << "(" << point\_array[0].x << ", " << point\_array[0].y << ") " << point\_array[0].f << std::endl;

}

std::cout << std::setprecision(11) << point\_array[0].x << " " << point\_array[0].y << " " << point\_array[0].f << std::endl;

std::cout << "Iter: " << iter;

}

};

Method\_variable\_metric.cpp

#pragma once

#include "lib.cpp"

class method\_variable\_metric

{

OneDimensionFunction function;

double delta = 1E-8;

double diff\_x(point point1) //маcсив аргументов/по какой переменной/в какую функцию подставляем/шаг

{

double h = 10E-7;

double d0 = point1.x;

point1.x += h;

double f\_right = function(point1);

point1.x = d0 - h;

double f\_left = function(point1);

count\_f += 2;

point1.x = d0;

return (f\_right - f\_left) / (2 \* h);

}

double diff\_y(point point1) //маcсив аргументов/по какой переменной/в какую функцию подставляем/шаг

{

double h = 10E-7;

double d0 = point1.y;

point1.y += h;

double f\_right = function(point1);

point1.y = d0 - h;

double f\_left = function(point1);

count\_f += 2;

point1.y = d0;

return (f\_right - f\_left) / (2 \* h);

}

point find\_grad(point p)

{

point grad;

grad.x = diff\_x(p);

grad.y = diff\_y(p);

grad.f = function(grad);

return grad;

}

void IntervalSearch(double lymbda, double\* a, double\* b, point point1, point p0)

{

double h = 0;

double lymbda\_previous = lymbda;

//шаг 1. определяем направление поиска.

point x1 = point1 + lymbda \* p0;

point x2 = point1 + (lymbda + delta) \* p0;

double f1 = function(x1);

if (f1 > function(x2))

{

lymbda += delta;

h = delta;

}

else

{

lymbda -= delta;

h = -delta;

}

h \*= 2;

lymbda\_previous = lymbda;

lymbda = lymbda + h;

double f2 = function(point1 + lymbda \* p0);

count\_f += 3;

while (f1 > f2)

{

h \*= 2;

lymbda\_previous = lymbda;

lymbda = lymbda + h;

f1 = f2;

f2 = function(point1 + lymbda \* p0);

count\_f++;

}

\*a = lymbda\_previous;

\*b = lymbda;

}

double GoldenSectionMethod(double a, double b, point point1, point pk)

{

int i;

const double A\_COEFF = ((3 - sqrt(5.0)) / 2);

const double B\_COEFF = ((sqrt(5.0) - 1) / 2);

double x1 = a + A\_COEFF \* (b - a);

double x2 = a + B\_COEFF \* (b - a);

point p1 = point1 + x1 \* pk;

point p2 = point1 + x2 \* pk;

double f1 = function(p1);

double f2 = function(p2);

count\_f += 2;

for (i = 0; abs(b - a) > 1e-8; i++)

{

if (f1 > f2)

{

a = x1;

x1 = x2;

f1 = f2;

x2 = a + B\_COEFF \* (b - a);

p2 = point1 + x2 \* pk;

f2 = function(p2);

}

else

{

b = x2;

x2 = x1;

f2 = f1;

x1 = a + A\_COEFF \* (b - a);

p1 = point1 + x1 \* pk;

f1 = function(p1);

}

count\_f++;

if (i > 100) break;

}

return (a + b) / 2.0;

}

double search\_lambd(double lambd, point point1, point pk)

{

double a = -1, b = 1;

IntervalSearch(lambd, &a, &b, point1, pk);

lambd = GoldenSectionMethod(a, b, point1, pk);

return lambd;

}

void Gessian(double (&H)[2][2], point dxk, point gk)

{

point z = dxk - H \* gk;

double div = z \* gk;

H[0][0] += z.x \* z.x / div;

H[0][1] += z.x \* z.y / div;

H[1][0] += z.y \* z.x / div;

H[1][1] += z.y \* z.y / div;

}

public:

method\_variable\_metric(point x, double eps, OneDimensionFunction function, std::string fname)

{

this->function = function;

FILE\* out;

FILE\* out2;

const char\* c = fname.c\_str();

const char\* c1 = "trajectory.txt";

fopen\_s(&out, c, "w");

fopen\_s(&out2, c1, "w");

if (out)

{

fprintf(out, "| ITERS | (Xi,Yi) | f(Xi,Yi)| (s1,s2) | Lambd |/Xi-Xi-1/| angle | H |\n");

fprintf(out, "| | | | | |/Yi-Yi-1/| | |\n");

fprintf(out, "| | | | | |/fi-fi-1/| | |\n");

}

int iterations = 0;

double lambd = 0;

int n = 2;

double angle;

double df;

count\_f = 0;

point grad1 = { 0,0 }, grad2, pk, gk, dxk, xk1;

double H0[2][2] = { 0 };

H0[0][0] = H0[1][1] = 1;//единичная матрица

grad1 = find\_grad(x);

if (out) {

fprintf(out, "|%- 7i|(%- 7f, %- 7f)|%- 7f| | | | | |\n", iterations, x.x, x.y, function(x));

}

if (out2) {

fprintf(out2, "%- 7f %- 7f\n", x.x, x.y);

}

while (norm(grad1) > eps)

{

pk = -1 \* (H0 \* grad1);

lambd = search\_lambd(lambd, x, pk);

xk1 = x + lambd \* pk;

grad2 = find\_grad(xk1);

dxk = xk1 - x;

gk = grad2 - grad1;

Gessian(H0, dxk, gk);

df = function(xk1) - function(x);

x = xk1;

angle = x \* pk / (norm(x) \* norm(pk));

iterations++;

if (out) {

fprintf(out, "|%- 7i|(%- 7f, %- 7f)|%- 7f|(%- 7f, %- 7f)|%- 7f|%- 7f|%- 7f|[%- 7f %- 7f]|\n", iterations, xk1.x, xk1.y, function(xk1), -pk.x, -pk.y, lambd, abs(dxk.x), angle, H0[0][0], H0[0][1]);

fprintf(out, "| | | | | |%- 7f| |[%- 7f %- 7f]|\n", abs(dxk.y), H0[1][0], H0[1][1]);

fprintf(out, "| | | | | |%- 7f| | |\n", abs(df));

}

if (out2) {

fprintf(out2, "%- 7f %- 7f\n", xk1.x, xk1.y);

}

grad1 = grad2;

//if (iterations % n == 0) //обновление алгоритма после n итераций

//{

// H0[0][0] = H0[1][1] = 1;//единичная матрица

// grad1 = find\_grad(x);

//}

if (iterations > 50)

break;

std::cout << x.x << " " << x.y << std::endl;

}

if (out)

fclose(out);

if (out2)

fclose(out2);

std::cout << "Func: " << function(x) << std::endl;

std::cout << "Iterations: " << iterations << std::endl;

std::cout << "Fcount: " << count\_f << std::endl;

std::cout << std::endl;

}

};

Trajectory.py

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.tri as tri

x, y = [], []

fig, ax = plt.subplots()

plt.grid(True)

val = 0.

try:

file = open('trajectory.txt', 'r')

coords = [line.split() for line in file]

for (c1, c2) in coords:

x.append(float(c1))

y.append(float(c2))

x1 = np.arange(-0.1, 2)

y1 = np.arange(-0.1, 2)

X, Y = np.meshgrid(x1, y1)

f = (100 \* (Y - X) \* (Y - X) + (1 - X) \* (1 - X))

ax.contour(X, Y, f, levels=30, linewidths = 1)

plt.plot(x,y, '-o' )

file = open('trajectory1.txt', 'r')

coords = [line.split() for line in file]

for (c1, c2) in coords:

x.append(float(c1))

y.append(float(c2))

plt.gca().set\_aspect('equal', adjustable='box')

plt.plot(x,y,'ro-', lw=0.7, alpha=1, markersize=1.5)

plt.savefig(fname='pic\_1.png', dpi = 150)

except IOError:

print("Can't open file")

ax.set\_xlabel("x",fontsize = 14, loc = "right")

ax.set\_ylabel("y", fontsize = 14, loc = "top", rotation='horizontal')

plt.show()