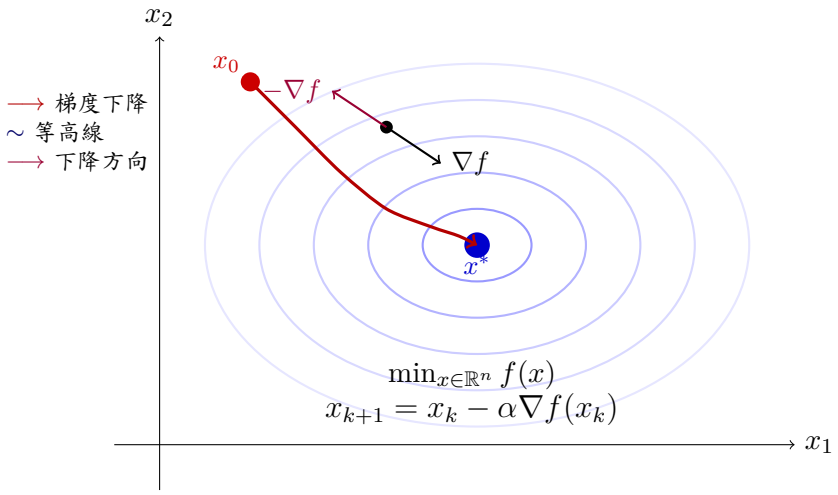
數學建模補充教材

無約束優化

理論與應用



程淑鍵 Kenneth

最後更新：July 7, 2025

透過數學優化達成優雅解決方案

Contents

Chapter 1

無約束優化基礎

1.1 介紹

無約束優化構成了許多數學建模應用的基石，從機器學習參數調優到工程設計優化。本章建立了理解優化演算法及其實際實現所需的理論基礎。

定義 **1.1** (無約束優化問題)**.** 無約束優化問題尋求找到：

$$\min_{x \in \mathbb{R}^n} f(x)$$

其中 $f : \mathbb{R}^n \to \mathbb{R}$ 是目標函數，$x \in \mathbb{R}^n$ 是決策變數向量。

1.2 最優性條件

理解何時一個點是最優的對於開發有效演算法至關重要。

定理

一階必要條件 **(FONC)**
設 $f : \mathbb{R}^n \to \mathbb{R}$ 連續可微。若 x^* 是 f 的局部最小值，則：

$$\nabla f(x^*) = 0$$

這樣的點稱為駐點或臨界點。

定理

二階必要條件 **(SONC)**
設 $f : \mathbb{R}^n \to \mathbb{R}$ 二次連續可微（即 $f \in C^2$）。若 x^* 是 f 的局部最小值，則：

1. $\nabla f(x^*) = 0$
2. $\nabla^2 f(x^*) \succeq 0$（Hessian 矩陣為半正定）

定理

二階充分條件 **(SOSC)**
設 $f:\mathbb{R}^n \to \mathbb{R}$ 二次連續可微。若：

1. $\nabla f(x^*) = 0$
2. $\nabla^2 f(x^*) \succ 0$（Hessian 矩陣為正定）

則 x^* 是 f 的嚴格局部最小值。

1.3 凸性與全域最優性

定義 **1.2** (凸函數)**.** 函數 $f:\mathbb{R}^n \to \mathbb{R}$ 是凸的，如果對於所有 $x, y \in \mathbb{R}^n$ 和 $\lambda \in [0,1]$：

$$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda) f(y)$$

定理

凸函數的全域最優性
若 f 是凸的且 x^* 是駐點（即 $\nabla f(x^*) = 0$），則 x^* 是 f 的全域最小值。

1.4 實際範例

範例 **1.1** (二次函數)**.** 考慮二次函數：

$$f(x) = \frac{1}{2} x^T Q x + c^T x + d$$

其中 $Q \in \mathbb{R}^{n\times n}$ 是對稱的，$c \in \mathbb{R}^n$，$d \in \mathbb{R}$。
梯度為：$\nabla f(x) = Qx + c$ Hessian 為：$\nabla^2 f(x) = Q$
若 $Q \succ 0$，則 f 是嚴格凸的，唯一的全域最小值為：

$$x^* = -Q^{-1} c$$

實務應用

投資組合優化
在金融領域，馬可維茲投資組合優化問題尋求在達到目標報酬的同時最小化風險。問題可以公式化為：

$$\min_w \frac{1}{2} w^T \Sigma w$$

受制於 $\sum_{i=1}^n w_i = 1$ 和 $\mu^T w = r_{\text{target}}$
其中 w 是投資組合權重向量，Σ 是資產報酬的共變異數矩陣，μ 是期望報酬向量。

Python 程式碼

```
import numpy as np
import matplotlib.pyplot as plt
```

```python
from scipy.optimize import minimize

def quadratic_function(x, Q, c):
    """
    計算二次函數 f(x) = 0.5 * x^T Q x + c^T x
    """
    return 0.5 * x.T @ Q @ x + c.T @ x

def quadratic_gradient(x, Q, c):
    """
    計算二次函數的梯度
    """
    return Q @ x + c

# 範例：2D 二次函數
Q = np.array([[2, 0.5], [0.5, 1]])
c = np.array([1, -2])

# 解析解
x_optimal = -np.linalg.solve(Q, c)
print(f"最優解: {x_optimal}")
print(f"最優值: {quadratic_function(x_optimal, Q, c)}")

# 驗證最優性條件
grad_at_optimal = quadratic_gradient(x_optimal, Q, c)
print(f"最優點的梯度: {grad_at_optimal}")
print(f"Hessian 特徵值: {np.linalg.eigvals(Q)}")
```

探索

視覺化練習

創建上述二次函數的等高線圖，並驗證梯度指向最陡上升方向。觀察等高線如何揭示函數的形狀和最小值的位置。

練習 **1.1.** 證明若 $f : \mathbb{R}^n \to \mathbb{R}$ 二次連續可微且為凸函數，則對於所有 $x \in \mathbb{R}^n$ 有 $\nabla^2 f(x) \succeq 0$。

練習 **1.2.** 考慮函數 $f(x, y) = x^4 + y^4 - 4xy$。找出所有駐點並使用二階條件分類它們。

Chapter 2

基於梯度的優化演算法

2.1 梯度下降法

梯度下降是無約束優化的基本演算法，是許多高級方法的基礎。

定義 **2.1** (梯度下降演算法)**.** 從初始點 x_0 開始，梯度下降法透過以下方式產生序列 $\{x_k\}$：

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

其中 $\alpha_k > 0$ 是第 k 次迭代的步長（學習率）。

2.1.1 步長選擇

步長的選擇對收斂行為有關鍵影響：

- 固定步長：對所有 k 有 $\alpha_k = \alpha$
- 精確線搜索：$\alpha_k = \arg\min_{\alpha>0} f(x_k - \alpha \nabla f(x_k))$
- **Armijo** 回溯：選擇 α_k 滿足充分下降條件

定理

梯度下降的收斂性
假設 f 連續可微且 $\inf f > -\infty$。若步長滿足：

$$\sum_{k=0}^{\infty} \alpha_k = \infty \quad \text{且} \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty$$

則 $\lim_{k\to\infty} \|\nabla f(x_k)\| = 0$。

2.2 牛頓法

牛頓法使用二階資訊來實現更快的收斂。

定義 **2.2** (牛頓法)**.** 牛頓迭代為：

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

假設 $\nabla^2 f(x_k)$ 是正定的。

定理

牛頓法的二次收斂
假設 f 二次連續可微，x^* 是具有 $\nabla^2 f(x^*) \succ 0$ 的局部最小值，且 x_0 足夠接近 x^*。則牛頓法二次收斂到 x^*：

$$\|x_{k+1} - x^*\| \leq C\|x_k - x^*\|^2$$

對某個常數 $C > 0$。

2.3 擬牛頓法

擬牛頓法近似 Hessian 以減少計算成本，同時保持超線性收斂。

2.3.1 BFGS 演算法

Broyden-Fletcher-Goldfarb-Shanno (BFGS) 方法是最受歡迎的擬牛頓演算法。

定義 **2.3** (BFGS 更新)**.** 從 $B_0 = I$（或另一個正定矩陣）開始，BFGS 方法更新 Hessian 近似：

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

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

實務應用

機器學習應用
優化演算法在訓練機器學習模型中至關重要：

- 線性回歸：最小化 $\|Ax - b\|_2^2$
- 邏輯回歸：最小化交叉熵損失
- 神經網路：帶梯度下降變體的反向傳播

廣泛用於深度學習的 Adam 優化器結合了梯度下降與動量和自適應步長的思想。

Python 程式碼

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

def rosenbrock(x):
    """Rosenbrock 函數 - 經典測試函數"""
    return 100 * (x[1] - x[0]**2)**2 + (1 - x[0])**2

def rosenbrock_grad(x):
    """Rosenbrock 函數的梯度"""
    grad = np.zeros_like(x)
    grad[0] = -400 * x[0] * (x[1] - x[0]**2) - 2 * (1 - x[0])
```

```
    grad[1] = 200 * (x[1] - x[0]**2)
    return grad

def gradient_descent(f, grad_f, x0, alpha=0.001, max_iter=1000, tol=1e-6):
    """
    梯度下降實現
    """
    x = x0.copy()
    trajectory = [x.copy()]

    for i in range(max_iter):
        grad = grad_f(x)
        if np.linalg.norm(grad) < tol:
            break
        x = x - alpha * grad
        trajectory.append(x.copy())

    return x, np.array(trajectory)

# 範例：優化 Rosenbrock 函數
x0 = np.array([-1.0, 1.0])
x_opt, trajectory = gradient_descent(rosenbrock, rosenbrock_grad, x0)

print(f"起始點: {x0}")
print(f"最優點: {x_opt}")
print(f"最優值: {rosenbrock(x_opt)}")
print(f"迭代次數: {len(trajectory)}")

# 與 scipy 實現比較
result = minimize(rosenbrock, x0, method='BFGS', jac=rosenbrock_grad)
print(f"SciPy BFGS 結果: {result.x}")
print(f"SciPy BFGS 值: {result.fun}")
```

2.4 收斂分析

2.4.1 線性收斂

定義 **2.4** (線性收斂). 序列 $\{x_k\}$ 線性收斂到 x^*，如果存在常數 $C > 0$ 和 $0 < r < 1$ 使得：

$$\|x_{k+1} - x^*\| \leq r\|x_k - x^*\|$$

常數 r 稱為收斂率。

定理

梯度下降的收斂率
對於具有 Lipschitz 連續梯度的強凸函數，帶適當步長的梯度下降達到線性收斂，收斂率為：

$$r = \frac{\kappa - 1}{\kappa + 1}$$

其中 $\kappa = \frac{L}{\mu}$ 是條件數（L 是 Lipschitz 常數，μ 是強凸性參數）。

2.4.2 超線性和二次收斂

定義 **2.5** (超線性收斂). 序列 $\{x_k\}$ 超線性收斂到 x^*，如果：

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0$$

探索

數值實驗
實現並比較以下方法的收斂行為：

1. 固定步長的梯度下降
2. 帶線搜索的梯度下降
3. 牛頓法
4. BFGS 方法

在不同條件數的函數上測試，觀察收斂率如何依賴於問題特性。

Python 程式碼

```python
def visualize_optimization_path(f, trajectory, title="優化路徑"):
    """
    在等高線圖上視覺化優化軌跡
    """
    # 創建等高線圖的網格
    x_range = np.linspace(-2, 2, 100)
    y_range = np.linspace(-1, 3, 100)
    X, Y = np.meshgrid(x_range, y_range)
    Z = np.zeros_like(X)

    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Z[i, j] = f([X[i, j], Y[i, j]])

    plt.figure(figsize=(10, 8))
    plt.contour(X, Y, Z, levels=50, alpha=0.6)
    plt.colorbar(label='函數值')

    # 繪製優化路徑
    plt.plot(trajectory[:, 0], trajectory[:, 1], 'ro-',
             markersize=3, linewidth=1, alpha=0.8)
    plt.plot(trajectory[0, 0], trajectory[0, 1], 'go',
             markersize=10, label='起始')
    plt.plot(trajectory[-1, 0], trajectory[-1, 1], 'ro',
             markersize=10, label='結束')

    plt.xlabel('x ')
    plt.ylabel('x ')
    plt.title(title)
    plt.legend()
```

```
    plt.grid(True, alpha=0.3)
    plt.show()

# 視覺化軌跡
visualize_optimization_path(rosenbrock, trajectory,
                            "Rosenbrock 函數上的梯度下降")
```

練習 **2.1.** 實現 Armijo 回溯線搜索，並在 Rosenbrock 函數上比較其與固定步長梯度下降的性能。

練習 **2.2.** 證明對於二次函數 $f(x) = \frac{1}{2}x^T Qx + c^T x$，其中 $Q \succ 0$，牛頓法不論起始點如何都能在一次迭代內收斂。

練習 **2.3.** 推導 L-BFGS 演算法（有限記憶 BFGS）並為儲存完整 Hessian 近似不可行的大規模問題實現它。

Chapter 3

進階優化方法

3.1 共軛梯度法

共軛梯度法結合了梯度下降的簡單性和牛頓法對二次函數的快速收斂特性。

定義 **3.1** (共軛方向)**.** 向量集合 $\{d_0, d_1, \ldots, d_{k-1}\}$ 相對於正定矩陣 A 是共軛的，如果：

$$d_i^T A d_j = 0 \quad \text{對所有} i \neq j$$

定義 **3.2** (共軛梯度演算法)**.** 共軛梯度法使用以下方式產生序列 $\{x_k\}$：

$$x_{k+1} = x_k + \alpha_k d_k \tag{3.1}$$
$$d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k \tag{3.2}$$

其中 α_k 由線搜索確定，β_k 選擇以確保共軛性。

3.1.1 Fletcher-Reeves 和 Polak-Ribière 公式

β_k 的不同選擇導致不同的共軛梯度變體：

- **Fletcher-Reeves**：$\beta_k^{FR} = \frac{\|\nabla f(x_{k+1})\|^2}{\|\nabla f(x_k)\|^2}$
- **Polak-Ribière**：$\beta_k^{PR} = \frac{\nabla f(x_{k+1})^T (\nabla f(x_{k+1}) - \nabla f(x_k))}{\|\nabla f(x_k)\|^2}$
- **Hestenes-Stiefel**：$\beta_k^{HS} = \frac{\nabla f(x_{k+1})^T (\nabla f(x_{k+1}) - \nabla f(x_k))}{d_k^T (\nabla f(x_{k+1}) - \nabla f(x_k))}$

定理

二次函數的有限終止性
對於二次函數 $f(x) = \frac{1}{2} x^T A x - b^T x$，其中 $A \succ 0$，共軛梯度法在最多 n 步內終止於精確解，其中 n 是問題的維度。

3.2 信賴域方法

信賴域方法透過基於局部模型可信度約束步長，提供線搜索方法的穩健替代方案。

定義 **3.3** (信賴域子問題)**.** 在每次迭代中，信賴域方法求解：

$$\min_d m_k(d) = f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T B_k d$$

受制於 $\|d\| \leq \Delta_k$，其中 $\Delta_k > 0$ 是信賴域半徑，B_k 近似 $\nabla^2 f(x_k)$。

3.3 加速梯度法

加速方法使用動量來實現比標準梯度下降更好的收斂率。

3.3.1 重球法

重球法在梯度下降中加入動量：

定義 **3.4** (重球法)**.**

$$x_{k+1} = x_k - \alpha \nabla f(x_k) + \beta (x_k - x_{k-1})$$

其中 $\alpha > 0$ 是步長，$\beta \geq 0$ 是動量參數。

3.3.2 Nesterov 加速梯度

Nesterov 方法在「前瞻」點評估梯度：

定義 **3.5** (Nesterov 加速梯度)**.**

$$y_k = x_k + \frac{k-1}{k+2}(x_k - x_{k-1}) \tag{3.3}$$

$$x_{k+1} = y_k - \alpha \nabla f(y_k) \tag{3.4}$$

定理

最優收斂率
對於具有 Lipschitz 連續梯度的凸函數，Nesterov 加速梯度達到最優收斂率 $O(1/k^2)$，相比標準梯度下降的 $O(1/k)$。

3.4 現代自適應方法

自適應方法自動為每個參數調整學習率，使它們在機器學習應用中特別有效。

3.4.1 AdaGrad

AdaGrad 基於歷史梯度適應學習率：

定義 **3.6** (AdaGrad 演算法)**.**

$$G_k = G_{k-1} + \nabla f(x_k) \odot \nabla f(x_k) \tag{3.5}$$

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{G_k + \epsilon}} \odot \nabla f(x_k) \tag{3.6}$$

其中 $\odot$ 表示逐元素乘法，$\epsilon > 0$ 是小常數。

3.4.2 RMSprop

RMSprop 使用指數移動平均來防止學習率下降太快：

定義 **3.7** (RMSprop 演算法)**.**

$$v_k = \rho v_{k-1} + (1-\rho)\nabla f(x_k) \odot \nabla f(x_k) \tag{3.7}$$

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{v_k + \epsilon}} \odot \nabla f(x_k) \tag{3.8}$$

其中 $\rho \in (0,1)$ 是衰減率。

3.4.3 Adam 優化器

Adam 結合 AdaGrad 和 RMSprop 與動量的優點：

定義 **3.8** (Adam 演算法)**.**

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\nabla f(x_k) \tag{3.9}$$

$$v_k = \beta_2 v_{k-1} + (1-\beta_2)\nabla f(x_k) \odot \nabla f(x_k) \tag{3.10}$$

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k} \tag{3.11}$$

$$\hat{v}_k = \frac{v_k}{1-\beta_2^k} \tag{3.12}$$

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{\hat{v}_k + \epsilon}} \odot \hat{m}_k \tag{3.13}$$

其中 $\beta_1, \beta_2 \in (0,1)$ 是指數衰減率。

實務應用

深度學習應用
現代自適應優化器在訓練深度神經網路中至關重要：

- 電腦視覺：用於圖像分類的 CNN 通常使用 Adam 或 RMSprop
- 自然語言處理：Transformer 模型通常採用帶學習率調度的 Adam
- 強化學習：策略梯度方法受益於自適應步長
- 生成模型：GAN 和 VAE 需要仔細的優化器調優以實現穩定訓練

優化器的選擇可以顯著影響訓練速度和最終模型性能。

探索

優化器比較研究
設計實驗來比較不同的優化方法：

1. 在各種函數類型上測試（凸、非凸、病態條件）
2. 分析收斂率和計算成本

3. 研究對超參數的敏感性
4. 調查在高維問題上的性能
5. 比較實際時間與迭代次數

這將提供何時在實踐中使用每種方法的見解。

練習 **3.1.** 實現 AdaMax 優化器（Adam 的變體）並在 Rosenbrock 函數上比較其與 Adam 的性能。

練習 **3.2.** 證明對於強凸函數，Nesterov 加速梯度達到 $O(1/k^2)$ 收斂率。

練習 **3.3.** 設計一個混合優化演算法，根據收斂行為自動在不同方法之間切換。