|  |  |
| --- | --- |
| 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 №1** | |
| 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)  «27» ноября 2023 г. |
|  | Преподаватель, кандидат физико-\_  (должность, ученое звание)  математических наук\_\_\_\_\_\_\_\_\_\_\_\_  Яковлев Анатолий Александрович\_  (подпись) (ФИО)  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г. |
| г. Владивосток  2023 | |

**Постановка задачи:**

Минимизировать функцию , где ,

Проверим, что матрицы является положительно определённой. Для этого достаточно убедиться, что собственные значения матрицы положительны:

**Метод градиента:**

*,* где

Первая производная функции: .

Приравнивая производную к нулю, получаем вектор .

Алгоритм отработал за 4326 шагов. Условие выхода из цикла:

*Промежуточные результаты:*

*Промежуточные значения функционала:*

Значение функционала в точке -0.67026 (точное решение)

*Погрешности метода градиента:*

Привести график зависимости значения функции от номера шага методом градиентного спуска.

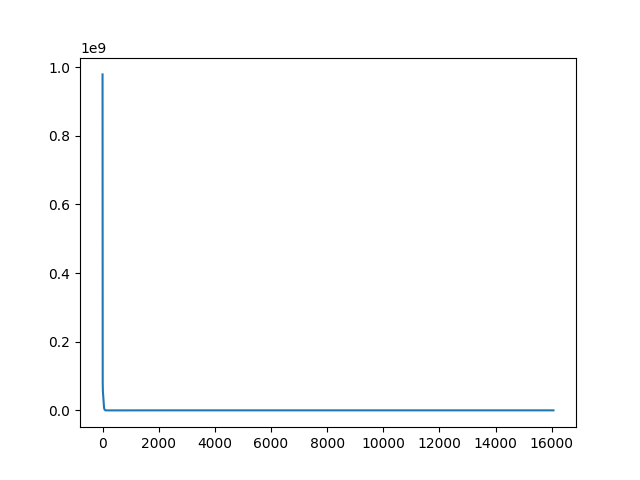


Рисунок 1 – График зависимости значения функции от номера шага методом градиентного спуска

Алгоритм содержится в приложении 1.

**Приложения**

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <Eigen/Dense>  
#include <chrono>  
#include <random>  
  
Eigen::MatrixXd f(const Eigen::VectorXd& x, const Eigen::MatrixXd& A, const Eigen::VectorXd& b) {  
 Eigen::MatrixXd xt = x.transpose();  
 Eigen::MatrixXd bt = b.transpose();  
 return 0.5 \* xt \* A \* x + bt \* x;  
}  
  
Eigen::VectorXd f\_dif(const Eigen::VectorXd& x, const Eigen::MatrixXd& A, const Eigen::VectorXd& b) {  
 return A \* x + b;  
}  
  
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(1.0000000, 15.0000000);  
  
  
 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;  
}  
  
Eigen::VectorXd gen\_vec(int dim) {  
 Eigen::VectorXd A(dim);  
 std::random\_device rd;  
 std::mt19937 gen(rd());  
 std::uniform\_real\_distribution<> dis(1.0000000, 15.0000000);  
  
 for(int i = 0; i < dim ; i++) {  
 A(i) = dis(gen);  
 }  
  
 return A;  
}  
  
  
Eigen::VectorXd gradient(const Eigen::VectorXd& x, const Eigen::MatrixXd& A, const Eigen::VectorXd& b) {  
 return x - 1e-4 \* f\_dif(x, A, b);  
}  
  
  
  
void calculate\_gradient(const Eigen::VectorXd x, const Eigen::MatrixXd& A, const Eigen::VectorXd& b,  
 std::vector<Eigen::VectorXd> list\_of\_x, int& count, Eigen::MatrixXd&M ) {  
  
 list\_of\_x.push\_back(x);  
 list\_of\_x.push\_back(gradient(x,A,b));  
  
 double norm = (list\_of\_x[1] - list\_of\_x[0]).norm();  
  
 while(norm > 1e-6) {  
 list\_of\_x.push\_back(gradient(list\_of\_x[count], A, b));  
 std::cout<<"this is the "<< count<<" iterator"<<std::endl;  
 std::cout<<list\_of\_x[count]<<std::endl;  
 std::cout<<"this is f(x) at x["<<count<<"] "<<f(list\_of\_x[count],A,b)<<std::endl;  
 count++;  
 norm = (list\_of\_x[count] - list\_of\_x[count - 1]).norm();  
 }  
  
}  
  
  
int main() {  
 int dim = 6;  
 Eigen::MatrixXd A = gen\_matrix(dim);  
 Eigen::VectorXd x = gen\_vec(dim);  
 Eigen::VectorXd b = gen\_vec(dim);  
 std::cout<<"this is matrix A: "<<std::endl;  
 std::cout<<A<<std::endl;  
 std::cout<<"this is vector x: "<<std::endl;  
 std::cout<<x<<std::endl;  
 std::cout<<"this is vector b: "<<std::endl;  
 std::cout<<b<<std::endl;  
  
  
 int count = 1;  
 Eigen::MatrixXd M(dim,dim);  
 std::vector<Eigen::VectorXd> list\_of\_x;  
  
 calculate\_gradient(x,A,b,list\_of\_x,count,M);  
  
 std::cout<<"the sum of iterator: "<<count<<std::endl;  
  
  
  
}