



# 最优化理论与方法期中作业

## 作业三

姓名 \_\_\_\_\_ 梁艺琳

学号 \_\_\_\_\_ 23354099

学院 \_\_\_\_\_ 智能工程学院

专业 \_\_\_\_\_ 智能科学与技术

2025 年 12 月 12 日

# 目录

<b>1 第一题</b>	<b>1</b>
1.1 问题分析	1
1.2 详细求解过程	1
1.2.1 迭代过程的伪代码	1
1.2.2 迭代过程的可视化分析	2
1.3 求解结果	2
<b>2 第二题</b>	<b>3</b>
2.1 问题分析	3
2.2 详细求解过程	3
2.2.1 子问题求解策略 (牛顿方向截断)	3
2.2.2 迭代过程的可视化分析	4
2.3 求解结果	4
<b>3 第三题</b>	<b>5</b>
3.1 问题分析	5
3.2 详细求解过程	5
3.2.1 迭代过程的伪代码	5
3.2.2 迭代过程的可视化分析	6
3.3 求解结果	6
<b>4 第四题</b>	<b>6</b>
4.1 问题分析	6
4.2 详细求解过程	7
4.2.1 线性共轭梯度法迭代公式	7
4.2.2 迭代过程的可视化分析	7
4.3 求解结果	8
<b>5 前半学期课程总结</b>	<b>8</b>
5.1 知识点总结与分析	8
5.1.1 迭代优化的基础框架	8
5.1.2 经典下降算法	9
5.1.3 高效二阶近似算法	9
5.1.4 特殊结构优化方法	10
5.2 个人角度的课程难点	10
5.3 前半学期的最优化方法在自己其他课程和科研工作中的应用设想	11

<b>附录</b>	<b>11</b>
<b>A 代码</b>	<b>11</b>
A.1 问题一代码 . . . . .	11
A.2 问题二代码 . . . . .	12
A.3 问题三代码 . . . . .	15
A.4 问题四代码 . . . . .	16

# 1 第一题

## 1.1 问题分析

目标函数  $f(\mathbf{x}) = x_1^2 + 6x_2^2 + 3x_1x_2$  是关于  $\mathbf{x} = [x_1, x_2]^T$  的二次函数。计算梯度向量  $\nabla f(\mathbf{x})$  和 Hesse 矩阵  $\mathbf{H}$ :

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 + 3x_2 \\ 3x_1 + 12x_2 \end{bmatrix} \quad (1)$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 3 & 12 \end{bmatrix} \quad (2)$$

计算  $\mathbf{H}$  的一阶主子式  $D_1 = 2 > 0$ , 二阶主子式  $D_2 = 2 \times 12 - 3 \times 3 = 15 > 0$ 。因此  $\mathbf{H}$  为正定矩阵, 该问题是严格凸二次规划问题, 存在唯一全局极小值点。理论最优解满足  $\nabla f(\mathbf{x}^*) = 0$ , 解得  $\mathbf{x}^* = [0, 0]^T$ 。

最速下降法采用负梯度方向作为搜索方向, 即  $\mathbf{d}_k = -\mathbf{g}_k = -\nabla f(\mathbf{x}_k)$ 。

本题采用精确线搜索方法确定步长  $\alpha_k$ 。对于正定二次函数, 使  $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$  最小的  $\alpha_k$  具有解析解:

$$\alpha_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T \mathbf{H} \mathbf{g}_k} \quad (3)$$

## 1.2 详细求解过程

### 1.2.1 迭代过程的伪代码

---

#### Algorithm 1 基于精确线搜索的最速下降法

---

**Require:** 目标函数  $f(\mathbf{x})$ , 初始点  $x^{(0)}$ , 精度  $\epsilon$ , 最大迭代次数  $K_{max}$

**Ensure:** 最优解  $x^*$

```

1:  $k \leftarrow 0, \quad x \leftarrow x^{(0)}$ 
2: while  $k < K_{max}$  do
3:    $g \leftarrow \nabla f(x)$                                  $\triangleright$  计算当前点的梯度
4:   if  $\|g\| < \epsilon$  then                          $\triangleright$  检查梯度模长是否满足收敛条件
5:     break                                          $\triangleright$  跳出循环
6:   end if
7:    $p \leftarrow -g,$                                 $\triangleright$  设置搜索方向 (负梯度方向)
8:    $\alpha \leftarrow \frac{p^T p}{p^T H p}$                        $\triangleright$  计算二次函数的精确步长
9:    $x \leftarrow x + \alpha p, \quad k \leftarrow k + 1$        $\triangleright$  更新当前点, 迭代计数加一
10: end while
11: return  $x$ 

```

---

完整代码见附录 A.1。

### 1.2.2 迭代过程的可视化分析

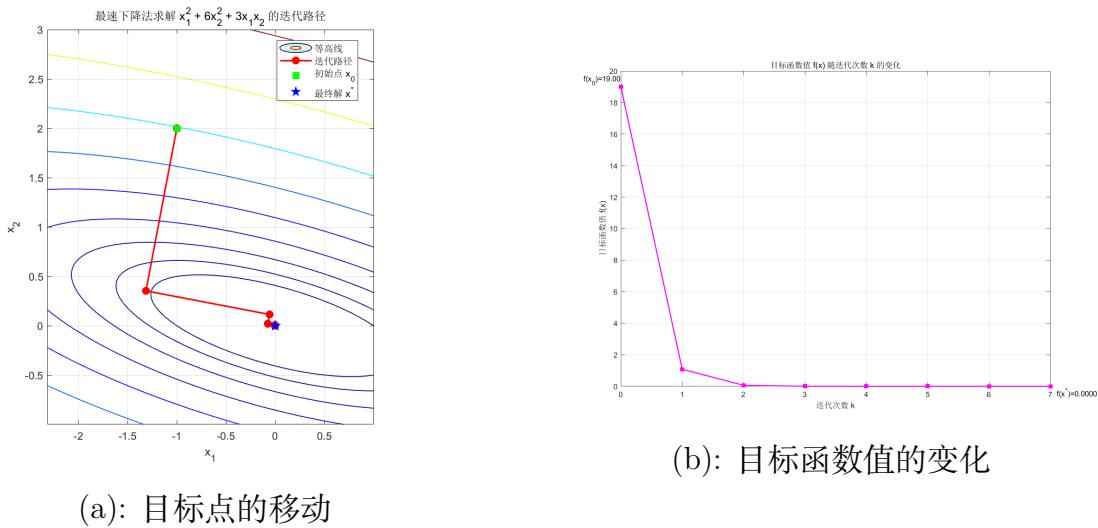


图 1: 迭代过程的可视化图

### 1.3 求解结果

#### 命令行窗口

```
--- 开始最速下降法迭代 ---
迭代 0: x = (-1.0000, 2.0000), f(x) = 19.0000, ||g(x)|| = 21.377558
迭代 1: x = (-1.3137, 0.3533), f(x) = 1.0823, ||g(x)|| = 1.595614
迭代 2: x = (-0.0570, 0.1139), f(x) = 0.0616, ||g(x)|| = 1.217705
迭代 3: x = (-0.0748, 0.0201), f(x) = 0.0035, ||g(x)|| = 0.090889
迭代 4: x = (-0.0032, 0.0065), f(x) = 0.0002, ||g(x)|| = 0.069363
迭代 5: x = (-0.0043, 0.0011), f(x) = 0.0000, ||g(x)|| = 0.005177
迭代 6: x = (-0.0002, 0.0004), f(x) = 0.0000, ||g(x)|| = 0.003951
迭代 7: x = (-0.0002, 0.0001), f(x) = 0.0000, ||g(x)|| = 0.000295
```

=====

最速下降法结果:

最优解  $x^*$  为: (-0.00024279, 6.5296e-05)

此时的最优函数值  $f(x^*)$  为: 3.6969e-08

图 2: 问题一 Matlab 运行结果

所以最优解  $x^* = (0, 0)$ , 最小函数值  $f(x^*) = 0$ 。

## 2 第二题

### 2.1 问题分析

目标函数  $f(x) = x_1^4 + x_2^2 + 3x_1x_2$  的梯度向量  $\nabla f(x)$  为:

$$\nabla f(x) = \begin{bmatrix} 4x_1^3 + 3x_2 \\ 2x_2 + 3x_1 \end{bmatrix} \quad (4)$$

Hesse 矩阵  $B(x)$  为:

$$B(x) = \nabla^2 f(x) = \begin{bmatrix} 12x_1^2 & 3 \\ 3 & 2 \end{bmatrix} \quad (5)$$

算法迭代通过求解信赖域子问题并根据下降比  $r_k$  动态调整信赖域半径  $\Delta_k$  实现。

在第  $k$  步, 求解子问题:

$$\min_{d \in \mathbb{R}^2} m_k(d) = f(x_k) + g_k^T d + \frac{1}{2} d^T B_k d \quad s.t. \quad \|d\| \leq \Delta_k \quad (6)$$

### 2.2 详细求解过程

#### 2.2.1 子问题求解策略 (牛顿方向截断)

采用以下近似策略确定步长  $d_k$ :

1. 计算全牛顿步  $d_{full} = -B_k^{-1}g_k$ 。

2. 步长选择  $d_k$ :

$$d_k = \begin{cases} d_{full} & \text{if } \|d_{full}\| \leq \Delta_k \\ \frac{\Delta_k}{\|d_{full}\|} d_{full} & \text{if } \|d_{full}\| > \Delta_k \end{cases} \quad (7)$$

**半径调整与迭代更新:**

下降比  $r_k$  用于衡量近似模型质量:

$$r_k = \frac{A_{red_k}}{P_{red_k}} = \frac{f(x_k) - f(x_k + d_k)}{-(g_k^T d_k + \frac{1}{2} d_k^T B_k d_k)} \quad (8)$$

**更新规则 ( $\eta = 0.1$ ):**

- **接受步长 & 扩大半径:** 若  $r_k > 0.75$  且  $\|d_k\| = \Delta_k$ , 则  $\Delta_{k+1} = \min(2\Delta_k, \hat{\Delta})$ , 且  $x_{k+1} = x_k + d_k$ 。
- **接受步长 & 半径不变:** 若  $0.25 \leq r_k \leq 0.75$  或  $r_k > 0.75$  且  $\|d_k\| < \Delta_k$ , 则  $\Delta_{k+1} = \Delta_k$ , 且  $x_{k+1} = x_k + d_k$ 。
- **缩小半径 & 拒绝步长:** 若  $r_k < 0.25$ , 则  $\Delta_{k+1} = 0.25\Delta_k$ , 且  $x_{k+1} = x_k$ 。
- **接受新点条件:** 要求  $r_k > \eta = 0.1$ 。

完整代码见附录 A.2。

### 2.2.2 迭代过程的可视化分析

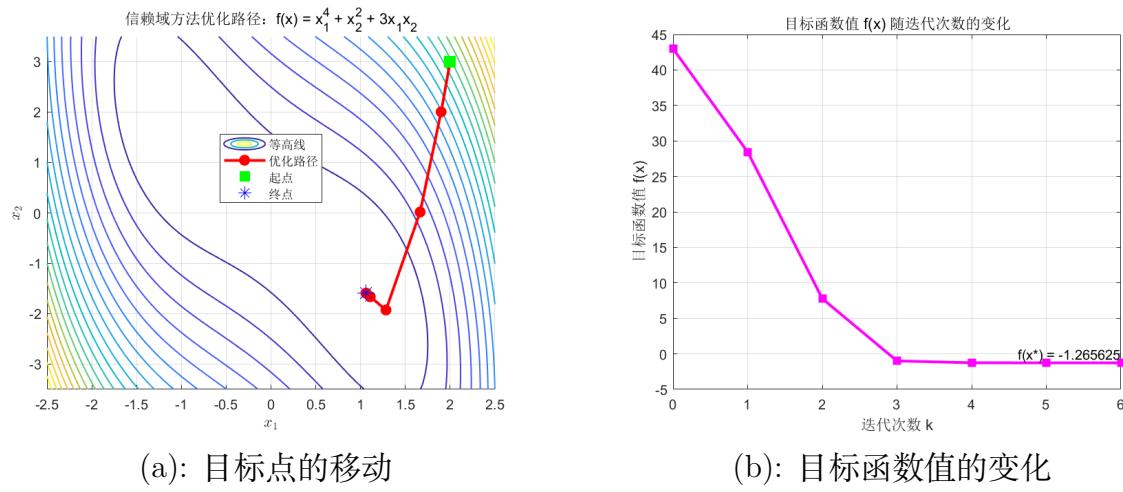


图 3: 迭代过程的可视化图

### 2.3 求解结果

命令行窗口						
--- 信赖域方法优化过程 ---						
Iter	x1	x2	f (x)	g	Delta	
0	2.0000	3.0000	43.0000	42.7200	1.0000	
1	1.8990	2.0051	28.4475	34.7887	2.0000	
2	1.6649	0.0189	7.7779	19.1879	2.0000	
3	1.2836	-1.9254	-0.9925	2.6831	2.0000	
4	1.1079	-1.6618	-1.2551	0.4538	2.0000	
5	1.0635	-1.5953	-1.2656	0.0258	2.0000	
6	1.0607	-1.5910	-1.2656	0.0001	2.0000	

最终解:  $x^* = (1.060672, -1.591007)$   
目标函数值:  $f(x^*) = -1.265625$

图 4: 问题二 Matlab 运行结果

所以最优解  $x^* = (1.060672, -1.591007)$ , 最小函数值  $f(x^*) = -1.265625$ 。

### 3 第三题

#### 3.1 问题分析

目标函数  $f(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2$  的梯度向量  $\nabla f(\mathbf{x})$  和 Hesse 矩阵  $\mathbf{H}(\mathbf{x})$ :

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 400x_1(x_1^2 - x_2) + 2(x_1 - 1) \\ -200(x_1^2 - x_2) \end{bmatrix} \quad (9)$$

$$\mathbf{H}(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix} \quad (10)$$

牛顿法的核心在于利用目标函数的二次近似模型:

$$m_k(\mathbf{d}) = f(\mathbf{x}_k) + \mathbf{d}^T \mathbf{g}_k + \frac{1}{2} \mathbf{d}^T \mathbf{H}_k \mathbf{d} \quad (11)$$

通过求  $m_k(\mathbf{d})$  的最小值, 得到牛顿步  $\mathbf{d}_k = -\mathbf{H}_k^{-1} \mathbf{g}_k$ 。步长恒取 1, 无线搜索。

#### 3.2 详细求解过程

##### 3.2.1 迭代过程的伪代码

---

##### Algorithm 2 牛顿最优化法

---

**Require:** 目标函数  $f(x)$ , 初始点  $x^{(0)}$ , 精度  $\epsilon = 0.001$ , 最大迭代次数  $K_{max}$

**Ensure:** 最优解  $x^*$

```

1:  $k \leftarrow 0, x \leftarrow x^{(0)}$ 
2: while  $k < K_{max}$  do
3:    $g \leftarrow \nabla f(x), H \leftarrow \nabla^2 f(x)$             $\triangleright$  计算梯度  $g$  和 Hesse 矩阵  $H$ 
4:   if  $\|g\| \leq \epsilon$  then                       $\triangleright$  检查梯度模长是否满足收敛条件
5:     break                                      $\triangleright$  跳出循环
6:   end if
7:   if  $H$  奇异或接近奇异 then           $\triangleright$  求解牛顿方向  $d$ :  $Hd = -g$ 
8:      $H_{reg} \leftarrow H + \delta I$                    $\triangleright$  应用 Tikhonov 正则化 ( $\delta$  很小, 例如  $10^{-3}$ )
9:      $d \leftarrow H_{reg}^{-1}(-g)$ 
10:  else
11:     $d \leftarrow H^{-1}(-g)$ 
12:  end if
13:   $x \leftarrow x + d, k \leftarrow k + 1$             $\triangleright$  使用单位步长  $\alpha = 1$  更新解
14: end while
15: return  $x$ 

```

---

完整代码见附录 A.3。

### 3.2.2 迭代过程的可视化分析

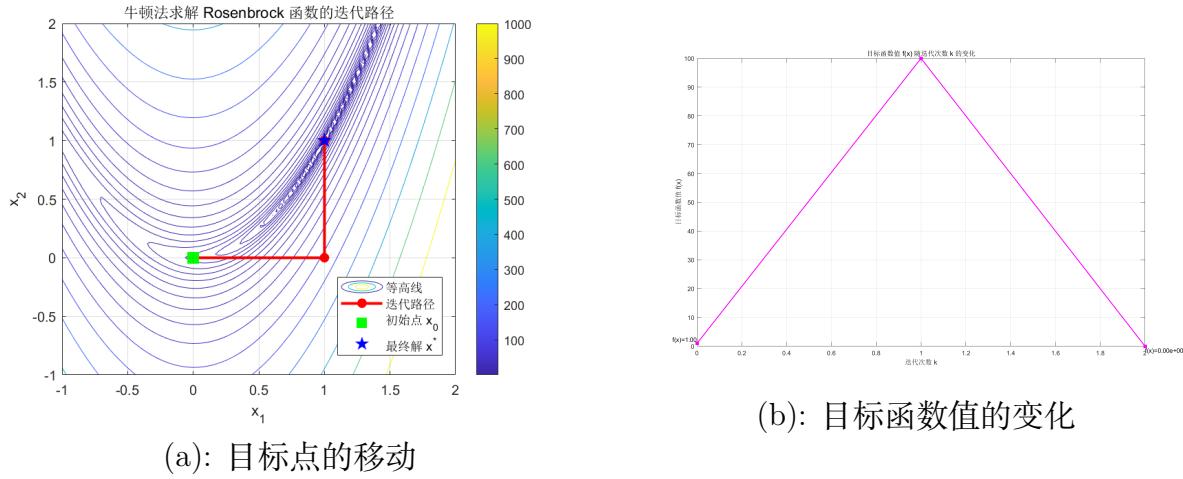


图 5: 迭代过程的可视化图

### 3.3 求解结果

```
命令行窗口
--- 开始牛顿法迭代 ---
迭代 0: x = (0.0000, 0.0000), f(x) = 1.0000, ||g(x)|| = 2.000000
迭代 1: x = (1.0000, 0.0000), f(x) = 100.0000, ||g(x)|| = 447.213595
迭代 2: x = (1.0000, 1.0000), f(x) = 0.0000, ||g(x)|| = 0.000000

==== 最终结果 ====
最优解 x*: (1.0000, 1.0000)
最优值 f(x*): 0.000000e+00
```

图 6: 问题三 Matlab 运行结果

所以最优解  $x^* = (1.0000, 1.0000)$ , 最小函数值  $f(x^*) = 0$ 。  
由目标函数  $f(\mathbf{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 \geq 0$  我们也可以看出全局极小值点  $\mathbf{x}^* = (1, 1)$ , 在此点  $f(\mathbf{x}^*) = 0$  且  $\nabla f(\mathbf{x}^*) = \mathbf{0}$ 。

## 4 第四题

### 4.1 问题分析

该无约束最小化问题  $\min_{\mathbf{x} \in \mathbb{R}^2} f(\mathbf{x}) = 2x_1^2 + 4x_2^2 + 5x_1x_2 - x_2$  形式为严格凸二次规划:

$$\min_{\mathbf{x} \in \mathbb{R}^2} \frac{1}{2} \mathbf{x}^\top A \mathbf{x} - \mathbf{b}^\top \mathbf{x} \quad (12)$$

其中，矩阵  $A$  和向量  $b$  分别为：

$$A = \begin{bmatrix} 4 & 5 \\ 5 & 8 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (13)$$

最优解  $x^*$  满足最优性条件  $\nabla f(x^*) = Ax^* - b = 0$ ，即  $Ax^* = b$ 。

## 4.2 详细求解过程

### 4.2.1 线性共轭梯度法迭代公式

迭代过程基于以下公式：

1. **梯度:**  $g^{(k)} = Ax^{(k)} - b$
2. **步长 (精确线搜索):**  $\alpha_k = \frac{g^{(k)\top} g^{(k)}}{d^{(k)\top} Ad^{(k)}}$
3. **更新点:**  $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$
4. **共轭系数 (FR 公式):**  $\beta_k = \frac{g^{(k+1)\top} g^{(k+1)}}{g^{(k)\top} g^{(k)}}$
5. **搜索方向:**  $d^{(k+1)} = -g^{(k+1)} + \beta_k d^{(k)}$

完整代码见附录 A.4。

### 4.2.2 迭代过程的可视化分析

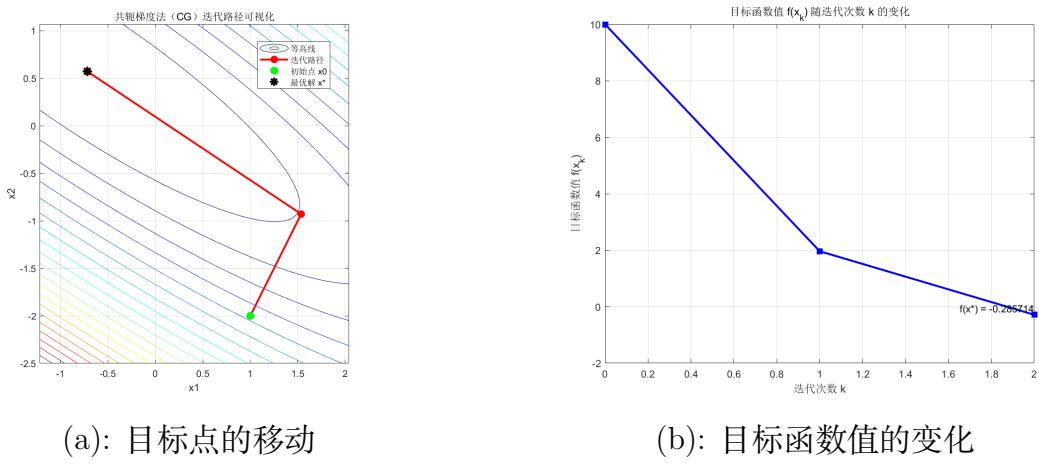


图 7: 迭代过程的可视化图

### 4.3 求解结果

```
命令行窗口
--- 开始共轭梯度法迭代 ---
k          x1          x2      ||g_k||
0      1.000000    -2.000000  1.341641e+01
1      1.535714    -0.928571  1.677051e+00
2     -0.714286     0.571429  1.848180e-14

最优解 x* = (-0.714286, 0.571429)
最小函数值 f(x*) = -0.285714
```

图 8: 问题四 Matlab 运行结果

所以最优解  $x^* = (-0.714286, 0.571429)$ , 最小函数值  $f(x^*) = -0.285714$ 。

## 5 前半学期课程总结

### 5.1 知识点总结与分析

#### 5.1.1 迭代优化的基础框架

##### 1. 线搜索方法

- **迭代公式:**  $x_{k+1} = x_k + \alpha_k d_k$ 。
- **步骤分解:** 首先确定一个下降方向  $d_k$  (满足  $d_k^T \nabla f(x_k) < 0$ ), 然后确定一个步长  $\alpha_k > 0$ 。
- **步长确定:**
  - **精确线搜索:** 求解  $\alpha_k = \arg \min_{\alpha > 0} f(x_k + \alpha d_k)$ 。
  - **非精确线搜索:** 寻求一个“满意”的下降步长。常用的规则包括 Armijo、Goldstein 和 Wolfe 规则。其中, Wolfe 规则在理论上和数值上都更稳定。

##### 2. 信赖域方法

- **核心思想:** 与线搜索不同, 信赖域方法是同时选择方向和步长。

- **子问题:** 求解目标函数  $f(x)$  在  $x_k$  处的二阶近似模型  $m_k(d)$  在一个有界区域（信赖域） $\|d\| \leq \Delta_k$  内的极小值点  $d_k$ :

$$\min_{d \in \mathbb{R}^n} m_k(d) = f(x_k) + g_k^T d + \frac{1}{2} d^T B_k d, \quad \text{s.t.} \quad \|d\| \leq \Delta_k$$

- **信赖域半径更新:** 根据实际下降量  $f(x_k) - f(x_{k+1})$  与模型预测下降量  $m_k(0) - m_k(d_k)$  的比值  $\rho_k$  来动态调整信赖域半径  $\Delta_k$ , 以控制模型近似的准确性。

### 5.1.2 经典下降算法

#### 1. 最速下降法

- **原理:** 沿负梯度方向  $-\nabla f(x_k)$  搜索, 这是局部下降最快的方向。
- **特点:** 算法简单, 对目标函数要求低。但收敛速度通常只有线性收敛, 且在目标函数等值线扁平时, 相邻搜索方向近乎正交, 导致“锯齿现象”或低效的“之”字形迭代路径。

#### 2. 牛顿法

- **原理:** 利用目标函数在当前点的二阶泰勒展开来近似目标函数, 并直接求解近似函数的极小值点。
- **搜索方向:**  $d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$ 。
- **特点:** 在最优解附近具有二次收敛速度, 收敛极快。缺点是需要计算、存储和求逆或求解 Hesse 矩阵  $\nabla^2 f(x_k)$  相关的线性系统, 计算量大; 当 Hesse 矩阵不正定时, 方向可能不是下降方向, 算法不稳定。

### 5.1.3 高效二阶近似算法

#### 1. 拟牛顿法

- **研究动机:** 结合最速下降法的简单性和牛顿法的快速收敛性。
- **核心思想:** 用一个易于计算、维护的矩阵  $B_k$  来近似 Hessian 矩阵  $\nabla^2 f(x_k)$ 。
- **拟牛顿条件 (割线方程):**  $B_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$ 。
- **校正公式:** 常见的有 DFP 和 BFGS 秩二校正公式。BFGS 在数值实验中通常表现最佳, 它具有超线性收敛速度。

#### 2. 共轭梯度法

- **原理:** 最初用于求解对称正定线性方程组  $Ax = b$ , 通过构造一组  $A$ -共轭方向  $d_k$  进行搜索, 保证不走回头路。

- **线性 CG:** 对二次函数在至多  $n$  步内收敛到精确解（二次终止性）。
- **非线性 CG:** 推广至一般非线性函数。搜索方向  $d_{k+1} = -g_{k+1} + \beta_k d_k$ , 通过  $\beta_k$  的不同计算公式区分，如 Fletcher-Reeves (FR) 和 Polak-Ribiere (PR) 公式。PR 公式数值效果通常更优。

#### 5.1.4 特殊结构优化方法

##### 1. 最小二乘问题

- **数学模型:** 最小化残量函数的平方和:  $\min_{x \in \mathbb{R}^n} \frac{1}{2} \|r(x)\|^2$ 。
- **线性最小二乘:**  $r(x) = Ax - b$ 。最优解由正规方程  $A^T A x = A^T b$  给出。
- **非线性最小二乘:**
  - **Gauss-Newton 法:** 利用一阶泰勒展开近似  $r(x)$ , 每一步都求解一个线性最小二乘子问题。
  - **Levenberg-Marquardt (LM) 法:** 对 Gauss-Newton 法的改进。在子问题中加入了正则项，平衡了 Gauss-Newton 方向和最速下降方向，尤其适用于初始点远离最优解时。

##### 2. 交替极小化方法

- **适用问题:** 变量可分为  $s$  个块的优化问题  $\min_{x_1, \dots, x_s} \Psi(x_1, \dots, x_s)$ 。
- **核心思想:** 固定其他  $s - 1$  个块变量，仅对剩余一个块变量进行最小化，然后循环交替。
- **特点:** 通过“化整为零”简化了优化子问题，适用于具有和式或块结构的凸优化问题，例如矩阵分解等。

#### 5.2 个人角度的课程难点

- **理解和证明:** 相比于将算法应用于例题，对各种算法的收敛性条件的理解和证明过程是难点。例如，精确理解 Wolfe 准则中两个条件（充分下降和曲率条件）的几何意义及其对保证收敛的重要性；信赖域方法中 Cauchy 点的下降量估计在证明全局收敛性中的作用。
- **算法的代码实现:** 在将理论算法转化为代码时，存在许多挑战。例如，信赖域方法子问题的代码实现，如何确定步长  $d_k$ 。
- **复杂算法的参数选择:** 信赖域方法、拟牛顿法和 Levenberg-Marquardt 法中存在关键参数（如拟牛顿法中的  $\gamma$ , 信赖域半径  $\Delta_k$ , LM 法中的正则化参数  $\lambda$ ），这些参数的选择会极大影响算法性能。

### 5.3 前半学期的最优化方法在自己其他课程和科研工作中的应用设想

- 在机器学习与深度学习中的应用:

- 梯度下降族算法: 最速下降法是训练神经网络的基础。理解其收敛慢的本质原因有助于理解 Momentum、Adagrad、Adam 等现代优化器如何通过调整梯度方向来加速收敛。
- 拟牛顿法: 在小规模的机器学习任务中, 如训练传统的支持向量机 (SVM) 或逻辑回归模型时, L-BFGS 收敛速度快且对内存友好, 可以作为比小批量随机梯度下降更高效的替代方案。

- 在信号/图像处理与计算机视觉中的应用:

- 交替极小化方法: 广泛应用于涉及矩阵分解 (如推荐系统中的协同过滤)、低秩恢复或一些图像去噪算法中。这些模型的目标函数往往具有分块结构, 通过 AM/块坐标下降 (BCD) 可以将一个复杂的优化问题分解为多个易于求解的子问题。
- 信赖域与 LM 法: 在计算机视觉的定位、三维重建等问题中, 需要求解大规模的非线性最小二乘问题。这些算法在处理非线性度和保证下降方向方面, 提供了比简单梯度下降更优越的稳定性和收敛性。

## 附录

### A 代码

#### A.1 问题一代码

```

1 f = @(x) x(1)^2 + 6*x(2)^2 + 3*x(1)*x(2); % 目标函数
2 grad_f = @(x) [2*x(1) + 3*x(2); 12*x(2) + 3*x(1)]; % 目标函数的梯度
3 x0 = [-1; 2]; % 初始值
4 epsilon = 0.001; % 精度
5 max_iter = 1000; % 最大迭代次数
6
7 x = x0; % 初始点
8 iter = 0;
9 history = zeros(2, max_iter + 1); % 存储迭代历史, 预留一列给x0
10 history(:, 1) = x0;
11
12 fprintf('--- 开始最速下降法迭代 ---\n');
13
14 % 最速下降法主循环

```

```

15 while iter < max_iter
16     grad = grad_f(x); % 计算当前点的梯度
17     p = -grad;          % 搜索方向为负梯度方向
18
19 fprintf('迭代 %d: x = (%.4f, %.4f), f(x) = %.4f, ||g(x)|| = %.6f\n',
20         iter, x(1), x(2), f(x), norm(grad));
21
22 % 检查梯度模长, 满足停止条件则跳出
23 if norm(grad) < epsilon
24     break;
25 end
26
27 % 使用精确线搜索的解析解(针对二次函数)
28 H = [2, 3; 3, 12]; % 此时的 Hesse 矩阵
29 alpha = (p' * p) / (p' * H * p);
30
31 % 更新 x, 使用计算的步长
32 x_new = x + alpha * p;
33
34 x = x_new; % 更新 x, 为下一次迭代做准备
35 iter = iter + 1; % 增加迭代计数
36 history(:, iter + 1) = x; % 记录新的迭代点
37
38 % 截断历史记录
39 history = history(:, 1:iter + 1);
40
41 disp(' ');
42 disp('=====');
43 disp('最速下降法结果:');
44 disp(['最优解 x* 为: (' num2str(x(1)) ', ' num2str(x(2)) ')']);
45 disp(['此时的最优函数值 f(x*) 为: ' num2str(f(x))]);
46 disp('=====');

```

## A.2 问题二代码

```

1 x = [2; 3];           % 初始值
2 epsilon = 0.001;       % 精度要求
3 delta = 1.0;           % 初始信赖域半径
4 max_delta = 2.0;        % 最大信赖域半径
5 eta = 0.1;             % 步长接受阈值
6 max_iter = 1000;        % 最大迭代次数
7 k = 0;
8

```

```

9  % 存储迭代点的历史记录
10 x_history = x;
11 f_history = x(1)^4 + x(2)^2 + 3*x(1)*x(2);
12
13 fprintf('--- 信赖域方法优化过程 ---\n');
14 fprintf(
15   ' Iter \t\t x1 \t\t x2 \t\t f(x) \t\t ||g|| \t\t Delta\n');
16 fprintf(
17   '-----\n');
18 % 目标函数
19 f_val = x(1)^4 + x(2)^2 + 3*x(1)*x(2);
20 % 梯度
21 g = [4*x(1)^3 + 3*x(2); 2*x(2) + 3*x(1)];
22 % Hesse 矩阵
23 B = [12*x(1)^2, 3; 3, 2];
24 fprintf(' %d \t %.4f \t %.4f \t %.4f \t %.4f \t %.4f\n', k, x(1), x(2),
25   f_val, norm(g), delta);
26
27 while norm(g) > epsilon && k < max_iter
28   % 求解信赖域子问题: min m(d) s.t. ||d|| <= delta
29   % 采用 Dogleg/全牛顿步的简单变体: 计算全牛顿步 d_full, 若超出信赖域则简
30   % 单缩放
31   d_full = -B \ g;
32
33   if norm(d_full) <= delta
34     d = d_full;
35     is_boundary = false; % 步长在信赖域内部
36   else
37     % 当牛顿步超出信赖域时, 沿着牛顿方向进行截断
38     d = (delta / norm(d_full)) * d_full;
39     is_boundary = true; % 步长达到信赖域边界
40   end
41
42   % 计算比值 r_k (实际下降量 / 预测下降量)
43   % 预测下降量
44   pred_red = -(g' * d + 0.5 * d' * B * d);
45
46   % 计算新点函数值
47   x1 = x + d;
48   f_new = x1(1)^4 + x1(2)^2 + 3*x1(1)*x1(2);
49
50   % 实际下降量
51   actual_red = f_val - f_new;

```

```
50
51 % 比值 r
52 if pred_red > 0
53     r = actual_red / pred_red;
54 else
55     % 预测下降量非正，说明近似模型有问题，直接缩小信赖域
56     r = -1;
57 end

58
59 % 更新信赖域半径 delta
60 if r < 0.25
61     delta = 0.25 * delta; % 实际下降远小于预测，缩小半径
62 elseif r > 0.75 && is_boundary % 实际下降与预测吻合，且步长达到边界，考虑扩大半径
63     delta = min(2 * delta, max_delta);
64 end

65
66 % 决定是否更新点位
67 if r > eta
68     x = x + d; % 接受新点
69     x_history = [x_history, x]; % 存储新点
70     f_history = [f_history, f_new];
71 end

72
73 % 目标函数
74 f_val = x(1)^4 + x(2)^2 + 3*x(1)*x(2);
75 % 梯度
76 g = [4*x(1)^3 + 3*x(2); 2*x(2) + 3*x(1)];
77 % Hesse 矩阵
78 B = [12*x(1)^2, 3; 3, 2];

79
80 k = k + 1;
81 % 打印迭代信息
82 fprintf(' %d \t %.4f \t %.4f \t %.4f \t %.4f \t %.4f\n', k, x(1), x(2), f_val, norm(g), delta);
83 end

84
85 % 结果输出
86 final_f = x(1)^4 + x(2)^2 + 3*x(1)*x(2);
87 fprintf('
88     -----\n');
89 fprintf(' 最终解: x* = (%.6f, %.6f)\n', x(1), x(2));
90 fprintf(' 目标函数值: f(x*) = %.6f\n', final_f);
```

### A.3 问题三代码

```

1 x = [0; 0];           % 初始点
2 epsilon = 0.001;       % 精度要求
3 k = 0;                 % 迭代次数计数器
4 max_iter = 100;        % 最大迭代次数
5 history = struct('x', zeros(2, max_iter), 'f', zeros(1, max_iter), '
6     grad_norm', zeros(1, max_iter));
7
8 fprintf('--- 开始牛顿法迭代 ---\n');
9
10 % 迭代循环
11 while k < max_iter
12     x1 = x(1);
13     x2 = x(2);
14
15     % 计算函数值
16     f = 100*(x1^2 - x2)^2 + (x1 - 1)^2;
17
18     % 计算梯度
19     g1 = 400*x1*(x1^2 - x2) + 2*(x1 - 1);
20     g2 = -200*(x1^2 - x2);
21     g = [g1; g2];
22
23     grad_norm = norm(g); % 梯度向量的欧几里得范数
24
25     fprintf('迭代 %d: x = (%.4f, %.4f), f(x) = %.4f, ||g(x)|| = %.6f\n', k,
26             x1, x2, f, grad_norm);
27
28     k = k + 1;
29     % 记录历史
30     history.x(:, k) = x;
31     history.f(k) = f;
32     history.grad_norm(k) = grad_norm;
33
34     % 检查停止准则
35     if grad_norm <= epsilon
36         break;
37     end
38
39     % 计算 Hesse 矩阵 H
40     H11 = 1200*x1^2 - 400*x2 + 2;
41     H22 = 200;
42     H12 = -400*x1;
43     H = [H11, H12; H12, H22];

```

```

42
43 % 求解牛顿方向 d = -H^-1 * g
44 try
45     step_direction = H \ (-g);
46 catch ME
47     % 捕捉奇异矩阵错误
48     fprintf('警告: Hesse 矩阵奇异或接近奇异。使用修正方向。\\n');
49     % 使用一个小的正则化项确保可逆性
50     H_reg = H + 1e-3 * eye(2);
51     step_direction = H_reg \ (-g);
52 end
53
54 % 更新迭代点
55 x = x + step_direction;
56 end
57
58 % 截断历史记录到实际迭代次数
59 x_opt = x;          % 最优解
60 f_opt = f;          % 最优函数值
61 k_actual = k;
62 history.x = history.x(:, 1:k_actual);
63 history.f = history.f(1:k_actual);
64 history.grad_norm = history.grad_norm(1:k_actual);
65
66 fprintf('\\n==== 最终结果 ===\\n');
67 fprintf('最优解 x*: (%.4f, %.4f)\\n', x_opt(1), x_opt(2));
68 fprintf('最优值 f(x*): %.6e\\n', f_opt);

```

## A.4 问题四代码

```

1 % 构造二次型 f(x) = 0.5*x'*A*x - b'*x
2 A = [4 5; 5 8]; % Hessian (对称正定矩阵)
3 b = [0; 1];      % 线性项向量
4
5 x = [1; -2];    % 初始点 x0
6 eps = 0.001;     % 精度要求
7 maxIter = 1000;  % 最大迭代步数
8
9 % 用于存储迭代点的历史记录
10 x_history = x;
11
12 g = A*x - b;   % 初始梯度
13 d = -g;         % 初始搜索方向
14 k = 0;          % 迭代计数器

```

```
15
16 fprintf('--- 开始共轭梯度法迭代 ---\n');
17 fprintf(' k          x1          x2          ||g_k||\n');
18 fprintf('%2d  %10.6f  %10.6f  %10.6e\n', k, x(1), x(2), norm(g));
19
20 % 迭代过程
21 while norm(g) > eps && k < maxIter
22     % 步长
23     alpha = (g' * g) / (d' * A * d);
24
25     % 更新迭代点
26     x = x + alpha * d;
27
28     % 记录当前迭代点
29     x_history = [x_history, x];
30
31     % 计算新的梯度
32     g_new = A * x - b;
33
34     k = k + 1;
35     fprintf('%2d  %10.6f  %10.6f  %10.6e\n', k, x(1), x(2), norm(g_new));
36
37     % 共轭系数
38     beta = (g_new' * g_new) / (g' * g);
39
40     % 新搜索方向
41     d = -g_new + beta * d;
42
43     % 为下一步迭代做准备
44     g = g_new;
45 end
46
47 % 输出最终结果
48 x_opt = x; % 最优解近似
49 f_opt = 2*x_opt(1)^2 + 4*x_opt(2)^2 + 5*x_opt(1)*x_opt(2) - x_opt(2);
50
51 fprintf('\n最优解 x* = (%.6f, %.6f)\n', x_opt(1), x_opt(2));
52 fprintf('最小函数值 f(x*) = %.6f\n', f_opt);
```