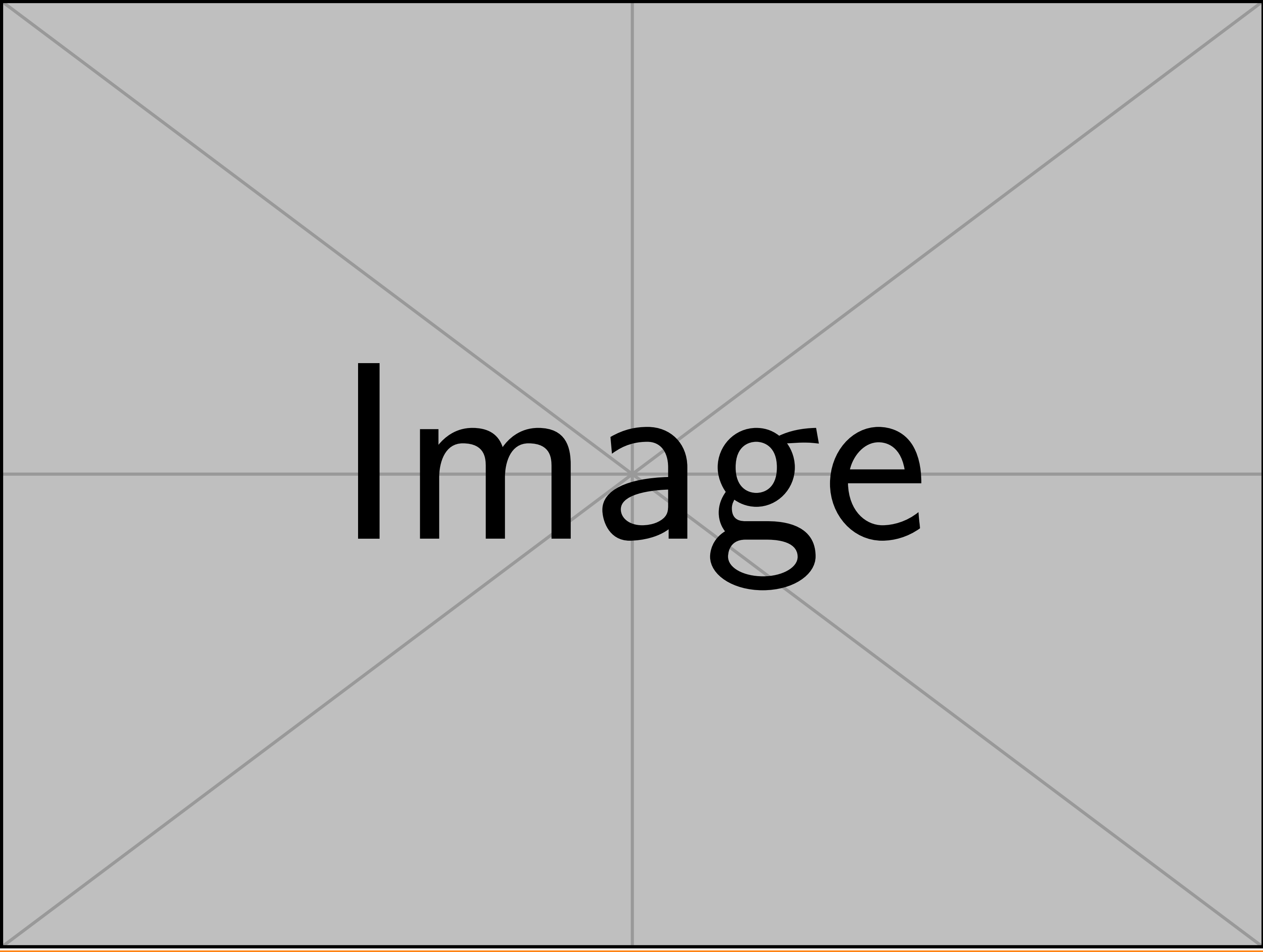

作者：龚梓阳

时间：2021 年 10 月 26 日

目录

第 1 章 引论

1.1 预备知识

1.1.1 范数

定义 1.1 (范数)

设 X 是域 $\mathbb{K}$（实数域或复数域）上的线性空间，函数 $\|\cdot\|: X \to \mathbb{R}$ 满足：

1. （正定性）$\forall x \in X, \|x\| \geq 0$；$\|x\| = 0 \iff x = 0$；
2. （齐次性）$\forall x \in X, \alpha \in \mathbb{K}, \|\alpha x\| = |\alpha| \cdot \|x\|$；
3. （次可加性）$\forall x, y \in X, \|x + y\| \leq \|x\| + \|y\|$。

则称 $\|\cdot\|$ 是 X 上的一个范数。

例题 1.1常见范数 常见范数有：

1. 空间 $\mathbb{R}$：$\forall x \in \mathbb{X}$

$$\|x\| := |x| \tag{1.1}$$

即范数 $\|x\|$ 为 x 的绝对值。

2. 空间 $\mathbb{R}^n$：$\forall \mathbf{x} \in \mathbb{R}^n, \mathbf{x} = (x_1, x_2, \ldots, x_n)'$

(a). 欧几里得范数：

$$\|\mathbf{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2} \tag{1.2}$$

(b). l_p 范数 $(p \geq 1)$：

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \tag{1.3}$$

若 $p = 1$，则

$$\|\mathbf{x}\|_1 := \sum_{i=1}^{n} |x_i| \tag{1.4}$$

若 $p = \infty$，则

$$\|\mathbf{x}\|_\infty := \max_{i=1,\ldots,n} |x_i| \tag{1.5}$$

(c). l_0 范数：

$$\|\mathbf{x}\|_0 = |\{x_i : x_i \neq 0, i = 1, \ldots, n\}| \tag{1.6}$$

即向量 $\mathbf{x}$ 中分量 x_i 不为零的个数。

1.1.2 梯度向量与 Hessien 矩阵

定义 1.2 (梯度向量)

对于多变量函数 $f:\mathbb{R}^n\to\mathbb{R}$，即 $f(\mathbf{x})=f(x_1,x_2,\cdots,x_n)$，如果 f 在点 $\mathbf{x}_0$ 关于每一个变量 x_i 都有偏导数 $\frac{\partial f}{\partial x_i}(\mathbf{x}_0)$ 存在，则在点 $\mathbf{x}$ 上，这些偏导数定义了一个向量:

$$\nabla f(\mathbf{x}_0)=\left(\frac{\partial f}{\partial x_1}(\mathbf{x}_0),\ldots,\frac{\partial f}{\partial x_n}(\mathbf{x}_0)\right)' \tag{1.7}$$

该向量称为 f 在点 $\mathbf{x}_0$ 的梯度向量。♣

定义 1.3 (Hessian 矩阵)

对于多变量函数 $f:\mathbb{R}^n\to\mathbb{R}$, 如果 f 在点 $\mathbf{x}_0$ 的所有二阶偏导数都存在，那么函数 f 在点 $\mathbf{x}_0$ 的 Hessian 矩阵为

$$\boldsymbol{H}(\mathbf{x}_0)=\begin{bmatrix}\frac{\partial^2 f}{\partial x_1^2}(\mathbf{x}_0) & \frac{\partial^2 f}{\partial x_1\partial x_2}(\mathbf{x}_0) & \cdots & \frac{\partial^2 f}{\partial x_1\partial x_n}(\mathbf{x}_0)\\ \frac{\partial^2 f}{\partial x_2\partial x_1}(\mathbf{x}_0) & \frac{\partial^2 f}{\partial x_2^2}(\mathbf{x}_0) & \cdots & \frac{\partial^2 f}{\partial x_2\partial x_n}(\mathbf{x}_0)\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial^2 f}{\partial x_n\partial x_1}(\mathbf{x}_0) & \frac{\partial^2 f}{\partial x_n\partial x_2}(\mathbf{x}_0) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(\mathbf{x}_0)\end{bmatrix} \tag{1.8}$$

或使用下标记号表示为

$$\boldsymbol{H}_{ij}(\mathbf{x}_0)=\frac{\partial^2 f}{\partial x_i\partial x_j}(\mathbf{x}_0) \tag{1.9}$$

♣

1.1.3 凸集与凸函数

定义 1.4 (凸集)

设集合 $C\subset\mathbb{R}^n$。若对 $\forall\mathbf{x},\mathbf{y}\in C$，有

$$\theta\mathbf{x}+(1-\theta)\mathbf{y}\in C,\quad \theta\in[0,1] \tag{1.10}$$

则称 C 为凸集。♣

定义 1.5 (凸函数)

设集合 $C\subset\mathbb{R}^n$ 为非空凸集, 函数 $f:C\to\mathbb{R}$。若对 $\forall\mathbf{x},\mathbf{y}\in C$，有

$$f(\theta\mathbf{x}+(1-\theta)\mathbf{y})\le\theta f(\mathbf{x})+(1-\theta)f(\mathbf{y}),\quad \theta\in[0,1] \tag{1.11}$$

则称 f 为 C 上的凸函数。若上述不等式对 $\mathbf{x}\neq\mathbf{y}$ 严格成立，则称 f 为 C 上的严格凸函数。♣

定理 1.1 (凸函数的一阶判定条件)

设集合 $C \subset \mathbb{R}^n$ 为非空开凸集，函数 $f : C \to \mathbb{R}$ 可微，则

1. $f(x)$ 是凸函数当且仅当对 $\forall \mathbf{x}, \mathbf{y} \in C$, 有

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})'(\mathbf{y} - \mathbf{x}) \tag{1.12}$$

2. $f(\mathbf{x})$ 是严格凸函数当且仅当对 $\forall \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}$, 有

$$f(\mathbf{y}) > f(\mathbf{x}) + \nabla f(\mathbf{x})'(\mathbf{y} - \mathbf{x}) \tag{1.13}$$

定理 1.2 (凸函数的二阶判定条件)

设集合 $C \subset \mathbb{R}^n$ 为非空开凸集，函数 $f : C \to \mathbb{R}$ 二阶连续可微，则

1. $f(x)$ 是凸函数当且仅当对 $\forall \mathbf{x} \in C$，Hessien 矩阵 $\boldsymbol{H}(\mathbf{x})$ 半正定；
2. 若对 $\forall \mathbf{x} \in C$，Hessien 矩阵 $\boldsymbol{H}(\mathbf{x})$ 正定，则 f 是严格凸函数。

1.1.4 常用矩阵求导公式

$$\frac{\partial \boldsymbol{\beta}'\mathbf{x}}{\partial \mathbf{x}} = \boldsymbol{\beta} \tag{1.14}$$

$$\frac{\partial \mathbf{x}'\mathbf{x}}{\partial \mathbf{x}} = 2\mathbf{x} \tag{1.15}$$

$$\frac{\partial \mathbf{x}'\mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \left(\mathbf{A} + \mathbf{A}'\right)\mathbf{x} \tag{1.16}$$

第 2 章 无约束最优化方法的基本结构

2.1 最优性条件

2.2 方法的特性

对于无约束最优化问题

$$\min_{\mathbf{x}\in\mathbb{R}^n} f\left(\mathbf{x}\right) \tag{2.1}$$

一种常用的解决该问题的迭代算法如下：

> 选定初始点 $\mathbf{x}_0 \in \mathbb{R}^n$，在 $\mathbf{x}_0$ 处，确定一个使函数值下降的方向 $\mathbf{d}$ 与步长 α，继而求得下一个迭代点，以此类推，产生一个迭代点列 $\{\mathbf{x}_k\}$，$\{\mathbf{x}_k\}$ 或者其子列应收敛于最优解。当给定的某种终止准则满足时，或者 $\mathbf{x}_k$ 已满足要求的近似最优解的精度，或者算法已经无力进一步改善迭代点，则迭代结束。

注 对于上述迭代算法，下降方向与步长的选取顺序不同，导致产生不同类型的方法。

- 线搜索方法：在 $\mathbf{x}_k$ 点求得下降方向 $\mathbf{d}_k$，再沿 $\mathbf{d}_k$ 确定步长 α_k；
- 信赖域方法：在 $\mathbf{x}_k$ 点，先限定步长的范围，再同时确定下降方法 $\mathbf{d}_k$ 和步长 α_k。

2.3 线搜索准则

例题 2.1 考虑对正定二次函数 $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}'\mathbf{G}\mathbf{x} + \mathbf{b}'\mathbf{x}$，在点 $\mathbf{x}_k$，求出沿下降方向 $\mathbf{d}_k$ 作精确线搜索的步长 α_k。

证明 若要给出沿下降方向 $\mathbf{d}_k$ 作精确线搜索的步长 α_k，则应对于 $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k$，满足

$$\mathbf{d}_k'\nabla f(\mathbf{x}_{k+1}) = 0 \tag{2.2}$$

由于 $f(\mathbf{x})$ 为正定二次函数，则 $\mathbf{G}$ 为对称正定矩阵，则有 $\mathbf{G}' = \mathbf{G}$，因此

$$\nabla f(\mathbf{x}) = \mathbf{G}\mathbf{x} + \mathbf{b} \tag{2.3}$$

即

$$\mathbf{d}_k'\nabla f(\mathbf{x}_{k+1}) = \mathbf{d}_k'\left[\mathbf{G}\left(\mathbf{x}_k + \alpha_k\mathbf{d}_k\right) + \mathbf{b}\right] = 0 \tag{2.4}$$

由于 $\mathbf{G}$ 为正定矩阵，有 $\mathbf{d}_k'\mathbf{G}\mathbf{d}_k > 0$，因此，上式可化简为

$$\alpha_k = -\frac{\mathbf{d}_k'\left(\mathbf{G}\mathbf{x}_k + \mathbf{b}\right)}{\mathbf{d}_k'\mathbf{G}\mathbf{d}_k} = -\frac{\mathbf{d}_k'\nabla f(\mathbf{x}_k)}{\mathbf{d}_k'\mathbf{G}\mathbf{d}_k} \tag{2.5}$$

```python
def unconstrained_optimize(f, x0, epsilon=1e-8, max_iter=1000):
    x = np.empty((max_iter+1, x0.shape[0]))  # 定义 x 初始存储空间

    x[0] = x0
    for k in range(max_iter):
        d = search_desc_direction(f, x[k], ...)

        phi = lambda alpha: f(x[k] + alpha * d)
        alpha = search_step_length(phi, ...)

        x[k+1] = x[k] + alpha * d

        # if np.linalg.norm(g(x[k+1])) <= epsilon:
        # if f(x[k]) - f(x[k+1]) <= epsilon:
        if np.linalg.norm(x[k] - x[k+1]) <= epsilon:
            break

    return x[k+1], f(x[k+1])
```

Listing 2.1: 线搜索方法的基本结构：Python 实现

2.4 线搜索求步长

无约束最优化算法中线搜索方法的基本结构如下：

Algorithm 1: 线搜索方法的基本结构（P15）

Input: 目标函数 $f(\mathbf{x})$，初始点 $\mathbf{x}_0 \in \mathbb{R}^n$ 以及终止准则
Output: 最优解 $\mathbf{x}^*$ 以及 $f(\mathbf{x}^*)$

for $k = 0, \dots,$ **do**
  确定下降方向 $\mathbf{d}_k$，使得 $\nabla f(\mathbf{x}_k)' \mathbf{d}_k < 0$;
  确定步长 α_k 使得 $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k)$;
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{d}_k$;
  if $\mathbf{x}_k$ 满足给定的终止准则 **then**
    break;
  end
end
$\mathbf{x}^* \leftarrow \mathbf{x}_k$，$f(\mathbf{x}^*) \leftarrow f(\mathbf{x}_k)$;

2.4.1 精确线搜索方法

2.4.1.1 确定步长搜索区间

从一点出发，按一定步长，试图确定函数值呈现“高-低-高”的三点，从而得到一个近

```python
def search_unimodal_interval(phi, alpha0, gamma=0.1, t=2, max_iter=100):
    alphas = np.empty(max_iter + 1)

    alphas[0] = alpha0
    for i in range(max_iter):
        alphas[i+1] = alphas[i] + gamma
        if phi(alphas[i+1]) >= phi(alphas[i]) or alphas[i+1] <= 0:
            if i == 0:
                gamma = -gamma
                alpha = alphas[i+1]
            else:
                break
        else:
            gamma = t * gamma
            alpha = alphas[i]
            alphas[i] = alphas[i+1]

    return min(alpha, alphas[i+1]), max(alpha, alphas[i+1])
```

Listing 2.2: 进退法求初始搜索区间：Python 实现

似的单峰区间。

Algorithm 2: 进退法求初始搜索区间（P26）

Input: 一元函数 $\phi(\alpha)$，初始点 α_0，初始步长 γ_0，步长因子 t

Output: 初始区间 $[a, b]$

for $i = 0, \ldots,$ **do**

  $\alpha_{i+1} \leftarrow \alpha_i + \gamma_i$;

  if $\phi(\alpha_{i+1}) \geq \phi(\alpha_i)$ 或 $\alpha_{i+1} \leq 0$ **then**

    if $i = 0$ **then** $\gamma_{i+1} \leftarrow -\gamma_i$，$\alpha \leftarrow \alpha_{i+1}$ **else** break;

  else

    $\gamma_{i+1} \leftarrow t\gamma_i$，$\alpha \leftarrow \alpha_i$，$\alpha_i \leftarrow \alpha_{i+1}$;

  end

end

$a \leftarrow \min\{\alpha, \alpha_{i+1}\}$，$b \leftarrow \max\{\alpha, \alpha_{i+1}\}$;

注 在该算法中 α_i，α_{i+1}, α 分别起到了什么作用？

2.4.1.2 缩小步长搜索区间

2.4.1.2.1 0.618 方法 通过试探点函数值的比较，使包含极值点的搜索区间不断缩小。

2.4.1.2.2 多项式插值法 通过在搜索区间中不断地使用二次多项式去近似目标函数，并逐步用插值多项式的极小点去逼近线搜索问题的极小点。

Algorithm 3: 0.618 方法求一元函数 $\phi(\alpha)$ 的近似极小点（P27）

Input: 一元函数 $\phi(\alpha)$，初始搜索区间 $[a_0, b_0]$ 满足 $a_0 > b_0 > 0$，容许误差 $\varepsilon > 0$
Output: 近似极小点 α^*

```
τ ← (√5−1)/2 ≈ 0.618;
for i = 0, ..., do
    a_i^l ← a_i + (1 − τ)(b_i − a_i)， a_i^r ← a_i + τ(b_i − a_i);
    if φ(a_i^l) < φ(a_i^r) then
        a_{i+1} ← a_i， b_{i+1} ← a_i^r;
    else
        a_{i+1} ← a_i^l， b_{i+1} ← b_i;
    end
    if b_{i+1} − a_{i+1} < ε then break;
end
a* ← (b_i + a_i)/2;
```

```python
def search_step_length_gold(phi, a0, b0, epsilon=1e-8, max_iter=1000):
    a, b = np.empty(max_iter + 1), np.empty(max_iter + 1)

    a[0], b[0] = a0, b0
    tau = (np.sqrt(5) - 1) / 2
    for i in range(max_iter):
        a_l = a[i] + (1 - tau) * (b[i] - a[i])
        a_r = a[i] + tau * (b[i] - a[i])
        if phi(a_l) < phi(a_r):
            a[i+1], b[i+1] = a[i], a_r
        else:
            a[i+1], b[i+1] = a_l, b[i]
        if b[i+1] - a[i+1] < epsilon:
            break
    return (a[i+1] + b[i+1]) / 2
```

Listing 2.3: 0.618 方法求一元函数 $\phi(\alpha)$ 的近似极小点：Python 实现

设 $\alpha_1 < \alpha_2 < \alpha_3$，$\phi(\alpha_1) > \phi(\alpha_2)$，$\phi(\alpha_2) < \phi(\alpha_3)$，拟合如下的二次插值多项式:

$$p(\alpha) = a\alpha^2 + b\alpha + c \tag{2.6}$$

满足插值条件：

$$\begin{cases} p(\alpha_1) = a\alpha_1^2 + b\alpha_1 + c = \phi(\alpha_1) \\ p(\alpha_2) = a\alpha_2^2 + b\alpha_2 + c = \phi(\alpha_2) \\ p(\alpha_3) = a\alpha_3^2 + b\alpha_3 + c = \phi(\alpha_3) \end{cases} \tag{2.7}$$

从极值的必要条件知 $p'(\alpha_p) = 2a\alpha_p + b = 0$，求得

$$\alpha_p = -\frac{b}{2a} \tag{2.8}$$

从而可以算出

$$\alpha_p = \frac{1}{2}\left(\alpha_1 + \alpha_2 - \frac{c_1}{c_2}\right) \tag{2.9}$$

其中

$$c_1 = \frac{\phi(\alpha_3) - \phi(\alpha_1)}{\alpha_3 - \alpha_1}, \quad c_2 = \frac{\frac{\phi(\alpha_2)-\phi(\alpha_1)}{\alpha_2-\alpha_1} - c_1}{\alpha_2 - \alpha_3} \tag{2.10}$$

Algorithm 4: 多项式插值法（三点二次插值法）求一维函数 $\phi(\alpha)$ 的近似极小点

Input: 一元函数 $\phi(\alpha)$，初始搜索区间 $[a_0, b_0]$ 满足 $a_0 > b_0 > 0$，容许误差 $\varepsilon > 0$

Output: 近似极小点 α^*

任取 $c_0 \in [a_0, b_0]$;

for $i = 0, \ldots,$ **do**

  $\alpha_p \leftarrow \frac{1}{2}\left(a_i + b_i - \frac{c_1}{c_2}\right)$，其中 $c_1 = \frac{\phi(c_i)-\phi(a_i)}{c_i-a_i}, c_2 = \frac{\frac{\phi(b_i)-\phi(a_i)}{b_i-a_i}-c_1}{b_i-c_i}$;

  if $\phi(c_i) \le \phi(\alpha_p)$ **then**

    if $c_i \le \alpha_p$ **then**

      $a_{i+1} \leftarrow a_i$，$c_{i+1} \leftarrow c_i$，$b_{i+1} \leftarrow \alpha_p$;

    else

      $a_{i+1} \leftarrow \alpha_p$，$c_{i+1} \leftarrow c_i$，$b_{i+1} \leftarrow b_i$;

    end

  else

    if $c_i \le \alpha_p$ **then**

      $a_{i+1} \leftarrow c_i$，$c_{i+1} \leftarrow \alpha_p$，$b_{i+1} \leftarrow b_i$;

    else

      $a_{i+1} \leftarrow a_i$，$c_{i+1} \leftarrow \alpha_p$，$b_{i+1} \leftarrow c_i$;

    end

  end

  if $b_{i+1} - a_{i+1} < \varepsilon$ **then** break;

end

$\alpha^* \leftarrow \alpha_p$;

2.4.1.3 牛顿切线法

```python
def search_step_length_poly32(phi, a0, b0, epsilon=1e-8, max_iter=1000):
    a, b, c = np.empty(max_iter + 1), np.empty(max_iter + 1), np.empty(max_iter + 1)

    a[0], c[0], b[0] = a0, (a0 + b0) / 2, b0
    for i in range(max_iter):
        c1 = (phi(c[i]) - phi(a[i])) / (c[i] - a[i])
        c2 = ((phi(b[i]) - phi(a[i])) / (b[i] - a[i]) - c1) / (b[i] - c[i])
        alpha_p = 0.5 * (a[i] + b[i] - c1 / c2)
        if phi(c[i]) <= phi(alpha_p):
            if c[i] <= alpha_p:
                a[i+1], c[i+1], b[i+1] = a[i], c[i], alpha_p
            else:
                a[i+1], c[i+1], b[i+1] = alpha_p, c[i], b[i]
        else:
            if c[i] <= alpha_p:
                a[i+1], c[i+1], b[i+1] = c[i], alpha_p, b[i]
            else:
                a[i+1], c[i+1], b[i+1] = a[i], alpha_p, c[i]
        if b[i+1] - a[i+1] < epsilon:
            break

    return alpha_p
```

Listing 2.4: 多项式插值法（三点二次插值法）求一维函数 $\phi(\alpha)$ 的近似极小点：Python 实现

Algorithm 5: 牛顿切线法

Input: 一元函数 $\phi(\alpha)$ 及其一阶导函数 $\phi'(\alpha)$ 与二阶导函数 $\phi''(\alpha)$，初始点 α_0，容许误差 $\varepsilon > 0$

Output: 近似极小点 α^*

1 **for** $i = 0, \ldots,$ **do**

2 $\quad \alpha_{i+1} = \alpha_i - \frac{\phi'(\alpha_i)}{\phi''(\alpha_i)}$;

3 $\quad$ **if** $|\alpha_{i+1} - \alpha_i| < \varepsilon$ **then** break;

4 **end**

5 $\alpha^* \leftarrow \alpha_{i+1}$;

```python
def search_step_length_newton(phi, phi_grad, phi_hess, alpha0, epsilon=1e-8, max_iter=1000):
    alpha = np.empty(max_iter + 1)

    alpha[0] = alpha0
    for i in range(max_iter):
        alpha[i+1] = alpha[i] - phi_grad(alpha[i]) / phi_hess(alpha[i])
        if abs(alpha[i+1] - alpha[i]) < epsilon:
            break

    return alpha[i+1]
```

Listing 2.5: 牛顿切线法：Python 实现

第 3 章 负梯度方法与 Newton 型方法

3.1 最速下降法

例题 3.1 用最速下降方法求解

$$\min\ f(\mathbf{x})=\frac{1}{2}\mathbf{x}'\mathbf{G}\mathbf{x}+\mathbf{b}'\mathbf{x}+c,\quad \mathbf{G}\geq \tag{3.1}$$

解 对 $f(\mathbf{x})$ 求导可得，

$$g(\mathbf{x})=\mathbf{G}\mathbf{x}+\mathbf{b} \tag{3.2}$$

因此，最速下降方向为

$$\mathbf{d}_k=-g(\mathbf{x}_k)=-(\mathbf{G}\mathbf{x}_k+\mathbf{b}) \tag{3.3}$$

通过一维精确线搜索求得步长 α_k，即

$$\alpha_k=\arg\min_{\alpha}\ \phi(\alpha) \tag{3.4}$$

其中，

$$\phi(\alpha)=f(\mathbf{x}_k+\alpha\mathbf{d}_k)=\frac{1}{2}(\mathbf{x}_k+\alpha\mathbf{d}_k)'\mathbf{G}(\mathbf{x}_k+\alpha\mathbf{d}_k)+\mathbf{b}'(\mathbf{x}_k+\alpha\mathbf{d}_k)+c \tag{3.5}$$

对 $\phi(\alpha_k)$ 求导可得，

$$\begin{aligned}\phi'(\alpha)&=[\mathbf{G}(\mathbf{x}_k+\alpha\mathbf{d}_k)]'\mathbf{d}_k+\mathbf{b}'\mathbf{d}_k\\&=[\mathbf{G}(\mathbf{x}_k+\alpha\mathbf{d}_k)+\mathbf{b}]'\mathbf{d}_k\\&=[(\mathbf{G}\mathbf{x}_k+\mathbf{b})+\alpha\mathbf{G}\mathbf{d}_k]'\mathbf{d}_k\\&=-\mathbf{d}_k'\mathbf{d}_k+\alpha\mathbf{d}_k'\mathbf{G}\mathbf{d}_k\end{aligned} \tag{3.6}$$

令 $\phi'(\alpha)=0$，可解得

$$\alpha_k=\frac{\mathbf{d}_k'\mathbf{d}_k}{\mathbf{d}_k'\mathbf{G}\mathbf{d}_k} \tag{3.7}$$

因此，迭代公式为

$$\mathbf{x}_{k+1}=\mathbf{x}_k+\alpha_k\mathbf{d}_k=\mathbf{x}_k+\frac{\mathbf{d}_k'\mathbf{d}_k}{\mathbf{d}_k'\mathbf{G}\mathbf{d}_k}\mathbf{d}_k=\mathbf{x}_k-\frac{\mathbf{g}_k'\mathbf{g}_k}{\mathbf{g}_k'\mathbf{G}\mathbf{g}_k}\mathbf{g}_k \tag{3.8}$$

其中，

$$\mathbf{g}_k=g(\mathbf{x}_k)=\mathbf{G}\mathbf{x}_k+\mathbf{b} \tag{3.9}$$

✍ **练习 3.1** 用最速下降法求解问题

$$\min\ f(\mathbf{x})=x_1^2+2x_2^2+4x_1+4x_2 \tag{3.10}$$

设 $\mathbf{x}^{(1)}=(0,0)'$。证明：

$$\mathbf{x}^{(n+1)}=\left(\frac{2}{3^n}-2,\left(-\frac{1}{3}\right)^n-1\right)' \tag{3.11}$$

解 该函数可重写为正定二次函数形式，即

$$f(\mathbf{x})=\frac{1}{2}\mathbf{x}'\mathbf{G}\mathbf{x}+\mathbf{b}'\mathbf{x}+c=\frac{1}{2}\begin{pmatrix}x_1\\x_2\end{pmatrix}'\begin{pmatrix}2&0\\0&4\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}+\begin{pmatrix}4\\4\end{pmatrix}'\begin{pmatrix}x_1\\x_2\end{pmatrix} \tag{3.12}$$

其中，

$$\mathbf{G}=\begin{pmatrix}2&0\\0&4\end{pmatrix},\quad \mathbf{b}=\begin{pmatrix}4\\4\end{pmatrix},\quad c=0 \tag{3.13}$$

则，根据最速下降法关于正定二次函数的通项公式有

$$\mathbf{x}^{(k+1)}=\mathbf{x}^{(k)}-\frac{\mathbf{g}_k'\mathbf{g}_k}{\mathbf{g}_k'\mathbf{G}\mathbf{g}_k}\mathbf{g}_k \tag{3.14}$$

其中，

$$\mathbf{g}_k=\mathbf{G}\mathbf{x}_k+\mathbf{b}=\begin{pmatrix}2&0\\0&4\end{pmatrix}\begin{pmatrix}x_1^{(k)}\\x_2^{(k)}\end{pmatrix}+\begin{pmatrix}4\\4\end{pmatrix}=\begin{pmatrix}2x_1^{(k)}+4\\4x_2^{(k)}+4\end{pmatrix} \tag{3.15}$$

1. 当 $n=0$ 时，结论显然成立，即

$$\mathbf{x}^{(1)}=\left(\frac{2}{3^0}-2,\left(-\frac{1}{3}\right)^0-1\right)'=(0,0)' \tag{3.16}$$

2. 当 $n=k-1$ 时，假设结论成立，则有

$$\mathbf{x}^{(k)}=\left(\frac{2}{3^{k-1}}-2,\left(-\frac{1}{3}\right)^{k-1}-1\right)' \tag{3.17}$$

3. 当 $n=k$ 时，根据通项公式以及可以求得，

$$\mathbf{g}_k=4\cdot\left(\frac{1}{3^{k-1}},\left(\frac{-1}{3}\right)^{k-1}\right)' \tag{3.18}$$

则有，

$$\mathbf{g}_k'\mathbf{g}_k=16\cdot\left[\frac{1}{3^{2(k-1)}}+\left(\frac{-1}{3}\right)^{2(k-1)}\right]=\frac{32}{3^{2(k-1)}}$$
$$\mathbf{g}_k'\mathbf{G}\mathbf{g}_k=16\cdot\left[2\cdot\frac{1}{3^{2(k-1)}}+4\cdot\left(\frac{-1}{3}\right)^{2(k-1)}\right]=\frac{96}{3^{2(k-1)}}$$

因此，

$$\begin{aligned}\mathbf{x}^{(k+1)}=&\mathbf{x}^{(k)}-\frac{\mathbf{g}_k'\mathbf{g}_k}{\mathbf{g}_k'\mathbf{G}\mathbf{g}_k}\mathbf{g}_k\\=&\left(\frac{2}{3^{k-1}}-2,\left(-\frac{1}{3}\right)^{k-1}-1\right)'-\frac{4}{3}\cdot\left(\frac{1}{3^{k-1}},\left(\frac{-1}{3}\right)^{k-1}\right)'\\=&\left(\frac{2}{3^k}-2,\left(-\frac{1}{3}\right)^k-1\right)'\end{aligned} \tag{3.19}$$