线性搜索法

首先，线性搜索法的核心思想为：

1. 选取一个搜索方向p_k
2. 选择合适的步长α_k

3. $x_{k+1} = x_k + \alpha_k p_k$.

由于大部分的线性搜索法要求p_k是下降方向，以保证算法的有效性，所以p_k往往有如下形式：

$$p_k = -B_k^{-1}\nabla f_k$$

其中，B_k是对称正定阵。

步长选取

线性搜索法的核心模型本质上是一个关于步长的一元函数：

$$\phi(\alpha) = f(x_k + \alpha p_k)$$

在选取步长的时候，我们希望选取使ϕ 尽可能小的α，最理想的情况是找到ϕ 的极小值点，然而极小化的过程往往代价很大，所以我们需要一些更具有实践意义的办法。
在提出算法之前，我们需要有一些判断α_k 是否合理的准则，因为我们不仅仅希望函数值下降，还希望函数值下降得足够多。

Wolfe Conditions

(1) of Wolfe Conditions: Armijo condition

$$f(x_k + \alpha p_k) \leq f_k + c_1\alpha_k\nabla f_k^T p_k$$

其中$c_1 \in (0,1)$,
这个式子等价于$\phi(\alpha_k) \leq \phi(0) + c_1\alpha_k f'(0)$
这个条件保证了新迭代点的函数值小于当前函数值
实际应用中，c_1 往往取得比较小，比如10^{-4}.

(2) of Wlofe Condition: curvature conditioon

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2\nabla f_k^T p_k$$

其中$c_2 \in (c_1,1)$,
这个式子等价于$\phi'(\alpha_k) \geq c_2\phi'(0)$
即我们希望新的迭代点比当前节点“更平缓”

我们把以上两个式子称为 **Wolfe Conditions**

然而注意到第二个条件可能会导致出现新节点的导数是一个很大的正数的情况，与我们的目标不太符合，因此，可以改为如下条件：

$$|\nabla f(x_k + \alpha_k p_k)^T p_k| \leq c_2|\nabla f_k^T p_k|$$

其中$c_2 \in (c_1, 1)$,
这个式子等价于$|\phi'(\alpha_k)| \leq c_2|\phi'(0)|$，保证了新的迭代点的确是比当前位置更平缓的要求。

我们把修改后的条件称为 **Strong Wolfe condition**

不难证明如果f 连续且下有界，我们一定能找到满足上述条件的α_k。

这一准则在线性搜索法中有着广泛应用，在*拟牛顿法*中尤为重要。

The Goldstein Conditions

$$f_k + (1-c)\alpha_k \nabla f_k^T p_k \leq f(x_k + \alpha_k p_k) \leq f_k + c\alpha_k \nabla f_k^T p_k$$

其中，$0 < c < 1/2$，第二个式子其实是 Armijo condition
这个条件等价于 $\phi(0) + (1-c)\alpha_k f'(0) \leq \phi(\alpha_k) \leq \phi(0) + c\alpha_k f'(0)$
它保证了α_k 不会太小。
然而这个准则相对于Wolfe Conditions来说，缺点在于它的第一个不等式可能会把极小点排除。然而这两个准则在理论上的收敛性是相似的。
在实际应用中，Goldstein Conditions在牛顿法中比较有效而在拟牛顿法中不太合适。

一个简单的线性搜索法

> Backtracking Line Search

```
Choose alpha0 > 0,  0<rho<1, 0<c<1; set alpha <- alpha0;
while f(xk + alpha*pk) > f(xk) + c*alpha*f'(xk)**pk
    alpha <- rho*alpha;
end
return alphak = alpha
```

在牛顿法和拟牛顿法中我们往往取初值为1，而其他算法则可以取别的值。
注：实际应用中，收缩子ρ 往往是可以根据迭代情况在一定范围内变化的。

这个简单常用的策略在牛顿法中表现很好，而在拟牛顿法和共轭梯度法中则不太合适。

线性搜索法的收敛性

为了保证全局收敛，我们不但需要找到合适的迭代步长α_k，还需要找到合适的搜索方向p_k。
本节我们主要研究一个关键属性：

$$\cos\theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\|\|p_k\|}$$

Theorem 3.2(Zoutendijk)

对于一个二次可微下有界且梯度满足Lipschitz连续条件的f，我们用满足Wolfe conditions的线性搜索法可以得到如下的全局收敛性：

$$\sum_{k\geq 0}\cos^2\theta_k\|\nabla f_k\|^2<\infty$$

我们从这个定理可以推出：

$$\cos^2\theta_k\|\nabla f_k\|^2<\infty\rightarrow 0$$

因此，如果我们选取的p_k能够保证θ_k与90°有个足够大的差距，即存在常数$\delta>0$ s.t.

$$\cos\theta_k\geq k>0,\quad \forall k$$

那么我们就有：(3.18)

$$\lim_{k\rightarrow\infty}\|\nabla f_k\|=0$$

我们把满足(3.18)的算法称为是具有全局收敛性的。
注意：这个全局收敛性的条件是我们能够得到的最强的条件了，但是它并不能保证我们找到最小值，只能保证我们找到一个稳定点。只有在提供了更多信息的情况下，如Hessian矩阵信息，我们才能保证找到局部最小点。

利用上述结论，我们显然可以得到，最速下降法是全局收敛的。那么对于牛顿法和拟牛顿法，若B_k是正定且条件数有界的，即

$$\|B_k\|\|B_k^{-1}\|\leq M,\quad \forall k$$

于是有$\cos\theta_k\geq 1/M$
从而，$\lim_{k\rightarrow\infty}\|\nabla f_k\|=0$
即当B_k正定且条件数有界，满足Wolfe conditions的牛顿法和拟牛顿法也是全局收敛的

而对于包括共轭梯度法在内的其它一些算法，我们只能证明一个较弱的结论：

$$\liminf_{k\rightarrow\infty}||\nabla f||=0$$

即存在子列全局收敛，细节会在后续章节讨论

于是我们利用类似的思路可以构造这样的算法：(1) 在每一步下降 (2) 每m步使用最速下降法——它既改善了最速下降法的性能，又利用最速下降法保证了全局收敛性

收敛速率

前一节的结论似乎说明：构造一个全局收敛的算法很简单，我们只要在每一步迭代的时候计算$\cos\theta_k$，当$\cos\theta_k$小于预设阈值时，将p_k向最速下降方向进行调整即可
然而这种方法是不可靠的，有如下两点原因：

1. 这样做的收敛速率可能很慢，因为有时候我们需要很接近梯度的法方向的搜索方向，如在Hessian矩阵非良性的时候
2. 这种做法破坏了拟牛顿法的尺度信息(invariance properties)

我们需要在收敛速度和全局收敛性之间做一个权衡，如最速下降法是一个很经典的全局收敛算法，然而它的速度很慢，相反，牛顿法在接近最优值的时候收敛很快然而在远离最优解的时候甚至可能不收敛。

最速下降法的收敛速率

我们从一个理想模型，即目标函数为二次函数的情形展开分析，对于更复杂的非线性问题，其性态也是相似的。

$$f(x)=\frac{1}{2}x^TQx-bx$$

其中Q 是对称正定的。于是$\nabla f(x)=Qx-b$，函数的极小点x^*是线性方程组$Qx=b$的唯一解。

$$\begin{aligned}\phi(\alpha)&=f(x_k-\alpha\nabla f_k)\\&=\frac{1}{2}(x_k-\alpha\nabla f_k)^TQ(x_k-\alpha\nabla f_k)-b^T(x_k-\alpha\nabla f_k)\\\phi'(\alpha)&=-\nabla f_k^TQ(x_k-\alpha\nabla f_k)+b^T\nabla f_k\\\Rightarrow\alpha_k&=\frac{\nabla f_k^T\nabla f_k}{\nabla f_k^TQ\nabla f_k}\end{aligned}$$

最速下降迭代为：

$$x_{k+1}=x_k-\left(\frac{\nabla f_k^T\nabla f_k}{\nabla f_k^TQ\nabla f_k}\right)\nabla f_k$$

其中$\nabla f_k=Qx_k-b$

为了分析收敛速率，我们引入Q范数：

$$\|x\|_Q^2=x^TQx$$

我们有

$$\frac{1}{2}\|x-x^*\|_Q^2=f(x)-f(x^*)$$

$$\|x_{k+1}-x^*\|_Q^2=\left\{1-\frac{(\nabla f_k^T\nabla f_k)^2}{(\nabla f_k^TQ\nabla f_k)(\nabla f_k^TQ^{-1}\nabla f_k)}\right\}\|x_k-x^*\|_Q^2$$

由于表达式复杂，所以我们希望通过分析条件数对算法进行估计

Theorem 3.3(*Luenberger*)

$$\|x_{k+1}-x^*\|_Q^2 \leq \left(\frac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1}\right)\|x_k-x^*\|_Q^2$$

其中 $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ 是 Q的特征值

这个定理对最速下降法的性能做了一个估计，注意到当所有特征值都相等时，最速下降法是单步收敛的。假设每次迭代步长都是搜索方向上的全局最优步长，我们有如下结论：

Theorem 3.4

假设f 二次可微，且最优解x^*处的Hessian矩阵$\nabla^2 f(x^*)$是正定的，对于最速下降法，任取$r \in R$ s.t.

$$r \in \left(\frac{\lambda_n-\lambda_1}{\lambda_n+\lambda_1}, 1\right)$$

我们有如下估计：

$$f(x_{k+1})-f(x^*) \leq r^2[f(x_k)-f(x^*)]$$

一般来说，当步长不是全局最优步长的时候，效果是不会改进的，因此，上述定理说明了最速下降法的效率不会太好。举个例字，对于$\kappa(Q)=800, f(x_0)=1, f(x^*)=0$的问题，定理3.4 给出了这样的估计：函数值经过500次最速下降法迭代后大概扔保持在0.08，经过1000次迭代后大概为0.006.

牛顿法的收敛速率

接下来我们分析一下牛顿法：

$$p_k^N = -\nabla^2 f_k^{-1}\nabla f_k$$

这里我们先只考虑x_k在x^*附近，即$\nabla^2 f_k$ 正定的情况：

Theorem 3.5

假设f 二次可微且$\nabla^2 f$ 在x^*附近Lipschitz连续，那么关于牛顿法，我们有如下结论：

1. 如果x_0充分靠近x^*，算法收敛至x^*
2. $\{x_k\}$的收敛速率是二次的
3. $\{\|\nabla f_k\|\}$二次收敛到0.

上述结论可由泰勒定理和Lipschitz连续的性质证得

我们会在定理3.6说明，当迭代点足够接近最优解时，$\alpha_k = 1$是满足Wolfe条件的。

拟牛顿法的收敛速率

拟牛顿法的形式如下：

$$p_k = -B_k^{-1}\nabla f_k$$

其中B_k 是一个随拟牛顿法迭代更新的对称正定阵。和牛顿法一样，我们在选取初值的时候会先尝试$\alpha_k = 1$，然后再根据Wolfe条件调整，这个初值选取的过程对于算法的收敛速率是很重要的。

下面的结果将说明如果拟牛顿法足够接近牛顿法，那么步长1是满足Wolfe条件的。此外，它还给出了一个线性搜索法达到超线性收敛的必要条件。为了定理的普适性，我们将用一般的线性搜索法去进行分析

Theorem 3.6

假设f 是二次可微的。考虑线性搜索法$x_{k+1} = x_k + \alpha_k p_k$，
其中p_k 是一个下降方向，而α_k 是一个满足$c < 1/2$的Wolfe条件的步长。如果$\{x_k\}$收敛到x^*且$\nabla f(x^*) = 0$，$\nabla^2 f^*$对称正定，且搜索方向满足(3.35)

$$\lim_{k\to\infty} \frac{\|\nabla f_k + \nabla^2 f_k p_k\|}{\|p_k\|} = 0$$

那么

1. 对于充分大的k，$\alpha_k = 1$ 是满足Wolfe条件的
2. 如果$\alpha_k = 1, \forall k > k_0$,$\{x_k\}$超线性收敛

对于拟牛顿法，(3.35)等价于(3.36)

$$\lim_{k\to\infty} \frac{\|(B_k - \nabla^2 f(x^*))p_k\|}{\|p_k\|} = 0$$

这个式子是令人惊讶的，因为它说明了，拟牛顿法要达到超线性收敛，并不要求B_k收敛到$\nabla^2 f(x^*)$，只要求B_k沿着搜索方向p_k逼近$\nabla^2 f(x^*)$ 即可。
特别地，**(3.36)其实是拟牛顿法达到超线性收敛的充分必要条件**。

Theorem 3.7

假设f 二次可微，且x^*满足二次充分条件，对于拟牛顿法，我们取$\alpha_k = 1$，则$\{x_k\}$超线性收敛当且仅当(3.36)成立。

基于修正Hessian矩阵的牛顿法

当远离极值的时候，Hessian矩阵可能并不正定所以方程(3.38)

$$\nabla^2 f(x_k) p_k^N = -\nabla f(x_k)$$

定义的p_k^N 也许不是一个下降方向。下面我们将利用线性代数里的技巧，对Hessian矩阵进行正定近似，进而进行求解。大致算法如下：

Algorithm3.2: Line Search Newton with Modification

```
Given initial point x0;
for k = 0,1,2,...
    Factorize the matrix Bk = H + Ek, where
        if H is p.d. Ek = 0
        else Ek is sufficiently large to ensure Bk is p.d.
    Solve Bk pk = -df(xk)
    set x(k+1) <- xk + alphak*pk, where alphak satisfies the Wolfe,
        Goldstein, or Armijo backtracking conditions;
end
```

有些方法并不直接计算E_k，而是引入一些额外的步骤，并用标准分解程序检验是否成功。
下面我们将介绍基于Cholesky分解和基于将Hessian矩阵进行对称正定分解的方法。
利用之前的结论，我们可以知道当$E_k(B_k)$满足有界分解条件，即：(3.39)

$$\kappa(B_k) = \|B_k\|\|B_k^{-1}\| \leq C, \quad \text{some } C > 0, \forall k = 0, 1, 2, ...$$

时，Algorithm 3.2是一个具有实践意义，全局收敛的牛顿方法。

Theorem 3.8

在前述条件满足且$\mathcal{L} = \{x \in \mathcal{D} : f(x) < f(x_0)\}$是紧集的条件下

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

$\nabla^2 f(x^*)$ 条件好的时候，算法3.2可以退化到牛顿法，因此具有二阶收敛性
而$\nabla^2 f(x^*)$ 接近奇异的时候，它的收敛速率可能只是线性
自然地，我们希望修正分解的计算代价比较小
为了进一步分析算法3.2，我们假设$\nabla^2 f(x^*)$的特征值分解是可以实现的。然而这个假设在规模大的问题中是不现实的，然而它可以给我们一些启示。

特征值分解

有许多修正策略可以改善Hessian矩阵的性态，并且也没有孰强孰弱的定论。
实际上，用不同的标准可以推导出不同的策略，如果我们希望修正前后ΔA的*Frobenius 范数*最小，那么我们就会得到(3.43)

$$\Delta A = Q\text{diag}(\tau_\text{i})\text{Q}^\text{T}, \quad \text{with } \tau_\text{i} = \begin{cases} 0, & \lambda_\text{i} \geq \delta \\ \delta - \lambda_\text{i}, & \lambda_\text{i} < \delta \end{cases}$$

如果我们希望修正前后ΔA的*Euclidean 范数*最小，那么我们就会得到(3.44)

$$\Delta A = \tau I, \quad \text{with} \tau = \max(0, \delta - \lambda_{\min}(A))$$

这也说明了对角校正和非对角校正都是可以考虑的。市面上有很多基于对角校正或非对角校正的算法，但是它们都不用特征值分解，因为特征值分解往往代价太大，惯用策略是在使用*高斯消元法*分解

矩阵的过程中间接做一些修正以期产生一些好的结果。数值实验说明这样的策略经常是有效的。

Adding a Multiple of the Identity

最简单的想法应该是找一个正数$\tau > 0$，使得$H + \tau I$正定。从前述分析我们知道τ需要满足(3.44)，然而Hessian矩阵的最小特征值往往是无法得到的，下面是我们的一个算法

Algorithm 3.3 Cholesky with Added Multiple of the Identity

```
choose beta>0;
if min(a(i,i))>0
    set tau(0) <- 0;
else
    tau0 = -min(a(i,i)) + beta;
end
for k = 0,1,2,...
    try do Cholesky algorithm to obtain LL' = A + tau(k)I
    if succeessful
        return L;
    else
        tau(k+1) <- max(2tau(k),beta)
    end
end
```

β 的选择是启发式的，常用的数值是10^{-3}.
需要注意如果算法运行太多次，代价是很大的，所以我们应该用更大的因子10而不是2 在最后调整的时候。

Modified Cholesky Factorization

另一个改良Hessian矩阵的做法是对Hessian矩阵做Cholesky分解，在分解过程中增大对角元使得结果是充分正定的。
在分析算法前，我们先简单回顾一下Cholesky分解的步骤

Algorithm 3.4 Cholesky Factorization LDL^T Form

```
for j = 1,2,...,n
    c(j,j) <- a(j,j) - sum(@s,1,j-1,d(s)*l(j,s)^2);
    d(j) <- c(j,j);
    for i = j+1,...,n
        c(i,j) <- a(i,j) - sum(@s,1,j-1,d(s)*l(i,s)*l(j,s));
        l(i,j) <- c(i,j)/d(j);
    end
end
```

我们可以在分解过程中对A 进行修正，从而使得D充分正定，同时D和L的元素也不能过大，为了控制修正的效果，我们引入两个参数β和δ，要求:(3.48)

$$d_j \geq \delta, \quad |m_{ij}| \leq \beta, \quad i = j+1, ..., n,$$

其中 $m_{ij} = l_{ij}\sqrt{d_j}$. 为了满足这个要求，我们只需对3.4做一点改变：

$$d_j = \max\left(|c_{jj}|, \left(\frac{\theta_j}{\beta}\right)^2, \delta\right), \quad \text{with } \theta_j = \max_{j<i\leq n} |c_{ij}|.$$

此时(3.48)自动满足。
注意到θ_j可以比d_j先算出来，这也是我们引入c_{ij}的原因。
实质上，这个方法是一种对角校正。可以证明这个算法得到B_k的条件数是有界的。

Modified Symmetric Indefinite Factorization

另一种修正Hessian矩阵的方法是基于对称分解进行的，对于任意的对称矩阵A，有(3.51)

$$PAP^T = LBL^T$$

其中L是下三角阵，B是分块对角阵，每一块的维度是1或2，P是置换矩阵。
这个方法是先将原矩阵进行(3.51)的对称分解，然后对B进行特征值分解，再构造修正矩阵F使得

$$L(B+F)L^T$$

是正定的，由前面的特征值分解结果(3.43)启发，我们引入参数$\delta > 0$，使得

$$F = Q\text{diag}(\tau_i)Q^T, \quad \tau_i = \begin{cases} 0, & \lambda_i \geq \delta \\ \delta - \lambda_i, & \lambda_i < \delta \end{cases}$$

其中λ_i是B的特征值。构造F的目的是当A的$\lambda_{\min} < \delta$，使得$\lambda_{\min}(A+E) \approx \delta$。然而具体是否能达到这个目标，是不清楚的。

步长选择算法

下面我们考虑如何寻找步长，即极小化(3.54)

$$\phi(\alpha) = f(x_k + \alpha p_k)$$

或者是找到一个满足前述条件的α_k。这里我们假设p_k是下降方向，即$\phi'(0) < 0$。

线性搜索算法分为使用导数信息和不使用导数信息的算法，不使用导数信息会导致算法很低效，因此这里只考虑使用导数信息的算法。
典型的线性搜索算法包括两个过程：1. 确定区间 2. 选择步长

插值法

假设我们已经有初值α_0，如果初值满足条件，则返回初值
否则，利用$\phi(0), \phi'(0), \phi(\alpha_0)$进行二次插值，记最小值点为$\alpha_1$ (3.58):

$$\alpha_1 = -\frac{\phi'(0)\alpha_0^2}{2[\phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0]}$$

如果 α_1 满足条件，则返回α_1
否则，利用$\phi(0), \phi'(0), \phi(\alpha_0), \phi(\alpha_1)$进行三次插值，记最小点为$\alpha_2$：

$$\alpha_2 = \frac{-b + \sqrt{b^2 - 3a\phi'(0)}}{3a}$$

若满足条件，则终止，否则反复利用$\phi(0), \phi'(0)$ 和两个最近的点的信息对步长进行更新，如果α_i和α_{i-1}太接近，我们取$\alpha_i = \alpha_{i-1}/2$.
如果导数信息比较容易获得，我们也可以利用导数进行三次插值。使用三次插值的原因是，它常常能提供二次收敛速率。

步长初值

对于牛顿和拟牛顿法，$\alpha_0 = 1$。

而对于那些对步长没有信息的算法，初值估计是一个很重要的事。

一个比较常用的策略是假设一阶改变相同，即：

$$\alpha_0 = \alpha_{k-1}\frac{\nabla f_{k-1}^T p_{k-1}}{\nabla f_k^T p_k}$$

另一个比较有用的方法是对$f(x_{k-1}), f(x_k), \nabla f_{k-1}^T p_{k-1}$进行二次插值，取极小点为$\alpha_0$ (3.60)

$$\alpha_0 = \frac{2(f_k - f_{k-1})}{\phi'(0)}$$

如果$x_k \to x^*$ 是超线性收敛，那么上述比值收敛到1。如果我们把(3.60)设为

$$\alpha_0 \leftarrow \min(1, 1.01\alpha_0)$$

我们发现最终$\alpha_0 = 1$，便得到了牛顿法和拟牛顿法的超线性收敛性质。

基于Wolfe条件的线性搜索算法

这里我们假设p_k是下降方向，f在p_k方向上下有界。
这个算法分两步：

1. 首先从α_1开始，然后不断增加它直到它符合要求，或者得到了一个长度合适的区间
2. 当范围缩小到某个区间后，我们利用**zoom**算法不断缩小区间至找到合适的步长

我们把Wolfe 条件的两个条件分别称为*充分下降条件*和*曲率条件*，α_m 是用户可以定义的最大步长

Algorithm 3.5 (Line Search Algorithm).

```
Set alpha0 <- 0, choose alpha_m>0 and alpha(1) in (0,alpha_m);
i <- 1;
repeat
    Evaluate phi(alpha(i));
    if phi(alpha(i)) > phi(0) + c1*alpha(i)*phi'(0)
        or [phi(alpha(i)) >= phi(alphi(i-1)) and i>1]
        alpha* <- zoom(alpha(i-1),alpha(i)) and stop;
    Evaluate phi'(alpha(i));
    if |phi'(alpha(i))| <= -c2*phi'(0)
        alpha* <- alphi(i) and stop;
    if phi'(alphi(i)) >= 0
        alpha* <- zoom(alpha(i-1),alpha(i)) and stop;
    Choose alpha(i+1) in (alphi(i),alphi_m);
    i <- i+1;
end
```

最后一步选择新的步长可以用我们之前设计的插值算法或者别的能在有限步内能快速达到α_m的算法。

关于*zoom*算法，我们记其形式为$zoom(\alpha_l, \alpha_h)$:

1. α_l, α_h 包含的区间内存在满足Wolfe条件的步长
2. α_l 是满足充分下降条件且函数值最小的步长
3. α_h 满足$\phi'(\alpha_l)(\alpha_h - \alpha_l) < 0$

Algorithm 3.6 (zoom).

```
repeat
    Interpolate (using quadratic, cubic, or bisection) to find
        a trial step length alpha(j) between alpha_l and alpha_h;
    Evaluate phi(alpha(j));
    if phi(alpha(j)) > phi(0) + c1*alpha(j)*phi'(0) or phi(alpha(j)) >= phi(alpha_l)
        alpha_h <- alpha(j);
    else
        Evaluate phi'(alpha(j));
        if |phi'(alpha(j))| <= -c2*phi(0)
            alpha* <- alpha(j) and stop
        if phi'(alpha(j))(alpha_h - alpha_l) >= 0
            alpha_h <- alpha_l;
        alpha_l <- alpha(j)
end
```

线性搜索法可能在有限步内无法停止，因此需要设计一些终止条件。

使用Wolfe条件的经验参数是$c_1 = 10^{-4}, c_2 = 0.9$。

由于强Wolfe条件和Wolfe条件的运算代价差不多，但是控制强Wolfe条件的c_2可以直接控制搜索的质量，这对最速下降法和非线性共轭梯度法是很有意义的，因此强Wolfe条件的应用更广泛。