**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.**

**1 вариант решения:**

*#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;*

*}*

**2 вариант решения:***#include <iostream>*

*#include <cmath>*

*// Определение целевой функции*

*double target\_function(double x1, double x2) {*

*return 4 \* x1 \* x1 + 2 \* x1 \* x2 + 5 \* x2 \* x2 + 9 \* x1;*

*}*

*int main() {*

*double x1 = 0.0; // Начальное значение x1*

*double x2 = 0.0; // Начальное значение x2*

*double tolerance = 0.0001; // Заданная точность*

*int iterations = 0;*

*while (true) {*

*double gradient\_x1 = 8 \* x1 + 2 \* x2 + 9;*

*double gradient\_x2 = 2 \* x1 + 10 \* x2;*

*// Обновление значений переменных*

*x1 -= 0.1 \* gradient\_x1; // Пример шага спуска*

*x2 -= 0.1 \* gradient\_x2; // Пример шага спуска*

*double change = sqrt(gradient\_x1 \* gradient\_x1 + gradient\_x2 \* gradient\_x2);*

*iterations++;*

*// Проверка условия окончания итераций*

*if (change < tolerance) {*

*break;*

*}*

*}*

*double result = target\_function(x1, x2);*

*std::cout << "x^ = (" << x1 << "; " << x2 << "), f(x^) = " << result << std::endl;*

*std::cout << "Количество итераций: " << iterations << std::endl;*

*return 0;*

*}*

**7.**

*#include <iostream>*

*#include <cmath>*

*using namespace std;*

*// Функция для вычисления значения функции f(x1, x2)*

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

*return 4 \* x1 \* x1 + 2 \* x1 \* x2 + 5 \* x2 \* x2 + 9 \* x1;*

*}*

*// Функция для вычисления градиента функции*

*void gradient(double x1, double x2, double& gradX1, double& gradX2) {*

*gradX1 = 8 \* x1 + 2 \* x2 + 9;*

*gradX2 = 2 \* x1 + 10 \* x2;*

*}*

*// Функция для вычисления матрицы Гессе*

*void Gesse(double x1, double x2, double& a11, double& a12, double& a21, double& a22) {*

*a11 = 8;*

*a12 = 2;*

*a21 = 2;*

*a22 = 10;*

*}*

*int main() {*

*double x1 = 0, x2 = 0; // начальные значения переменных*

*double deltaX1, deltaX2;*

*double epsilon = 0.0001; // заданная точность*

*double gradX1, gradX2;*

*double a11, a12, a21, a22;*

*int iterations = 0;*

*do {*

*gradient(x1, x2, gradX1, gradX2);*

*Gesse(x1, x2, a11, a12, a21, a22);*

*double determinant = a11 \* a22 - a12 \* a21;*

*double inverseA11 = a22 / determinant;*

*double inverseA12 = -a12 / determinant;*

*double inverseA21 = -a21 / determinant;*

*double inverseA22 = a11 / determinant;*

*deltaX1 = inverseA11 \* gradX1 + inverseA12 \* gradX2;*

*deltaX2 = inverseA21 \* gradX1 + inverseA22 \* gradX2;*

*x1 -= deltaX1;*

*x2 -= deltaX2;*

*iterations++;*

*} while (sqrt(deltaX1 \* deltaX1 + deltaX2 \* deltaX2) > epsilon);*

*cout << "x^ = (" << x1 << "; " << x2 << "), ";*

*cout << "f(x^) = " << objectiveFunction(x1, x2) << endl;*

*cout << "Number of iterations: " << iterations << 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.**

**Решение на питоне 1:**

*import numpy as np*

*from scipy.optimize import minimize*

*# Определение целевой функции*

*def objective(x):*

*x1, x2 = x*

*return 4 \* x1\*\*2 + 2 \* x1 \* x2 + 5 \* x2\*\*2 + 9 \* x1*

*# Начальное приближение*

*x0 = [0, 0]*

*# Вызов функции минимизации с методом симплекса (Nelder-Mead)*

*result = minimize(objective, x0, method='nelder-mead', tol=0.0001)*

*# Вывод результатов*

*print("Минимум найден в точке:", result.x)*

*print("Значение целевой функции в минимуме:", result.fun)*

*print("Количество итераций:", result.nit)*

**Решение на питоне 2:**

*import numpy as np*

*# Определение функции*

*def f(x):*

*x1, x2 = x*

*return 4 \* x1\*\*2 + 2 \* x1 \* x2 + 5 \* x2\*\*2 + 9 \* x1*

*# Градиент функции*

*def grad\_f(x):*

*x1, x2 = x*

*df\_dx1 = 8 \* x1 + 2 \* x2 + 9*

*df\_dx2 = 2 \* x1 + 10 \* x2*

*return np.array([df\_dx1, df\_dx2])*

*# Градиентный спуск*

*def gradient\_descent(x, learning\_rate, iterations, precision):*

*for i in range(iterations):*

*gradient = grad\_f(x)*

*new\_x = x - learning\_rate \* gradient*

*if np.linalg.norm(new\_x - x) < precision:*

*break*

*x = new\_x*

*return x, f(x)*

*# Начальные значения и параметры*

*starting\_point = np.array([0, 0])*

*learning\_rate = 0.01*

*max\_iterations = 10000*

*precision = 0.0001*

*# Запуск градиентного спуска*

*result, min\_value = gradient\_descent(starting\_point, learning\_rate, max\_iterations, precision)*

*print("Минимум достигается в точке:", result)*

*print("Значение функции в минимуме:", min\_value)*

**Решение на питоне 1:**

**Решение на питоне 1:**

**Решение на питоне 1:**