

Image

作者：龚梓阳

时间：2021 年 10 月 26 日

目录

1	引论	1
1.1	预备知识	1
2	无约束最优化方法的基本结构	4
2.1	最优性条件	4
2.2	方法的特性	4
2.3	线搜索准则	4
2.4	线搜索求步长	4
3	负梯度方法与 Newton 型方法	10
3.1	最速下降法	10

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 $\|\cdot\|: X \rightarrow \mathbb{R}$ 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} : $\forall x \in \mathbb{R}$

$$\|x\| := |x| \quad (1.1)$$

即范数 $\|x\|$ 为 x 的绝对值。

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

(a). 欧几里得范数:

$$\|\mathbf{x}\|_2 := \sqrt{x_1^2 + \dots + x_n^2} \quad (1.2)$$

(b). l_p 范数 ($p \geq 1$):

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

若 $p = 1$, 则

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

若 $p = \infty$, 则

$$\|\mathbf{x}\|_\infty := \max_{i=1, \dots, n} |x_i| \quad (1.5)$$

(c). l_0 范数:

$$\|\mathbf{x}\|_0 = |\{x_i : x_i \neq 0, i = 1, \dots, n\}| \quad (1.6)$$

即向量 \mathbf{x} 中分量 x_i 不为零的个数。

1.1.2 梯度向量与 Hessian 矩阵

定义 1.2 (梯度向量)

对于多变量函数 $f: \mathbb{R}^n \rightarrow \mathbb{R}$, 即 $f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$, 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:

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

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 $f: \mathbb{R}^n \rightarrow \mathbb{R}$, 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

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

或使用下标记号表示为

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

**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

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

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 $f: C \rightarrow \mathbb{R}$. 若对 $\forall \mathbf{x}, \mathbf{y} \in C$, 有

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

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 $f: C \rightarrow \mathbb{R}$ 可微, 则

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

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

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

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

**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 $f: C \rightarrow \mathbb{R}$ 二阶连续可微, 则

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

**1.1.4 常用矩阵求导公式**

$$\frac{\partial \beta' \mathbf{x}}{\partial \mathbf{x}} = \beta \quad (1.14)$$

$$\frac{\partial \mathbf{x}' \mathbf{x}}{\partial \mathbf{x}} = 2\mathbf{x} \quad (1.15)$$

$$\frac{\partial \mathbf{x}' \mathbf{A} \mathbf{x}}{\partial \mathbf{x}} = (\mathbf{A} + \mathbf{A}') \mathbf{x} \quad (1.16)$$

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

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

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

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

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

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

即

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

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

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

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：

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，从而得到一个近似的单峰区间。

注 在该算法中 α_i ， α_{i+1} ， α 分别起到了什么作用？

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 $f(\mathbf{x}^*)$

```

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

```

```

1 def unconstrained_optimize(f, x0, epsilon=1e-8, max_iter=1000):
2     x = np.empty((max_iter+1, x0.shape[0])) # 定义 x 初始存储空间
3
4     x[0] = x0
5     for k in range(max_iter):
6         d = search_desc_direction(f, x[k], ...)
7
8         phi = lambda alpha: f(x[k] + alpha * d)
9         alpha = search_step_length(phi, ...)
10
11        x[k+1] = x[k] + alpha * d
12
13        # if np.linalg.norm(g(x[k+1])) <= epsilon:
14        # if f(x[k]) - f(x[k+1]) <= epsilon:
15        if np.linalg.norm(x[k] - x[k+1]) <= epsilon:
16            break
17
18    return x[k+1], f(x[k+1])

```

Listing 2.1: 线搜索方法的基本结构: Python 实现**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 $[a, b]$

```

1 for  $i = 0, \dots$ , do
2    $\alpha_{i+1} \leftarrow \alpha_i + \gamma_i$ ;
3   if  $\phi(\alpha_{i+1}) \geq \phi(\alpha_i)$  或  $\alpha_{i+1} \leq 0$  then
4     if  $i = 0$  then  $\gamma_{i+1} \leftarrow -\gamma_i$ ,  $\alpha \leftarrow \alpha_{i+1}$  else break;
5   else
6      $\gamma_{i+1} \leftarrow t\gamma_i$ ,  $\alpha \leftarrow \alpha_i$ ,  $\alpha_i \leftarrow \alpha_{i+1}$ ;
7   end
8 end
9  $a \leftarrow \min\{\alpha, \alpha_{i+1}\}$ ,  $b \leftarrow \max\{\alpha, \alpha_{i+1}\}$ ;

```

```

1 def search_unimodal_interval(phi, alpha0, gamma=0.1, t=2, max_iter=100):
2     alphas = np.empty(max_iter + 1)
3
4     alphas[0] = alpha0
5     for i in range(max_iter):
6         alphas[i+1] = alphas[i] + gamma
7         if phi(alphas[i+1]) >= phi(alphas[i]) or alphas[i+1] <= 0:
8             if i == 0:
9                 gamma = -gamma
10                alpha = alphas[i+1]
11            else:
12                break
13        else:
14            gamma = t * gamma
15            alpha = alphas[i]
16            alphas[i] = alphas[i+1]
17
18    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

2.4.1.2 缩小小步长搜索区间

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。

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^*

```

1  $\tau \leftarrow \frac{\sqrt{5}-1}{2} \approx 0.618$ ;
2 for  $i = 0, \dots$ , do
3      $a_i^l \leftarrow a_i + (1 - \tau)(b_i - a_i)$ ,  $a_i^r \leftarrow a_i + \tau(b_i - a_i)$ ;
4     if  $\phi(a_i^l) < \phi(a_i^r)$  then
5          $a_{i+1} \leftarrow a_i$ ,  $b_{i+1} \leftarrow a_i^r$ ;
6     else
7          $a_{i+1} \leftarrow a_i^l$ ,  $b_{i+1} \leftarrow b_i$ ;
8     end
9     if  $b_{i+1} - a_{i+1} < \varepsilon$  then break;
10 end
11  $a^* \leftarrow \frac{b_i + a_i}{2}$ ;

```

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。

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:

$$p(\alpha) = a\alpha^2 + b\alpha + c \quad (2.6)$$

满足插值条件:

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

```

1 def search_step_length_gold(phi, a0, b0, epsilon=1e-8, max_iter=1000):
2     a, b = np.empty(max_iter + 1), np.empty(max_iter + 1)
3
4     a[0], b[0] = a0, b0
5     tau = (np.sqrt(5) - 1) / 2
6     for i in range(max_iter):
7         a_l = a[i] + (1 - tau) * (b[i] - a[i])
8         a_r = a[i] + tau * (b[i] - a[i])
9         if phi(a_l) < phi(a_r):
10             a[i+1], b[i+1] = a[i], a_r
11         else:
12             a[i+1], b[i+1] = a_l, b[i]
13         if b[i+1] - a[i+1] < epsilon:
14             break
15     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

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

$$\alpha_p = -\frac{b}{2a} \quad (2.8)$$

从而可以算出

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

其中

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

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^*

```

1 任取  $c_0 \in [a_0, b_0]$ ;
2 for  $i = 0, \dots$ , do
3      $\alpha_p \leftarrow \frac{1}{2} \left( a_i + b_i - \frac{c_1}{c_2} \right)$ , 其中  $c_1 = \frac{\phi(c_i) - \phi(a_i)}{c_i - a_i}$ ,  $c_2 = \frac{\frac{\phi(b_i) - \phi(a_i)}{b_i - a_i} - c_1}{b_i - c_i}$ ;
4     if  $\phi(c_i) \leq \phi(\alpha_p)$  then
5         if  $c_i \leq \alpha_p$  then
6              $a_{i+1} \leftarrow a_i$ ,  $c_{i+1} \leftarrow c_i$ ,  $b_{i+1} \leftarrow \alpha_p$ ;
7         else
8              $a_{i+1} \leftarrow \alpha_p$ ,  $c_{i+1} \leftarrow c_i$ ,  $b_{i+1} \leftarrow b_i$ ;
9         end
10    else
11        if  $c_i \leq \alpha_p$  then
12             $a_{i+1} \leftarrow c_i$ ,  $c_{i+1} \leftarrow \alpha_p$ ,  $b_{i+1} \leftarrow b_i$ ;
13        else
14             $a_{i+1} \leftarrow a_i$ ,  $c_{i+1} \leftarrow \alpha_p$ ,  $b_{i+1} \leftarrow c_i$ ;
15        end
16    end
17    if  $b_{i+1} - a_{i+1} < \varepsilon$  then break;
18 end
19  $\alpha^* \leftarrow \alpha_p$ ;

```

```

1 def search_step_length_poly32(phi, a0, b0, epsilon=1e-8, max_iter=1000):
2     a, b, c = np.empty(max_iter + 1), np.empty(max_iter + 1), np.empty(max_iter + 1)
3
4     a[0], c[0], b[0] = a0, (a0 + b0) / 2, b0
5     for i in range(max_iter):
6         c1 = (phi(c[i]) - phi(a[i])) / (c[i] - a[i])
7         c2 = ((phi(b[i]) - phi(a[i])) / (b[i] - a[i]) - c1) / (b[i] - c[i])
8         alpha_p = 0.5 * (a[i] + b[i] - c1 / c2)
9         if phi(c[i]) <= phi(alpha_p):
10             if c[i] <= alpha_p:
11                 a[i+1], c[i+1], b[i+1] = a[i], c[i], alpha_p
12             else:
13                 a[i+1], c[i+1], b[i+1] = alpha_p, c[i], b[i]
14         else:
15             if c[i] <= alpha_p:
16                 a[i+1], c[i+1], b[i+1] = c[i], alpha_p, b[i]
17             else:
18                 a[i+1], c[i+1], b[i+1] = a[i], alpha_p, c[i]
19         if b[i+1] - a[i+1] < epsilon:
20             break
21
22     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

2.4.1.3 牛顿切线法

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  $i = 0, \dots$ , do
2      $\alpha_{i+1} = \alpha_i - \frac{\phi'(\alpha_i)}{\phi''(\alpha_i)}$ ;
3     if  $|\alpha_{i+1} - \alpha_i| < \varepsilon$  then break;
4 end
5  $\alpha^* \leftarrow \alpha_{i+1}$ ;

```

```
1 def search_step_length_newton(phi, phi_grad, phi_hess, alpha0, epsilon=1e-8, max_iter=1000):
2     alpha = np.empty(max_iter + 1)
3
4     alpha[0] = alpha0
5     for i in range(max_iter):
6         alpha[i+1] = alpha[i] - phi_grad(alpha[i]) / phi_hess(alpha[i])
7         if abs(alpha[i+1] - alpha[i]) < epsilon:
8             break
9
10    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

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

证明 对 $f(\mathbf{x})$ 求导可得,

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

因此, 最速下降方向为

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

通过一维精确线搜索求得步长 α_k , 即

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

其中,

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

对 $\phi(\alpha_k)$ 求导可得,

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

令 $\phi'(\alpha) = 0$, 可解得

$$\alpha_k = \frac{\mathbf{d}_k' \mathbf{d}_k}{\mathbf{d}_k' \mathbf{G} \mathbf{d}_k} \quad (3.7)$$

因此, 迭代公式为

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

其中,

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