# PART I. OPTIMIZATION: CLASSICAL APPROACHES
(LECTURE 2)

Shpilev Petr Valerievich

Faculty of Mathematics and Mechanics, SPbU

September, 2025



Санкт-Петербургский государственный университет



## Comments

In today's lecture, we will explore the issue of convergence in Newton-type methods. We will also consider approaches to constructing Hessian approximations in quasi-Newton methods, specifically, algorithms based on Cholesky factorization.

Newton iteration defines the search direction as

$$p_k = -\left(\nabla^2 f_k\right)^{-1} \nabla f_k$$

where $\nabla^2 f_k$ is the Hessian matrix at $x_k$.

**Theorem 8**

Suppose that f is twice differentiable and that $\nabla^2 f(x)$ is Lipschitz continuous near a solution $x^*$ satisfying the second-order sufficient conditions (Theorem 4). Consider the iteration $x_{k+1} = x_k + p_k$, where $p_k$ is the Newton step.
Then:

(i) If the starting point $x_0$ is sufficiently close to $x^*$, the sequence of iterates converges to $x^*$.

(ii) The convergence rate of $\{x_k\}$ is quadratic.

(iii) The sequence $\{\|\nabla f_k\|\}$ converges quadratically to zero.



## Comments

We now consider the Newton iteration, for which the search direction is defined as $p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$.

As discussed previously, Newton's method can be extremely effective — it converges rapidly under favorable conditions. However, such conditions are typically satisfied only near the solution. Away from the solution, the Newton step may not even be a descent direction, which raises issues with global convergence. It is therefore essential to distinguish between two situations: the behavior far from the minimizer (which may require modifications), and the behavior close to the minimizer. In the future, we will look at the approaches used in the first situation, and now we will discuss the second in detail.

The following theorem shows that if the initial point is sufficiently close to the solution, and the function is well-behaved (in particular, the Hessian is positive definite and Lipschitz continuous), then Newton's method performs extremely well: the iterates converge quadratically, and the gradient norms go to zero at a quadratic rate. This is the key advantage of Newton's method compared to first-order methods like steepest descent.

The following theorem describes the behavior near the solution. We assume that the function f is twice differentiable, and that the Hessian of f is Lipschitz continuous in a neighborhood of a solution $x^*$. We also assume that $\nabla f(x^*) = 0$ and that $\nabla^2 f(x^*)$ is positive definite. Under these assumptions, consider the Newton iteration: $x_{k+1} = x_k + p_k$. Then the following conclusions hold:

First, if the initial point $x_0$ is sufficiently close to $x^*$, then the sequence $\{x_k\}$ converges to $x^*$. Second, the convergence is quadratic. Third, the sequence $\{\|\nabla f(x_k)\|\}$ converges to zero quadratically.

## Proof of Theorem 8 (I)



**Proof:** From the Newton step and the optimality condition $\nabla f(x^*) = 0$, we have

$$x_{k+1} - x^* = x_k + p_k - x^* = x_k - x^* - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

$$= \left(\nabla^2 f(x_k)\right)^{-1} \left[\nabla^2 f(x_k)(x_k - x^*) - (\nabla f(x_k) - \nabla f(x^*))\right]$$

By Taylor's theorem (Theorem 1):

$$\nabla f(x_k) - \nabla f(x^*) = \int_0^1 \nabla^2 f(x_k + t(x^* - x_k))(x_k - x^*)dt$$

Hence, $\nabla^2 f(x_k)(x_k - x^*) - (\nabla f(x_k) - \nabla f(x^*)) =$

$$\int_0^1 \left[\nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k))\right](x_k - x^*)dt \quad \Rightarrow$$

$$\|x_{k+1} - x^*\| \leq \|\nabla^2 f(x_k)^{-1}\| \cdot \int_0^1 \left\|\nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k))\right\| \cdot \|x_k - x^*\|dt$$



### Comments

From the definition of the Newton step and the optimality condition $\nabla f(x^*) = 0$, we can write the Newton update as follows:

$$x_{k+1} - x^* = x_k - x^* - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k).$$

We now factor out the inverse Hessian. This gives:

$$x_{k+1} - x^* = \left(\nabla^2 f(x_k)\right)^{-1} \left[\nabla^2 f(x_k)(x_k - x^*) - (\nabla f(x_k) - \nabla f(x^*))\right].$$

Now we apply Taylor's theorem to the gradient:

$$\nabla f(x_k) - \nabla f(x^*) = \int_0^1 \nabla^2 f(x_k + t(x^* - x_k))(x_k - x^*).dt$$

Substituting this into the previous formula, we obtain an upper bound for the norm of the difference between the Newton iteration and the solution. This gives us the basic inequality we will now use to derive a quadratic bound on the distance to the solution.

Since $\nabla^2 f$ is Lipschitz continuous near $x^*$, there exists $L > 0$ such that

$$\|\nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k))\| \le L \cdot t \cdot \|x_k - x^*\|$$

and we obtain:

$$\|x_{k+1} - x^*\| \le \|\nabla^2 f(x_k)^{-1}\| \cdot \int_0^1 Lt\|x_k - x^*\|^2 dt$$

$$= \|\nabla^2 f(x_k)^{-1}\| \cdot L \cdot \|x_k - x^*\|^2 \cdot \int_0^1 t dt = \frac{L}{2} \cdot \|\nabla^2 f(x_k)^{-1}\| \cdot \|x_k - x^*\|^2$$

Since $\nabla^2 f(x^*)$ is nonsingular there exist a constant $r > 0$ such that for all $x \in B_r(x^*)$:

$$\left\|\nabla^2 f(x)^{-1}\right\| \le 2 \left\|\nabla^2 f(x^*)^{-1}\right\|$$



## Comments

Since the Hessian of the function f is Lipschitz continuous near the point $x^*$, there exists a positive constant L such that $\|\nabla^2 f(x_k) - \nabla^2 f(x_k + t(x^* - x_k))\| \le L \cdot t \cdot \|x_k - x^*\|$. Substituting this bound into the previous estimate for the Newton update, we obtain: $\|x_{k+1} - x^*\| \le \|\nabla^2 f(x_k)^{-1}\| \cdot \int_0^1 Lt\|x_k - x^*\|^2 dt$.

Since $\|x_k - x^*\|$ does not depend on the integration variable, we factor it out. Then the result becomes: $\|x_{k+1} - x^*\| \le \|\nabla^2 f(x_k)^{-1}\| \cdot L \cdot \|x_k - x^*\|^2 \cdot \int_0^1 t dt$.

The value of that integral is $\frac{1}{2}$. So we arrive at the final bound: $\|x_{k+1} - x^*\| \le \frac{L}{2} \cdot \|\nabla^2 f(x_k)^{-1}\| \cdot \|x_k - x^*\|^2$. Next, since $\nabla^2 f(x^*)$ is nonsingular (by the condition of the theorem), and the function f is twice continuously differentiable, by continuity there exists constants $r > 0$ such that for all $x \in B_r(x^*)$: $\|\nabla^2 f(x)^{-1}\| \le 2\|\nabla^2 f(x^*)^{-1}\|$.

## Proof of Theorem 8 (conclusion)



Thus, for all $x_k \in B_r(x^*)$:

$$\|x_{k+1} - x^*\| \le \bar{L}\|x_k - x^*\|^2, \text{ where } \bar{L} = L\left\|\nabla^2 f(x^*)^{-1}\right\|$$

Choosing $x_0$ so that $\|x_0 - x^*\| \le \min(r, 1/(2\bar{L}))$ we can use this inequality inductively to deduce that the sequence converges to $x^*$, and the rate of convergence is quadratic.

By using the relations $x_{k+1} - x_k = p_k$ with $p_k = -\left(\nabla^2 f(x_k)\right)^{-1}\nabla f(x_k)$ (i.e. $\nabla f(x_k) + \nabla^2 f(x_k)p_k = 0$), we obtain that
$\|\nabla f(x_{k+1})\| = \|\nabla f(x_{k+1}) - \nabla f(x_k) - \nabla^2 f(x_k)p_k\| =$

$$= \left\|\int_0^1 \nabla^2 f(x_k + tp_k)(x_{k+1} - x_k)dt - \nabla^2 f(x_k)p_k\right\| \le$$

$$\le \int_0^1 \|\nabla^2 f(x_k + tp_k) - \nabla^2 f(x_k)\| \cdot \|p_k\|dt \le$$

$$\le \frac{1}{2}L\|p_k\|^2 \le \frac{1}{2}L\|\nabla^2 f(x_k)^{-1}\|^2\|\nabla f(x_k)\|^2 \le 2L\|\nabla^2 f(x^*)^{-1}\|^2\|\nabla f(x_k)\|^2$$

proving that the gradient norms converge to zero quadratically. □



### Comments

Thus, for all $x_k \in B_r(x^*)$, $\|x_{k+1} - x^*\| \le \bar{L}\|x_k - x^*\|^2$, where $\bar{L} = L\|\nabla^2 f(x^*)^{-1}\|$. Choosing $x_0$ so that $\|x_0 - x^*\| \le \min(r, 1/(2\bar{L}))$, we inductively deduce that $\{x_k\}$ converges to $x^*$ quadratically.

It remains for us to prove the third part of the theorem. Using the Newton step $x_{k+1} - x_k = p_k$, where $p_k = -\left(\nabla^2 f(x_k)\right)^{-1}\nabla f(x_k)$, we bound $\|\nabla f(x_{k+1})\|$. This norm equals $\left\|\int_0^1 \nabla^2 f(x_k + tp_k)p_k dt - \nabla^2 f(x_k)p_k\right\|$. Since the Hessian is Lipschitz continuous with constant $L$, this is at most $\int_0^1 Lt\|p_k\|^2 dt = \frac{1}{2}L\|p_k\|^2$. Substituting $p_k$, we get $\|p_k\|^2 \le \|\nabla^2 f(x_k)^{-1}\|^2\|\nabla f(x_k)\|^2$. Since for $x_k$ from the corresponding neighborhood of $x^*$ we have $\|\nabla^2 f(x_k)^{-1}\| \le 2\|\nabla^2 f(x^*)^{-1}\|$, it follows that $\|\nabla f(x_{k+1})\| \le 2L\|\nabla^2 f(x^*)^{-1}\|^2\|\nabla f(x_k)\|^2$, proving the gradient norms converge quadratically.

## Quasi-Newton Methods



In Quasi-Newton methods instead of the true Hessian $\nabla^2 f_k$, a matrix $B_k$ is used and updated after each step to reflect new curvature information.

**The idea:**

By Taylor's theorem we have:

$$\nabla f(x+p) = \nabla f(x) + \nabla^2 f(x)p + \int_0^1 [\nabla^2 f(x+tp) - \nabla^2 f(x)]pdt$$

Since $\nabla^2 f(x)$ is continuous, the integral term is $o(\|p\|)$. Letting $x = x_k$, $p = x_{k+1} - x_k$, we get:

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

Thus, when $x_k$ and $x_{k+1}$ lie in a region near the solution $x^*$

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

**Secant equation:**

$$B_{k+1}s_k = y_k, \quad s_k = x_{k+1} - x_k, \quad y_k = \nabla f_{k+1} - \nabla f_k$$



### Comments

The main drawback of the Newton direction is the need for the Hessian. Explicit computation of this matrix of second derivatives can sometimes be a cumbersome, error-prone, and expensive process. When Hessian is not positive definite, the Newton direction may not even be defined, since the inverse Hessian may not exist. Even when it is defined, it may not satisfy the descent property $\nabla f(x_k)^T p_k < 0$, in which case it is unsuitable as a search direction. In these situations, line search methods modify the definition of $p_k$ to make it satisfy the descent condition while retaining the benefit of the second-order information contained in Hessian.

Quasi-Newton methods provide an attractive alternative to Newton's method, as they avoid computing the Hessian while still achieving superlinear convergence. Instead of using the true Hessian, these methods use an approximation $B_k$, which is updated after each iteration based on observed gradient changes. The key idea is based on Taylor's theorem. Assume that the function f satisfies its conditions. Then $\nabla f(x+p) = \nabla f(x) + \nabla^2 f(x)p + \int_0^1 [\nabla^2 f(x+tp) - \nabla^2 f(x)]pdt$. Since the function is twice continuously differentiable, the integral is $o(\|p\|)$, so we approximate $\nabla f(x_{k+1}) - \nabla f(x_k) \approx \nabla^2 f(x_k)(x_{k+1} - x_k)$. This leads to the secant equation, which the updated matrix $B_{k+1}$ is required to satisfy.

We choose $B_{k+1}$ so that it satisfies the secant equation

$$B_{k+1}s_k = y_k, \quad s_k = x_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

Additional conditions are typically imposed:

- $B_{k+1}$ should be symmetric (like the true Hessian),
- $B_{k+1} - B_k$ should be of low rank.

**SR1 update:**

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

**BFGS update:**

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

SR1: rank-one update. BFGS: rank-two update, preserves positive definiteness if $B_0 \succ 0$ and $s_k^T y_k > 0$.



## Comments

To construct effective Quasi-Newton methods, we choose the Hessian approximation matrix $B_{k+1}$ to satisfy the secant equation $B_{k+1}s_k = y_k$, which ensures it mimics the Hessian's behavior along the step direction. We impose additional requirements on $B_{k+1}$: it should be symmetric, like the true Hessian of a twice-differentiable function, and the difference $B_{k+1} - B_k$ should be low-rank to keep updates computationally efficient. Two widely used Quasi-Newton updates are the Symmetric Rank-One, or SR1, and the Broyden-Fletcher-Goldfarb-Shanno, or BFGS, formulas.

The SR1 update is defined as: $B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$. This is a rank-one update, meaning the change $B_{k+1} - B_k$ has rank one, but it does not guarantee that $B_{k+1}$ remains positive definite, which can affect convergence.

The BFGS update is: $B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$. This is a rank-two update, as the change has rank two. BFGS preserves positive definiteness of $B_{k+1}$ if the initial matrix $B_0$ is positive definite and $s_k^T y_k > 0$, ensuring robust convergence for convex problems.

## Convergence of Quasi-Newton Methods



Assume the search direction is given by $p_k = -B_k^{-1}\nabla f(x_k)$ where $B_k$ is symmetric positive definite and updated at each iteration by a quasi-Newton formula.

### Theorem 9

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. Let the iteration be: $x_{k+1} = x_k + \alpha_k p_k$ with $p_k$ a descent direction and $\alpha_k$ satisfying Wolfe conditions with $c_1 \leq \frac{1}{2}$. If the sequence $\{x_k\}$ converges to a point $x^*$ such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, and if the search direction satisfies

$$\lim_{k\to\infty} \frac{\|\nabla f(x_k) + \nabla^2 f(x_k)p_k\|}{\|p_k\|} = 0$$

then:

(i) the step length $\alpha_k = 1$ is admissible for all k greater than a certain index $k_0$.

(ii) if $\alpha_k = 1$ for all $k > k_0$, then $\{x_k\}$ converges to $x^*$ superlinearly.

Proof in: Dennis, J. E., Schnabel, R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, 1996 (see Theorem 6.3.4, p. 123).



## Comments

We consider a search direction defined as $p_k = -B_k^{-1}\nabla f(x_k)$. The matrix $B_k$ is assumed to be symmetric and positive definite, and it is updated at each iteration using a quasi-Newton formula.

Theorem 9 states the following. Let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable. Suppose the sequence $\{x_k\}$ is generated by $x_{k+1} = x_k + \alpha_k p_k$, where $p_k$ is a descent direction. The step $\alpha_k$ satisfies the Wolfe conditions with parameter $c_1 \leq \frac{1}{2}$.

Suppose further that $\{x_k\}$ converges to a point $x^*$ such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Assume also that $\lim_{k\to\infty} \frac{\|\nabla f(x_k)+\nabla^2 f(x_k)p_k\|}{\|p_k\|} = 0$.

Then two conclusions follow. First: the step length $\alpha_k = 1$ is admissible for all k greater than a certain index $k_0$. Second: if $\alpha_k = 1$ for all $k > k_0$, then $\{x_k\}$ converges to $x^*$ superlinearly.

We will not prove this theorem. Its proof is similar to the proof of proposition 1.3.2 from Dimitri Bersekas' book "Nonlinear Programming".

Theorem 9 provides sufficient conditions for superlinear convergence of an iterative method in which the search direction is based on a quasi-Newton approximation. If the function is twice continuously differentiable, the search direction is a descent direction, and the step size satisfies the Wolfe conditions with a parameter less than or equal to one-half, and if the iterates converge to a point at which the gradient vanishes and the Hessian is positive definite, then two conclusions hold. First, a unit step is admissible for all sufficiently large iterations. Second, if a unit step is used at all sufficiently large iterations, the convergence is superlinear. By definition that means the ratio of the error at the next step to the current error tends to zero.

The condition in the theorem — that $\lim_{k\to\infty} \frac{\|\nabla f(x_k)+\nabla^2 f(x_k)p_k\|}{\|p_k\|} = 0$ — is automatically satisfied by Newton's method. In that case, $\nabla f(x_k) + \nabla^2 f(x_k)p_k = 0$, so the ratio is zero for all k. As a result, implementations of Newton's method with Wolfe or Goldstein line search conditions, and which always try a unit step first, will accept that step for all large k and achieve quadratic convergence locally.

This observation is also important in the analysis of quasi-Newton methods. Hence, we have the beautiful result that a superlinear convergence rate can be attained even if the sequence of quasi-Newton matrices $\{B_k\}$ does not converge to $\nabla^2 f(x^*)$; it suffices that the matrices $B_k$ become increasingly accurate approximations to the Hessian along the search directions $p_k$.

## Theorem 10

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. Consider the iteration $x_{k+1} = x_k + p_k$ (i.e. the step length $\alpha_k$ is uniformly 1), where $p_k = -B_k^{-1}\nabla f(x_k)$ and $B_k$ is symmetric and positive definite. Assume that $\{x_k\}$ converges to a point $x^*$ such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x)$ is positive definite at $x^*$ and Lipschitz continuous near $x^*$. Then $\{x_k\}$ converges to $x^*$ superlinearly if and only if

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

**Proof:** Assume that

$$\frac{\|(B_k - \nabla^2 f(x^*))p_k\|}{\|p_k\|} \to 0.$$

This condition is equivalent to $p_k - p_k^N = o(\|p_k\|)$, where $p_k^N = -\nabla^2 f(x_k)^{-1}\nabla f(x_k)$ is the Newton step. Indeed, since $\nabla^2 f(x_k)^{-1}$ is bounded above for $x_k$ sufficiently close to $x^*$ we have

$$p_k - p_k^N = \nabla^2 f(x_k)^{-1}(\nabla^2 f(x_k) - B_k)p_k = O(\|(\nabla^2 f(x_k) - B_k)p_k\|) = o(\|p_k\|).$$



## Comments

In the previous lecture, we discussed Theorem 9. That result provided sufficient conditions for superlinear convergence of line search methods. It stated that, if the step satisfies Wolfe conditions and the search direction approximates the Newton direction well enough, then a unit step becomes admissible after some point, and using unit steps leads to superlinear convergence. Now we formulate a sharper result — Theorem 10 — which provides necessary and sufficient conditions for superlinear convergence in the case where a quasi-Newton approximation to the Hessian is used.

Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, and the sequence $\{x_k\}$ converges to a point $x^*$ such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite and Lipschitz continuous near $x^*$. Suppose also that the search direction is given by $p_k = -B_k^{-1}\nabla f(x_k)$. Then, $\{x_k\}$ converges superlinearly if and only if $\lim_{k\to\infty} \frac{\|(B_k - \nabla^2 f(x^*))p_k\|}{\|p_k\|} = 0$.

This means that the quasi-Newton approximation does not have to converge to the full Hessian. It is enough that the approximation becomes increasingly accurate in the directions of the iterates.

We now prove Theorem 10. Assume that $\frac{\|(B_k - \nabla^2 f(x^*))p_k\|}{\|p_k\|} \to 0$.

The proof is based on showing that this condition is equivalent to $p_k - p_k^N = o(\|p_k\|)$, where $p_k^N = -\nabla^2 f(x_k)^{-1}\nabla f(x_k)$ is the Newton step. Indeed, $p_k - p_k^N = \nabla^2 f(x_k)^{-1}(\nabla^2 f(x_k) - B_k)p_k$. Since $\nabla^2 f(x_k)^{-1}$ is bounded above for $x_k$ sufficiently close to $x^*$, we have $p_k - p_k^N = O(\|(\nabla^2 f(x_k) - B_k)p_k\|) = o(\|p_k\|)$.

Since the Newton step converges quadratically (see Theorem 8) and by previous estimation, we get

$$\|x_k + p_k - x^*\| \le \|x_k + p_k^N - x^*\| + \|p_k - p_k^N\| = O(\|x_k - x^*\|^2) + o(\|p_k\|).$$

On the other hand, by Taylor's theorem (as we applied on slide 5) we have

$$\nabla f(x_k) = \nabla^2 f(x^*)(x_k - x^*) + o(\|x_k - x^*\|),$$

thus (since $\nabla^2 f(x^*)$ and $B_k$ are positive define)

$$\|p_k\| = \| - B_k^{-1}\nabla f(x_k)\| \le \| - B_k^{-1}\| \left(\|\nabla^2 f(x^*)\| \cdot \|(x_k - x^*)\| + o(\|x_k - x^*\|)\right) =$$

$$O(\|(x_k - x^*)\|) + o(\|x_k - x^*\|) = O(\|(x_k - x^*)\|)$$

hence:

$$\|x_{k+1} - x^*\| \le O(\|x_k - x^*\|^2) + o(O(\|(x_k - x^*)\|)) = o(\|x_k - x^*\|),$$

which completes the proof. □



## Comments

Next, we estimate $\|x_{k+1} - x^*\|$. This distance is bounded by $\|x_k + p_k^N - x^*\| + \|p_k - p_k^N\|$. The first term decreases quadratically — it is $O(\|x_k - x^*\|^2)$ — and the second term is $o(\|p_k\|)$.

On the other hand, by Taylor's theorem, $\nabla f(x_k) = \nabla^2 f(x^*)(x_k - x^*) + o(\|x_k - x^*\|)$. Therefore, $\|p_k\| = \| - B_k^{-1}\nabla f(x_k)\| \le \|B_k^{-1}\| \left(\|\nabla^2 f(x^*)\| \cdot \|x_k - x^*\| + o(\|x_k - x^*\|)\right) = O(\|x_k - x^*\|)$.

Substituting this back into our previous bound, we conclude that $\|x_{k+1} - x^*\| \le O(\|x_k - x^*\|^2) + o(\|x_k - x^*\|) = o(\|x_k - x^*\|)$. This is exactly the definition of superlinear convergence. The theorem is proved.

When $\nabla^2 f(x_k)$ is not positive definite, the Newton step

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

may not define a descent direction. To ensure positive definiteness, we replace the Hessian by a modified matrix:

$$B_k = \nabla^2 f(x_k) + E_k,$$

where $E_k = 0$ if the Hessian is sufficiently positive definite, and otherwise chosen to ensure $B_k \succ 0$.

**Algorithm 1: Line Search Newton with Modification**

1: **Given:** initial point $x_0$
2: for $k = 0, 1, 2, \ldots$ do
3: Compute $B_k = \nabla^2 f(x_k) + E_k$
4: Solve $B_k p_k = -\nabla f(x_k)$
5: Update $x_{k+1} = x_k + \alpha_k p_k$, where $\alpha_k$ satisfies Wolfe or Armijo conditions
6: end for





## Comments

Away from the solution, $\nabla^2 f(x_k)$ may not be positive definite. In this situation, the classic Newton step $\nabla^2 f(x_k) p_k = -\nabla f(x_k)$ may not produce a descent direction. This is problematic because descent directions are essential for line search strategies to work.

To address this, we replace the Hessian by a modified matrix $B_k = \nabla^2 f(x_k) + E_k$. If the Hessian is already sufficiently positive definite, $E_k = 0$. Otherwise, $E_k$ is chosen to ensure $B_k \succ 0$.

This leads to Algorithm 1. The method begins with an initial point $x_0$. At each iteration, we compute the modified matrix $B_k$, solve the linear system $B_k p_k = -\nabla f(x_k)$ to obtain the step direction, and update the iterate $x_{k+1} = x_k + \alpha_k p_k$. The step length $\alpha_k$ is determined by line search conditions such as Wolfe or Armijo. This modified Newton method can be applied from any starting point. It ensures that the search direction is always a descent direction and allows global convergence, provided the modification ensures that the matrices $B_k$ remain well-conditioned throughout the process.

## Theorem 11

Let f be twice continuously differentiable on an open set $D \subset \mathbb{R}^n$, and let $x_0 \in D$ be such that

$$\text{the level set} \quad L = \{x \in D : f(x) \le f(x_0)\} \quad \text{is compact.}$$

If the sequence $\{B_k\}$ generated by Algorithm 1 satisfies the bounded modified factorization property:

$$\kappa(B_k) = \|B_k\| \cdot \|B_k^{-1}\| \le C \text{ for all } k,$$

then

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

**Proof in**: J. J. Moré, D. Sorensen, *Newton's Method*, Argonne National Laboratory,. Argonne, Illinois, 1982 (see Theorem 4.15, p. 23).

**Interpretation:** If the step directions remain well-conditioned, the algorithm converges globally.



## Comments

We now state Theorem 11, which is a key result guaranteeing global convergence of the modified Newton method. Assume the function is twice continuously differentiable on an open set. The starting point is chosen so that the level set — that is, the set of points where the function value is less than or equal to its value at the starting point — is compact. This level set is denoted L, and being compact means, it is closed and bounded.

Next, assume that the matrices $B_k$, constructed by the algorithm, satisfy the bounded modified factorization property. That is, the condition number of each matrix $B_k$, defined as $\|B_k\| \cdot \|B_k^{-1}\|$, is uniformly bounded above by a constant C, for all iterations. Then the theorem asserts that $\lim_{k\to\infty} \|\nabla f(x_k)\| = 0$ — in other words, the method converges to a stationary point.

This result is crucial because it ensures that the modified Newton method converges globally, even when the Hessian is not positive definite at early steps. As long as the modifications maintain good numerical properties — especially well-conditioned search directions — the algorithm is guaranteed to work. Hence, Algorithm 1 is globally reliable under the assumptions of this theorem.

Assume that the iterates $\{x_k\}$ generated by Algorithm 1 converge to a point $x^*$, and that the Hessian $\nabla^2 f(x^*)$ is positive definite for all sufficiently large k. Then, for all sufficiently large k, the correction vanishes:

$$E_k = 0 \quad \Rightarrow \quad B_k = \nabla^2 f(x_k).$$

By Theorem 10, assuming $\alpha_k = 1$, the method converges quadratically:

$$\|x_{k+1} - x^*\| = O(\|x_k - x^*\|^2).$$

**Caveat:** If $\nabla^2 f(x^*)$ is nearly singular, $E_k$ may not vanish. Then the convergence rate may be only linear.



## Comments

Let us now examine the convergence rate of the modified Newton method. Suppose that the sequence of iterates converges to a point $x^*$, and the Hessian at that point is sufficiently positive definite. Then, after some point, the modification term $E_k$ becomes zero, and $B_k$ equals the actual Hessian. The method thus becomes the pure Newton method.

Under these conditions, and assuming that unit steps are accepted, the convergence becomes quadratic. That is, $\|x_{k+1} - x^*\| = O(\|x_k - x^*\|^2)$.

However, this favorable outcome does not always occur. If the Hessian at the solution is nearly singular — for example, poorly conditioned or close to losing positive definiteness — then the modification $E_k$ may never vanish. In such cases, the method does not revert to the classical Newton regime. The convergence may then be only linear. This limitation is essential to keep in mind in practical implementations.

In addition to ensuring that the modified matrix $B_k$ has a bounded condition number (to satisfy the requirements of Theorem 11), we aim to minimize the magnitude of the modification to preserve the second-order information encoded in the Hessian as much as possible. Furthermore, we seek a modified factorization that can be computed efficiently with reasonable computational cost.

**Goal:** Replace $\nabla^2 f(x_k)$ by a positive definite $B_k$ that is close to it, well-conditioned, and cheap to compute.

**Step 1:** Compute symmetric factorization

$$\nabla^2 f(x_k) = Q \Lambda Q^T,$$

where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$

**Step 2:** Modify negative or small eigenvalues

$$\hat{\lambda}_i = \max\{\lambda_i, \delta\} \quad \text{with small } \delta > 0$$

**Step 3:** Reconstruct

$$B_k = Q \, \hat{\Lambda} \, Q^T$$

**Guarantees:** $B_k$ is symmetric, positive definite and preserves curvature.



## Comments

We now introduce the first practical technique for modifying the Hessian — based on eigenvalue correction. The goal of this approach is to construct a positive definite matrix $B_k$ that:

(1) preserves as much second-order information as possible, (2) remains well-conditioned (so Theorem 11 applies), and is computationally feasible.

The method proceeds in three steps. In the first step, we compute a symmetric factorization of the Hessian: $\nabla^2 f(x_k) = Q \Lambda Q^T$, where $Q$ is orthogonal and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ contains the eigenvalues. In the second step, we correct the spectrum: any negative or very small eigenvalue is replaced by a small positive threshold $\delta$. This guarantees positive definiteness. In the third step, we reconstruct the matrix using the same orthogonal transformation and the modified eigenvalues: $B_k = Q \hat{\Lambda} Q^T$, where $\hat{\Lambda} = \mathrm{diag}(\hat{\lambda}_1, \ldots, \hat{\lambda}_n)$ and $\hat{\lambda}_i = \max\{\lambda_i, \delta\}$. The resulting matrix $B_k$ is symmetric, positive definite, and stays close to the original Hessian.

This method is attractive because it maintains curvature information, ensures theoretical convergence guarantees, and introduces minimal perturbation to the problem. Moreover, it can be implemented efficiently if the eigen structure is already available or cheaply approximated.

**Example:** Let

$$\nabla f(x_k) = \begin{pmatrix} 1 \\ -4 \\ 4 \end{pmatrix}, \quad \nabla^2 f(x_k) = \mathrm{diag}(8,\ 1,\ -2)$$

Then Newton step solves $p_k^N = -\nabla^2 f(x_k)^{-1} \nabla f(x_k) = (-0.125, 4, 2)^T$ and $\nabla f(x_k)^T p_k^N = 7.975 > 0$.
By the spectral decomposition theorem we have

$$\nabla^2 f(x_k) = Q \Lambda Q^T = \sum_{i=1}^{n} \lambda_i q_i q_i^T$$

where $Q = I$ and $\lambda_1 = 8$, $\lambda_2 = 1$, $\lambda_3 = -2$. Modify eigenvalues: set $\hat{\lambda}_3 = \delta = 10^{-8}$

$$B_k = \sum_{i=1}^{2} \lambda_i q_i q_i^T + \delta q_3 q_3^T = \mathrm{diag}(8,\ 1,\ 10^{-8})$$

Then the modified Newton step is

$$p_k = -B_k^{-1} \nabla f_k = -\sum_{i=1}^{2} \frac{1}{\lambda_i} q_i (q_i^T \nabla f_k) - \frac{1}{\delta} q_3 (q_3^T \nabla f_k) \approx -(4 \times 10^8) q_3$$



## Comments

Now we will consider a specific example that illustrates how a poor choice of $\delta$ can adversely affect the efficiency of the method. Let the gradient at the current point be $\nabla f(x_k) = (1, -4, 4)^T$, and let the Hessian be $\nabla^2 f(x_k) = \mathrm{diag}(8, 1, -2)$, which is clearly indefinite. The Newton step is given by $p_k^N = -\nabla^2 f(x_k)^{-1} \nabla f(x_k) = (-0.125, 4, 2)^T$. The inner product $\nabla f(x_k)^T p_k^N = 7.975 > 0$. Therefore, the Newton direction is not a descent direction.

To construct a suitable approximation, we apply the spectral decomposition theorem. The Hessian is equal to $\nabla^2 f(x_k) = Q \Lambda Q^T = \sum_{i=1}^{n} \lambda_i q_i q_i^T$. In this case, $Q = I$ and the eigenvalues are $\lambda_1 = 8$, $\lambda_2 = 1$, $\lambda_3 = -2$. We now modify the spectrum by replacing the negative eigenvalue with a small positive number $\delta = 10^{-8}$. The resulting modified matrix is $B_k = \mathrm{diag}(8, 1, 10^{-8})$.

The modified Newton step is given by $p_k = -B_k^{-1} \nabla f_k = -\sum_{i=1}^{2} \frac{1}{\lambda_i} q_i (q_i^T \nabla f_k) - \frac{1}{\delta} q_3 (q_3^T \nabla f_k) \approx -(4 \times 10^8) q_3$. Although f decreases along the direction $p_k$, its extreme length violates the spirit of Newton's method, which relies on a quadratic approximation of the objective function that is valid in a neighborhood of the current iterate $x_k$. It is therefore not clear that this search direction is effective.

Let $A$ be symmetric with spectral decomposition $A = Q\Lambda Q^T$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.

**Frobenius-norm-optimal correction:**

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

$$A + \Delta A = Q(\Lambda + \mathrm{diag}(\tau_i))Q^T$$

**Euclidean-norm-optimal diagonal correction:**

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

$$A + \Delta A = A + \tau I$$



## Comments

We now consider different strategies for modifying a Hessian matrix so that it becomes positive definite. A number of alternatives to the standard eigenvalue thresholding are possible. For example, we may flip the signs of negative eigenvalues; in the previous example, this would correspond to setting $\delta = 1$. Another option is to completely remove the component of the search direction along the negative curvature directions, which amounts to dropping the last term in the modified Newton step formula. Yet another approach is to choose $\delta$ adaptively, so that the resulting step does not become excessively large. This approach has similarities with trust-region methods.

These examples illustrate that there is a wide variety of possible modification strategies, and there is currently no agreement in the field about which is best. Some strategies prioritize preserving curvature, others focus on controlling the step norm, and others prioritize computational simplicity. Setting aside the question of how to choose $\delta$ for now, we examine how to construct a modification that is in a certain sense optimal.

Suppose that the matrix $A$ is symmetric and has a spectral decomposition: $A = Q\Lambda Q^T$, where $Q$ is an orthogonal matrix, and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.

To ensure that all eigenvalues of the modified matrix are greater than or equal to a small positive threshold $\delta$, we define a correction matrix $\Delta A$ as follows. It is given by $\Delta A = Q\,\mathrm{diag}(\tau_i)\,Q^T$, where each $\tau_i = 0$ if $\lambda_i \geq \delta$, and $\tau_i = \delta - \lambda_i$ otherwise. The resulting modified matrix is $A + \Delta A = Q(\Lambda + \mathrm{diag}(\tau_i))Q^T$. This correction has the smallest Frobenius norm among all possible corrections that guarantee the minimum eigenvalue is at least $\delta$.

Alternatively, we can apply a uniform diagonal shift. We compute a scalar $\tau = \max(0, \delta - \lambda_{\min}(A))$. The correction matrix is then $\Delta A = \tau I$, and the modified matrix becomes $A + \tau I$. This modification has the smallest Euclidean norm among all diagonal corrections that achieve the same eigenvalue bound.

The Frobenius-optimal correction changes only the problematic eigenvalues and leaves the large eigenvalues untouched. The diagonal shift is simpler and affects the entire spectrum equally. Both strategies are useful, and the choice between them depends on computational considerations and desired accuracy.

## Adding a Multiple of the Identity

**Goal:** Make $\nabla^2 f(x_k) + \tau I$ sufficiently positive definite by choosing $\tau > 0$

**Algorithm 2** (Cholesky with Added Multiple of the Identity):

Choose parameter $\beta > 0$
if all diagonal elements of $\nabla^2 f(x_k)$ are positive then
  $\tau_0 \leftarrow 0$
else
  $\tau_0 \leftarrow -\min a_{ii} + \beta$
end if
for $j = 0, 1, 2, \ldots$ do
  if **Cholesky factorization of** $A + \tau_k I$ **succeeds** (i.e. $A + \tau_k I \succ 0$ ) then
    **stop and return factor** $L$ **(**$LL^T = A + \tau_k I$**)**
  else
    $\tau_{j+1} \leftarrow \max(2\tau_j,\ \beta)$
  end if
end for

**Note:** The choice of $\beta$ is heuristic; a typical value is $\beta = 10^{-3}$





## Comments

This slide presents Algorithm 2, which modifies the Hessian in Newton's method to ensure it is positive definite, as part of Algorithm 1. The goal is to find a scalar $\tau > 0$ such that the matrix $\nabla^2 f(x_k) + \tau I$ is positive definite, ensuring the Newton step is a descent direction.

The algorithm starts by selecting a positive parameter $\beta$, typically $10^{-3}$, a heuristic choice. If all diagonal elements of the Hessian are positive, we set $\tau_0 = 0$, as the Hessian may already be positive definite. Otherwise, $\tau_0$ is set to $-\min a_{ii} + \beta$, providing an initial shift. The algorithm then loops over indices $j$, attempting the Cholesky factorization of the Hessian plus $\tau_j I$.

Why Cholesky? Computing $A + \tau I$ is simple, but we need to confirm the matrix is positive definite, meaning all eigenvalues are positive. Cholesky factorization tests this efficiently: it succeeds only if the matrix is positive definite. It also provides the factor $L$ to solve the Newton system, $(\nabla^2 f(x_k) + \tau I)p_k = -\nabla f(x_k)$, without additional cost.

If the factorization succeeds, we return the factor, as the matrix is positive definite. If it fails, $\tau_j$ is too small, so we set $\tau_{j+1} = \max(2\tau_j, \beta)$ and try again. The heuristic $\beta$ balances making $\tau$ large enough for positive definiteness and small enough to preserve the Hessian's information. This method is simple but may require multiple factorizations, which can be costly, as each trial of $\tau$ needs a new factorization.

# Cholesky Factorization for Newton's Method



## Definition

For a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$, the Cholesky factorization is:

$$A = LL^T,$$

where L is lower triangular with positive diagonal elements.

**Application to Newton System:**

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

Using $\nabla^2 f(x_k) = LL^T$:

1. Solve $Lz = -\nabla f(x_k)$ (forward substitution).
2. Solve $L^T p_k = z$ (backward substitution).

**Why Cholesky?**

- Tests positive definiteness: Factorization exists only if A is positive definite.
- Efficient: Costs $\frac{1}{3}n^3$ flops, versus $\frac{2}{3}n^3$ for LU factorization.
- Solves system directly: Triangular systems are fast ($O(n^2)$ flops).



## Comments

The Cholesky factorization applies to a symmetric positive definite matrix A, such as the Hessian or a modified Hessian in Newton's method. It decomposes A into $LL^T$, where L is a lower triangular matrix with positive diagonal elements. This factorization is crucial for solving the Newton system, $\nabla^2 f(x_k)p_k = -\nabla f(x_k)$, which determines the search direction in Algorithm 1. Using the factorization, we replace the Hessian with $LL^T$. This transforms the system into two easier steps.

First, we solve $Lz = -\nabla f(x_k)$, a forward substitution since L is lower triangular, which is computationally simple. Second, we solve $L^T p_k = z$, a backward substitution, also straightforward due to the triangular structure.

Why use Cholesky factorization? It serves two purposes.

First, it tests whether the matrix is positive definite. The factorization exists only if all eigenvalues of A are positive, confirming that the Newton direction is a descent direction. This is why Algorithm 2 uses Cholesky to check if $\nabla^2 f(x_k) + \tau I$ is positive definite.

Second, it's efficient. Computing the Cholesky factorization costs about $\frac{1}{3}n^3$ flops, half the cost of a general LU factorization. Solving the two triangular systems costs $O(n^2)$ flops, much faster than direct methods.

In Algorithm 2, Cholesky ensures the modified Hessian, $\nabla^2 f(x_k) + \tau I$, is positive definite and provides the factors to solve the Newton system efficiently. This method avoids expensive eigenvalue computations while ensuring a robust and fast solution process. Next, we'll explore the modified Cholesky factorization, which adjusts the Hessian during factorization to handle non-positive definite cases.

## Modified Cholesky Factorization

**Goal:** Modify $\nabla^2 f(x_k)$ during factorization to ensure positive definiteness, avoiding multiple factorizations (see Algorithm 2).

**Algorithm 3** (Modified Cholesky, $LDL^T$ Form):

1: for $j = 1, 2, \ldots, n$ do
2: $\quad c_{jj} \leftarrow a_{jj} - \sum_{x=1}^{j-1} d_x l_{jx}^2$
3: $\quad \theta_j \leftarrow \max_{j<i\le n} |c_{ij}|$, where $c_{ij} \leftarrow a_{ij} - \sum_{x=1}^{j-1} d_x l_{ix} l_{jx}$
4: $\quad d_j \leftarrow \max\left(|c_{jj}|, \left(\frac{\theta_j}{\beta}\right)^2, \delta\right)$
5: $\quad$ for $i = j+1, \ldots, n$ do
6: $\quad\quad c_{ij} \leftarrow a_{ij} - \sum_{x=1}^{j-1} d_x l_{ix} l_{jx}$
7: $\quad\quad l_{ij} \leftarrow c_{ij}/d_j$
8: $\quad$ end for
9: end for

**Note:** Parameters $\delta, \beta > 0$ ensure $d_j \ge \delta$ and bounded factors. Produces $PAP^T + E = LDL^T$, where E is diagonal.





## Comments

Now let's consider the Modified Cholesky Factorization as a more efficient alternative to Algorithm 2 for ensuring a positive definite Hessian in Newton's method. The goal is to modify the Hessian, $\nabla^2 f(x_k)$, during its factorization to make it positive definite, avoiding the repeated factorizations required in Algorithm 2.

Algorithm 3 computes an $LDL^T$ factorization, where L is lower triangular with unit diagonal, and D is diagonal with positive elements. For each column j, it calculates $c_{jj}$ as the diagonal element adjusted by prior terms, and $\theta_j$ as the maximum absolute off-diagonal element in the remaining columns. The key modification is in setting $d_j$, the diagonal element of D, to the maximum of $|c_{jj}|$, $(\theta_j/\beta)^2$, and a positive parameter $\delta$. This ensures $d_j$ is at least $\delta$, guaranteeing positive definiteness, and controls the size of the factors to keep them bounded. For each row $i > j$, it computes $c_{ij}$ and sets $l_{ij} = c_{ij}/d_j$, forming the off-diagonal elements of L. Parameters $\delta$ and $\beta$, both positive, ensure $d_j$ is sufficiently large and the factors L and D are not too large, maintaining numerical stability.

The algorithm may include row and column permutations, represented by a permutation matrix P, producing $PAP^T + E = LDL^T$, where E is a nonnegative diagonal matrix that's zero if the Hessian is already positive definite. This approach modifies the Hessian only when necessary, preserving its second-order information while ensuring a descent direction for Algorithm 1.

Next, we'll illustrate this with an example to show how the modification works in practice.

## Example of Modified Cholesky Factorization

**Example 1:** Let $A = \begin{bmatrix} 4 & 2 & 1 \\ 2 & -1 & 0 \\ 1 & 0 & 2 \end{bmatrix}$, EV $= 5, \sqrt{3}, -\sqrt{3}$. Set $\delta = 0.1$, $\beta = 1$.

**Steps:**

- j $= 1$: $c_{11} = a_{11} = 4$, $\theta_1 = \max(|a_{21}|, |a_{31}|) = 2$, $d_1 = \max(4, 4, 0.1) = 4$.

$$l_{21} = a_{21}/d_1 = 0.5, \quad l_{31} = a_{31}/d_1 = 0.25.$$

- j $= 2$: $c_{22} = a_{22} - d_1 l_{21}^2 = -1 - 4 \cdot 0.5^2 = -2$, $\theta_2 = |a_{32} - d_1 l_{31} l_{21}| = 0.5$,

$$d_2 = \max(2, 0.25, 0.1) = 2, \quad l_{32} = (a_{32} - d_1 l_{31} l_{21})/d_2 = -0.25.$$

- j $= 3$: $c_{33} = a_{33} - d_1 l_{31}^2 - d_2 l_{32}^2 = 2 - 4 \cdot 0.25^2 - 2 \cdot (-0.25)^2 = 1.625$,

$$d_3 = \max(1.625, 0, 0.1) = 1.625.$$

**Result:** $A + E = LDL^T$, with $E = \mathrm{diag}(0, 4, 0)$, where $E_{jj} = d_j - c_{jj}$.





### Comments

Let's consider a three-by-three symmetric matrix $A = \begin{bmatrix} 4 & 2 & 1 \\ 2 & -1 & 0 \\ 1 & 0 & 2 \end{bmatrix}$. The eigenvalues of this matrix are 5, $\sqrt{3}$, and $-\sqrt{3}$, indicating that A is indefinite and needs modification before it can be used in Newton's method.

We now apply the Modified Cholesky Factorization. We choose $\delta = 0.1$ and $\beta = 1$. Then, step by step, we compute the values used in the factorization. For each index j from 1 to 3, we calculate the corrected diagonal entry $d_j$ and the multipliers $l_{ij}$ according to the algorithm.

Once all steps are completed, we can compute the correction matrix E as the difference between $d_j$ and the elements $c_{jj}$. This gives a diagonal matrix $E = \mathrm{diag}(0, 4, 0)$. Thus, we obtain that $A + E = LDL^T$, where L is unit lower triangular and D is diagonal with strictly positive entries.

This example shows how, using Algorithm 3, we can modify the matrix A to ensure positive definiteness with minimal changes. In the next part, we will explain why the specific formula for $d_j$ used in step 4 is chosen to balance the need for positive definiteness and numerical stability.

## Cholesky Factorization, $LDL^T$ Form

Every symmetric positive definite matrix A can be written as $A = LDL^T$ where L is a lower triangular matrix with unit diagonal elements and D is a diagonal matrix with positive elements on the diagonal.

**Example 2:** Consider the case $n = 3$. The equation $A = LDL^T$ is given by

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \begin{bmatrix} 1 & l_{21} & l_{31} \\ 0 & 1 & l_{32} \\ 0 & 0 & 1 \end{bmatrix}.$$

$$a_{11} = d_1,$$

$$a_{21} = d_1 l_{21} \implies l_{21} = \frac{a_{21}}{d_1},$$

$$a_{31} = d_1 l_{31} \implies l_{31} = \frac{a_{31}}{d_1},$$

$$a_{22} = d_1 l_{21}^2 + d_2 \implies d_2 = a_{22} - d_1 l_{21}^2,$$

$$a_{32} = d_1 l_{31} l_{21} + d_2 l_{32} \implies l_{32} = \frac{a_{32} - d_1 l_{31} l_{21}}{d_2},$$

$$a_{33} = d_1 l_{31}^2 + d_2 l_{32}^2 + d_3 \implies d_3 = a_{33} - d_1 l_{31}^2 - d_2 l_{32}^2.$$





## Comments

It is well-known from algebra, that every symmetric positive definite matrix A can be written as $A = LDL^T$ where L is a lower triangular matrix with unit diagonal elements and D is a diagonal matrix with positive elements on the diagonal.

Let us illustrates the Cholesky factorization in $LDL^T$ form for a three-by-three symmetric positive definite matrix A. We express A as $LDL^T$. The matrices L and D are shown on the slide.

By equating the elements of A with those of $LDL^T$ column by column, we derive formulas for the elements of L and D. For the first column, $a_{11} = d_1$, and the subdiagonal elements yield $l_{21} = a_{21}/d_1$ and $l_{31} = a_{31}/d_1$.

For the second column, $a_{22} = d_1 l_{21}^2 + d_2$, giving $d_2 = a_{22} - d_1 l_{21}^2$, and $a_{32}$ yields $l_{32} = (a_{32} - d_1 l_{31} l_{21})/d_2$.

The third column gives $d_3 = a_{33} - d_1 l_{31}^2 - d_2 l_{32}^2$. For a positive definite matrix A, the values $d_j$ computed in these formulas are equivalent to $c_{jj}$ in Algorithm 3, ensuring positive diagonal elements.

Further, we will discuss why Algorithm 3 uses a modified formula for $d_j$ to handle cases where A is not positive definite, ensuring numerical stability and positive definiteness.

## Modified Cholesky Factorization. Formula for $d_j$

**Problem with standard $LDL^T$:**

- For indefinite $\nabla^2 f(x_k)$, $A = LDL^T$ may not exist (e.g., $c_{jj} \leq 0$).
- If it exists, L, D elements can grow arbitrarily large, causing numerical instability.
- Post-factorization fix (e.g., forcing $d_j > 0$) may drastically alter A.

**Algorithm 3 solution:**

$$d_j = \max\left(|c_{jj}|, \left(\frac{\theta_j}{\beta}\right)^2, \delta\right),$$

ensures:

- $d_j \geq \delta > 0$: Positive definiteness of $A + E$.
- $d_j \geq \left(\frac{\theta_j}{\beta}\right)^2$: Bounds $|l_{ij}| \leq \beta$, controlling L.
- $d_j \geq |c_{jj}|$: Minimizes $E_{jj} = d_j - c_{jj}$.

**Result:** $A + E = LDL^T$, with bounded L, D and small E.

**Note:** Parameters $\delta, \beta > 0$ balance stability and fidelity to $\nabla^2 f(x_k)$.





## Comments

As we just saw in a standard $LDL^T$ factorization, we compute A as $LDL^T$, setting $d_j$ equal to $c_{jj}$ at each step. For the Hessian, $\nabla^2 f(x_k)$, this assumes positive definiteness. If the Hessian is indefinite, with negative or zero eigenvalues, $c_{jj}$ may be negative or zero, causing the factorization to fail. Even if the factorization exists for an indefinite matrix, the elements of L and D can become arbitrarily large, leading to numerical instability. For example, small diagonal elements amplify off-diagonal elements in L, making the factorization unreliable.

One might consider computing the standard factorization and then adjusting D afterward to make its elements positive. However, this post-hoc modification can significantly change the matrix, producing a result far from the original Hessian, which loses valuable second-order information needed for Algorithm 1.

Instead, Algorithm 3 modifies the Hessian during factorization by setting $d_j$ to the maximum of $|c_{jj}|$, $(\theta_j/\beta)^2$, and $\delta$, where $\theta_j$ is the largest absolute off-diagonal element in the remaining columns, and $\delta$ and $\beta$ are positive parameters. This formula achieves three goals.

First, $d_j \geq \delta$, ensuring all diagonal elements of D are positive, so $A + E$, where E is the modification matrix, is positive definite, suitable for solving the Newton system.

Second, $d_j \geq (\theta_j/\beta)^2$, which bounds the off-diagonal elements $l_{ij}$, computed as $c_{ij}/d_j$. Since $c_{ij}$ is at most $\theta_j$, this ensures $|l_{ij}| \leq \beta$, keeping L well-conditioned and preventing large factors that cause instability.

Third, $d_j = |c_{jj}|$ when $c_{jj}$ is sufficiently large and positive, minimizing $E_{jj} = d_j - c_{jj}$. This keeps $A + E$ close to the original Hessian, preserving its curvature information for effective Newton steps.

The result is $A + E = LDL^T$, where E is a diagonal matrix with entries $d_j - c_{jj}$, and both L and D have bounded elements, ensuring numerical stability and a positive definite matrix. Parameters $\delta$ and $\beta$ balance the trade-off between making E large enough for positive definiteness and small enough to stay close to the Hessian.

This approach improves on Algorithm 2 by performing a single, stable factorization, as seen in our first example. Next, we'll discuss the modified symmetric indefinite factorization, another method for handling indefinite Hessians.

## Example: Symmetric Indefinite Factorization

Any symmetric matrix A, whether positive definite or not, can be written as $PAP^T = LBL^T$ (where P: permutation, L: unit lower triangular, B: block diagonal with blocks of dimension 1 or 2).

### Definition

Inertia of a matrix is the number of positive, negative, and zero eigenvalues.

**Factorization Example:** Let's $P = \begin{bmatrix} e_1 & e_4 & e_3 & e_2 \end{bmatrix}$,

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & -1 & 1 & 2 \\ 1 & 1 & 2 & 1 \\ 0 & 2 & 1 & 3 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & \frac{1}{3} & 1 & 0 \\ 2 & \frac{2}{3} & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & \frac{2}{3} & -\frac{5}{3} \\ 0 & 0 & -\frac{5}{3} & -\frac{19}{3} \end{bmatrix}.$$

**Inertia:** B has one $1 \times 1$ positive block, one $2 \times 2$ block (three positive, zero zero, and one negative eigenvalue).





### Comments

We mentioned earlier that attempting to compute the $LDL^T$ factorization of an indefinite matrix, where D is a diagonal matrix, is inadvisable because even if the factors L and D are well defined, they may contain entries that are larger than the original elements of A, thus amplifying rounding errors that arise during the computation.

However, by using the block diagonal matrix B, which allows $2 \times 2$ blocks as well as $1 \times 1$ blocks on the diagonal, we can guarantee that the $LBL^T$ factorization always exists and can be computed by a numerically stable process. The symmetric indefinite factorization allows us to determine the inertia of a matrix, that is, the number of positive, zero, and negative eigenvalues.

One can show that the inertia of B equals the inertia of A. Moreover, the $2 \times 2$ blocks in B are always constructed to have one positive and one negative eigenvalue. Thus, the number of positive eigenvalues in A equals the number of positive $1 \times 1$ blocks plus the number of $2 \times 2$ blocks.

Here is an example of such a factorization for an indefinite $4 \times 4$ symmetric matrix A. The matrices P, L, and B are computed to satisfy $PAP^T = LBL^T$, revealing the inertia of A: three positive, zero zero, and one negative eigenvalue.

## Modified Symmetric Indefinite Factorization

We now consider techniques for finding a minimum of the one-dimensional function

**Modification Steps:**

1. Compute $B = Q\Lambda Q^T$, define $F = Q\,\mathrm{diag}(\tau_i)Q^T$, where

$$\tau_i = \begin{cases} 0, & \lambda_i \geq \delta, \\ \delta - \lambda_i, & \lambda_i < \delta, \end{cases}$$

   for a positive $\delta$.

2. Form modified factorization: $P(A+E)P^T = L(B+F)L^T$, where $E = P^T L F L^T P$.

**Exercise:** For $A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & -1 & 1 & 2 \\ 1 & 1 & 2 & 1 \\ 0 & 2 & 1 & 3 \end{bmatrix}$, modify B to ensure $\lambda_{\min}(B+F) \geq \delta$.

**Note:** F ensures positive definiteness with minimal Frobenius norm.





## Comments

An indefinite symmetric factorization algorithm can be modified to ensure that the modified factors are the factors of a positive definite matrix. The idea is to modify the matrix B so that as a result of this modification, all the eigenvalues become positive. To do this, first we need to calculate the spectral decomposition of this matrix. The spectral decomposition of B is computed as $B = Q\Lambda Q^T$, where $\Lambda$ contains the eigenvalues $\lambda_i$.

A modification matrix F is defined as $F = Q\,\mathrm{diag}(\tau_i)Q^T$, where $\tau_i = 0$ if $\lambda_i \geq \delta$, and $\tau_i = \delta - \lambda_i$ otherwise, for a positive $\delta$. This ensures that $B + F$ has a minimum eigenvalue of at least $\delta$. The modification F is designed to minimize the Frobenius norm while ensuring positive definiteness. The modified factorization becomes $P(A+E)P^T = L(B+F)L^T$, where $E = P^T L F L^T P$. (Note that E will not be diagonal, in general.)

Hence, in contrast to the modified Cholesky approach, this modification strategy changes the entire matrix A, not just its diagonal. The goal is for the modified matrix to satisfy that the smallest eigenvalue of $A + E$ is approximately equal to $\delta$, whenever the smallest eigenvalue of the original matrix A is less than $\delta$. However, it is unclear whether this goal is always achieved.

As a small exercise, you can try to modify matrix B from the previous example and construct a modified factorization for matrix A.

Minimize one-dimensional function:

$$\phi(\alpha) = f(x_k + \alpha p_k). \qquad (*)$$

Assume $p_k$ is a descent direction ($\phi'(0) < 0$), so $\alpha > 0$.

**Convex Quadratic:**

$$f(x) = \frac{1}{2}x^T Q x - b^T x, \quad \alpha_k = -\frac{\nabla f_k^T p_k}{p_k^T Q p_k}. \qquad (**)$$

**Procedure:**

- Bracketing phase: Find interval $[\bar{a}, \bar{b}]$ with acceptable step lengths.
- Selection phase: Zoom in to locate final step length.

**Note:** Derivative-based algorithms are efficient, often terminating after one evaluation near solutions.



## Comments

Let us consider techniques for finding a minimum of the one-dimensional function $\phi(\alpha) = f(x_k + \alpha p_k)$, as given in equation $(*)$. We assume that $p_k$ is a descent direction—that is, $\phi'(0) < 0$—so that our search can be confined to positive values of $\alpha$. This ensures that moving along $p_k$ will reduce the function value for some positive step length. For a convex quadratic function, defined as $f(x) = \frac{1}{2}x^T Q x - b^T x$, the one-dimensional minimizer along the ray $x_k + \alpha p_k$ can be computed analytically. This minimizer is given by $\alpha_k = -\frac{\nabla f(x_k)^T p_k}{p_k^T Q p_k}$, as shown in equation $(**)$.

For general nonlinear functions, it is necessary to use an iterative procedure, as finding the exact minimizer is too costly. Line search procedures consist of two phases: a bracketing phase that finds an interval containing acceptable step lengths, and a selection phase that zooms in to locate the final step length. Algorithms that use derivative information are more efficient, as they can often determine whether a suitable step length has been located after just one evaluation, particularly when the current iterate is close to the solution. This efficiency is critical for the robustness and performance of nonlinear optimization methods.

The aim is to find a value of $\alpha$ that satisfies the sufficient decrease condition (1), without being "too small." The sufficient decrease condition is:

$$\phi(\alpha) \leq \phi(0) + c_1 \alpha \phi'(0).$$

If the initial trial value $\alpha_0 > 0$ does not satisfy this condition, an improved estimate can be computed using interpolation.

**Quadratic interpolation:**
Constructed using function values at $\alpha_0 > 0$ and 0, and the derivative at 0. Then $m(\alpha)$ has the form:

$$m(\alpha) = \phi(0) + \phi'(0)\alpha + \left[\frac{\phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0}{\alpha_0^2}\right]\alpha^2,$$

and its minimizer is:

$$\alpha = -\frac{\phi'(0)\alpha_0^2}{2\left[\phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0\right]}.$$



## Comments

The purpose of using interpolation in the line search procedure is to find a value of $\alpha$ that satisfies the sufficient decrease condition without being too small. Recall that the sufficient decrease condition is: $\phi(\alpha) \leq \phi(0) + c_1 \alpha \phi'(0)$. The constant $c_1$ is usually chosen to be small in practice (say $c_1 = 10^{-4}$). We typically start with a positive initial guess $\alpha_0$. If this initial trial step does not satisfy the condition, then a refined estimate can be generated using interpolation based on previously computed function and derivative values.

This slide describes the quadratic interpolation strategy. A quadratic model $m(\alpha)$ is constructed using the value of the function at 0, its derivative at 0, and the value of the function at $\alpha_0$. This model approximates $\phi$ locally and provides an analytical formula for the minimum of the quadratic, which serves as a new candidate for the step length. It's important to be cautious when using such formulas. In some cases, the denominator might be close to zero or even negative, leading to unreliable or invalid step sizes. Therefore, the result of interpolation must always be checked before being accepted as the next trial step.

If the sufficient decrease condition is satisfied at $\alpha_1$, we terminate the search. Otherwise, we build a cubic function $\phi_c(\alpha)$ interpolating:

$$\phi(0), \quad \phi'(0), \quad \phi(\alpha_0), \quad \phi(\alpha_1).$$

The cubic model takes the form:

$$\phi_c(\alpha) = a\alpha^3 + b\alpha^2 + \alpha\phi'(0) + \phi(0),$$

where coefficients a and b are given by:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\alpha_0^2\alpha_1^2(\alpha_1 - \alpha_0)} \begin{bmatrix} \alpha_0^2 & -\alpha_1^2 \\ -\alpha_0^3 & \alpha_1^3 \end{bmatrix} \begin{bmatrix} \phi(\alpha_1) - \phi(0) - \phi'(0)\alpha_1 \\ \phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0 \end{bmatrix}.$$

The minimizer of $\phi_c(\alpha)$ is:

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





## Comments

If the sufficient decrease condition is satisfied at $\alpha_1$, we terminate the search. Otherwise, we construct a cubic model that interpolates four known values: $\phi(0)$, $\phi'(0)$, and $\phi(\alpha_0)$ and $\phi(\alpha_1)$. That is, the model must match the values of the function $\phi$ and its derivative at these points. The cubic model has an explicit form with coefficients a and b, which are determined by solving a linear system.

These coefficients depend on the differences between the function values and their linear approximations at $\alpha_0$ and $\alpha_1$. In addition, the model provides an analytic formula for its minimizer. If the discriminant under the square root in this formula is negative, it indicates that the cubic model is unreliable. In such cases, we discard the interpolation result and fall back to a safer choice - for example, halving the previous step length. This safeguard ensures steady progress and prevents the algorithm from stalling. If, at the i-th iteration, the computed $\alpha$ is either too close to the previous value or significantly smaller, we do the same: we reset $\alpha_i$ to be equal to half of $\alpha_{i-1}$, where $\alpha_i$ and $\alpha_{i-1}$ denote the step lengths used at iterations i and $i-1$ of the optimization algorithm, respectively.

Why not just always halve the step size instead of using interpolation? Because interpolation uses available information - function and derivative values - to more intelligently predict a suitable step. This often leads to finding an acceptable step with fewer function evaluations. Halving remains a fallback strategy when interpolation fails.

When both function and directional derivative values are available at little cost, we can interpolate $\phi$ and $\phi'$ at two previous points $\alpha_{i-1}$ and $\alpha_i$:

- A unique cubic interpolant always exists;
- Its minimizer $\alpha_{i+1}$ lies in $[\bar{a}, \bar{b}]$ and is given by:

$$\alpha_{i+1} = \alpha_i - (\alpha_i - \alpha_{i-1}) \left[ \frac{\phi'(\alpha_i) + d_2 - d_1}{\phi'(\alpha_i) - \phi'(\alpha_{i-1}) + 2d_2} \right],$$

where

$$d_1 = \phi'(\alpha_{i-1}) + \phi'(\alpha_i) - 3\frac{\phi(\alpha_{i-1}) - \phi(\alpha_i)}{\alpha_{i-1} - \alpha_i},$$

$$d_2 = \text{sign}(\alpha_i - \alpha_{i-1}) \left[ d_1^2 - \phi'(\alpha_{i-1})\phi'(\alpha_i) \right]^{1/2}.$$



## Comments

This slide presents an alternative interpolation strategy that uses both function and derivative values at two points. The key idea is that if evaluating the derivative is not much more expensive than evaluating the function itself, we can leverage more information to build a better model. In particular, we use a cubic polynomial that interpolates both function and derivative values at $\alpha_{i-1}$ and $\alpha_i$. This interpolation always yields a unique cubic, whose minimizer is given by the formula at the top. Specifically, the new trial point $\alpha_{i+1}$ is equal to $\alpha_i - (\alpha_i - \alpha_{i-1}) \left[ \frac{\phi'(\alpha_i)+d_2-d_1}{\phi'(\alpha_i)-\phi'(\alpha_{i-1})+2d_2} \right]$.

The quantity $d_1$ is the sum of the derivatives at $\alpha_{i-1}$ and $\alpha_i$, minus three times the difference in function values divided by the difference in step lengths. The value $d_2$ incorporates curvature information and includes a square root, which can be complex. In such case, the interpolation is deemed unreliable, and the algorithm discards it, reverting to a safer strategy like halving the previous step length to maintain steady progress. If the interpolation is successful, the process can be repeated by replacing either the data at $\alpha_{i-1}$ or $\alpha_i$ with the new function and derivative values at $\alpha_{i+1}$, depending on the line search termination conditions. The decision on which of $\alpha_{i-1}$ and $\alpha_i$ should be kept and which discarded depends on the specific conditions used to terminate the line search; we discuss this issue next in the context of the Wolfe conditions.

In general, cubic interpolation is a powerful strategy because it accounts for changes in the function's curvature, often leading to a quadratic rate of convergence, which means the algorithm can find an optimal step length more quickly than simpler methods.

- Newton/quasi-Newton: Set $\alpha_k^{(0)} = 1$.
- Poorly scaled methods (e.g., steepest descent):
  - Match first-order change:

$$\alpha_k^{(0)} = \alpha_{k-1} \frac{\nabla f_{k-1}^T p_{k-1}}{\nabla f_k^T p_k}.$$

**Note:** For the first iteration ($k = 0$) set $\alpha_0^{(0)} = 1$ for Newton methods; for others, use $\alpha_0^{(0)} = \min\left(1, \frac{c}{\|\nabla f(x_0)\|}\right)$ or a task-dependent value or a task-dependent value.



## Comments

Denote by $\alpha_k^{(0)}$ the initial approximation of the step length at the k-th iteration of the linear search. Selecting an effective initial step length is crucial for optimization efficiency. For Newton and quasi-Newton methods, $\alpha_k^{(0)}$ is set to 1 on every iteration to prioritize unit steps, leveraging their fast convergence when conditions like sufficient decrease are satisfied. For methods with poorly scaled directions, such as steepest descent a popular strategy is to assume that the first-order change in the function at iterate $x_k$ will be the same as that obtained at the previous step. This causes $\alpha_k^{(0)}$ to be $\alpha_{k-1}$ times the ratio of the dot product of the gradient at $x_{k-1}$ with the previous direction to the dot product at $x_k$ with the current direction.

For the first iteration ($k = 0$), where prior data is unavailable, Newton and quasi-Newton methods use $\alpha_0^{(0)} = 1$ to exploit their scaling properties. For poorly scaled methods, $\alpha_0^{(0)}$ is set to $\min\left(1, \frac{c}{\|\nabla f(x_0)\|}\right)$, where c is a positive constant scale the step to fit the task, or a task-dependent value like a small constant, providing a robust starting point.

## Algorithm 4: Line Search

- Finds $\alpha_k$ satisfying strong Wolfe conditions for direction $p_k$.
- Parameters: $0 < c_1 < c_2 < 1$, $\alpha_{max} > 0$, e.g., $c_1 = 10^{-4}$, $c_2 = 0.9$.

**Algorithm 4** (Line Search Algorithm):

Set: $\alpha_0 \leftarrow 0$, choose $\alpha_{max} > 0$ and $\alpha_1 \in (0, \alpha_{max})$ , $i \leftarrow 1$
repeat
  if $\phi(\alpha_i) > \phi(0) + c_1\alpha_i\phi'(0)$ **or** $(\phi(\alpha_i) \geq \phi(\alpha_{i-1})$ and $i > 1)$ then
    $\alpha_k \leftarrow \text{zoom}(\alpha_{i-1}, \alpha_i)$
    **return** $\alpha_k$
  end if
  if $|\phi'(\alpha_i)| \leq -c_2\phi'(0)$ then $\alpha_k \leftarrow \alpha_i$
    **return** $\alpha_k$
  end if
  if $\phi'(\alpha_i) \geq 0$ then $\alpha_k \leftarrow \text{zoom}(\alpha_i, \alpha_{i-1})$
    **return** $\alpha_k$
  end if
  Choose $\alpha_{i+1} \in (\alpha_i, \alpha_{max})$
  $i \leftarrow i + 1$
until false





### Comments

Algorithm 4 searches for a step length $\alpha_k$ satisfying strong Wolfe conditions. It starts with $\alpha_0 = 0$ and a trial $\alpha_1$, iteratively increasing $\alpha_i$ while checking sufficient decrease and curvature conditions. If conditions fail, it calls a function called zoom to refine the step within an interval. Parameters $c_1$ and $c_2$ (e.g., $10^{-4}$, 0.9) ensure robust step selection.

## Algorithm 5: Zoom

- Refines step length $\alpha_k \in (\alpha_{lo}, \alpha_{hi})$ satisfying strong Wolfe conditions.
- Parameters: $0 < c_1 < c_2 < 1$, e.g., $c_1 = 10^{-4}$, $c_2 = 0.9$.

**Algorithm 5** (Zoom):

```
loop
    Interpolate to find α_j ∈ (α_lo, α_hi)
    if φ(α_j) > φ(0) + c_1 α_j φ'(0) or φ(α_j) ≥ φ(α_lo) then
        α_hi ← α_j
    else
        if |φ'(α_j)| ≤ −c_2 φ'(0) then α_k ← α_j
            return α_k
        end if
        if φ'(α_j)(α_hi − α_lo) ≥ 0 then α_hi ← α_lo
        end if
        α_lo ← α_j
    end if
end loop
```





## Comments

Algorithm 5 refines a step length $\alpha_k$ within an interval $(\alpha_{lo}, \alpha_{hi})$ containing steps satisfying strong Wolfe conditions. It interpolates to find a trial step $\alpha_j$, evaluates $\phi$ and $\phi'$, and updates the interval endpoints to maintain the smallest function value and derivative conditions. It stops when $\alpha_j$ satisfies both Wolfe conditions.

As we mentioned earlier, the interpolation step for selecting $\alpha_j$ must be safeguarded to prevent the new step from being too close to the interval endpoints. Practical line search algorithms leverage properties of interpolating polynomials to make informed predictions about the next step length. Near the solution, function values $f(x_k)$ and $f(x_{k-1})$ may become indistinguishable due to finite-precision arithmetic, so the line search should include a stopping criterion if no lower function value is found after a set number of trials (typically ten) or if the change in x approaches machine precision or a user-defined threshold.

Strong Wolfe conditions with parameters like $c_1 = 10^{-4}$ and $c_2 = 0.9$ have similar computational costs to regular Wolfe conditions but offer better control over search quality because by decreasing $c_2$, we can bring the assumed value of $\alpha$ closer to a local minimum, which is crucial for steepest descent and nonlinear conjugate gradient methods.