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 прикладной математики**

**ОТЧЕТ**

**по дисциплине**

**«Методы оптимизации»**

Работу выполнил М. А. Пузырёв

Работу принял преподаватель Е. С. Троценко

Краснодар

2024

1. **Постановка задачи**

Пусть дана функция , ограниченная снизу на множестве *R*n и имеющая непрерывные частные производные во всех его точках.

Требуется найти локальный минимум функции двух переменных  методом Ньютона-Рафсона при начальных значениях , т.е. такие точки , что .

1. **Стратегия поиска**

Стратегия метода Ньютона-Рафсона состоит в построении последовательности точек , k=0, 1, …, таких, что ,

k = 0, 1, … . Точки последовательности вычисляются по правилу

где x0 задаётся пользователем, а величина шага tk определяется из условия

Задача может решаться либо аналитически с использованием необходимого условия минимума с последующей проверкой достаточного условия , либо численно как задача

*,* ,

где интервал [a, b] задаётся пользователем.

Если функция достаточно сложна, то возможна её замена полиномом второй или третьей степени и тогда шаг может быть определён из условия при выполнении условия .

При численном решении задачи определения величины шага степень близости найденного значения к оптимальному значению , удовлетворяющему условиям , , зависит от задания интервала и точности методов одномерной оптимизации.

Построение последовательности заканчивается в точке , для которой , где – заданное число, или при (M – предельное число итераций), или при двукратном одновременном выполнении двух неравенств , , где – малое положительное число.

1. **Алгоритм**

*Шаг l*. Задать предельное число итераций . Найти градиент и матрицу Гессе .

*Шаг 2*. Положить

*Шаг 3*. Вычислить.

*Шаг 4*. Проверить выполнение критерия окончания :

а) если критерий выполнен, то ;

б) если критерий не выполнен, то перейти к шагу 5.

*Шаг 5*. Проверить выполнение неравенства :

а) если неравенство выполнено, то ;

б) если нет, то перейти к шагу 6.

*Шаг 6*. Вычислить матрицу .

*Шаг 7*. Вычислить матрицу .

*Шаг 8*. Проверить выполнение условия :

а) если условие выполняется, то найти ;

б) если нет, то положить

*Шаг 9.* Определить .

*Шаг 10*. Найти шаг из условия .

*Шаг 11*. Вычислить .

*Шаг 12.* Проверить выполнение условий

а) если оба условия выполнены при текущем значении и , то расчет окончен, ;

б) в противном случае положить и перейти к шагу 3.

**4 Код программы**

Реализация кода представлена на языке C++:

#include <iostream>

#include <cmath>

#include <math.h>

using namespace std;

double F(double x, double y) {

return pow(x, 2) + 4 \* pow(y, 2) + x \* y + x;

}

double Fx(double x, double y) {

return 2 \* x + y + 1;

}

double Fy(double x, double y) {

return x + 8 \* y;

}

double L(double x, double y) {

double l = pow(x, 2) + pow(y, 2);

return sqrt(l);

}

void M(double\*\* A, double\*\* B, int n) {

for (int k = 0; k < n; ++k)

{

double div = A[k][k];

for (int j = 0; j < n; ++j)

{

A[k][j] /= div;

B[k][j] /= div;

}

for (int i = k + 1; i < n; ++i)

{

double multi = A[i][k];

for (int j = 0; j < n; ++j)

{

A[i][j] -= multi \* A[k][j];

B[i][j] -= multi \* B[k][j];

}

}

}

for (int k = n - 1; k > 0; --k)

{

for (int i = k - 1; i >= 0; --i)

{

double multi = A[i][k];

for (int j = 0; j < n; ++j)

{

A[i][j] -= multi \* A[k][j];

B[i][j] -= multi \* B[k][j];

}

}

}

}

double G(double x, double y, double t) { // функция f(t)

return F(x - t \* Fx(x, y), y - t \* Fy(x, y));

}

double Zol(double x, double y) { // метод нахождения минимума функции f(t)

double a = -1;

double b = 1;

double k = 0.38196;

double l = 0.01;

double t1 = a + k \* (b - a);

double t2 = a + b - t1;

do {

if (G(x, y, t1) <= G(x, y, t2)) {

b = t2;

t2 = t1;

t1 = a + b - t2;

}

else {

a = t1;

t1 = t2;

t2 = a + b - t1;

}

} while (fabs(b - a) > l);

return (b + a) / 2;

}

int main() {

setlocale(LC\_ALL, "rus");

int m = 50, it = 0;

double eps1 = 0.001, eps2 = 0.0015;

double x, y, t, x01, y01, x00, y00, d1, d2, x0, y0, X0, Y0;

double\*\* H = new double\* [2];

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

H[i] = new double[2];

}

double\*\* H1 = new double\* [2];

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

H1[j] = new double[2];

}

H[0][0] = 2.0, H[0][1] = 1.0, H[1][0] = 1.0, H[1][1] = 8.0;

H1[0][0] = 1.0, H1[0][1] = 0.0, H1[1][0] = 0.0, H1[1][1] = 1.0;

M(H, H1, 2); // получение матрицы H-1

x00 = x01 = x0 = 3, y00 = y01 = y0 = 1;

if (H1[0][0] > 0 && (H1[0][1] < 0)) {

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

if (L(Fx(x01, y01), Fy(x01, y01)) < eps1)

break;

d1 = -H1[0][0] \* Fx(x01, y01) - H1[0][1] \* Fy(x01, y01);

d2 = -H1[1][0] \* Fx(x01, y01) - H1[1][1] \* Fy(x01, y01);

t = Zol(x01, y01);

x = x01 + t\*d1;

y = y01 + t\*d2;

if (L(x - x01, y - y01) < eps2 && fabs(F(x, y) - F(x01, y01)) < eps2 && L(x01 - x00, y01 - y00) < eps2 && fabs(F(x01, y01) - F(x00, y00)) < eps2) {

x01 = x;

y01 = y;

it++;

break;

}

x00 = x01, y00 = y01;

x01 = x, y01 = y;

if (i == 0) {

X0 = x01;

Y0 = y01;

}

it++;

}

}

else {

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

if (L(Fx(x01, y01), Fy(x01, y01)) < eps1)

break;

d1 = -Fx(x01, y01);

d2 = -Fy(x01, y01);

t = Zol(x01, y01);

x = x01 + t \* d1;

y = y01 + t \* d2;

if (L(x - x01, y - y01) < eps2 && fabs(F(x, y) - F(x01, y01)) < eps2 && L(x01 - x00, y01 - y00) < eps2 && fabs(F(x01, y01) - F(x00, y00)) < eps2) {

x01 = x;

y01 = y;

it++;

break;

}

x00 = x01, y00 = y01;

x01 = x, y01 = y;

if (i == 0) {

X0 = x01;

Y0 = y01;

}

it++;

}

}

double m1;

double l1 = (H[0][0] + H[1][1] + sqrt(pow((H[0][0] + H[1][1]), 2) - 4 \* (H[0][0] \* H[1][1] - H[0][1] \* H[1][0]))) / 2;

double l2 = (H[0][0] + H[1][1] - sqrt(pow((H[0][0] + H[1][1]), 2) - 4 \* (H[0][0] \* H[1][1] - H[0][1] \* H[1][0]))) / 2;

if (l1 >= l2)

m1 = l2;

else

m1 = l1;

bool flag = false;

if (pow(L(x01 - x0, y01 - y0), 2) / m1 >= L(X0 - x0, Y0 - y0))

flag = true;

cout << "Алгоритм завершился за " << it << " итераций.\n";

cout << "Точка минимума X = (" << x01 << "; " << y01 << ")\n";

cout << "Значение функции в минимуме: F = " << F(x01, y01) << endl;

if (flag)

cout << "Алгоритм сходится к точке минимума с квадратичной скоростью." << endl;

return 0;

}

**5 Сходимость**

Пусть дважды непрерывно дифференцируема сильно выпуклая функция с константой на и удовлетворяет условию

Тогда последовательность сходится независимо от выбора начальной точки к точке минимума с квадратичной скоростью

,

где m – оценка наименьшего собственного значения матрицы Гёссе.

**6 Вывод**

Для решения задачи по поиску безусловного минимума функции методом Ньютона-Рафсона была написана программа на языке C++. Результатом работы программы является вычисленное минимальное значение функции, равное -0,0577486 в точке минимума X = (-0,1654; 0,163857) (рис.1). При изменении параметра и , точка минимума, значение функции в ней и сходимость меняются. При уменьшении и число итераций, необходимых для нахождения безусловного минимума увеличивается с 14 до 39, а вычисленное значение минимума функции F стало равно -0,266602, при этом X = (-0,526855; 0,0683779) (рис.2). В обоих случаях условие сходимости было выполнено, из чего следует, что алгоритм сошёлся в точке минимума с квадратичной скоростью.

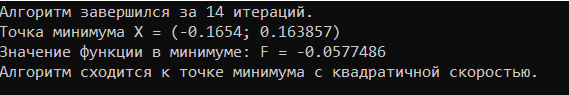


Рисунок 1 – Результат работы программы

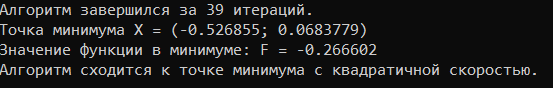


Рисунок 2 – Результат работы программы