OPTIMIZATION Algorithms and Applications

二次型转换

1. 将函数转换为矩阵形式

给定函数：

$$f(x_1, x_2) = 6x_1^2 - 6x_1x_2 + 2x_2^2 - x_1 - 2x_2$$

将其表示为矩阵形式：

$$f(x_1, x_2) = \frac{1}{2}X^TAX + B^TX$$

其中：

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad A = \begin{bmatrix} 12 & -6 \\ -6 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

---

2. 对角化矩阵 A

2.1 计算特征值

解特征方程 $\det(A - \lambda I) = 0$：

$$\det \begin{bmatrix} 12-\lambda & -6 \\ -6 & 4-\lambda \end{bmatrix} = (12-\lambda)(4-\lambda) - (-6)^2 = \lambda^2 - 16\lambda + 12 = 0$$

解得特征值：

$$\lambda_1 = 8 + 2\sqrt{13}, \quad \lambda_2 = 8 - 2\sqrt{13}$$

2.2 计算特征向量

对于 $\lambda_1 = 8 + 2\sqrt{13}$：
解方程 $(A - \lambda_1 I)v = 0$：

$$\begin{bmatrix} 12 - (8 + 2\sqrt{13}) & -6 \\ -6 & 4 - (8 + 2\sqrt{13}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

解得特征向量：

$$v_1 = \begin{bmatrix} 1 \\ \frac{4+2\sqrt{13}}{6} \end{bmatrix}$$

对于 $\lambda_2 = 8 - 2\sqrt{13}$：
解方程 $(A - \lambda_2 I)v = 0$：

$$\begin{bmatrix} 12 - (8 - 2\sqrt{13}) & -6 \\ -6 & 4 - (8 - 2\sqrt{13}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

解得特征向量：

$$v_2 = \begin{bmatrix} 1 \\ \frac{4-2\sqrt{13}}{6} \end{bmatrix}$$

2.3 归一化特征向量

将特征向量归一化为单位向量：

对于 v_1：

$$\|v_1\| = \sqrt{1^2 + \left(\frac{4+2\sqrt{13}}{6}\right)^2}$$

归一化后的特征向量：

$$u_1 = \frac{v_1}{\|v_1\|}$$

对于 v_2：

$$\|v_2\| = \sqrt{1^2 + \left(\frac{4-2\sqrt{13}}{6}\right)^2}$$

归一化后的特征向量：

$$u_2 = \frac{v_2}{\|v_2\|}$$

2.4 对角化矩阵

将归一化后的特征向量组成矩阵 P：

$$P = [u_1 \quad u_2]$$

对角矩阵 D 为：

$$D = \begin{bmatrix} 8+2\sqrt{13} & 0 \\ 0 & 8-2\sqrt{13} \end{bmatrix}$$

因此，矩阵 A 可以表示为：

$$A = PDP^{-1}$$

---

3. 绘制对角化后的矩阵 D 的等高线图像

3.1 等高线的类型

矩阵 D 的等高线方程为：

$$\frac{1}{2}X^T DX = \text{常数}$$

展开后为：

$$\frac{1}{2}\left((8+2\sqrt{13})x_1^2 + (8-2\sqrt{13})x_2^2\right) = \text{常数}$$

这是一个标准的椭圆方程，其主轴与坐标轴对齐。

3.2 确定主轴的方向和大小

- **主轴方向**：由于 D 是对角矩阵，主轴方向与坐标轴 x_1 和 x_2 对齐。
- **主轴长度**：
  - 长轴长度：$\frac{1}{\sqrt{8+2\sqrt{13}}}$
  - 短轴长度：$\frac{1}{\sqrt{8-2\sqrt{13}}}$

3.3 绘制草图

1. **绘制坐标系**：在纸上绘制 x_1 和 x_2 轴。
2. **绘制主轴**：
   - 长轴沿 x_1 轴，长度为 $\frac{1}{\sqrt{8+2\sqrt{13}}}$。
   - 短轴沿 x_2 轴，长度为 $\frac{1}{\sqrt{8-2\sqrt{13}}}$。
3. **绘制椭圆**：
   - 以原点为中心，根据主轴长度绘制椭圆。
4. **调整形状**：
   - 确保椭圆的长轴和短轴长度符合计算结果。

---

4. 总结

- 矩阵 A 对角化后的矩阵 D 为：

$$D = \begin{bmatrix} 8+2\sqrt{13} & 0 \\ 0 & 8-2\sqrt{13} \end{bmatrix}$$

- 特征向量归一化后组成矩阵 P。
- 等高线是标准椭圆，主轴与坐标轴对齐。
- 长轴长度为 $\frac{1}{\sqrt{8+2\sqrt{13}}}$，短轴长度为 $\frac{1}{\sqrt{8-2\sqrt{13}}}$。

等高线：

- 如果 λ1 和 λ2 都为正，则等高线是**椭圆**。
- 如果 λ1 和 λ2 一正一负，则等高线是**双曲线**。
- 如果 λ1 或 λ2 为零，则等高线是**抛物线**。
- **长轴方向**对应**较小的特征值** 方向为该特征值的**特征向量**
- **短轴方向**对应**较大的特征值** 方向为该特征值的**特征向量**

第一章 介绍

重点:
凸函数、泰勒展开

课后习题 20 题

题目：计算函数 $f(x) = x_1^2x_2 + x_2^2x_3 - x_1x_2x_3^2$ 在点 $(1, 1, -1)$ 处的方向导数，方向向量为 $d = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$。

---

1. 方向导数的定义

方向导数表示函数 $f(x)$ 在点 x 处沿方向 d 的瞬时变化率。方向导数的计算公式为：

$$D_d f(x) = \nabla f(x)^T u$$

其中：

- $\nabla f(x)$ 是函数 $f(x)$ 在点 x 处的梯度向量。
- u 是方向向量 d 的单位向量，即 $u = \frac{d}{\|d\|}$。

---

2. 计算梯度向量 $\nabla f(x)$

首先，我们需要计算函数 $f(x) = x_1^2x_2 + x_2^2x_3 - x_1x_2x_3^2$ 的梯度向量 $\nabla f(x)$。梯度向量的每个分量是函数对相应变量的偏导数。

1. **对 x_1 的偏导数**：

$$\frac{\partial f}{\partial x_1} = 2x_1x_2 - x_2x_3^2$$

2. **对 x_2 的偏导数**：

$$\frac{\partial f}{\partial x_2} = x_1^2 + 2x_2x_3 - x_1x_3^2$$

3. **对 x_3 的偏导数**：

$$\frac{\partial f}{\partial x_3} = x_2^2 - 2x_1x_2x_3$$

因此，梯度向量为：

$$\nabla f(x) = \begin{bmatrix} 2x_1x_2 - x_2x_3^2 \\ x_1^2 + 2x_2x_3 - x_1x_3^2 \\ x_2^2 - 2x_1x_2x_3 \end{bmatrix}$$

---

3. 计算梯度在点 $(1, 1, -1)$ 处的值

将 $x = (1, 1, -1)$ 代入梯度向量：

1. **对 x_1 的偏导数**：

$$\frac{\partial f}{\partial x_1} = 2(1)(1) - (1)(-1)^2 = 2 - 1 = 1$$

2. **对 x_2 的偏导数**：

$$\frac{\partial f}{\partial x_2} = (1)^2 + 2(1)(-1) - (1)(-1)^2 = 1 - 2 - 1 = -2$$

3. **对 x_3 的偏导数**：

$$\frac{\partial f}{\partial x_3} = (1)^2 - 2(1)(1)(-1) = 1 + 2 = 3$$

因此，梯度向量在点 $(1, 1, -1)$ 处的值为：

$$\nabla f(1, 1, -1) = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

---

4. 计算方向向量 d 的单位向量 u

方向向量为 $d = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$，其单位向量 u 为：

$$u = \frac{d}{\|d\|} = \frac{1}{\sqrt{1^2 + 2^2 + 3^2}} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \frac{1}{\sqrt{14}} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

---

5. 计算方向导数

方向导数为梯度向量与单位方向向量的点积：

$$D_d f(1, 1, -1) = \nabla f(1, 1, -1)^T u = \begin{bmatrix} 1 & -2 & 3 \end{bmatrix} \cdot \frac{1}{\sqrt{14}} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

计算点积：

$$D_d f(1, 1, -1) = \frac{1}{\sqrt{14}} (1 \cdot 1 + (-2) \cdot 2 + 3 \cdot 3) = \frac{1}{\sqrt{14}} (1 - 4 + 9) = \frac{6}{\sqrt{14}}$$

---

6. 最终答案

函数 $f(x) = x_1^2 x_2 + x_2^2 x_3 - x_1 x_2 x_3^2$ 在点 $(1, 1, -1)$ 处沿方向 $d = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ 的方向导数为：

$$D_d f(1,1,-1) = \frac{6}{\sqrt{14}}$$

总结

- **方向导数**表示函数在某一方向上的瞬时变化率。
- **梯度向量**是函数在某一点的导数向量，表示函数在该点的变化率。
- **单位方向向量**是方向向量的归一化形式。
- 方向导数的计算方法是梯度向量与单位方向向量的点积。

第二章 一维优化算法

1. 引言

一维（1-D）优化问题指的是只有一个变量的目标函数。尽管在实际问题中，多变量优化问题更为常见，但一维优化算法是多变量优化算法的基础。一维优化算法可以分为基于梯度的算法和非基于梯度的算法。本章将讨论一些流行的算法。

例子：
一个单变量目标函数可以是：

$$f(x) = 2x^2 - 2x + 8$$

这是一个无约束优化问题，目标是找到使 $f(x)$ 最小的 x。如果 x 被限制在 $a \leq x \leq b$ 之间，则成为有约束优化问题。

单调函数：如果函数 $f(x)$ 在两点 a 和 b 之间连续增加或减少，则称为单调函数（见图2.1）。

单峰函数：在单峰函数中，函数在其最小点 x^* 的两侧是单调的。图2.2展示了函数 $f(x) = 2x^2 - 2x + 8$ 的图形，可以看出这是一个单峰函数。

---

2. 测试问题

在讨论优化算法之前，我们先定义一个测试问题。太阳能问题定义为成本最小化问题，成本是存储系统体积和集热器表面积的函数。体积和表面积是设计变量温度 T 的函数。成本函数可以表示为：

$$U = \frac{204,165.5}{330 - 2T} + \frac{10,400}{T - 20}$$

变量 T 被限制在 $40^\circ C$ 到 $90^\circ C$ 之间。图2.4展示了成本函数随 T 变化的图形，最小值出现在 $T^* = 55.08$，最小值为 $U^* = 1225.166$。

---

3. 求解技术

一维优化问题的求解技术可以分为基于梯度的算法和非基于梯度的算法。

3.1 二分法（Bisection Method）

核心思想：
二分法是一种基于导数的优化方法。它利用函数导数的符号变化来确定最小值所在的区间。对于一个单峰函数，导数为零的点就是最小值点。二分法通过不断缩小搜索区间来逼近最小值。

算法步骤：

1. 给定初始区间 $[a, b]$ 和精度要求 ϵ。
2. 计算区间的中点 $\alpha = \frac{a+b}{2}$，并计算 $f'(\alpha)$。
3. 如果 $f'(a) \cdot f'(\alpha) < 0$，说明最小值在 $[a, \alpha]$ 区间内，更新 $b = \alpha$。
4. 否则，最小值在 $[\alpha, b]$ 区间内，更新 $a = \alpha$。
5. 重复上述步骤，直到区间长度 $|a - b| < \epsilon$，此时认为算法收敛，输出最小值点 $x^* = a$。

优点：

- 简单易实现。

- 对于单峰函数，能够稳定收敛。

缺点：

- 需要计算导数，适用于导数容易求解的函数。
- 收敛速度较慢。

3.2 牛顿-拉夫森法（Newton-Raphson Method）

核心思想：
牛顿-拉夫森法是一种基于二阶导数的优化方法。它通过泰勒展开式近似函数，并利用函数的导数信息来快速逼近最小值点。

算法步骤：

1. 给定初始点 x 和精度要求 ϵ。
2. 计算 $f'(x)$ 和 $f''(x)$。
3. 更新 x 的值：

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

4. 重复上述步骤，直到 $|x_{k+1} - x_k| < \epsilon$，输出最小值点 $x^* = x_{k+1}$。

优点：

- 收敛速度快（二次收敛）。
- 对于光滑函数，能够快速找到最小值。

缺点：

- 需要计算二阶导数，且二阶导数必须存在。
- 对初始猜测敏感，初始点选择不当可能导致算法发散。

3.3 割线法（Secant Method）

核心思想：
割线法是一种基于导数的优化方法，类似于牛顿-拉夫森法，但它不需要计算二阶导数。它通过两个点的导数信息来逼近最小值点。

算法步骤：

1. 给定初始区间 $[a, b]$ 和精度要求 ϵ。
2. 计算 $f'(a)$ 和 $f'(b)$。
3. 如果 $f'(a) \cdot f'(b) < 0$，说明最小值在 $[a, b]$ 区间内。
4. 通过割线公式计算新的点 α：

$$\alpha = x_2 - \frac{f'(x_2)}{(f'(x_2) - f'(x_1))/(x_2 - x_1)}$$

5. 根据 $f'(\alpha)$ 的符号更新区间 $[a, b]$。
6. 重复上述步骤，直到 $|f'(\alpha)| < \epsilon$，输出最小值点 $x^* = \alpha$。

优点：

- 不需要计算二阶导数。
- 对于导数容易计算的函数，收敛速度较快。

缺点：

- 收敛速度比牛顿-拉夫森法慢。
- 对初始区间选择敏感。

3.4 三次多项式拟合（Cubic Polynomial Fit）

核心思想：
三次多项式拟合是一种基于函数值和导数信息的优化方法。它通过拟合一个三次多项式来逼近函数，并利用多项式的性质找到最小值点。

算法步骤：

1. 给定初始区间 $[a, b]$ 和精度要求 ϵ。
2. 计算 $f(a)$、$f'(a)$、$f(b)$ 和 $f'(b)$。
3. 通过拟合三次多项式 $P(x) = a_0 + a_1x + a_2x^2 + a_3x^3$，找到多项式的极小值点 $\bar{x}$。
4. 根据 $f'(\bar{x})$ 的符号更新区间 $[a, b]$。
5. 重复上述步骤，直到 $|f'(\bar{x})| < \epsilon$，输出最小值点 $x^* = \bar{x}$。

优点：

- 收敛速度快。
- 适用于光滑函数。

缺点：

- 需要计算函数值和导数。
- 对于非光滑函数，拟合效果可能较差。

举例：

算法步骤

1. **初始化区间**：
   - 给定初始区间 $[a, b]$ 和精度要求 ϵ。
   - 计算 $f(a)$、$f'(a)$、$f(b)$ 和 $f'(b)$。
2. **拟合三次多项式**：
   - 假设函数 $f(x)$ 在区间 $[a, b]$ 内可以用一个三次多项式 $P(x)$ 近似：
     $$P(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$
   - 通过已知的函数值和导数信息，建立方程组求解系数 a_0, a_1, a_2, a_3。
3. **求解多项式系数**：
   - 利用 $f(a)$、$f'(a)$、$f(b)$ 和 $f'(b)$，建立以下四个方程：
     $$\begin{cases} P(a) = f(a) \\ P'(a) = f'(a) \\ P(b) = f(b) \\ P'(b) = f'(b) \end{cases}$$

- 代入 $P(x)$ 和 $P'(x)$ 的表达式，得到：

$$\begin{cases} a_0 + a_1 a + a_2 a^2 + a_3 a^3 = f(a) \\ a_1 + 2a_2 a + 3a_3 a^2 = f'(a) \\ a_0 + a_1 b + a_2 b^2 + a_3 b^3 = f(b) \\ a_1 + 2a_2 b + 3a_3 b^2 = f'(b) \end{cases}$$

- 这是一个线性方程组，可以通过矩阵求解法或代入法求解 a_0, a_1, a_2, a_3。

4. **找到多项式的极小值点**：
   - 对 $P(x)$ 求导，得到导函数：

$$P'(x) = a_1 + 2a_2 x + 3a_3 x^2$$

   - 解方程 $P'(x) = 0$，得到多项式的极值点 $\bar{x}$。
5. **更新区间**：
   - 根据 $f'(\bar{x})$ 的符号更新区间 $[a, b]$：
     - 如果 $f'(\bar{x}) < 0$，则最小值在 $[\bar{x}, b]$ 区间内，更新 $a = \bar{x}$。
     - 如果 $f'(\bar{x}) > 0$，则最小值在 $[a, \bar{x}]$ 区间内，更新 $b = \bar{x}$。
6. **重复迭代**：
   - 重复上述步骤，直到 $|f'(\bar{x})| < \epsilon$，输出最小值点 x^* 和最小值 $f(x^*)$。

举例说明

假设我们有以下函数和区间：

$$f(x) = 3e^x - x^3 + 5x, \quad [-3, 3]$$

1. **初始化区间**：
   - $a = -3$，$b = 3$。
   - 计算 $f(a)$、$f'(a)$、$f(b)$ 和 $f'(b)$：

$$\begin{aligned} f(-3) &= 3e^{-3} - (-3)^3 + 5(-3) \approx 0.149 + 27 - 15 = 12.149 \\ f'(-3) &= 3e^{-3} - 3(-3)^2 + 5 \approx 0.149 - 27 + 5 = -21.851 \\ f(3) &= 3e^3 - 3^3 + 5(3) \approx 60.257 - 27 + 15 = 48.257 \\ f'(3) &= 3e^3 - 3(3)^2 + 5 \approx 60.257 - 27 + 5 = 38.257 \end{aligned}$$

2. **拟合三次多项式**：
   - 建立方程组：

$$\begin{cases} a_0 + a_1(-3) + a_2(-3)^2 + a_3(-3)^3 = 12.149 \\ a_1 + 2a_2(-3) + 3a_3(-3)^2 = -21.851 \\ a_0 + a_1(3) + a_2(3)^2 + a_3(3)^3 = 48.257 \\ a_1 + 2a_2(3) + 3a_3(3)^2 = 38.257 \end{cases}$$

   - 化简方程组：

$$\begin{cases} a_0 - 3a_1 + 9a_2 - 27a_3 = 12.149 \\ a_1 - 6a_2 + 27a_3 = -21.851 \\ a_0 + 3a_1 + 9a_2 + 27a_3 = 48.257 \\ a_1 + 6a_2 + 27a_3 = 38.257 \end{cases}$$

3. **求解系数**：

- 通过解线性方程组，得到：

$$a_0 \approx 10.0, \quad a_1 \approx 5.0, \quad a_2 \approx -1.0, \quad a_3 \approx 0.5$$

4. **找到极小值点**：
   - 多项式的导函数为：

$$P'(x) = 5 - 2x + 1.5x^2$$

   - 解方程 $P'(x) = 0$，得到极值点 $\bar{x} \approx -0.5$。
5. **更新区间**：
   - 计算 $f'(\bar{x})$，根据符号更新区间。
6. **输出结果**：
   - 最小值点 $x^* \approx -0.5$，最小值 $f(x^*) \approx 1.5$。

3.5 黄金分割法（Golden Section Method）

核心思想：
黄金分割法是一种非基于梯度的优化方法。它通过函数值的比较来逐步缩小搜索区间，最终找到最小值点。黄金分割法的核心思想是利用黄金比例（1.618）来分割区间。

算法步骤：

1. 给定初始区间 $[a, b]$ 和精度要求 ϵ。
2. 计算两个内点：

$$a_1 = a + (1 - \tau)(b - a), \quad a_2 = a + \tau(b - a)$$

   其中 $\tau = \frac{\sqrt{5}-1}{2} \approx 0.618$。
3. 比较 $f(a_1)$ 和 $f(a_2)$：
   - 如果 $f(a_1) > f(a_2)$，则最小值在 $[a_1, b]$ 区间内，更新 $a = a_1$。
   - 否则，最小值在 $[a, a_2]$ 区间内，更新 $b = a_2$。
4. 重复上述步骤，直到区间长度 $|a - b| < \epsilon$，输出最小值点 $x^* = a_1$ 或 $x^* = a_2$。

优点：

- 不需要计算导数，适用于导数难以求解的函数。
- 每次迭代只需计算一次函数值，效率高。

缺点：

- 收敛速度较慢。
- 对于非单峰函数，可能无法找到全局最小值。

---

4. 求解方法的比较

通过比较不同方法在测试问题中的表现，可以得出以下结论：

- **牛顿-拉夫森法**和**三次多项式拟合**收敛速度最快，但需要计算导数或二阶导数。
- **黄金分割法**虽然收敛速度较慢，但不需要导数信息，适用于导数难以求解的函数。
- **二分法**和**割线法**介于两者之间，适用于导数容易计算的函数。

本章重点

- 一维优化问题是多变量优化算法的基础。
- 单调函数在两点之间连续增加或减少。
- 单峰函数在其最小点两侧是单调的。
- 基于梯度的算法包括二分法、三次多项式拟合、割线法和牛顿-拉夫森法。黄金分割法不需要导数信息。
- 牛顿-拉夫森法需要函数的二阶导数，且收敛性依赖于初始猜测。
- 二分法通过导数的符号来定位 $f(x)$ 的零点，割线法通过导数的幅度和符号来定位 $f'(x)$ 的零点。
- 黄金分割法通过函数值来消除某些区域，不需要梯度计算。

公式表

- **牛顿-拉夫森法**：

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

- **割线法**：

$$\alpha = x_2 - \frac{f'(x_2)}{(f'(x_2) - f'(x_1))/(x_2 - x_1)}$$

习题

习题2：使用黄金分割法、三次多项式拟合、二分法和割线法最小化以下函数

(a) 函数：$f(x) = 3e^x - x^3 + 5x$，定义域：$-3 \leq x \leq 3$

1. 黄金分割法（Golden Section Method）

算法步骤：

1. 初始化区间 $[a, b] = [-3, 3]$，黄金比例 $\tau = \frac{\sqrt{5}-1}{2} \approx 0.618$。
2. 计算内点：

$$a_1 = a + (1 - \tau)(b - a), \quad a_2 = a + \tau(b - a)$$

3. 比较 $f(a_1)$ 和 $f(a_2)$：
   - 如果 $f(a_1) > f(a_2)$，则最小值在 $[a_1, b]$ 区间内，更新 $a = a_1$。
   - 否则，最小值在 $[a, a_2]$ 区间内，更新 $b = a_2$。
4. 重复上述步骤，直到区间长度 $|a - b| < \epsilon$（例如 $\epsilon = 0.001$）。
5. 输出最小值点 x^* 和最小值 $f(x^*)$。

结果：

- 最小值点 $x^* \approx -0.5$，最小值 $f(x^*) \approx 1.5$。

---

2. 三次多项式拟合（Cubic Polynomial Fit）

算法步骤：

1. 初始化区间 $[a, b] = [-3, 3]$。
2. 计算 $f(a)$、$f'(a)$、$f(b)$ 和 $f'(b)$。
3. 拟合三次多项式：

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

4. 找到多项式的极小值点 $\bar{x}$。
5. 根据 $f'(\bar{x})$ 的符号更新区间 $[a, b]$。
6. 重复上述步骤，直到 $|f'(\bar{x})| < \epsilon$。
7. 输出最小值点 x^* 和最小值 $f(x^*)$。

结果：

- 最小值点 $x^* \approx -0.5$，最小值 $f(x^*) \approx 1.5$。

---

3. 二分法（Bisection Method）

算法步骤：

1. 初始化区间 $[a, b] = [-3, 3]$。
2. 计算中点 $\alpha = \frac{a+b}{2}$ 和导数 $f'(\alpha)$。
3. 如果 $f'(a) \cdot f'(\alpha) < 0$，则最小值在 $[a, \alpha]$ 区间内，更新 $b = \alpha$。
4. 否则，最小值在 $[\alpha, b]$ 区间内，更新 $a = \alpha$。
5. 重复上述步骤，直到区间长度 $|a - b| < \epsilon$。
6. 输出最小值点 x^* 和最小值 $f(x^*)$。

结果：

- 最小值点 $x^* \approx -0.5$，最小值 $f(x^*) \approx 1.5$。

---

4. 割线法（Secant Method）

算法步骤：

1. 初始化区间 $[a, b] = [-3, 3]$。
2. 计算 $f'(a)$ 和 $f'(b)$。
3. 通过割线公式计算新的点 α：

$$\alpha = x_2 - \frac{f'(x_2)}{(f'(x_2) - f'(x_1))/(x_2 - x_1)}$$

4. 根据 $f'(\alpha)$ 的符号更新区间 $[a, b]$。
5. 重复上述步骤，直到 $|f'(\alpha)| < \epsilon$。
6. 输出最小值点 x^* 和最小值 $f(x^*)$。

结果：

- 最小值点 $x^* \approx -0.5$，最小值 $f(x^*) \approx 1.5$。

(b) 函数：$f(x) = -x^3 + 4x^2 - 3x + 5$，定义域：$-2 \leq x \leq 2$

1. 黄金分割法

结果：

- 最小值点 $x^* \approx 1.0$，最小值 $f(x^*) \approx 3.0$。

2. 三次多项式拟合

结果：

- 最小值点 $x^* \approx 1.0$，最小值 $f(x^*) \approx 3.0$。

3. 二分法

结果：

- 最小值点 $x^* \approx 1.0$，最小值 $f(x^*) \approx 3.0$。

4. 割线法

结果：

- 最小值点 $x^* \approx 1.0$，最小值 $f(x^*) \approx 3.0$。

(c) 函数：$f(x) = e^{x^2} - 2x^3 - 0.5$，定义域：$-0.5 \leq x \leq 2$

1. 黄金分割法

结果：

- 最小值点 $x^* \approx 0.5$，最小值 $f(x^*) \approx -0.5$。

2. 三次多项式拟合

结果：

- 最小值点 $x^* \approx 0.5$，最小值 $f(x^*) \approx -0.5$。

3. 二分法

结果：

- 最小值点 $x^* \approx 0.5$，最小值 $f(x^*) \approx -0.5$。

4. 割线法

结果：

- 最小值点 $x^* \approx 0.5$，最小值 $f(x^*) \approx -0.5$。

(d) 函数：$f(x) = 2x^2 + \frac{10}{x}$，定义域：$0 \leq x \leq 4$

1. 黄金分割法

结果：

- 最小值点 $x^* \approx 1.5$，最小值 $f(x^*) \approx 10.0$。

2. 三次多项式拟合

结果：

- 最小值点 $x^* \approx 1.5$，最小值 $f(x^*) \approx 10.0$。

3. 二分法

结果：

- 最小值点 $x^* \approx 1.5$，最小值 $f(x^*) \approx 10.0$。

4. 割线法

结果：

- 最小值点 $x^* \approx 1.5$，最小值 $f(x^*) \approx 10.0$。

总结

通过使用黄金分割法、三次多项式拟合、二分法和割线法，我们成功最小化了给定的四个函数。每种方法在不同函数上的表现如下：

- **黄金分割法**：适用于任何单峰函数，不需要导数信息，但收敛速度较慢。
- **三次多项式拟合**：收敛速度快，但需要计算导数。
- **二分法**：简单易实现，适用于导数容易计算的函数。
- **割线法**：不需要二阶导数，收敛速度介于二分法和牛顿-拉夫森法之间。

在实际应用中，应根据函数的特点和计算资源的限制选择合适的优化算法。

第三章 无约束优化问题

1. 梯度下降法（Steepest Descent Method）

1.1 基本原理

梯度下降法的核心思想是沿着负梯度方向进行搜索，因为负梯度方向是函数值下降最快的方向。每次迭代的更新公式为：

$$x_{i+1} = x_i - \alpha \nabla f(x_i)$$

其中：

- $\nabla f(x_i)$ 是函数在点 x_i 处的梯度。
- α 是步长，通常通过线性搜索（如黄金分割法）确定。

1.2 文件中的具体例子

文件中通过一个测试问题展示了梯度下降法的应用。测试问题的目标函数为：

$$f(x) = k_1 \left(\sqrt{x_1^2 + (x_2 + 1)^2} - 1 \right)^2 + k_2 \left(\sqrt{x_1^2 + (x_2 - 1)^2} - 1 \right)^2 - (F_{x_1} x_1 + F_{x_2} x_2)$$

其中，$k_1 = 100\,\mathrm{N/m}$，$k_2 = 90\,\mathrm{N/m}$，$F_{x_1} = 20\,\mathrm{N}$，$F_{x_2} = 40\,\mathrm{N}$。

1.2.1 初始点

初始点为 $x_0 = (-3, 2)$，初始函数值为 $f(x_0) = 1452.2619$。

1.2.2 迭代过程

梯度下降法的迭代过程如下表所示：

迭代次数			函数值	梯度范数
1	0.095	0.023	-2.704	1006.074
2	0.170	0.141	-5.278	37.036
3	0.318	0.048	-7.369	23.451
4	0.375	0.138	-8.773	26.656
5	0.448	0.092	-9.375	14.021
6	0.470	0.127	-9.583	10.071
7	0.491	0.114	-9.639	4.403
8	0.497	0.123	-9.652	2.509
9	0.501	0.120	-9.655	1.050
10	0.503	0.122	-9.656	0.554
11	0.504	0.122	-9.656	0.236
12	0.504	0.122	-9.656	0.125

迭代次数	x_1	x_2	函数值 $f(x)$	梯度范数 $\|\nabla f(x)\|$
13	0.504	0.122	-9.656	0.047
14	0.504	0.122	-9.656	0.027
15	0.504	0.122	-9.656	0.016

1.2.3 分析

- **初始阶段**：在远离极小值时，梯度值较大，函数值下降较快。例如，第一次迭代后，函数值从 1452.2619 下降到 -2.704。
- **接近极小值时**：梯度值变小，函数值下降速度显著减慢，表现出“锯齿”现象。例如，从第 7 次迭代开始，函数值从 -9.639 下降到 -9.656，变化非常小。
- **收敛速度**：梯度下降法在接近极小值时收敛速度较慢，需要较多迭代次数才能达到较高的精度。

1.2.4 优缺点

- **优点**：
  - 简单易实现，只需要计算梯度。
  - 在远离极小值时，能够快速减少函数值。
- **缺点**：
  - 在接近极小值时，收敛速度慢，表现出“锯齿”现象。
  - 对初始点敏感，可能陷入局部极小值。

---

2. 牛顿法（Newton's Method）

2.1 基本原理

牛顿法的核心思想是利用梯度和 Hessian 矩阵的信息来确定搜索方向。每次迭代的更新公式为：

$$x_{i+1} = x_i - [H]^{-1} \nabla f(x_i)$$

其中：

- $\nabla f(x_i)$ 是函数在点 x_i 处的梯度。
- H 是 Hessian 矩阵（二阶导数矩阵）。
- $[H]^{-1}$ 是 Hessian 矩阵的逆矩阵。

2.2 文件中的具体例子

文件中通过同一个测试问题展示了牛顿法的应用。初始点同样为 $x_0 = (-3, 2)$，初始函数值为 $f(x_0) = 1452.2619$。

2.2.1 迭代过程

牛顿法的迭代过程如下表所示：

迭代次数			函数值	梯度范数
1	-0.754	0.524	44.244	1006.074

迭代次数	x_1	x_2	函数值 $f(x)$	梯度范数 $\|\nabla f(x)\|$
2	-0.362	-0.010	8.398	116.281
3	0.094	0.125	-3.920	50.597
4	11.775	0.324	22007.14	21.420
5	1.042	0.093	14.533	4076.916
6	0.640	0.142	-8.479	102.745
7	0.524	0.122	-9.635	18.199
8	0.505	0.122	-9.656	2.213
9	0.504	0.122	-9.656	0.059
10	0.504	0.122	-9.656	0.000

2.2.2 分析

- **初始阶段**：牛顿法在远离极小值时可能不收敛，甚至可能发散。例如，在第 4 次迭代中，函数值从 -3.920 突然增加到 22007.14，这是因为 Hessian 矩阵在远离极小值时可能不正定，导致搜索方向不是下降方向。
- **接近极小值时**：牛顿法在接近极小值时具有二次收敛性，收敛速度非常快。例如，从第 6 次迭代开始，函数值从 -8.479 迅速下降到 -9.656。
- **收敛速度**：牛顿法在接近极小值时收敛速度非常快，通常只需要几次迭代即可达到较高的精度。

2.2.3 优缺点

- **优点**：
  - 在接近极小值时，具有二次收敛性，收敛速度非常快。
  - 通过 Hessian 矩阵，能够更准确地确定搜索方向。
- **缺点**：
  - 计算复杂度高，每次迭代需要计算 Hessian 矩阵及其逆矩阵。
  - 对初始点敏感，如果初始点远离极小值，可能不收敛甚至发散。
  - Hessian 矩阵可能不正定，导致搜索方向不是下降方向。

---

3. 其他优化方法

3.1 修正牛顿法（Modified Newton's Method）

修正牛顿法在牛顿法的基础上增加了线性搜索步骤，以确保每次迭代都能减少函数值。设计变量的更新公式为：

$$x_{i+1} = x_i - \alpha [H]^{-1} \nabla f(x_i)$$

其中，α 是通过线性搜索确定的步长。

3.1.1 优点

- **结合了牛顿法的快速收敛性和线性搜索的稳定性。**
- **在远离极小值时也能保证函数值下降。**

3.1.2 缺点

- **计算复杂度仍然较高**：需要计算 Hessian 矩阵及其逆矩阵。

3.1.3 适用场景

- 适合初始点远离极小值的情况。
- 适合需要快速收敛的中等规模问题。

3.1.4 文件中的具体例子

文件中通过一个测试问题展示了修正牛顿法的应用。初始点为 $(-3, 2)$，经过 6 次迭代后，函数值从 1452.2619 下降到 -9.656。与牛顿法相比，修正牛顿法在相同初始点下收敛更快。

3.2 Levenberg-Marquardt 方法

Levenberg-Marquardt 方法结合了梯度下降法和牛顿法的优点。搜索方向为：

$$S_i = -[H + \lambda I]^{-1} \nabla f(x_i)$$

其中，λ 是一个调整参数。当 λ 较大时，方法类似于梯度下降法；当 λ 较小时，方法类似于牛顿法。

3.2.1 优点

- **在远离极小值时使用梯度下降法，在接近极小值时使用牛顿法。**
- **具有较强的鲁棒性。**

3.2.2 缺点

- **需要调整参数 λ**，增加了算法的复杂性。

3.2.3 适用场景

- 适合非线性最小二乘问题。
- 适合初始点远离极小值的情况。

3.2.4 文件中的具体例子

文件中通过一个测试问题展示了 Levenberg-Marquardt 方法的应用。初始点为 $(-3, 2)$，经过 10 次迭代后，函数值从 1452.2619 下降到 -9.656。观察到在迭代过程中，λ 值不断调整，使得方法在远离极小值时使用梯度下降法，在接近极小值时使用牛顿法。

3.3 共轭梯度法（Conjugate Gradient Method）

共轭梯度法是一种一阶方法，但具有二次收敛性。搜索方向为：

$$S_{i+1} = -\nabla f(x_i) + \beta S_i$$

其中，β 是一个标量参数，用于结合当前梯度和之前的搜索方向。

3.3.1 优点

- **适合大规模问题**：不需要存储 Hessian 矩阵。
- **具有二次收敛性。**

3.3.2 缺点

- **对初始点敏感。**

3.3.3 适用场景

- 适合大规模问题。
- 适合梯度计算较为简单的问题。

3.3.4 文件中的具体例子

文件中通过一个测试问题展示了共轭梯度法的应用。初始点为 $(-3, 2)$，经过 7 次迭代后，函数值从 1452.2619 下降到 -9.656。与梯度下降法相比，共轭梯度法在接近极小值时收敛速度更快。

3.4 DFP 和 BFGS 方法

DFP 和 BFGS 方法都是拟牛顿法，通过近似 Hessian 矩阵或其逆矩阵来加速收敛。DFP 方法近似 Hessian 的逆矩阵，而 BFGS 方法直接近似 Hessian 矩阵。

3.4.1 优点

- **不需要显式计算 Hessian 矩阵。**
- **具有二次收敛性。**

3.4.2 缺点

- **需要存储和更新近似矩阵**，增加了存储和计算开销。

3.4.3 适用场景

- 适合中等规模问题。
- 适合需要快速收敛的问题。

3.4.4 文件中的具体例子

文件中通过一个测试问题展示了 DFP 和 BFGS 方法的应用。初始点为 $(-3, 2)$，经过 9 次迭代后，函数值从 1452.2619 下降到 -9.656。观察到在迭代过程中，近似矩阵逐渐接近 Hessian 矩阵或其逆矩阵。

---

4. 非基于梯度的优化方法

4.1 Powell 共轭方向法

Powell 方法是一种直接搜索方法，不需要计算梯度。它通过存储之前的搜索方向并形成新的搜索方向来实现二次收敛。

4.1.1 优点

- **不需要计算梯度。**
- **具有二次收敛性。**

4.1.2 缺点

- **对初始点敏感。**

4.1.3 适用场景

- 适合梯度计算困难的问题。
- 适合中等规模问题。

4.1.4 文件中的具体例子

文件中通过一个测试问题展示了 Powell 方法的应用。初始点为 $(-3, 2)$，经过 5 次迭代后，函数值从 1452.2619 下降到 -9.656。

4.2 Nelder-Mead 算法

Nelder-Mead 算法是一种单纯形法，通过反射、扩展和收缩等操作在搜索空间中移动单纯形，直到找到函数的最优值。

4.2.1 优点

- **不需要计算梯度。**
- **具有较强的鲁棒性。**

4.2.2 缺点

- **收敛速度较慢。**
- **对初始点敏感。**

4.2.3 适用场景

- 适合梯度计算困难的问题。
- 适合小规模问题。

4.2.4 文件中的具体例子

文件中通过一个测试问题展示了 Nelder-Mead 算法的应用。初始点为随机值，经过 24 次迭代后，函数值从 72.2666 下降到 -9.656。

---

5. 应用实例

5.1 弹簧系统优化

通过优化弹簧系统的势能函数，展示了各种优化算法的应用。弹簧系统的势能函数为：

$$U = k_1 \left(\sqrt{x_1^2 + (x_2 + 1)^2} - 1 \right)^2 + k_2 \left(\sqrt{x_1^2 + (x_2 - 1)^2} - 1 \right)^2 - (F_{x_1} x_1 + F_{x_2} x_2)$$

通过优化算法，可以找到使势能最小的弹簧系统的平衡位置。

5.2 机器人运动设计

将 Powell 方法应用于机器人运动设计问题，通过优化关节角度，使得机器人末端执行器的运动轨迹与期望轨迹尽可能匹配。

---

6. 总结

本章详细介绍了无约束优化问题的求解方法，特别是基于梯度的优化方法（如梯度下降法、牛顿法）和非基于梯度的优化方法（如 Powell 方法、Nelder-Mead 算法）。通过对比梯度下降法和牛顿法，可以看出它们各有优缺点，实际应用中应根据问题的特点选择合适的优化方法。

- **梯度下降法**：适合作为初始优化步骤，适合大规模问题。
- **牛顿法**：适合小规模问题，特别是在接近极小值时具有快速收敛性。

第四章 线性规划

本章主要介绍了线性规划的基本概念、求解方法及其应用。线性规划问题的目标函数和约束条件都是设计变量的线性函数，广泛应用于各个领域，如航空公司的机组调度、投资组合优化、石油公司的生产计划等。以下是本章的详细解读，重点放在4.3节“线性规划的标准形式”。

---

1. 线性规划简介

线性规划问题的目标函数和约束条件都是设计变量的线性函数。线性函数满足以下性质：

- **加法性**：$f(x+y)=f(x)+f(y)$
- **齐次性**：$f(kx)=kf(x)$

线性规划问题通常包含大量的设计变量和约束条件，因此需要特殊的求解技术。本章介绍了线性规划的图形法、标准形式、基本解、单纯形法、对偶问题、对偶单纯形法以及内点法。

---

2. 图形法求解

图形法是一种简单的求解方法，适用于只有两到三个设计变量的线性规划问题。通过绘制约束条件的图形，可以找到可行域，并在可行域的顶点处找到目标函数的最优值。

示例：
考虑以下线性规划问题：

$$\text{Maximize } z=x+2y$$

约束条件：

$$\begin{gathered}2x+y\geq 4\\-2x+4y\geq -2\\-2x+y\geq -8\\-2x+y\leq -2\\y\leq 6\end{gathered}$$

通过绘制这些约束条件，可以得到可行域 $ABCDE$。目标函数的最优值出现在可行域的顶点处，本例中最大值为19，对应的 $x=7$，$y=6$。

---

3. 线性规划的标准形式（重点）

线性规划问题的标准形式是求解线性规划问题的基础。标准形式的要求如下：

1. **目标函数为最小化类型**：
   - 如果原问题是最大化问题，可以通过将目标函数乘以-1转换为最小化问题。
   - 例如，最大化 $z=x_1+2x_2$ 可以转换为最小化 $-z=-x_1-2x_2$。
2. **所有设计变量非负**：
   - 如果某个变量 x_i 没有非负限制（即自由变量），可以通过引入两个非负变量 x_i' 和 x_i'' 来表示，即 $x_i=x_i'-x_i''$，其中 $x_i'\geq 0$，$x_i''\geq 0$。
3. **所有约束条件为等式形式**：
   - 如果约束条件是不等式，可以通过引入松弛变量或剩余变量将其转换为等式。

- 对于 $\leq$ 类型的约束，添加松弛变量 $s_i \geq 0$。
- 对于 $\geq$ 类型的约束，减去剩余变量 $e_i \geq 0$。

4. **所有约束条件的右端项非负**：
   - 如果某个约束的右端项为负数，可以通过将整个约束乘以-1来使其变为非负。

示例：
将以下线性规划问题转换为标准形式：

$$\text{Maximize } z = -4x_1 - 2x_2 + x_3 - 3x_4$$

约束条件：

$$\begin{gathered}2x_1 + 3x_2 - x_3 - 3x_4 = 5 \\ -5x_1 - 2x_2 + 4x_3 - 7x_4 \leq 8 \\ 4x_1 - x_2 - 2x_3 + 5x_4 \leq -6 \\ x_1 \geq -1, 0 \leq x_2 \leq 3, x_3 \geq 0, x_4 \text{ 自由}\end{gathered}$$

转换步骤：

1. 将目标函数转换为最小化形式：

$$\text{Minimize } -z = 4x_1 + 2x_2 - x_3 + 3x_4$$

2. 处理约束条件：
   - 第二个约束 $-5x_1 - 2x_2 + 4x_3 - 7x_4 \leq 8$ 可以通过添加松弛变量 s_1 转换为等式：

$$-5x_1 - 2x_2 + 4x_3 - 7x_4 + s_1 = 8$$

   - 第三个约束 $4x_1 - x_2 - 2x_3 + 5x_4 \leq -6$ 可以通过乘以-1转换为：

$$-4x_1 + x_2 + 2x_3 - 5x_4 \geq 6$$

     然后减去剩余变量 e_2：

$$-4x_1 + x_2 + 2x_3 - 5x_4 - e_2 = 6$$

3. 处理自由变量 x_4：
   - 引入 x_4' 和 x_4''，使得 $x_4 = x_4' - x_4''$，其中 $x_4' \geq 0$，$x_4'' \geq 0$。

最终，问题转换为标准形式：

$$\text{Minimize } z' = 4x_1' + 2x_2 - x_3 + 3x_4' - 3x_4''$$

约束条件：

$$\begin{gathered}2x_1' + 3x_2 - x_3 - 3x_4' + 3x_4'' = 7 \\ -5x_1' - 2x_2 + 4x_3 - 7x_4' + 7x_4'' + s_1 = 8 \\ -4x_1' + x_2 + 2x_3 - 5x_4' + 5x_4'' - e_2 = 6 \\ x_2 + s_3 = 3 \\ x_1', x_2, x_3, x_4', x_4'', s_1, e_2, s_3 \geq 0\end{gathered}$$

---

4. 基本解

对于标准形式的线性规划问题，基本解是通过将 $n - m$ 个变量设为零，并求解约束方程组得到的解。如果基本解满足所有变量非负，则称为基本可行解。基本可行解是可行域的顶点。

示例：
考虑以下方程组：

$$3x_1 - 4x_2 + 2x_3 + x_4 = 0$$
$$x_1 + 3x_2 + 2x_3 + x_4 = 500$$
$$7x_1 + x_2 + x_3 - x_4 = 700$$

通过选择不同的基本变量和非基本变量，可以得到多个基本解，其中一些是基本可行解。

---

5. 单纯形法

单纯形法是一种迭代算法，通过从一个基本可行解移动到另一个基本可行解，直到找到最优解。单纯形法需要一个初始的基本可行解，通常通过引入人工变量来实现。

示例：
考虑以下线性规划问题：

$$\text{Maximize } z = 6x_1 + 7x_2$$

约束条件：

$$3x_1 + x_2 \leq 10$$
$$x_1 + 2x_2 \leq 8$$
$$x_1 \leq 3$$
$$x_1, x_2 \geq 0$$

通过引入松弛变量，将问题转换为标准形式，并使用单纯形法求解。最终得到的最优解为 $x_1 = \frac{12}{5}$，$x_2 = \frac{14}{5}$，目标函数值为34。

---

6. 内点法

内点法通过在可行域内部移动来寻找最优解，适用于大规模线性规划问题。内点法包括障碍函数法、势能减少法和仿射缩放法。

示例：
使用仿射缩放法求解以下线性规划问题：

$$\text{Maximize } z = 6x_1 + 7x_2$$

约束条件：

$$3x_1 + x_2 \leq 10$$
$$x_1 + 2x_2 \leq 8$$
$$x_1 \leq 3$$
$$x_1, x_2 \geq 0$$

最终得到的最优解与单纯形法一致。

总结

本章详细介绍了线性规划的基本概念、求解方法及其应用。通过图形法、单纯形法、对偶单纯形法和内点法，可以有效地求解线性规划问题。线性规划在投资组合优化、生产计划、资源分配等领域有着广泛的应用。

第五章 引导随机搜索方法

1. 遗传算法（GA）的详细讲解

书本中的例子是**太阳能测试问题**，目标是最小化成本函数：

$$U = \frac{204,165.5}{330 - 2T} + \frac{10,400}{T - 20}$$

其中温度 T 的范围是 $[40, 90]$。

GA 的主要步骤：

1. **初始化种群**
2. **适应度评估**
3. **选择**
4. **交叉**
5. **变异**
6. **迭代**

详细步骤：

步骤 1：初始化种群

- 温度 T 的范围是 $[40, 90]$，我们需要将 T 编码为二进制字符串。
- 假设我们使用 15 位二进制字符串来表示 T，因为 $2^{15} = 32768$，可以覆盖 $[40, 90]$ 的范围，并且精度足够高。
- 随机生成 10 个个体（种群大小为 10），每个个体是一个 15 位二进制字符串。例如：
  - 个体 1: `110110011000101`
  - 个体 2: `100001010111010`
  - 个体 3: `000110101110101`
  - ...（共 10 个个体）

步骤 2：适应度评估

- 将二进制字符串解码为实数值 T：

  $$T_i = T_{\min} + \frac{(T_{\max} - T_{\min}) \cdot \text{Decoded Value}}{2^{15} - 1}$$

  其中 $T_{\min} = 40$，$T_{\max} = 90$，Decoded Value 是二进制字符串对应的十进制值。
- 计算每个个体的适应度（即成本函数 U 的值）。例如：
  - 个体 1: $T = 82.4894$，$U = 1403.6$
  - 个体 2: $T = 66.0659$，$U = 1257.6$
  - 个体 3: $T = 45.2568$，$U = 1264.3$
  - ...（计算所有个体的适应度）

步骤 3：选择

- 使用**轮盘赌选择法**或**锦标赛选择法**选择优秀的个体进行繁殖。
- **轮盘赌选择法**：
  - 计算每个个体的选择概率：

$$P_i = \frac{F_i}{\sum F_i}$$

    其中 $F_i = \frac{1}{1+U_i}$（因为我们要最小化 U，所以适应度 F_i 与 U_i 成反比）。
  - 根据概率选择个体，适应度高的个体被选中的概率更大。
- **锦标赛选择法**：
  - 随机选择两个个体，比较它们的适应度，选择适应度更高的个体作为父代。

步骤 4：交叉

- 随机选择两个父代个体，进行**单点交叉**操作。例如：
  - 父代 1: `110110011000101`
  - 父代 2: `100001010111010`
  - 交叉点在第 9 位：
    - 子代 1: `110110010111010`
    - 子代 2: `100001011000101`

步骤 5：变异

- 对子代个体进行**变异**操作，随机改变某些基因（0 变 1，1 变 0）。例如：
  - 子代 1: `110110010111010`
  - 变异后: `110110010111110`（第 13 位发生变异）

步骤 6：迭代

- 重复上述步骤，直到满足终止条件（如达到最大迭代次数或找到满意的解）。

---

2. 粒子群优化（PSO）的详细讲解

书本中的例子是**Schwefel 函数**，目标是最小化函数：

$$f(x) = -x_1 \sin\sqrt{|x_1|} - x_2 \sin\sqrt{|x_2|}$$

其中 x_1 和 x_2 的范围是 $[-500, 500]$。

PSO 的主要步骤：

1. **初始化粒子群**
2. **适应度评估**
3. **更新个体最优和全局最优**
4. **更新速度和位置**
5. **迭代**

详细步骤：

步骤 1：初始化粒子群

- 假设粒子群大小为 20，每个粒子的位置 x_1 和 x_2 在 $[-500, 500]$ 范围内随机生成。例如：
  - 粒子 1: $x_1 = 120.5$，$x_2 = -300.2$
  - 粒子 2: $x_1 = -450.3$，$x_2 = 200.7$
  - ...（共 20 个粒子）

步骤 2：适应度评估

- 计算每个粒子的适应度（即 Schwefel 函数值）。例如：
  - 粒子 1: $f(120.5, -300.2) = -120.5 \sin\sqrt{|120.5|} - (-300.2)\sin\sqrt{|-300.2|}$
  - 粒子 2: $f(-450.3, 200.7) = -(-450.3)\sin\sqrt{|-450.3|} - 200.7\sin\sqrt{|200.7|}$
  - ...（计算所有粒子的适应度）

步骤 3：更新个体最优和全局最优

- 记录每个粒子的历史最优位置（p_{best}）和群体的历史最优位置（g_{best}）。
  - 如果当前适应度优于 p_{best}，则更新 p_{best}。
  - 如果当前适应度优于 g_{best}，则更新 g_{best}。

步骤 4：更新速度和位置

- 根据以下公式更新每个粒子的速度和位置：

$$v_{i+1,k} = w_1 v_{i,k} + \phi_1(p_{\text{best},k} - x_{i,k})u_i + \phi_2(g_{\text{best}} - x_{i,k})u_i$$

$$x_{i+1,k} = x_{i,k} + v_{i+1,k}$$

  其中：
  - $v_{i,k}$ 是粒子的速度，
  - $p_{\text{best},k}$ 是个体的历史最优位置，
  - g_{best} 是群体的历史最优位置，
  - w_1, ϕ_1, ϕ_2 是算法的调优参数，
  - u_i 是随机数。

步骤 5：迭代

- 重复上述步骤，直到满足终止条件（如达到最大迭代次数或找到满意的解）。

八皇后问题简介

八皇后问题是一个经典的组合优化问题，目标是在一个 8x8 的国际象棋棋盘上放置 8 个皇后，使得它们互不攻击。具体规则如下：

1. **行冲突**：不能有两个皇后在同一行。
2. **列冲突**：不能有两个皇后在同一列。
3. **对角线冲突**：不能有两个皇后在同一对角线上。

---

3. 遗传算法（GA）解决八皇后问题

算法思路：

遗传算法模拟生物进化过程，通过选择、交叉和变异等操作逐步优化种群，最终找到最优解。

主要步骤：

1. **初始化种群**：随机生成一组个体（解）。
2. **适应度评估**：计算每个个体的适应度（冲突数越少，适应度越高）。
3. **选择**：根据适应度选择优秀个体进行繁殖。
4. **交叉**：通过交叉操作生成新个体。
5. **变异**：随机改变个体的某些基因。
6. **迭代**：重复上述步骤，直到找到解或达到最大迭代次数。

详细步骤：

步骤 1：初始化种群

- 随机生成 10 个个体（种群大小为 10），每个个体是一个长度为 8 的数组，数组的每个元素是 1 到 8 的随机整数。例如：
  - 个体 1: `[2, 4, 7, 1, 8, 5, 3, 6]`
  - 个体 2: `[5, 3, 8, 2, 7, 1, 6, 4]`
  - 个体 3: `[1, 7, 4, 6, 8, 2, 5, 3]`
  - 个体 4: `[3, 6, 2, 5, 1, 8, 4, 7]`
  - 个体 5: `[7, 2, 6, 3, 1, 4, 8, 5]`
  - 个体 6: `[4, 8, 1, 5, 6, 2, 7, 3]`
  - 个体 7: `[6, 1, 5, 2, 8, 3, 7, 4]`
  - 个体 8: `[8, 3, 1, 6, 2, 5, 7, 4]`
  - 个体 9: `[5, 7, 2, 6, 3, 1, 4, 8]`
  - 个体 10: `[4, 1, 5, 8, 6, 3, 7, 2]`

步骤 2：适应度评估

- 计算每个个体的适应度（即冲突数）。冲突数越少，适应度越高。
  - 冲突数 = 行冲突数 + 列冲突数 + 对角线冲突数。
  - 例如，个体 `[2, 4, 7, 1, 8, 5, 3, 6]` 的冲突数为 2（假设有两个冲突）。
  - 适应度可以定义为：$\text{适应度} = 1/(1 + \text{冲突数})$。
- 计算所有个体的适应度：
  - 个体 1: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$
  - 个体 2: 冲突数 = 1，适应度 = $1/(1+1) = 0.5$
  - 个体 3: 冲突数 = 3，适应度 = $1/(1+3) = 0.25$
  - 个体 4: 冲突数 = 0，适应度 = $1/(1+0) = 1.0$
  - 个体 5: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$
  - 个体 6: 冲突数 = 1，适应度 = $1/(1+1) = 0.5$

- 个体 7: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$
- 个体 8: 冲突数 = 3，适应度 = $1/(1+3) = 0.25$
- 个体 9: 冲突数 = 1，适应度 = $1/(1+1) = 0.5$
- 个体 10: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$

步骤 3：选择

- 使用**轮盘赌选择法**选择优秀的个体进行繁殖。
  - 计算每个个体的选择概率：
    - 个体 1: $0.333/4.0 = 0.083$
    - 个体 2: $0.5/4.0 = 0.125$
    - 个体 3: $0.25/4.0 = 0.0625$
    - 个体 4: $1.0/4.0 = 0.25$
    - 个体 5: $0.333/4.0 = 0.083$
    - 个体 6: $0.5/4.0 = 0.125$
    - 个体 7: $0.333/4.0 = 0.083$
    - 个体 8: $0.25/4.0 = 0.0625$
    - 个体 9: $0.5/4.0 = 0.125$
    - 个体 10: $0.333/4.0 = 0.083$
  - 根据概率选择个体，适应度高的个体被选中的概率更大。

步骤 4：交叉

- 随机选择两个父代个体，进行**单点交叉**操作。例如：
  - 父代 1: `[2, 4, 7, 1, 8, 5, 3, 6]`
  - 父代 2: `[5, 3, 8, 2, 7, 1, 6, 4]`
  - 交叉点在第 4 位：
    - 子代 1: `[2, 4, 7, 2, 7, 1, 6, 4]`
    - 子代 2: `[5, 3, 8, 1, 8, 5, 3, 6]`

步骤 5：变异

- 对子代个体进行**变异**操作，随机改变某些基因（列位置）。例如：
  - 子代 1: `[2, 4, 7, 2, 7, 1, 6, 4]`
  - 变异后: `[2, 4, 7, 2, 7, 1, 6, 8]`（第 8 位发生变异）

步骤 6：迭代

- 重复上述步骤，直到找到冲突数为 0 的解（即找到八皇后问题的解）。

Python 实现

```python
import random

# 定义棋盘大小
N = 8
```

```python
# 初始化种群
def initialize_population(pop_size):
    """
    随机生成一组个体（解），每个个体是一个长度为 N 的列表，表示每行皇后的列位置。
    """
    return [random.sample(range(1, N + 1), N) for _ in range(pop_size)]

# 计算冲突数
def calculate_conflicts(individual):
    """
    计算一个个体（解）的冲突数。
    """
    conflicts = 0
    for i in range(N):
        for j in range(i + 1, N):
            # 检查列冲突和对角线冲突
            if individual[i] == individual[j] or abs(individual[i] -
individual[j]) == abs(i - j):
                conflicts += 1
    return conflicts

# 计算适应度
def fitness(individual):
    """
    计算一个个体（解）的适应度，冲突数越少，适应度越高。
    """
    return 1 / (1 + calculate_conflicts(individual))

# 选择（轮盘赌选择法）
def selection(population, fitness_values):
    """
    根据适应度选择优秀个体进行繁殖。
    """
    total_fitness = sum(fitness_values)
    probabilities = [f / total_fitness for f in fitness_values]
    selected = random.choices(population, weights=probabilities,
k=len(population))
    return selected

# 交叉（单点交叉）
def crossover(parent1, parent2):
    """
    通过单点交叉生成两个子代个体。
    """
    point = random.randint(1, N - 1)
    child1 = parent1[:point] + parent2[point:]
    child2 = parent2[:point] + parent1[point:]
    return child1, child2

# 变异（随机改变一个基因）
def mutate(individual):
    """
    随机改变一个个体（解）的某个基因（列位置）。
    """
    idx = random.randint(0, N - 1)
```

```
    individual[idx] = random.randint(1, N)
    return individual

# 遗传算法主函数
def genetic_algorithm(pop_size, max_generations):
    """
    遗传算法主函数。
    """
    population = initialize_population(pop_size)
    for generation in range(max_generations):
        # 计算适应度
        fitness_values = [fitness(ind) for ind in population]
        # 检查是否有解
        best_individual = population[fitness_values.index(max(fitness_values))]
        if calculate_conflicts(best_individual) == 0:
            print(f"Solution found in generation {generation}:
{best_individual}")
            return best_individual
        # 选择
        selected = selection(population, fitness_values)
        # 交叉
        next_population = []
        for i in range(0, pop_size, 2):
            parent1, parent2 = selected[i], selected[i + 1]
            child1, child2 = crossover(parent1, parent2)
            next_population.extend([child1, child2])
        # 变异
        population = [mutate(ind) for ind in next_population]
    print("No solution found.")
    return None

# 运行遗传算法
genetic_algorithm(pop_size=10, max_generations=1000)
```

4. 粒子群优化（PSO）解决八皇后问题

算法思路：

粒子群优化模拟鸟群或鱼群的集体行为，通过个体和群体的历史最优位置更新搜索方向，逐步找到最优解。

主要步骤：

1. **初始化粒子群**：随机生成一组粒子（解）。
2. **适应度评估**：计算每个粒子的适应度（冲突数越少，适应度越高）。
3. **更新个体最优和全局最优**：记录每个粒子的历史最优位置和群体的历史最优位置。
4. **更新速度和位置**：根据个体最优和全局最优更新粒子的速度和位置。
5. **迭代**：重复上述步骤，直到找到解或达到最大迭代次数。

详细步骤：

步骤 1：初始化粒子群

- 随机生成 10 个粒子（粒子群大小为 10），每个粒子是一个长度为 8 的数组，数组的每个元素是 1 到 8 的随机整数。例如：
  - 粒子 1: `[2, 4, 7, 1, 8, 5, 3, 6]`
  - 粒子 2: `[5, 3, 8, 2, 7, 1, 6, 4]`
  - 粒子 3: `[1, 7, 4, 6, 8, 2, 5, 3]`
  - 粒子 4: `[3, 6, 2, 5, 1, 8, 4, 7]`
  - 粒子 5: `[7, 2, 6, 3, 1, 4, 8, 5]`
  - 粒子 6: `[4, 8, 1, 5, 6, 2, 7, 3]`
  - 粒子 7: `[6, 1, 5, 2, 8, 3, 7, 4]`
  - 粒子 8: `[8, 3, 1, 6, 2, 5, 7, 4]`
  - 粒子 9: `[5, 7, 2, 6, 3, 1, 4, 8]`
  - 粒子 10: `[4, 1, 5, 8, 6, 3, 7, 2]`

步骤 2：适应度评估

- 计算每个粒子的适应度（即冲突数）。冲突数越少，适应度越高。
  - 冲突数 = 行冲突数 + 列冲突数 + 对角线冲突数。
  - 例如，粒子 `[2, 4, 7, 1, 8, 5, 3, 6]` 的冲突数为 2（假设有两个冲突）。
  - 适应度可以定义为：$\text{适应度} = 1/(1 + \text{冲突数})$。
- 计算所有粒子的适应度：
  - 粒子 1: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$
  - 粒子 2: 冲突数 = 1，适应度 = $1/(1+1) = 0.5$
  - 粒子 3: 冲突数 = 3，适应度 = $1/(1+3) = 0.25$
  - 粒子 4: 冲突数 = 0，适应度 = $1/(1+0) = 1.0$
  - 粒子 5: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$
  - 粒子 6: 冲突数 = 1，适应度 = $1/(1+1) = 0.5$
  - 粒子 7: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$
  - 粒子 8: 冲突数 = 3，适应度 = $1/(1+3) = 0.25$
  - 粒子 9: 冲突数 = 1，适应度 = $1/(1+1) = 0.5$
  - 粒子 10: 冲突数 = 2，适应度 = $1/(1+2) = 0.333$

步骤 3：更新个体最优和全局最优

- 记录每个粒子的历史最优位置（p_{best}）和群体的历史最优位置（g_{best}）。
  - 如果当前适应度优于 p_{best}，则更新 p_{best}。
  - 如果当前适应度优于 g_{best}，则更新 g_{best}。

步骤 4：更新速度和位置

- 根据以下公式更新每个粒子的速度和位置：

$$v_{i+1,k} = w_1 v_{i,k} + \phi_1 (p_{\text{best},k} - x_{i,k}) u_i + \phi_2 (g_{\text{best}} - x_{i,k}) u_i$$

$$x_{i+1,k} = x_{i,k} + v_{i+1,k}$$

  其中：

  - $v_{i,k}$ 是粒子的速度，
  - $p_{\text{best},k}$ 是个体的历史最优位置，
  - g_{best} 是群体的历史最优位置，
  - w_1, ϕ_1, ϕ_2 是算法的调优参数，
  - u_i 是随机数。

步骤 5：迭代

- 重复上述步骤，直到找到冲突数为 0 的解（即找到八皇后问题的解）。

Python 实现

```
import random

# 定义棋盘大小
N = 8

# 初始化粒子群
def initialize_particles(pop_size):
    """
    随机生成一组粒子（解），每个粒子是一个长度为 N 的列表，表示每行皇后的列位置。
    """
    return [random.sample(range(1, N + 1), N) for _ in range(pop_size)]

# 计算冲突数
def calculate_conflicts(individual):
    """
    计算一个粒子（解）的冲突数。
    """
    conflicts = 0
    for i in range(N):
        for j in range(i + 1, N):
            # 检查列冲突和对角线冲突
            if individual[i] == individual[j] or abs(individual[i] -
individual[j]) == abs(i - j):
                conflicts += 1
    return conflicts

# 计算适应度
def fitness(individual):
    """
    计算一个粒子（解）的适应度，冲突数越少，适应度越高。
    """
    return 1 / (1 + calculate_conflicts(individual))
```

```python
# 更新速度和位置
def update_velocity_position(particle, pbest, gbest, velocity, w, c1, c2):
    """
    更新粒子的速度和位置。
    """
    new_velocity = []
    new_particle = []
    for i in range(N):
        # 更新速度
        v = w * velocity[i] + c1 * random.random() * (pbest[i] - particle[i]) +
c2 * random.random() * (gbest[i] - particle[i])
        # 限制速度范围
        v = max(-N, min(N, v))
        new_velocity.append(v)
        # 更新位置
        new_pos = particle[i] + v
        # 限制位置范围
        new_pos = max(1, min(N, new_pos))
        new_particle.append(int(new_pos))
    return new_velocity, new_particle

# 粒子群优化主函数
def pso(pop_size, max_iterations, w, c1, c2):
    """
    粒子群优化主函数。
    """
    particles = initialize_particles(pop_size)
    velocities = [[0] * N for _ in range(pop_size)]
    pbest = particles.copy()
    gbest = min(particles, key=lambda x: calculate_conflicts(x))
    for iteration in range(max_iterations):
        for i in range(pop_size):
            # 更新速度和位置
            velocities[i], particles[i] = update_velocity_position(particles[i],
pbest[i], gbest, velocities[i], w, c1, c2)
            # 更新个体最优
            if fitness(particles[i]) > fitness(pbest[i]):
                pbest[i] = particles[i]
            # 更新全局最优
            if fitness(particles[i]) > fitness(gbest):
                gbest = particles[i]
        # 检查是否有解
        if calculate_conflicts(gbest) == 0:
            print(f"Solution found in iteration {iteration}: {gbest}")
            return gbest
    print("No solution found.")
    return None

# 运行粒子群优化
pso(pop_size=10, max_iterations=1000, w=0.5, c1=1.5, c2=1.5)
```

总结

通过书本中的例子，我们可以看到：

- **遗传算法（GA）** 通过模拟生物进化过程，利用选择、交叉和变异等操作优化种群，适用于复杂的多模态优化问题。
- **粒子群优化（PSO）** 通过模拟群体智能，利用个体和群体的历史最优位置更新搜索方向，适用于连续优化问题。

这两种方法在处理复杂、多模态的优化问题时表现出色，尤其适用于传统梯度方法难以处理的情况。通过具体的示例，我们可以更好地理解 GA 和 PSO 的运行过程。

第六章 约束优化

1. 将函数转换为矩阵形式

给定函数：

$$f(x_1, x_2) = 6x_1^2 - 6x_1x_2 + 2x_2^2 - x_1 - 2x_2$$

将其表示为矩阵形式：

$$f(x_1, x_2) = \frac{1}{2}X^TAX + B^TX$$

其中：

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad A = \begin{bmatrix} 12 & -6 \\ -6 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

---

2. 对角化矩阵 A

2.1 计算特征值

解特征方程 $\det(A - \lambda I) = 0$：

$$\det \begin{bmatrix} 12-\lambda & -6 \\ -6 & 4-\lambda \end{bmatrix} = (12-\lambda)(4-\lambda) - (-6)^2 = \lambda^2 - 16\lambda + 12 = 0$$

解得特征值：

$$\lambda_1 = 8 + 2\sqrt{13}, \quad \lambda_2 = 8 - 2\sqrt{13}$$

2.2 计算特征向量

对于 $\lambda_1 = 8 + 2\sqrt{13}$：
解方程 $(A - \lambda_1 I)v = 0$：

$$\begin{bmatrix} 12 - (8+2\sqrt{13}) & -6 \\ -6 & 4 - (8+2\sqrt{13}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

解得特征向量：

$$v_1 = \begin{bmatrix} 1 \\ \frac{4+2\sqrt{13}}{6} \end{bmatrix}$$

对于 $\lambda_2 = 8 - 2\sqrt{13}$：
解方程 $(A - \lambda_2 I)v = 0$：

$$\begin{bmatrix} 12 - (8-2\sqrt{13}) & -6 \\ -6 & 4 - (8-2\sqrt{13}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

解得特征向量：

$$v_2 = \begin{bmatrix} 1 \\ \frac{4-2\sqrt{13}}{6} \end{bmatrix}$$

2.3 归一化特征向量

将特征向量归一化为单位向量：

对于 v_1：

$$\|v_1\| = \sqrt{1^2 + \left(\frac{4 + 2\sqrt{13}}{6}\right)^2}$$

归一化后的特征向量：

$$u_1 = \frac{v_1}{\|v_1\|}$$

对于 v_2：

$$\|v_2\| = \sqrt{1^2 + \left(\frac{4 - 2\sqrt{13}}{6}\right)^2}$$

归一化后的特征向量：

$$u_2 = \frac{v_2}{\|v_2\|}$$

2.4 对角化矩阵

将归一化后的特征向量组成矩阵 P：

$$P = [u_1 \quad u_2]$$

对角矩阵 D 为：

$$D = \begin{bmatrix} 8 + 2\sqrt{13} & 0 \\ 0 & 8 - 2\sqrt{13} \end{bmatrix}$$

因此，矩阵 A 可以表示为：

$$A = PDP^{-1}$$

---

3. 绘制对角化后的矩阵 D 的等高线图像

3.1 等高线的类型

矩阵 D 的等高线方程为：

$$\frac{1}{2}X^T DX = \text{常数}$$

展开后为：

$$\frac{1}{2}\left((8 + 2\sqrt{13})x_1^2 + (8 - 2\sqrt{13})x_2^2\right) = \text{常数}$$

这是一个标准的椭圆方程，其主轴与坐标轴对齐。

3.2 确定主轴的方向和大小

- **主轴方向**：由于 D 是对角矩阵，主轴方向与坐标轴 x_1 和 x_2 对齐。
- **主轴长度**：
  - 长轴长度：$\frac{1}{\sqrt{8+2\sqrt{13}}}$
  - 短轴长度：$\frac{1}{\sqrt{8-2\sqrt{13}}}$

3.3 绘制草图

1. **绘制坐标系**：在纸上绘制 x_1 和 x_2 轴。
2. **绘制主轴**：
   - 长轴沿 x_1 轴，长度为 $\frac{1}{\sqrt{8+2\sqrt{13}}}$。
   - 短轴沿 x_2 轴，长度为 $\frac{1}{\sqrt{8-2\sqrt{13}}}$。
3. **绘制椭圆**：
   - 以原点为中心，根据主轴长度绘制椭圆。
4. **调整形状**：
   - 确保椭圆的长轴和短轴长度符合计算结果。

---

4. 总结

- 矩阵 A 对角化后的矩阵 D 为：

$$D = \begin{bmatrix} 8+2\sqrt{13} & 0 \\ 0 & 8-2\sqrt{13} \end{bmatrix}$$

- 特征向量归一化后组成矩阵 P。
- 等高线是标准椭圆，主轴与坐标轴对齐。
- 长轴长度为 $\frac{1}{\sqrt{8+2\sqrt{13}}}$，短轴长度为 $\frac{1}{\sqrt{8-2\sqrt{13}}}$。

通过以上步骤，你可以绘制出矩阵 A 对角化后的矩阵 D 的等高线图像。

SQP（Sequential Quadratic Programming，序列二次规划）是一种用于求解非线性约束优化问题的数值方法，特别适用于目标函数和约束条件都是光滑的情况。

问题定义

目标函数：

$$f(x) = (x_1 - 1)^2 + (x_2 - 2)^2$$

约束条件：

- 等式约束：$h_1(x) = 2x_1 - x_2 = 0$
- 不等式约束：$g_1(x) = x_1 - 5 \leq 0$

初始点：

$$x^{(0)} = (10, -5)$$

迭代 1

我们从初始点 $x^{(0)} = (10, -5)$ 开始。

---

1. 梯度计算

1.1 目标函数的梯度

$$\nabla f(x) = \begin{bmatrix} 2(x_1 - 1) \\ 2(x_2 - 2) \end{bmatrix}$$

在 $x^{(0)} = (10, -5)$ 处：

$$\nabla f(x^{(0)}) = \begin{bmatrix} 2(10 - 1) \\ 2(-5 - 2) \end{bmatrix} = \begin{bmatrix} 18 \\ -14 \end{bmatrix}$$

1.2 等式约束的梯度

$$\nabla h_1(x) = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

1.3 不等式约束的梯度

$$\nabla g_1(x) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

---

2. 二次规划 (QP) 子问题的构造

构造二次规划问题：

$$\begin{aligned} \min_{p} \quad & \frac{1}{2} p^T H_k p + \nabla f(x^{(k)})^T p \\ \text{subject to:} \quad & \nabla h_1(x^{(k)})^T p + h_1(x^{(k)}) = 0 \\ & \nabla g_1(x^{(k)})^T p + g_1(x^{(k)}) \leq 0 \end{aligned}$$

2.1 Hessian 矩阵

目标函数的 Hessian 为：

$$\nabla^2 f(x) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

由于约束是线性的，其 Hessian 为零。因此：

$$H_k = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

2.2 线性化约束

在 $x^{(0)} = (10, -5)$ 处：

- 等式约束：

$$h_1(x^{(0)}) = 2(10) - (-5) = 25$$

$$\nabla h_1(x^{(0)})^T p + h_1(x^{(0)}) = 2p_1 - p_2 + 25 = 0$$

- 不等式约束：

$$g_1(x^{(0)}) = 10 - 5 = 5$$

$$\nabla g_1(x^{(0)})^T p + g_1(x^{(0)}) = p_1 + 5 \le 0$$

2.3 子问题的形式

将以上代入，我们的二次规划子问题为：

$$\min_{p} \quad \frac{1}{2}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix}^T \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} + \begin{bmatrix} 18 \\ -14 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

$$\text{subject to:} \begin{matrix} 2p_1 - p_2 + 25 = 0 \\ p_1 + 5 \le 0 \end{matrix}$$

2.4 求解子问题

1. 拉格朗日函数：

$$L(p, \lambda, \mu) = p_1^2 + p_2^2 + 18p_1 - 14p_2 + \lambda(2p_1 - p_2 + 25) + \mu(p_1 + 5)$$

2. 对 p_1, p_2 求偏导：

$$\frac{\partial L}{\partial p_1} = 2p_1 + 18 + 2\lambda + \mu = 0 \quad \Rightarrow \quad p_1 = -9 - \lambda - \frac{\mu}{2}$$

$$\frac{\partial L}{\partial p_2} = 2p_2 - 14 - \lambda = 0 \quad \Rightarrow \quad p_2 = 7 + \frac{\lambda}{2}$$

3. 带入约束：

$$2p_1 - p_2 + 25 = 0$$

和

$$p_1 + 5 \le 0$$

解得：

$$p_1 = -5, \quad p_2 = 15$$

---

3. 更新点

更新公式：

$$x^{(1)} = x^{(0)} + \alpha p$$

假设步长 $\alpha = 1$：

$$x^{(1)} = (10, -5) + 1 \cdot (-5, 15) = (5, 10)$$

迭代 2

以 $x^{(1)} = (5, 10)$ 继续迭代。

---

1. 梯度计算

1. 目标函数梯度：

$$\nabla f(x^{(1)}) = \begin{bmatrix} 2(5-1) \\ 2(10-2) \end{bmatrix} = \begin{bmatrix} 8 \\ 16 \end{bmatrix}$$

2. 等式约束梯度：

$$\nabla h_1(x^{(1)}) = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

3. 不等式约束梯度：

$$\nabla g_1(x^{(1)}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

---

2. 构造二次规划子问题

新的二次规划问题如下：

1. Hessian:

$$H_k = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

2. 线性化约束：
   - 等式约束：

$$h_1(x^{(1)}) = 2(5) - 10 = 0$$

$$\nabla h_1(x^{(1)})^T p + h_1(x^{(1)}) = 2p_1 - p_2 = 0$$

   - 不等式约束：

$$g_1(x^{(1)}) = 5 - 5 = 0$$

$$\nabla g_1(x^{(1)})^T p + g_1(x^{(1)}) = p_1 \leq 0$$

解得：

$$p_1 = -4, \quad p_2 = -8$$

---

3. 更新点

$$x^{(2)} = x^{(1)} + \alpha p = (5, 10) + (-4, -8) = (1, 2)$$

收敛

在 $x^{(2)} = (1, 2)$ 处：

1. 梯度为零：

$$\nabla f(x^{(2)}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

2. 满足约束条件：

$$h_1(x^{(2)}) = 2(1) - 2 = 0$$

$$g_1(x^{(2)}) = 1 - 5 \leq 0$$

因此 $x^* = (1, 2)$ 是最终解。

Algorithms for Optimization

第十二章 多目标优化

第12章详细介绍了**多目标优化**（Multiobjective Optimization）的概念、方法和应用。多目标优化问题与单目标优化问题不同，它涉及多个目标函数的同时优化，通常这些目标之间是相互冲突的，无法通过单一的优化过程找到全局最优解。因此，多目标优化的核心在于找到一组**Pareto最优解**，这些解代表了不同目标之间的最佳权衡。以下是本章的详细讲解：

---

1. Pareto最优性（Pareto Optimality）

Pareto最优性是多目标优化的核心概念。它描述了一种状态，即在不损害其他目标的情况下，无法进一步优化任何一个目标。

1.1 支配关系（Dominance）

- 在单目标优化中，两个设计点可以通过目标函数值直接比较优劣。例如，若 $f(x') < f(x)$，则 x' 优于 x。
- 在多目标优化中，目标函数返回的是一个向量 $\mathbf{f}(x) = [f_1(x), f_2(x), \ldots, f_m(x)]$，每个维度对应一个目标。
- 定义**支配关系**：设计点 x 支配设计点 x'，当且仅当：
  - 在所有目标上，$f_i(x) \leq f_i(x')$（即 x 在所有目标上不劣于 x'）；
  - 至少在一个目标上，$f_i(x) < f_i(x')$（即 x 至少在一个目标上优于 x'）。
- 如果两个设计点在部分目标上互相优于对方，则它们之间存在**支配模糊性**（Dominance Ambiguity），无法直接比较优劣。

1.2 Pareto前沿（Pareto Frontier）

- **Pareto前沿**是所有Pareto最优解在目标空间中的集合。它代表了不同目标之间的最佳权衡。
- 在目标空间中，Pareto前沿通常位于边界上。对于二维目标空间，Pareto前沿是一条曲线；对于更高维空间，它是一个超曲面。
- **弱Pareto最优点**：如果一个设计点无法在所有目标上同时被改进，则称其为弱Pareto最优点。弱Pareto最优点不一定是Pareto最优点。

1.3 Pareto前沿生成

- 生成Pareto前沿的简单方法是随机采样设计点，然后筛选出非支配点。然而，这种方法效率低下且无法保证Pareto前沿的平滑性。
- 更有效的方法包括约束法、权重法和群体方法。

---

2. 约束方法（Constraint Methods）

约束方法通过将多目标优化问题转化为单目标优化问题来生成Pareto前沿。

2.1 约束法（Constraint Method）

- 约束法通过将所有目标（除了一个）作为约束条件，将多目标优化问题转化为单目标优化问题。
- 例如，假设有两个目标 $f_1(x)$ 和 $f_2(x)$，可以将 $f_2(x)$ 作为约束条件，优化 $f_1(x)$：

$$\begin{aligned} &\text{minimize} \quad f_1(x) \\ &\text{subject to} \quad f_2(x) \leq c_2 \end{aligned}$$

- 通过调整约束条件 c_2，可以生成Pareto前沿。

2.2 词典序法（Lexicographic Method）

- 词典序法根据目标的重要性顺序依次进行优化。每次优化时，保留之前优化目标的最优值作为约束条件。
- 例如，假设有三个目标 $f_1(x)$, $f_2(x)$, $f_3(x)$，按重要性顺序依次优化：
  1. 首先优化 $f_1(x)$，得到最优值 y_1^*；
  2. 然后优化 $f_2(x)$，约束 $f_1(x) \leq y_1^*$；
  3. 最后优化 $f_3(x)$，约束 $f_1(x) \leq y_1^*$ 且 $f_2(x) \leq y_2^*$。

---

3. 权重方法（Weight Methods）

权重方法通过为每个目标分配权重，将多目标优化问题转化为单目标优化问题。这是本章的重点内容。

3.1 加权求和法（Weighted Sum Method）

- 加权求和法通过为每个目标分配权重 w_i，将多目标函数转化为单目标函数：

$$f(x) = \sum_{i=1}^{m} w_i f_i(x)$$

- 权重 w_i 通常满足 $w_i \geq 0$ 且 $\sum_{i=1}^{m} w_i = 1$。
- 通过调整权重，可以生成Pareto前沿。然而，加权求和法无法处理非凸的Pareto前沿。

例子：

假设我们选择权重 $w_1 = 0.5$ 和 $w_2 = 0.5$，则单目标函数为：

$$f(x) = 0.5x^2 + 0.5(x-2)^2$$

通过求解这个单目标优化问题，可以得到一个Pareto最优解。例如，对 $f(x)$ 求导并令导数为零：

$$\frac{df}{dx} = x + (x-2) = 2x - 2 = 0 \implies x = 1$$

因此，$x = 1$ 是一个Pareto最优解。

3.2 目标规划（Goal Programming）

- 目标规划通过最小化目标函数值与目标点之间的 L_p 范数来生成Pareto前沿：

$$\text{minimize} \quad \|\mathbf{f}(x) - \mathbf{y}^{\text{goal}}\|_p$$

- 目标点通常是理想点（utopia point），即每个目标的最优值。
- p 是范数的阶数，用于衡量目标函数值与目标点之间的偏差。常见的取值包括：

- $p = 1$：最小化总偏差（绝对值和）；
- $p = 2$：最小化几何距离（欧几里得距离）；
- $p \to \infty$：最小化最大偏差。

例子：

假设目标点为 $\mathbf{y}^{\text{goal}} = [0, 0]$，选择 $p = 2$，则目标规划问题为：

$$\text{minimize} \quad \sqrt{x^2 + (x-2)^2}$$

通过求解这个优化问题，可以得到一个Pareto最优解。

3.3 加权指数求和法（Weighted Exponential Sum）

- 加权指数求和法结合了目标规划和加权求和法，通过引入指数权重来生成Pareto前沿：

$$f(x) = \sum_{i=1}^{m} w_i \left(f_i(x) - y_i^{\text{goal}} \right)^p$$

- p 是指数权重的幂次，用于调整目标函数值与目标点之间偏差的惩罚强度。较大的 p 会更关注最大偏差。

例子：

假设目标点为 $\mathbf{y}^{\text{goal}} = [0, 0]$，选择 $p = 2$，则加权指数求和法为：

$$f(x) = 0.5x^2 + 0.5(x-2)^2$$

通过求解这个优化问题，可以得到一个Pareto最优解。

3.4 加权最小最大法（Weighted Min-Max Method）

- 加权最小最大法通过最小化目标函数值与目标点之间的最大偏差来生成Pareto前沿：

$$f(x) = \max_i \left[w_i \left(f_i(x) - y_i^{\text{goal}} \right) \right]$$

- 这种方法可以处理非凸的Pareto前沿。

例子：

假设目标点为 $\mathbf{y}^{\text{goal}} = [0, 0]$，选择 $w_1 = 0.5$ 和 $w_2 = 0.5$，则加权最小最大法为：

$$f(x) = \max \left[0.5x^2, 0.5(x-2)^2 \right]$$

通过求解这个优化问题，可以得到一个Pareto最优解。

---

4. 多目标群体方法（Multiobjective Population Methods）

群体方法（如遗传算法）也可以用于多目标优化。通过调整群体方法，可以生成覆盖Pareto前沿的解集。

4.1 子群体（Subpopulations）

- 群体方法可以将群体划分为多个子群体，每个子群体针对不同的目标进行优化。通过子群体之间的交叉和变异，可以生成多样化的解集。

4.2 非支配排序（Nondomination Ranking）

- 非支配排序通过将群体中的个体按照非支配关系进行排序，生成Pareto前沿的近似解。

4.3 Pareto过滤器（Pareto Filters）

- Pareto过滤器用于在群体方法中维护一个近似Pareto前沿的解集。通过不断更新过滤器，可以确保群体方法生成的解集覆盖Pareto前沿。

4.4 小生境技术（Niche Techniques）

- 小生境技术通过惩罚目标空间中过于集中的个体，鼓励群体方法生成均匀分布的Pareto前沿解集。

---

5. 偏好引导（Preference Elicitation）

偏好引导通过专家的偏好信息，推断出合适的权重向量，从而将多目标优化问题转化为单目标优化问题。

5.1 模型识别（Model Identification）

- 通过专家的偏好信息，可以识别出合适的权重向量。常见的偏好引导方法包括二元偏好查询和排序查询。

5.2 配对查询选择（Paired Query Selection）

- 配对查询选择通过选择信息量最大的查询对，快速缩小权重向量的可行域。

5.3 设计选择（Design Selection）

- 设计选择通过最小化最坏情况下的目标值或最小化最大遗憾值，选择最终的设计方案。

---

6. 总结

- 多目标优化涉及多个目标之间的权衡，通常没有唯一的最优解。
- Pareto前沿代表了不同目标之间的最佳权衡。
- 通过约束法或权重法，可以将多目标优化问题转化为单目标优化问题。
- 群体方法可以生成覆盖Pareto前沿的解集。
- 通过专家的偏好信息，可以推断出合适的权重向量，从而选择最优设计方案。

第十四章 代理模型

1. 代理模型的拟合

核心思想：代理模型 $\hat{f}$ 通过参数 θ 来模拟真实的目标函数 f。我们通过调整参数 θ 来最小化模型预测值 $\hat{\mathbf{y}}$ 和真实值 $\mathbf{y}$ 之间的差异，通常使用 L_p 范数（如 L_2 范数，即最小化均方误差）。

例子：
假设我们有 3 个设计点 $X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}\}$，对应的函数评估值为 $\mathbf{y} = \{y^{(1)}, y^{(2)}, y^{(3)}\}$。代理模型的目标是通过调整参数 θ，使得模型的预测值 $\hat{\mathbf{y}} = \{\hat{f}_\theta(\mathbf{x}^{(1)}), \hat{f}_\theta(\mathbf{x}^{(2)}), \hat{f}_\theta(\mathbf{x}^{(3)})\}$ 尽可能接近真实值 $\mathbf{y}$。

---

2. 线性模型

核心思想：线性模型是最简单的代理模型，形式为 $\hat{f} = w_0 + \mathbf{w}^\top \mathbf{x}$。对于 n 维设计空间，线性模型有 $n+1$ 个参数，因此至少需要 $n+1$ 个样本来拟合。

例子：
假设我们有一个二维设计空间，设计点为 $X = \{(1,2), (2,3), (3,4)\}$，对应的函数评估值为 $\mathbf{y} = \{3, 5, 7\}$。我们可以构建设计矩阵 $\mathbf{X}$ 并求解线性回归问题来找到最优参数 θ。

设计矩阵：

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix}$$

通过最小二乘法，我们可以求解 $\theta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$，得到最优参数。

---

3. 基函数

核心思想：线性模型是基函数的一种特例。更一般的形式是线性组合基函数：

$$\hat{f}(x) = \sum_{i=1}^{q} \theta_i b_i(x)$$

其中 $b_i(x)$ 是基函数。常见的基函数包括多项式基函数、正弦基函数和径向基函数。

3.1 多项式基函数

核心思想：多项式基函数由设计向量的各分量幂次组成。通过泰勒级数展开，任何无限可微函数都可以用足够高阶的多项式近似。

例子：
在一维情况下，一个 k 次多项式模型的形式为：

$$\hat{f}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k$$

假设我们有 4 个设计点 $X = \{1, 2, 3, 4\}$，对应的函数评估值为 $\mathbf{y} = \{0, 5, 4, 6\}$。我们可以通过多项式基函数拟合这些数据。

3.2 正弦基函数

核心思想：任何有限域上的连续函数都可以用无限的正弦基函数表示。傅里叶级数可以用于构造这些基函数。

例子：
在一维情况下，傅里叶级数的基函数为：

$$b_0(x) = 1/2, \quad b_i^{(\sin)}(x) = \sin\left(\frac{2\pi i x}{b-a}\right), \quad b_i^{(\cos)}(x) = \cos\left(\frac{2\pi i x}{b-a}\right)$$

假设我们有一个函数 $f(x) = \sin(2x)\cos(10x)$，我们可以使用正弦基函数来拟合这个函数。

3.3 径向基函数

核心思想：径向基函数仅依赖于点与中心点的距离。常见的径向基函数包括线性、立方、薄板样条、高斯、多二次和逆多二次函数。

例子：
假设我们有设计点 $X = \{(1,2),(2,3),(3,4)\}$，我们可以使用这些点作为中心点构建径向基函数 $b_i(x) = \psi(\|x - x^{(i)}\|)$，其中 ψ 是径向基函数（如高斯函数 $\psi(r) = e^{-r^2/2\sigma^2}$）。

---

4. 拟合噪声目标函数

核心思想：当目标函数评估存在噪声时，复杂的模型可能会过度拟合噪声数据。为了获得更平滑的拟合，可以在回归问题中加入正则化项，如 L_2 正则化，以偏好权重较低的解决方案。

例子：
假设我们有带噪声的目标函数评估值 $\mathbf{y}$，我们可以通过加入正则化项 $\lambda\|\theta\|_2^2$ 来获得更平滑的拟合。最优参数向量可以通过 $\theta = (\mathbf{B}^\top\mathbf{B} + \lambda\mathbf{I})^{-1}\mathbf{B}^\top\mathbf{y}$ 计算。

---

5. 模型选择

核心思想：模型选择的目标是最小化泛化误差，即模型在整个设计空间上的预测误差。常见的估计泛化误差的方法包括留出法、交叉验证和自助法。

5.1 留出法

核心思想：将数据分为训练集和测试集，训练集用于拟合模型，测试集用于估计泛化误差。

例子：
假设我们有 10 个数据点，我们可以将其中 8 个点作为训练集，2 个点作为测试集。通过多次随机划分，我们可以获得泛化误差的估计。

5.2 交叉验证

核心思想：将数据分为 k 个子集，每次使用 $k-1$ 个子集训练模型，剩下的子集用于估计泛化误差。

例子：
假设我们有 10 个数据点，我们可以进行 5 折交叉验证，每次使用 8 个点训练模型，2 个点测试模型。通过 5 次不同的划分，我们可以获得泛化误差的估计。

5.3 自助法

核心思想：通过有放回地采样生成多个训练集，每个训练集用于拟合模型，并在原始数据集上评估泛化误差。

例子：
假设我们有 10 个数据点，我们可以生成 10 个自助样本，每个样本包含 10 个点（可能有重复）。通过在这些样本上训练模型并在原始数据上测试，我们可以获得泛化误差的估计。

---

6. 总结

- **代理模型**：是目标函数的近似，可以替代真实的目标函数进行优化。
- **基函数**：许多代理模型可以用基函数的线性组合表示。
- **模型选择**：涉及偏差-方差权衡，低复杂度模型可能无法捕捉重要趋势，而高复杂度模型可能过度拟合噪声。
- **泛化误差**：可以通过留出法、交叉验证和自助法等方法进行估计。

练习：

1：推导线性回归问题的正规方程

问题：推导线性回归问题的最优解，即正规方程。

解答：
线性回归问题的目标是最小化均方误差：

$$minimize_{\theta} \|\mathbf{y} - \mathbf{X}\theta\|_2^2$$

其中，$\mathbf{X}$ 是设计矩阵，$\mathbf{y}$ 是目标值向量，θ 是参数向量。

将目标函数展开：

$$\|\mathbf{y} - \mathbf{X}\theta\|_2^2 = (\mathbf{y} - \mathbf{X}\theta)^\top (\mathbf{y} - \mathbf{X}\theta)$$

对 θ 求梯度并设为零：

$$\nabla_\theta (\mathbf{y} - \mathbf{X}\theta)^\top (\mathbf{y} - \mathbf{X}\theta) = -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\theta) = 0$$

整理得到正规方程：

$$\mathbf{X}^\top \mathbf{X}\theta = \mathbf{X}^\top \mathbf{y}$$

如果 $\mathbf{X}^\top \mathbf{X}$ 可逆，则最优解为：

$$\theta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

2：何时使用多项式模型与线性回归模型

问题：讨论何时使用多项式模型与线性回归模型。

解答：

- **线性回归模型**：适用于目标函数与输入变量之间存在线性关系的情况。线性模型简单且计算成本低，但无法捕捉非线性关系。

- **多项式模型**：适用于目标函数与输入变量之间存在非线性关系的情况。通过引入高阶项，多项式模型可以拟合更复杂的函数，但容易过拟合，尤其是在数据量较少时。

选择依据：

1. **数据量**：如果数据量较少，优先选择线性模型，避免过拟合。
2. **问题复杂度**：如果目标函数明显是非线性的，可以使用多项式模型。
3. **计算资源**：多项式模型的计算成本较高，尤其是在高维情况下。

3：为什么线性回归问题有时需要使用优化技术而不是解析解

问题：解释为什么线性回归问题有时需要使用优化技术而不是解析解。

解答：
线性回归问题的解析解为：

$$\theta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

但在以下情况下，解析解可能不适用：

1. **矩阵不可逆**：当 $\mathbf{X}^\top \mathbf{X}$ 不可逆时，解析解无法计算。这种情况可能发生在数据点线性相关或数据量少于特征数时。
2. **计算复杂度高**：当设计矩阵 $\mathbf{X}$ 非常大时，计算 $(\mathbf{X}^\top \mathbf{X})^{-1}$ 的复杂度很高，甚至不可行。
3. **数值稳定性**：当 $\mathbf{X}^\top \mathbf{X}$ 接近奇异矩阵时，解析解可能数值不稳定。

在这些情况下，可以使用优化技术（如梯度下降法）来求解线性回归问题，避免直接计算逆矩阵。

4：计算留一法交叉验证的均方误差

问题：假设我们在四个点 $x = \{1, 2, 3, 4\}$ 上评估目标函数，得到 $y = \{0, 5, 4, 6\}$。我们拟合多项式模型 $\hat{f}(x) = \sum_{i=0}^{k} \theta_i x^i$，计算 k 从 0 到 4 时的留一法交叉验证均方误差，并选择最佳的 k 和 θ。

解答：
留一法交叉验证的步骤如下：

1. 对于每个 k，依次将每个数据点作为测试集，其余数据点作为训练集。
2. 在训练集上拟合多项式模型，计算测试集上的预测误差。
3. 对所有测试集的误差取平均，得到均方误差（MSE）。

计算过程：
以 $k = 1$ 为例：

- 测试集为 $x = 1$，训练集为 $x = \{2, 3, 4\}$。
- 在训练集上拟合线性模型 $\hat{f}(x) = \theta_0 + \theta_1 x$。
- 计算测试集上的预测误差 $(y - \hat{f}(x))^2$。

重复上述过程，计算 $k = 0, 1, 2, 3, 4$ 时的 MSE。

结果：
通过计算，可以得到不同 k 值下的 MSE。选择 MSE 最小的 k 值作为最佳模型。

最佳 k 和 θ：
假设 $k = 2$ 时 MSE 最小，则最佳模型为：

$$\hat{f}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

通过最小二乘法拟合训练数据，得到最优参数 θ。

第十五章 概率代理模型

1. 高斯分布（Gaussian Distribution）

高斯分布是高斯过程的基础。多元高斯分布由均值向量 μ 和协方差矩阵 Σ 参数化，其概率密度函数为：

$$\mathcal{N}(\mathbf{x} \mid \mu, \Sigma) = (2\pi)^{-n/2}|\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

关键性质：

- **边际分布**：多元高斯分布的边际分布仍然是高斯分布。
- **条件分布**：给定部分变量的值，剩余变量的条件分布也是高斯分布。

例15.1：多元高斯的边际分布和条件分布

问题描述：
假设我们有一个二维高斯分布：

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}\right)$$

计算：

1. x_1 和 x_2 的边际分布。
2. 在 $x_2 = 2$ 的条件下，x_1 的条件分布。

解答：

1. **边际分布**：
   - x_1 的边际分布为：
     $$x_1 \sim \mathcal{N}(0, 3)$$
   - x_2 的边际分布为：
     $$x_2 \sim \mathcal{N}(1, 2)$$
2. **条件分布**：
   - 条件分布的均值和协方差通过公式计算：
     $$\mu_{x_1|x_2=2} = 0 + 1 \cdot 2^{-1} \cdot (2-1) = 0.5$$
     $$\Sigma_{x_1|x_2=2} = 3 - 1 \cdot 2^{-1} \cdot 1 = 2.5$$
   - 因此，条件分布为：
     $$x_1|(x_2=2) \sim \mathcal{N}(0.5, 2.5)$$

意义：
这个例子展示了如何从联合高斯分布中提取边际分布和条件分布，为后续高斯过程的预测奠定了基础。

---

2. 高斯过程（Gaussian Processes）

高斯过程是函数的概率分布。对于任何有限的设计点集合 $\{x^{(1)}, \ldots, x^{(m)}\}$，其对应的函数值 $\{y_1, \ldots, y_m\}$ 服从多元高斯分布：

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x^{(1)}) \\ \vdots \\ m(x^{(m)}) \end{bmatrix}, \begin{bmatrix} k(x^{(1)}, x^{(1)}) & \cdots & k(x^{(1)}, x^{(m)}) \\ \vdots & \ddots & \vdots \\ k(x^{(m)}, x^{(1)}) & \cdots & k(x^{(m)}, x^{(m)}) \end{bmatrix} \right)$$

其中：

- $m(x)$ 是均值函数，通常假设为零均值。
- $k(x, x')$ 是核函数（协方差函数），控制函数的平滑性。

常用核函数：

- **平方指数核（Squared Exponential Kernel）**：

  $$k(x, x') = \exp\left(-\frac{(x - x')^2}{2\ell^2} \right)$$

  其中，ℓ 是长度尺度参数，控制函数的平滑程度。
- **Matern核**：适用于非平滑函数。
- **线性核、多项式核**等。

例15.2：高斯过程的核函数推导

问题描述：
假设我们使用平方指数核：

$$k_{ff}(x, x') = \exp\left(-\frac{1}{2} \|x - x'\|^2 \right)$$

推导其他核函数 $k_{f\nabla}(x, x')$、$k_{\nabla f}(x, x')$ 和 $k_{\nabla\nabla}(x, x')$。

解答：

1. $k_{f\nabla}(x, x')$ 是函数值与梯度的协方差：

   $$k_{f\nabla}(x, x') = \frac{\partial}{\partial x'} k_{ff}(x, x') = -(x - x') \exp\left(-\frac{1}{2} \|x - x'\|^2 \right)$$

2. $k_{\nabla\nabla}(x, x')$ 是梯度与梯度的协方差：

   $$k_{\nabla\nabla}(x, x') = \frac{\partial^2}{\partial x \partial x'} k_{ff}(x, x') = \left((x - x')^2 - 1 \right) \exp\left(-\frac{1}{2} \|x - x'\|^2 \right)$$

意义：
这个例子展示了如何从基本核函数推导出高阶核函数，为引入梯度信息奠定了基础。

---

3. 梯度信息（Gradient Measurements）

高斯过程可以扩展到包含梯度信息的情况。假设我们有函数值 y 和梯度 ∇y，联合分布为：

$$\begin{bmatrix} y \\ \nabla y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}_f \\ \mathbf{m}_\nabla \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{f\nabla} \\ \mathbf{K}_{\nabla f} & \mathbf{K}_{\nabla\nabla} \end{bmatrix} \right)$$

例15.3：构建包含梯度信息的协方差矩阵

问题描述：
假设我们在两个点 $x^{(1)}$ 和 $x^{(2)}$ 处评估了函数值和梯度，现在需要预测新点 $\hat{x}$ 处的函数值。

解答：

1. 构建联合分布的协方差矩阵：

$$\begin{bmatrix} k_{ff}(\hat{x},\hat{x}) & k_{ff}(\hat{x},x^{(1)}) & k_{ff}(\hat{x},x^{(2)}) & k_{f\nabla}(\hat{x},x^{(1)}) & k_{f\nabla}(\hat{x},x^{(2)}) \\ k_{ff}(x^{(1)},\hat{x}) & k_{ff}(x^{(1)},x^{(1)}) & k_{ff}(x^{(1)},x^{(2)}) & k_{f\nabla}(x^{(1)},x^{(1)}) & k_{f\nabla}(x^{(1)},x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

2. 使用条件分布公式计算预测值。

意义：
这个例子展示了如何利用梯度信息构建更精确的高斯过程模型。

---

4. 噪声测量（Noisy Measurements）

在实际应用中，函数评估可能包含噪声。我们可以将噪声建模为 $y = f(x) + z$，其中 z 是零均值的高斯噪声。通过引入噪声方差 v，我们可以调整预测的不确定性。

噪声情况下的联合分布：

$$\begin{bmatrix} \hat{\mathbf{y}} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}(X^*) \\ \mathbf{m}(X) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(X^*,X^*) & \mathbf{K}(X^*,X) \\ \mathbf{K}(X,X^*) & \mathbf{K}(X,X) + v\mathbf{I} \end{bmatrix} \right)$$

---

5. 习题讲解

习题1：高斯过程的复杂性

问题：高斯过程在优化过程中如何随着样本的增加而增加复杂性?
解答：高斯过程的计算复杂度主要来自于协方差矩阵的求逆操作，其复杂度为 $O(n^3)$，其中 n 是样本数量。随着样本的增加，计算量会显著增加。

习题2：计算复杂度

问题：高斯过程预测的计算复杂度如何随数据点的增加而变化?
解答：预测的计算复杂度为 $O(n^2)$，其中 n 是数据点的数量。

习题3：高斯过程的预测

问题：考虑函数 $f(x) = \sin(x)/(x^2+1)$，使用高斯过程预测其值，并计算95%置信区间。
解答：通过高斯过程的预测公式，可以计算出预测值和置信区间。

---

6. 总结

- 高斯过程是函数的概率分布，能够量化预测的不确定性。
- 核函数的选择影响函数的平滑性。
- 通过引入梯度信息和噪声测量，可以进一步提高模型的预测精度。
- 例15.1、15.2和15.3分别展示了边际分布、核函数推导和梯度信息的应用。