**2. Многомерная оптимизация**

**Цель работы:** Изучение различных методов безусловной многомерной оптимизации и сравнение эффективности их применения для конкретных целевых функций.

**Задание.** В соответствии с вариантом **м**инимизировать целевую функцию (табл.2.2) с точностью . Методы минимизации целевой функции указаны в таблице 2.1. Оценить эффективность методов по количеству итераций.

**Таблица 2.1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Метод первого порядка** | **Метод второго порядка** | **Метод нулевого порядка** |
| 22 | 4 | 7 | 9 |

Принятые в таблице 2.1. обозначения:

4 – Метод покоординатного спуска.

7 – Метод Ньютона - Рафсона.

9 – Симплексный метод.

**Варианты заданий**

**Таблица 2.2**

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант  задания | Вид функции | Ответ: | |
| 22 |  |  |  |

**4.**

*#include <iostream>*

*#include <cmath>*

*#include <string>*

*using namespace std;*

*double fun(double x1, double x2) {*

*return 4 \* pow(x1, 2) + 2 \* x1 \* x2 + 5 \* pow(x2, 2) + 9 \* x1;*

*}*

*double gradient\_x1(double x1, double x2) {*

*return 8 \* x1 + 2 \* x2 + 9;*

*}*

*double gradient\_x2(double x1, double x2) {*

*return 2 \* x1 + 10 \* x2;*

*}*

*int main() {*

*double x1 = 0, x2 = 0;*

*double step = 0.01;*

*double eps = 0.0001;*

*int k = 0;*

*double df;*

*do {*

*double temp\_x1 = x1 - step \* gradient\_x1(x1, x2);*

*double temp\_x2 = x2 - step \* gradient\_x2(x1, x2);*

*df = sqrt(pow(x1 - temp\_x1, 2) + pow(x2 - temp\_x2, 2));*

*x1 = temp\_x1;*

*x2 = temp\_x2;*

*k++;*

*cout << "Iteration " << k << " , x1 = " << x1 << " , x2 = " << x2 << " , f(x) = " << fun(x1, x2) << endl;*

*} while(df > eps);*

*return 0;*

*}*

**7.**

*#include <stdio.h>*

*#include <iostream>*

*#include <cstdlib>*

*#include <cmath>*

*#include <string>*

*using namespace std;*

*double fun(double x1, double x2) {*

*return 4 \* pow(x1, 2) + 2 \* x1 \* x2 + 5 \* pow(x2, 2) + 9 \* x1;*

*}*

*bool Sylvester(double a11, double a12, double a21, double a22) {*

*if (a11 > 0 && a11 \* a22 - a12 \* a21 > 0)*

*return true;*

*else*

*return false;*

*}*

*int main() {*

*double b11, b12, b22, b21, h, det, p1, p2;*

*double df = 1;*

*bool z;*

*double x1\_0 = 1.1842;*

*double x2\_0 = 0.2368;*

*double eps = 0.0001;*

*double x1 = x1\_0;*

*double x2 = x2\_0;*

*double c1, c2;*

*double a11, a12, a21, a22;*

*int k, m;*

*k = m = 0;*

*while (df > eps) {*

*a11 = 8;*

*a12 = 2;*

*a21 = 2;*

*a22 = 10;*

*z = Sylvester(a11, a12, a21, a22);*

*c1 = 8 \* x1 + 2 \* x2 + 9;*

*c2 = 2 \* x1 + 10 \* x2;*

*df = sqrt(pow(c1, 2) + pow(c2, 2));*

*det = a11 \* a22 - a12 \* a21;*

*if (z == true) {*

*k += 1;*

*b11 = a22 / det;*

*b12 = -a12 / det;*

*b21 = -a21 / det;*

*b22 = a11 / det;*

*p1 = b11 \* c1 + b12 \* c2;*

*p2 = b21 \* c1 + b22 \* c2;*

*h = 1.0;*

*} else {*

*m += 1;*

*p1 = c1;*

*p2 = c2;*

*h = 1.0;*

*}*

*x1 -= h \* p1;*

*x2 -= h \* p2;*

*cout << "Iteration " << k << " , x1 = " << x1 << " , x2 = " << x2 << " , f(x) = " << fun(x1, x2) << endl;*

*}*

*return 0;*

*}*

**Из файла Вадима, Ньютон-рафсон**

*#include <iostream>*

*#include <contio.h>*

*#include <cmath>*

*#include <windows.h>*

*#define sqr(x) (x)\*(x)*

*using namespace std;*

*int n;*

*const double dx = 0.0001;*

*double fun(double x1, double x2) {*

*return 4 \* pow(x1, 2) + 2 \* x1 \* x2 + 5 \* pow(x2, 2) + 9 \* x1;*

*}*

*void point(double \*\*2df, double \*x, double \*df, double h){*

*double s;*

*int i, j;*

*for (i = 0; i < n; i++){*

*s = 0;*

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

*s += d2f[i][j] \* df[j];*

*x[i] -= s\*h;*

*}*

*}*

*void Grad(double \*x, double \*df){*

*int i;*

*double f0;*

*f0 = f(x);*

*for (i=0; i<n;i++){*

*x[i] += dx;*

*df[i] = f(x) - f0 / dx;*

*x[i] -= dx;*

*}*

*}*

*double normGrad(double\*df){*

*int i;*

*double s;*

*s = 0;*

*for (i = 0; i < n; i++)*

*s += sqr(df[i]);*

*return sqrt(x: s);*

*}*

*void Gesse(double \*x, double \*\*d2f){*

*double s;*

*for (int i = 0; i < n; i++){*

*s = -2 \* f(x);*

*x[i] += dx;*

*s += f(x);*

*x[i] -= 2 \* dx;*

*s += f(x);*

*x[i] += dx;*

*d2f[i][i] = s/pow(x: dx, y: 2);*

*}*

*for (int i = 0; i< n - 1; i++)*

*for (int j = i+ 1; j < n; j++){*

*s = f(x);*

*x[i] -= dx;*

*x[j] -= dx;*

*s += f(x);*

*x[j] += dx;*

*s -= f(x)*

*x[j] += dx;*

*d2f[i][j] = s/pow(x:dx, y: 2);*

*d2f[j][i] = d2f[i][j];*

*}*

*}*

*int invert(double \*\*d2f){*

*double \*\*a, t;*

*int i, j, k, m, flag = 0;*

*m = 2 \* n;*

*a = new model \*[n];*

*for (i = 0; i < n; i++)*

*a[i] = new double [m];*

*for (i = 0; i < n; i++)*

*for (j = 0; j < m; j++)*

*if (j<n)*

*a[i][j] = d2f[i][j];*

*for (i = 0; i < n; i++){*

*k = 1;*

*while (!a[k][i]){*

*flag = 1;*

*if (k < n - 1) k++;*

*else return flag;*

*}*

*if (flag) for (j = 0; j < m; j++){*

*t = a[k][j];*

*a[k][j] = a[i][j];*

*a[i][j] = t;*

*}*

*}*

*}*

**9.**

Использование симплексного метода может быть нецелесообразно для решения данной задачи, поскольку симплекс-метод обычно используется для решения линейных задач оптимизации. Ваша функция (4x1^2 + 2x1x2 + 5x2^2 + 9x1) нелинейна. Поэтому для решения такого рода задачи вы можете использовать методы, предназначенные для решения нелинейных задач, такие как метод Ньютона или метод градиентного спуска.

Симплекс-метод требует наличия ограничений в виде системы линейных неравенств, а в вашей задаче таковых нет. Вы можете сформулировать для вашей функции систему ограничений и попробовать либо решить ее с использованием симплекс-метода, либо преобразовать вашу нелинейную функцию в линейную и снова применить симплекс-метод.

Если решение необходимо именно с помощью симплексного метода, то задача потребует существенной переформулировки, и, возможно, она не будет в точности соответствовать исходной задаче.

В общем случае, без установленных ограничений для переменных x1 и x2, эта задача не может быть решена с помощью симплексного метода. Однако, задачи минимизации с ограничениями могут быть успешно решены с помощью этого подхода. Если вы можете предоставить такие ограничения, задача может быть решена с использованием симплексного метода.

Поэтому, мы настоятельно рекомендуем оставить ваши исходные данные и использовать их с минимальной поправкой для построения надежных решений, используя, например, метод Ньютона.

Если вы все еще хотите использовать симплексный метод, пожалуйста, укажите явные ограничения для переменных x1 и x2.