|  |  |
| --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | |
| Федеральное государственное автономное образовательное учреждение высшего образования | |
| **«Дальневосточный федеральный университет»** (ДВФУ) | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | |
| **Департамент математического и компьютерного моделирования** | |
| **ЛАБОРАТОРНАЯ РАБОТА №2** | |
| По основной образовательной программе подготовки бакалавров  направлению 02.03.01 Математика и компьютерные науки  профиль «Сквозные цифровые технологии» | |
|  | Студент группы  Ле нгок куок лик  (подпись)  «20» декабря 2023 г. |
|  | Преподаватель кандидат физико-\_  (должность, ученое звание)  математических наук\_\_\_\_\_\_\_\_\_\_\_\_  Яковлев Анатолий Александрович\_  (подпись) (ФИО)  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. |
| г. Владивосток  2023 | |

**Постановка задачи:**

Найти минимум функции :



с условием .

**Исходные данные:**

 – произвольная симметрическая, невырожденная матрица, .

 – произвольный ненулевой вектор, 

 – произвольный начальный ненулевой вектор, 

 – радиус сферы

**Решение:**

Найдём функцию Лагранжа:

.

Найдём точки минимума. Для этого возьмём частную производную по  и приравняем её к нулю:

.

Рассмотрим два случая:

1. Пусть .

, тогда , где  – «подозрительная» на минимум точка.

Проверим, подходит ли данная точка под условие :

Условие не выполняется. Таким образом, найденная точка не подходит под ограничения и не будет рассматриваться при выборе итогового ответа.

2. Пусть .

Преобразуем  и получим следующую систему уравнений из пяти уравнений:

.

Для нахождения точек, подозрительных на оптимум, воспользуемся методом Ньютона:

,

где  – пятимерный вектор неизвестных, составленный из элементов вектора  и .

 – левая часть данной системы,

 – матрица Якоби данной системы уравнений.

.

Метод Ньютона будем запускать на нескольких начальных приближениях, т.к. функция может иметь несколько оптимальных точек. За начальное приближение берётся восемь точек:

Условие для выхода из цикла:

,

где .

В результате получаем несколько точек , подозрительных на оптимум:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | Начальное приближение |  |  |  |
| 1 | 5.66031e+13  7.69353e+13  1.90908e+13  -3.84365e+11 | -0.89372  -0.264877  -1.25682 0.53238 | 0.101775 | -0.690654 |
| 2 | 7.61231e+13  -4.10831e+13  8.56751e+13  -3.48418e+13 | -0.89372  -0.264877  -1.25682 0.53238 | 0.101775 | --0.690654 |
| 3 | 6.18606e+13  8.02614e+13  -6.28835e+13  9.26387e+13 | - 0.89372  -0.264877  -1.25682 0.53238 | 0.101775 | -0.690654 |
| 4 | 9.83035e+13  -5.5793e+13  6.93259e+13  5.07223e+13 | -0.89372  -0.264877  -1.25682 0.53238 | 0.101775 | -0.690654 |
| 5 | -1.11817e+13  6.53846e+13  -3.25615e+13  -1.27968e+13 | -0.89372  -0.264877  -1.25683 0.53238 | 0.101775 | -0.690654 |
| 6 | -4.074e+13  1.57089e+13  6.43449e+13  -7.30362e+13 | -0.89372  -0.264877  -1.25683 0.53238 | 0.101775 | -0.690654 |
| 7 | 5.59362e+13  -1.06265e+12  9.38423e+13  -1.35963e+13 | 0.591936 -1.4656 2.08454 -1.21384 | -0.169681 | 2.11311 |
| 8 | 3.82298e+13  -6.63072e+13  8.94689e+13  1.38456e+13 | -0.89372  -0.264877  -1.25682 0.53238 | 0.169681 | -0.690654 |

Выясним, в какой из данных точек функция принимает минимальное значение. Отбросим результаты, полученные при , и получим, что минимальное значение функции  при заданных ограничениях достигается в точке:

.

Минимальное значение функции:

**Приложения**

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <Eigen/Dense>  
#include <chrono>  
#include <random>  
using namespace Eigen;  
using namespace std;  
using vec = Matrix<double, Dynamic, 1>;  
using mat = Matrix<double, Dynamic, Dynamic>;  
  
Eigen::MatrixXd gen\_matrix( int dim) {  
 Eigen::MatrixXd random\_matrix(dim,dim) ;  
// std::minstd\_rand random (std::chrono::system\_clock::now().time\_since\_epoch().count());  
 std::random\_device rd;  
 std::mt19937 gen(rd());  
 std::uniform\_real\_distribution<> dis(1e-10, 1);  
  
  
 for(int i = 0; i < dim; i++)  
 for(int j = 0; j < dim; j++)  
 (i == j) ? (random\_matrix(j,i) = dis(gen)) : 0;  
  
 Eigen::MatrixXd A(dim,dim);  
  
 for(int i = 0; i < dim; i++)  
 for(int j = 0; j < dim; j++)  
 A(j,i) = dis(gen);  
  
  
 random\_matrix = A \* random\_matrix \* A.transpose();  
 return random\_matrix;  
}  
  
vec gen\_vec(double low = 1e-3, double high = 1, int dim = 4) {  
 random\_device rd;  
 mt19937 gen(rd());  
 uniform\_real\_distribution<> dis(low, high);  
  
 vec v(dim);  
 for (int i = 0; i < dim; ++i) {  
 v(i) = dis(gen);  
 }  
 return v;  
}  
  
double f\_origin(const mat& A, const vec& b, const vec& x) {  
 return (x.transpose() / 2 \* A \* x + b.transpose() \* x)(0);  
}  
  
vec gen\_vec\_nearby(const vec& x, int dim = 4) {  
 random\_device rd;  
 mt19937 gen(rd());  
 uniform\_real\_distribution<> dis(-1e+14, 1e+14);  
  
 vec v(dim);  
 for (int i = 0; i < dim; ++i) {  
 v(i) = dis(gen) + x(i);  
 }  
 return v;  
}  
  
vec f(const mat& A, const vec& b, double r, const vec& x, const vec& x0, double y) {  
 int dim = A.rows() + 1;  
 vec pre\_res = (A + 2 \* MatrixXd::Identity(dim-1, dim-1) \* y) \* x + b + 2 \* y \* x0;  
 vec result(dim);  
 result.head(dim-1) = pre\_res;  
 result(dim-1) = (x - x0).squaredNorm() - r\*r;  
 return result;  
}  
  
mat jacobian(const mat& A, const vec& x, const vec& x0, double y) {  
 int dim = A.rows() + 1;  
 mat m(dim, dim);  
 m.block(0, 0, dim-1, dim-1) = A + 2 \* MatrixXd::Identity(dim-1, dim-1) \* y;  
 m.block(0, dim-1, dim-1, 1) = 2 \* (x - x0);  
 m.block(dim-1, 0, 1, dim-1) = (2 \* (x - x0)).transpose();  
 m(dim-1, dim-1) = 0;  
 return m;  
}  
  
vec newton(const mat& A, const vec& b, double r, const vec& xk, const vec& x0) {  
 mat jac = jacobian(A, xk.head(xk.rows()-1), x0, xk(xk.rows()-1));  
 return xk - jac.inverse() \* f(A, b, r, xk.head(xk.rows()-1), x0, xk(xk.rows()-1));  
}  
  
void calc(const mat& A, const vec& b, const vec& x0,const double& r, const vector<vec>& x\_approx) {  
 double y = r;  
 vec xy = -A.fullPivLu().solve(b);  
 cout << "y = 0:" << endl;  
 cout << "x\* = " << xy.transpose() << endl;  
 cout << "f(x\*) = " << f\_origin(A, b, xy) << endl;  
 cout << "||x-x0|| <= r:" << endl;  
 cout << (xy - x0).norm() << " <= " << r << endl;  
 int i{};  
 for (const auto& x : x\_approx) {  
 vec x\_prev = x;  
 vec xk = newton(A, b, y, x\_prev, x0);  
 while ((xk - x\_prev).norm() > 1e-6) {  
 x\_prev = xk;  
 xk = newton(A, b, r, x\_prev, x0);  
 }  
 cout << "xk: " << xk.transpose() << endl;  
 cout<<"x["<<i<<"]"<<x<<endl;  
 cout << "f(x): " << f\_origin(A, b, xk.head(xk.rows()-1)) << endl;  
 i++;  
 }  
}  
  
int main() {  
 int dim = 4;  
 mat A = gen\_matrix(dim);  
 vec b = gen\_vec();  
 vec x = gen\_vec();  
 vec x0 = gen\_vec();  
  
 cout<<"matrix A:"<<endl<<A<<endl;  
 cout<<"vecotr b:"<<endl<<b<<endl;  
 cout<<"vecotr x:"<<endl<<x<<endl;  
 cout<<"vector x0:"<<endl<<x0<<endl;  
  
  
  
  
 const double r = M\_PI;  
  
 std::vector<VectorXd> x\_approx;  
 for (int i = 0; i < 8; ++i) {  
 VectorXd x = gen\_vec\_nearby(x0);  
 x.conservativeResize(x.size() + 1);  
 x(x.size() - 1) = r;  
 x\_approx.push\_back(x);  
 }  
  
  
  
 calc(A,b,x0,r,x\_approx);  
  
}