数学建模大作业报告

姓名	班级	学号
郑德凯	软件2202班	U202217216

题目选择

本次大作业选择 **问题E**，结合具体案例对 **最速下降法**、**牛顿法** 和 **DFP算法** 三个优化算法进行编程实现和比较。

一、案例描述

目标是寻找函数

$$f(x, y, z) = x^2 + 2y^2 + 3z^2$$

的极小点。设初始点为 $(1, 1, 1)$，收敛阈值为 $\epsilon = 0.01$。

二、算法实现

1. 最速下降法

原理：

最速下降法通过沿着目标函数梯度的负方向更新参数寻找极小点。算法核心步骤如下：

1. 计算当前点 x_k 的梯度 $\nabla f(x_k)$。
2. 确定步长 α_k，满足一定准则（例如 Wolfe 条件）。
3. 更新点 $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$。
4. 重复上述步骤，直到满足收敛条件 $\|\nabla f(x_k)\| < \epsilon$。

代码实现：

```
import numpy as np
```

```python
# 定义目标函数

def f(x):

    return x[0]**2 + 2 * x[1]**2 + 3 * x[2]**2


# 定义梯度函数

def grad_f(x):

    return np.array([2 * x[0], 4 * x[1], 6 * x[2]])


# 最速下降法

def gradient_descent(initial_x, epsilon, step_size):

    x = initial_x

    history = [x]

    while np.linalg.norm(grad_f(x)) >= epsilon:

        x = x - step_size * grad_f(x)

        history.append(x)

    return x, history


# 参数设置

initial_x = np.array([1, 1, 1])

epsilon = 0.01

step_size = 0.1


# 执行算法

optimal_x, history = gradient_descent(initial_x, epsilon, step_size)
```

```python
print("极小点:", optimal_x)

print("迭代次数:", len(history))
```

运行结果：

- 极小点：$[4.72236648e-034.73838134e-062.81474977e-10]$
- 迭代次数：25

---

2. 牛顿法

原理：

牛顿法使用目标函数的梯度和 Hessian 矩阵寻找极值点。公式如下：

$$x_{k+1} = x_k - H^{-1} \nabla f(x_k)$$

其中，H 为 Hessian 矩阵。

代码实现：

```python
# 牛顿法

def newton_method(initial_x):

    H_inv = np.linalg.inv(np.array([[2, 0, 0], [0, 4, 0], [0, 0, 6]]))  # Hessian 矩阵的逆

    x_newton = initial_x - H_inv.dot(grad_f(initial_x))

    return x_newton



# 执行算法

optimal_x = newton_method(initial_x)

print("极小点:", optimal_x)
```

运行结果：

- 极小点：$[0, 0, 0]$

3. DFP算法

原理：

DFP 算法通过更新逆 Hessian 矩阵的估计来寻找极小点，核心更新公式如下：

$$H_{k+1} = H_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k}$$

其中，$s_k = x_{k+1} - x_k$，$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$。

代码实现：

```
# DFP算法

def dfp_method(f, grad_f, initial_x, epsilon, max_iter=1000):

    x = initial_x

    H = np.eye(len(x))  # 初始逆 Hessian 矩阵

    g = grad_f(x)

    history = [x]

    for _ in range(max_iter):

        if np.linalg.norm(g) < epsilon:

            break

        p = -H.dot(g)

        alpha = 0.1

        x_new = x + alpha * p

        g_new = grad_f(x_new)

        s = x_new - x

        y = g_new - g

        sy = s.dot(y) + 1e-10  # 避免除零

        H = H + np.outer(s, s) / sy - H.dot(np.outer(y, y)).dot(H) /
(y.dot(H).dot(y) + 1e-10)
```

```
        x, g = x_new, g_new
        history.append(x)
    return x, history


# 执行算法
optimal_x, history = dfp_method(f, grad_f, initial_x, epsilon)
print("极小点:", optimal_x)
```

运行结果：

- 极小点：$[0.001883840.001403150.00117085]$
- 迭代次数：58

三、算法比较

算法	迭代次数	极小点	特点
最速下降法	25	$[4.72236648e-034.73838134e-062.81474977e-10]$	收敛较慢，适合初学者实现
牛顿法	1	$[0,0,0]$	对二次函数高效，但 Hessian 矩阵计算复杂
DFP算法	58	$[0.001883840.001403150.00117085]$	不需 Hessian 矩阵，适合大规模优化问题

四、总结

通过对比，可以发现牛顿法在二次函数上表现最佳，但对于复杂问题可能不适用。最速下降法简单直观，但收敛较慢。DFP 算法在兼顾效率和适用性上表现出色，是实际优化中的常用方法。