2021级最优化作业题

姓名	班级	学号	时间
			2023年11月8日

> 这里选择问题E,结合具体案例对最速下降法、牛顿法、 DFP算法三个算法进行编程实现。

一、案例

求函数$f(x)=2x_1^2+x_2^2$的极小点。设初始点为$x_0=(1,1)$，$\epsilon=\frac{1}{10}$。

二、最速下降法

2.1 原理

最速下降法（又称梯度下降法）是一种优化算法，它通过在每一步沿着目标函数梯度的负方向更新参数来寻找函数的极小点。对于函数 $f(\mathbf{x})=2x_1^2+x_2^2$，梯度向量可以通过对每个变量求偏导来计算：

$$\nabla f(\mathbf{x})=\left(\frac{\partial f}{\partial x_1},\frac{\partial f}{\partial x_2}\right)^T$$

对于 $f(\mathbf{x})=2x_1^2+x_2^2$，偏导数是：

$$\frac{\partial f}{\partial x_1}=4x_1,\quad \frac{\partial f}{\partial x_2}=2x_2$$

因此，梯度向量为 $\nabla f(\mathbf{x})=(4x_1,2x_2)^T$。

在最速下降法中，我们需要迭代以下步骤来更新 x：

1. 计算当前点$\mathbf{x}$的梯度 $\nabla f(\mathbf{x})$。
2. 确定步长α，这通常通过线搜索来完成，以满足一定的准则，例如Wolfe条件。
3. 更新$\mathbf{x}$为$\mathbf{x}-\alpha\nabla f(\mathbf{x})$。
4. 重复这个过程，直到$\|\nabla f(\mathbf{x})\|<\epsilon$，在这里$\epsilon=\frac{1}{10}$。

我们可以开始实现这个过程，设初始点为$\mathbf{x}_0=(1,1)^T$。我们需要一个合适的方法来选择步长 α，一个简单的选择是使用固定的步长，例如$\alpha=0.1$，或者实现一个简单的回溯线搜索。在这里，我们将使用一个较小的固定步长并观察收敛情况。

2.2 解答

使用python编程如下：

```python
import numpy as np

# 定义函数f(x)
def f(x):
    return 2*x[0]**2 + x[1]**2

# 定义梯度计算函数
def grad_f(x):
    return np.array([4*x[0], 2*x[1]])

# 最速下降法
def gradient_descent(initial_x, epsilon, step_size):
    x = initial_x
    history = [x] # 用于存储迭代过程中的x值
    while np.linalg.norm(grad_f(x)) ≥ epsilon:
        # 更新x
        x = x - step_size * grad_f(x)
        history.append(x)
    return x, history

# 初始点
initial_x = np.array([1, 1])
# 收敛阈值
epsilon = 1/10
# 步长
step_size = 0.1

# 执行最速下降法
optimal_x, history = gradient_descent(initial_x, epsilon, step_size)

print("极小点:", optimal_x)
print("迭代次数:", len(history))
```

运行上述程序，得到如下输出：

```
极小点: [0.00078364 0.04398047]
迭代次数: 15
```

使用最速下降法，我们找到了函数$f(x) = 2x_1^2 + x_2^2$的极小点大约在$(0.0008, 0.0440)$处。这个过程经历了 15 次迭代才满足梯度的模长小于$\epsilon = \frac{1}{10}$的收敛条件。

由于这个函数是一个凸函数，并且有一个明显的全局最小值在 (0, 0)，我们可以推断该方法找到的极小点非常接近真实的全局最小值，即使是在使用了固定步长的简化情况下。

三、牛顿法

3.1 原理

牛顿法是一种寻找函数零点或极值点的迭代算法。对于寻找函数极小点的问题，牛顿法迭代公式如下：

$$\mathbf{x}_{n+1} = \mathbf{x}_n - H^{-1}(\mathbf{x}_n)\nabla f(\mathbf{x}_n)$$

其中，$\mathbf{x}_n$ 是当前迭代值，$H(\mathbf{x}_n)$ 是函数 f 在 $\mathbf{x}_n$ 处的Hessian矩阵，$\nabla f(\mathbf{x}_n)$ 是函数 f 的梯度。

对于二次函数 $f(x) = 2x_1^2 + x_2^2$，Hessian矩阵是一个常数，因为二次项的二阶导数是常数。对于这个函数，Hessian矩阵是：

$$H(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$$

由于Hessian矩阵是对角矩阵且不依赖于 $\mathbf{x}$，所以其逆矩阵也是容易计算的。在这个特殊情况下，牛顿法会在一步之内直接到达极小点，因为我们是在用二次函数的牛顿法，它直接找到了函数的极小值。

3.2 解答

使用python编程如下：

```python
import numpy as np

# 定义梯度计算函数
def grad_f(x):
    return np.array([4*x[0], 2*x[1]])

# 牛顿法求解极小点
def newton_method(initial_x):
    # Hessian矩阵的逆矩阵，对于给定函数是常数
    H_inv = np.linalg.inv(np.array([[4, 0], [0, 2]]))
    # 进行一次牛顿迭代
    x_newton = initial_x - H_inv.dot(grad_f(initial_x))
    return x_newton

# 初始点
```

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

# 执行牛顿法
optimal_x = newton_method(initial_x)

print("极小点:", optimal_x)

```

程序输出结果如下：

```
极小点: [0. 0.]
```

用牛顿法，我们立即找到了函数 $f(x) = 2x_1^2 + x_2^2$ 的极小点，位于 $(0, 0)$。这是因为牛顿法对于二次函数是非常有效的，特别是当Hessian矩阵是常数时，它可以直接在一次迭代中找到极值点。在这种情况下，牛顿法展示了其对于二次问题的解析求解能力。

四、DFP算法

4.1 原理

DFP（Davidon-Fletcher-Powell）算法是一种求解无约束优化问题的迭代算法，属于拟牛顿方法的一种。DFP算法通过不断更新逆Hessian矩阵的估计来寻找函数的极小点，其基本步骤如下：

1. 选择一个初始点 $\mathbf{x}_0$ 和初始逆Hessian估计 H_0，通常 H_0 可以是单位矩阵。
2. 计算梯度 $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$。
3. 对于 $k = 0, 1, 2, \ldots$ 直到收敛：
   - 计算搜索方向 $\mathbf{p}_k = -H_k \mathbf{g}_k$。
   - 用线搜索找到步长 α_k，使得 $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ 相对 α_k 最小化。
   - 更新 $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$。
   - 计算新梯度 $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$。
   - 如果 $\|\mathbf{g}_{k+1}\|$ 足够小，则停止迭代。
   - 计算 $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ 和 $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$。
   - 更新逆Hessian估计 H_{k+1}。

函数 $f(\mathbf{x}) = 2x_1^2 + x_2^2$ 的梯度和Hessian矩阵都是很简单的。我们可以使用固定的步长，因为这是一个凸二次函数，这样可以简化线搜索过程。

4.2 解答

使用python编程如下：

```python
import numpy as np

# 定义函数f(x)
def f(x):
    return 2*x[0]**2 + x[1]**2

# 定义梯度计算函数
def grad_f(x):
    return np.array([4*x[0], 2*x[1]])

# DFP算法实现
def dfp_method(f, grad_f, initial_x, epsilon, max_iter=1000):
    x = initial_x
    n = len(x)
    H = np.eye(n)  # 初始化逆Hessian矩阵为单位矩阵
    g = grad_f(x)
    history = [x]  # 用于存储迭代过程中的x值

    for _ in range(max_iter):
        if np.linalg.norm(g) < epsilon:  # 检查收敛性
            break

        # 计算搜索方向
        p = -np.dot(H, g)

        # 这里我们使用一个简单的固定步长
        alpha = 0.1

        # 更新x值
        x_new = x + alpha * p
        g_new = grad_f(x_new)
        s = x_new - x
        y = g_new - g

        # 避免除以0，加入一个小的正数
        sy = np.dot(s, y)
        if sy == 0:
            sy += 1e-10

        # 更新逆Hessian矩阵
        Hs = np.dot(H, s)
        ys = np.dot(y, s)
```

```
        Hys = np.dot(H, y)
        sHs = np.dot(s, Hs)

        # DFP更新公式
        H = H + np.outer(s, s) / sy - np.outer(Hs, Hs) / sHs

        # 为下一次迭代做准备
        x = x_new
        g = g_new
        history.append(x)

    return x, history

# 初始点
initial_x = np.array([1, 1])
# 收敛阈值
epsilon = 1/10

# 执行DFP算法
optimal_x, history = dfp_method(f, grad_f, initial_x, epsilon)

print("极小点:", optimal_x)

```

结果如下：

```
极小点: [0.0014867  0.04557862]
```

使用DFP算法，我们找到了函数 $f(x) = 2x_1^2 + x_2^2$ 的极小点大约在 $(0.0015, 0.0456)$ 处。这个结果与最速下降法和牛顿法得到的结果是一致的，都非常接近于函数的真实极小点 $(0, 0)$。