Lê Văn Anh Khoa -- 102220153 -- 22T\_DT3

Bài tập:

1. Sử dụng các thuật toán (Gradient Descent, Gradient Descant Momentum, Newton) để thực hiện tối ưu hóa hàm số dưới:
2. Sử dụng thuật toán Lagrage Multipliers để thực hiện tối ưu hóa:

Với điều kiện

Với điều kiện

Bài làm:

1.

Source code:

#include <iostream>

#include <cmath>

#include <conio.h>

#define e 10e-5

using namespace std;

double x0 = -1, y\_zero = -1;  // Điểm bắt đầu

double gama = 0.001, alpha = 0.1;

// Hàm f với hai biến x và y

double f(double x, double y)

{

    return 20 \* x \* x + 14 \* x \* y + 20 \* y \* y - 4 \* x - 5 \* y;

}

// Đạo hàm riêng của f theo x

double f\_prime\_x(double x, double y)

{

    return 40 \* x + 14 \* y - 4;

}

// Đạo hàm riêng của f theo y

double f\_prime\_y(double x, double y)

{

    return 14 \* x + 40 \* y - 5;

}

// Gradient descent cho hàm hai biến

void gradientDescent()

{

    cout << "Gradient Descent: " << endl;

    double x\_prev = x0, y\_prev = y\_zero;

    double x\_next = x\_prev - gama \* f\_prime\_x(x\_prev, y\_prev);

    double y\_next = y\_prev - gama \* f\_prime\_y(x\_prev, y\_prev);

    int count = 1;

    while(fabs(x\_prev - x\_next) > e || fabs(y\_prev - y\_next) > e)

    {

        x\_prev = x\_next;

        y\_prev = y\_next;

        x\_next = x\_prev - gama \* f\_prime\_x(x\_prev, y\_prev);

        y\_next = y\_prev - gama \* f\_prime\_y(x\_prev, y\_prev);

        count++;

    }

    cout << "x = " << x\_next << ", y = " << y\_next << ", fmin = " << f(x\_next, y\_next) << ", so buoc = " << count << endl;

    cout << "--------------------------------------------" << endl;

}

// Gradient descent với momentum cho hàm hai biến

void gradientDescentMomentum()

{

    cout << "Gradient Descent Momentum: " << endl;

    double x\_prev = x0, y\_prev = y\_zero;

    double x\_next = x\_prev - gama \* f\_prime\_x(x\_prev, y\_prev);

    double y\_next = y\_prev - gama \* f\_prime\_y(x\_prev, y\_prev);

    int count = 1;

    double delta\_x = 0, delta\_y = 0;

    while(fabs(x\_prev - x\_next) > e || fabs(y\_prev - y\_next) > e)

    {

        delta\_x = x\_next - x\_prev;

        delta\_y = y\_next - y\_prev;

        x\_prev = x\_next;

        y\_prev = y\_next;

        x\_next = x\_prev - gama \* f\_prime\_x(x\_prev, y\_prev) + alpha \* delta\_x;

        y\_next = y\_prev - gama \* f\_prime\_y(x\_prev, y\_prev) + alpha \* delta\_y;

        count++;

    }

    cout << "x = " << x\_next << ", y = " << y\_next << ", fmin = " << f(x\_next, y\_next) << ", so buoc = " << count << endl;

    cout << "--------------------------------------------" << endl;

}

double f\_double\_prime\_xx(double x, double y) { return 40; }

double f\_double\_prime\_yy(double x, double y) { return 40; }

double f\_double\_prime\_xy(double x, double y) { return 14; }

// Phương pháp Newton cho hàm hai biến

void Newton()

{

    cout << "Newton:" << endl;

    double x\_prev = x0, y\_prev = y\_zero;

    int count = 1;

    double x\_next, y\_next;

    do {

        double det = f\_double\_prime\_xx(x\_prev, y\_prev) \* f\_double\_prime\_yy(x\_prev, y\_prev) - f\_double\_prime\_xy(x\_prev, y\_prev) \* f\_double\_prime\_xy(x\_prev, y\_prev);

        // Công thức Newton cho hai biến

        x\_next = x\_prev - (f\_prime\_x(x\_prev, y\_prev) \* f\_double\_prime\_yy(x\_prev, y\_prev) - f\_prime\_y(x\_prev, y\_prev) \* f\_double\_prime\_xy(x\_prev, y\_prev)) / det;

        y\_next = y\_prev - (f\_prime\_y(x\_prev, y\_prev) \* f\_double\_prime\_xx(x\_prev, y\_prev) - f\_prime\_x(x\_prev, y\_prev) \* f\_double\_prime\_xy(x\_prev, y\_prev)) / det;

        x\_prev = x\_next;

        y\_prev = y\_next;

        count++;

    } while (fabs(f\_prime\_x(x\_prev, y\_prev)) > e || fabs(f\_prime\_y(x\_prev, y\_prev)) > e);

    cout << "x = " << x\_next << ", y = " << y\_next << ", fmin = " << f(x\_next, y\_next) << ", so buoc = " << count << endl;

    cout << "--------------------------------------------" << endl;

}

int main()

{

    gradientDescent();

    gradientDescentMomentum();

    Newton();

    return 0;

}

Kết quả:

