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 безусловный минимум функции двух переменных  методом Флетчера–Ривса при начальных значениях , т.е. такие точки , что .

1. **Стратегия поиска**

Стратегия метода Флетчера–Ривса состоит в построении последовательности точек , k = 0,1,.., таких, что < , k=0,1,… .

Точки последовательности вычисляются по правилу:

, k=0,1, …;

;

;

.

Точка x0 задаётся пользователем, величина шага tk определяется для каждого значения k из условия

min.

Решение задачи одномерной минимизации может осуществляться либо из условия = 0, > 0, либо численно, с использованием методов одномерной минимизации, когда решается задача

min, tk [a, b].

При численном решении задачи определения величины шага степень близости найденного значения tk к оптимальному значению , удовлетворяющему условиям = 0, > 0, зависит от задания интервала и точности методов одномерной минимизации.

Вычисление величины по формуле обеспечивает для квадратичной формы = построение последовательности *H*-сопряжённых направлений d0, d1,…, dk,…, для которых = 0, j=0, 1,…, k, i j. При этом в точках последовательности градиенты функции взаимно перпендикулярны, т.е. , k=0, 1, … .

Для квадратичных функций с матрицей *H* > 0 метод Флетчера-Ривса является конечным и сходится за число шагов, не превышающее n – размерность вектора x.

1. **Алгоритм**

Шаг l. Задать предельное число итераций . Найти градиент функции в произвольной точке

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 при k = 0 перейти к шагу 6, а при k 1 перейти к шагу 7.

Шаг 6. Определить d0 = .

Шаг 7. Определить

, .

Шаг 8. Определить dk = .

Шаг 9. Найти из условия

Шаг 10. Вычислить .

Шаг 11. Проверить выполнение условий

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

б) если хотя бы одно из условий не выполнено, то положить и перейти к шагу 3.

**4 Код программы**

Реализация кода представлена на языке C++:

#include <iostream>

#include <cmath>

#include <math.h>

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);

}

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;

}

using namespace std;

int main() {

setlocale(LC\_ALL, "rus");

int m = 50, it = 1;

double eps1 = 0.1, eps2 = 0.15;

double x, y, x0, y0, t, x01, y01, x00, y00, d1, d2, d10, d20, b;

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

t = Zol(x00, y00);

x0 = x00 - t \* Fx(x00, y00);

y0 = y00 - t \* Fy(x00, y00);

d10 = -Fx(x0, y0);

d20 = -Fy(x0, y0);

for (int i = 1; i < m; i++) {

if (L(Fx(x0, y0), Fy(x0, y0)) < eps1)

break;

b = pow(L(Fx(x0, y0), Fy(x0, y0)), 2) / pow(L(Fx(x01, y01), Fy(x01, y01)), 2);

d1 = -Fx(x0, y0) + b \* d10;

d2 = -Fy(x0, y0) + b \* d20;

t = Zol(x0, y0);

x = x0 + t \* d1;

y = y0 + t \* d2;

if (L(x - x0, y - y0) < eps2 && fabs(F(x, y) - F(x0, y0)) < eps2 && L(x0 - x01, y0 - y01) < eps2 && fabs(F(x0, y0) - F(x01, y01)) < eps2) {

x0 = x;

y0 = y;

it++;

break;

}

d10 = d1;

d20 = d2;

x01 = x0, y01 = y0;

x0 = x, y0 = y;

it++;

}

// Матрица Гессе

double MG[2][2];

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

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

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

if (l1 > l2) {

double c = l1;

l1 = l2;

l2 = c;

}

double q = (pow((l1 / l2 - 1), 2)) / (pow((l1 / l2 + 1), 2));

double a = pow((L(x0 - x01, y0 - y01) / L(x01 - x00, y01 - y00)), it / 2);

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

cout << "Точка минимума X = (" << x0 << "; " << y0 << ")\n";

cout << "Значение функции в минимуме: F = " << F(x0, y0) << endl;

cout << "Сходимость функции находится в пределах: " << a << " <= q <= " << q << endl;

return 0;

}

**5 Сходимость**

Оценки скорости сходимости получены только для сильно выпуклых функций, когда последовательность {} сходится к точке минимума функции со скоростью

, k .

**6 Вывод**

Для решения задачи по поиску безусловного минимума функции методом Флетчера-Ривса была написана программа на языке C++. Результатом работы программы является вычисленное минимальное значение функции, равное -0,257762 в точке минимума X = (-0,589689; 0,112205). Сходимость алгоритма оказалась в пределах отрезка [0,0000410749; 0,4] (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, так, к примеру, для = 0,005 и = 0,01 число итераций увеличилось с 6 до 15, при этом значение функции стало равно -0,266612, в точке X = (-0,526492; 0,067479). Сходимость алгоритма оказалась в пределах отрезка [7,04838e-20; 0,4] (рис.2).

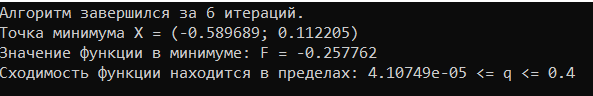


Рисунок 1 – Результат работы программы

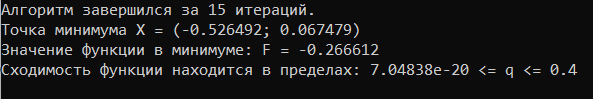


Рисунок 2 – Результат работы программы