Chapter 3

Newton method. Invariance to linear scaling. Global and local convergence. Matrix decompositions for solving linear systems. Hessian correction schemes.

Abstract:
We introduce Newton's method as a second-order optimization scheme based on local quadratic models. We explain why Newton steps require solving linear systems and review standard matrix factorizations (Cholesky, LDL, LU, QR, and spectral decompositions) that are used in practice. We then study two key theoretical properties: invariance of Newton iterates under linear changes of variables, and global vs. local convergence behavior. Finally, we discuss practical Hessian correction schemes that make Newton-like methods robust on nonconvex objectives, and illustrate the ideas on the Rosenbrock function and a one-dimensional example where Newton loses quadratic convergence.

3.1 Introduction

In Chapter 2 we studied first-order methods such as gradient descent and line-search procedures for choosing step sizes. In this chapter we take the next step: we include *second-order* information—the Hessian $\nabla^2 f(\mathbf{x})$—to build a local quadratic approximation of the objective and exploit curvature.

Newton's method is often dramatically faster near a solution (typically *quadratic* convergence under suitable assumptions), and it has a striking invariance to linear scaling of variables. However, it also comes with two important caveats:

- each iteration requires solving a linear system involving the Hessian, which is expensive for large n;
- for nonconvex objectives, the Hessian can be indefinite and the "raw" Newton direction may fail to be a descent direction.

We will address both issues. We start with linear-algebra tools for solving systems efficiently and stably, then derive Newton's method and its properties, and finally discuss Hessian correction schemes for nonconvex problems.

3.2 Solving linear systems without inverting matrices

The Newton step at $\mathbf{x}_k$ is defined via a linear system (see section 3.3)

$$\nabla^2 f(\mathbf{x}_k)\,\mathbf{d}_k = -\nabla f(\mathbf{x}_k).$$

Even if the matrix is invertible, one should *not* form $(\nabla^2 f(\mathbf{x}_k))^{-1}$ explicitly[1]. Computing inverses is typically more expensive and can be numerically unstable (especially for ill-conditioned matrices). In practice, we factorize the matrix once and then solve triangular systems.

3.2.1 A quick warm-up: square and rectangular systems

For a square system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \qquad \mathbf{A} \in \mathbb{R}^{n\times n},\ \det(\mathbf{A}) \neq 0,$$

the formal solution is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, but we will instead compute $\mathbf{x}$ via factorizations of $\mathbf{A}$.

For a rectangular matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m > n$, the system is typically overdetermined and may have no exact solution. A standard surrogate is the least-squares problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2,$$

whose normal equations give

$$\mathbf{x}^\star = (\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}^\top \mathbf{b},$$

assuming $\mathbf{A}^\top\mathbf{A}$ is invertible. However, solving least squares via QR is usually more stable than forming $\mathbf{A}^\top\mathbf{A}$ explicitly (we return to this in Section 3.2.6).

3.2.2 Permutation matrices and pivoting

Many factorizations involve row or column permutations. A *permutation matrix* $\mathbf{P}$ is obtained from the identity by permuting its rows. It is orthogonal:

$$\mathbf{P}^\top = \mathbf{P}^{-1}, \qquad \mathbf{P}^\top\mathbf{P} = \mathbf{I}.$$

Multiplying a matrix on the left permutes its rows, and multiplying on the right permutes

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Figure 3.1: Example of a permutation matrix $\mathbf{P}$ (exactly one 1 in each row and each column).

its columns:

$$\mathbf{PA} \text{ permutes the rows of } \mathbf{A}, \qquad \mathbf{AP} \text{ permutes the columns of } \mathbf{A}.$$

[1]Of course, if the matrix is low-dimensional, then inverting it may be preferable.

3.2.3 Cholesky factorization

If $\mathbf{A} = \mathbf{A}^\top \succ 0$, then there exists a unique lower triangular matrix $\mathbf{L}$ with positive diagonal such that

$$\mathbf{A} = \mathbf{L}\mathbf{L}^\top.$$

This is the *Cholesky factorization.* It can also be used as a practical test of positive definiteness: if the factorization algorithm runs to completion with all L_{ii} well-defined[2] and positive, then $\mathbf{A}$ is positive definite. To solve $\mathbf{Ax} = \mathbf{b}$, we perform two triangular solves:

$$\mathbf{Ly} = \mathbf{b}, \qquad \mathbf{L}^\top\mathbf{x} = \mathbf{y}.$$

The factorization costs $\approx \frac{1}{3}n^3$ flops, while each triangular solve costs $O(n^2)$.

3.2.4 LDL$^\top$ factorization

For a general symmetric matrix $\mathbf{A} = \mathbf{A}^\top$ (not necessarily positive definite), one often uses a modified LDL factorization:

$$\mathbf{P}^\top\mathbf{AP} = \mathbf{LDL}^\top,$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is unit lower triangular ($L_{ii} = 1$), and $\mathbf{D}$ is block diagonal with 1×1 and 2×2 blocks.

To solve $\mathbf{Ax} = \mathbf{b}$ using the factorization, write

$$\mathbf{PLDL}^\top\mathbf{P}^\top\mathbf{x} = \mathbf{b}.$$

One convenient sequence of solves is

$$\mathbf{z} = \mathbf{L}^{-1}\mathbf{P}^\top\mathbf{b}, \qquad \mathbf{y} = \mathbf{D}^{-1}\mathbf{z}, \qquad \mathbf{x} = \mathbf{PL}^{-\top}\mathbf{y},$$

where the triangular solves cost $O(n^2)$ and applying $\mathbf{D}^{-1}$ costs $O(n)$. Forming the factorization $\mathbf{P}^\top\mathbf{AP} = \mathbf{LDL}^\top$ for a symmetric matrix costs $\approx \frac{n^3}{3}$ flops (the same order as Cholesky), with a slightly larger constant due to pivoting and permutations.

Inertia and definiteness.

The block structure of $\mathbf{D}$ encodes the inertia of $\mathbf{A}$, i.e., the numbers of positive, negative, and zero eigenvalues. Concretely, each 1×1 block contributes according to the sign of its scalar entry, and each 2×2 block contributes according to the signs of its two eigenvalues; summing these contributions over all blocks yields the inertia of $\mathbf{A}$.

3.2.5 LU factorization

For a general (not necessarily symmetric) matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$, we can use an LU factorization

$$\mathbf{PA} = \mathbf{LU},$$

where $\mathbf{P} \in \mathbb{R}^{m\times m}$ permutes the rows of $\mathbf{A}$, $\mathbf{L} \in \mathbb{R}^{m\times n}$ is *unit lower trapezoidal* ($L_{ii} = 1$ for $i = 1, \ldots, n$), and $\mathbf{U} \in \mathbb{R}^{n\times n}$ is upper triangular. In the square case $m = n$, the cost of this factorization is $\approx \frac{2}{3}n^3$ flops.

[2] i.e., not NaN

If $m = n$, solving $\mathbf{Ax} = \mathbf{b}$ reduces to

$$\mathbf{Ly} = \mathbf{Pb}, \qquad \mathbf{Ux} = \mathbf{y},$$

again using triangular solves. If $m > n$, the system $\mathbf{Ax} = \mathbf{b}$ is generally overdetermined, and least-squares problems are better handled via QR (Section 3.2.6).

3.2.6 QR factorization and least squares

For $\mathbf{A} \in \mathbb{R}^{m\times n}$, QR factorization writes as

$$\mathbf{AP} = \mathbf{QR},$$

where $\mathbf{Q} \in \mathbb{R}^{m\times m}$ is orthogonal ($\mathbf{Q}^{-1} = \mathbf{Q}^\top$) and $\mathbf{R} \in \mathbb{R}^{m\times n}$ is upper trapezoidal. For a square $n \times n$ matrix, QR costs about $\frac{4}{3}n^3$ flops, which is why it is usually not the first choice for solving *square* systems (LU is cheaper).

Least squares via QR.

For $m > n$, QR is particularly useful for

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2.$$

Assume, for simplicity, that we do not use permutations (this is always possible, but it decreases the stability of the algorithm).[3]

$$\mathbf{A} = \mathbf{Q}\begin{pmatrix}\mathbf{R}_1\\ \mathbf{0}\end{pmatrix}, \qquad \mathbf{Q} = [\mathbf{Q}_1\ \mathbf{Q}_2],$$

where $\mathbf{Q}$ is orthogonal, $\mathbf{Q}_1 \in \mathbb{R}^{m\times n}$ has orthonormal columns spanning col($\mathbf{A}$), and $\mathbf{R}_1 \in \mathbb{R}^{n\times n}$ is upper triangular. Now use that orthogonal maps preserve Euclidean norms: for any $\mathbf{x}$,

$$\|\mathbf{Ax} - \mathbf{b}\| = \left\|\mathbf{Q}^\top(\mathbf{Ax} - \mathbf{b})\right\| = \left\|\begin{pmatrix}\mathbf{R}_1\mathbf{x} - \mathbf{Q}_1^\top\mathbf{b}\\ -\mathbf{Q}_2^\top\mathbf{b}\end{pmatrix}\right\|.$$

Therefore,

$$\|\mathbf{Ax} - \mathbf{b}\|^2 = \|\mathbf{R}_1\mathbf{x} - \mathbf{Q}_1^\top\mathbf{b}\|^2 + \|\mathbf{Q}_2^\top\mathbf{b}\|^2,$$

and the second term does not depend on $\mathbf{x}$. Hence the least-squares minimizer is obtained by solving the triangular system

$$\mathbf{R}_1\mathbf{x} = \mathbf{Q}_1^\top\mathbf{b},$$

i.e.,

$$\mathbf{x}^\star = \mathbf{R}_1^{-1}\mathbf{Q}_1^\top\mathbf{b}.$$

This approach avoids forming $\mathbf{A}^\top\mathbf{A}$ and is typically more stable.

[3]If we use column pivoting and factorize $\mathbf{AP} = \mathbf{QR}$, we can absorb $\mathbf{P}$ by a change of variables: write $\mathbf{x} = \mathbf{Py}$ and solve for $\mathbf{y}$.

3.2.7 Spectral decomposition and SVD

If $\mathbf{A} = \mathbf{A}^\top$, then it admits an eigenvalue decomposition (spectral decomposition)

$$\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top,$$

where $\mathbf{Q}$ is orthogonal and $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ contains the eigenvalues. This decomposition is extremely expensive.

For rectangular matrices, the analogue is the singular value decomposition (SVD). In many optimization pipelines, eigen/SVD computations are therefore "methods of last resort" for large dimensions.

3.2.8 Practical takeaway: small vs. large scale

The asymptotic complexities above hide an important practical point:

- for small matrices (say, sizes on the order of tens), most decompositions are fast enough and the choice is dominated by stability and convenience;
- for sizes in the hundreds or thousands, the difference between $\frac{1}{3}n^3$ and $\frac{4}{3}n^3$ becomes dramatic;
- for sizes in the tens of thousands, dense matrices may not even fit in memory, so one needs large-scale methods that avoid forming and factorizing the Hessian explicitly.

3.3 Newton's method from a quadratic model

Consider the unconstrained problem

$$f(\mathbf{x}) \to \min_{\mathbf{x}\in\mathbb{R}^n}, \qquad f \in C^2.$$

At iterate $\mathbf{x}_k$, we approximate f near $\mathbf{x}_k$ by a quadratic model

$$m_k(\mathbf{d}) := f(\mathbf{x}_k + \mathbf{d}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{d} + \frac{1}{2}\mathbf{d}^\top \mathbf{B}_k \mathbf{d}, \tag{3.1}$$

where $\mathbf{B}_k = \mathbf{B}_k^\top \succ 0$ is a symmetric positive definite matrix that plays the role of curvature. Minimizing (3.1) over $\mathbf{d}$ gives

$$\nabla_{\mathbf{d}} m_k(\mathbf{d}) = \nabla f(\mathbf{x}_k) + \mathbf{B}_k \mathbf{d} = 0 \quad \Longrightarrow \quad \mathbf{d}_k = -\mathbf{B}_k^{-1}\nabla f(\mathbf{x}_k).$$

Two important special choices are:

- $\mathbf{B}_k = \mathbf{I}$ gives the gradient direction $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$;
- $\mathbf{B}_k = \nabla^2 f(\mathbf{x}_k)$ gives the *Newton direction.*

Thus Newton's method can be viewed as "gradient descent with a matrix-valued step size" that adapts to curvature.

3.3.1 Newton direction and descent

With $\mathbf{B}_k = \nabla^2 f(\mathbf{x}_k)$, the Newton direction satisfies

$$\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k = -\nabla f(\mathbf{x}_k).$$

If $\nabla^2 f(\mathbf{x}_k) \succ 0$, then the direction is automatically a descent direction:

$$\nabla f(\mathbf{x}_k)^\top \mathbf{d}_k = -\nabla f(\mathbf{x}_k)^\top (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) < 0,$$

whenever $\nabla f(\mathbf{x}_k) \neq 0$. This is a reason why we require $\mathbf{B}_k \succ 0$ in eq. (3.1).

Away from the optimum in nonconvex problems, the Hessian may not be positive definite, and then the Newton direction may fail to be a descent direction. This is why practical Newton-type methods often use either line search (choose α_k) or a corrected Hessian (choose $\mathbf{B}_k \succ 0$); see Sections 3.4.1 and 3.5.

3.3.2 Newton on quadratic objectives

If f is a strictly convex quadratic

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}, \qquad \mathbf{A} = \mathbf{A}^\top \succ 0,$$

then $\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$ and $\nabla^2 f(\mathbf{x}) = \mathbf{A}$. Newton's method with a full step ($\alpha_k = 1$) solves $\mathbf{A}\mathbf{d}_k = -(\mathbf{A}\mathbf{x}_k - \mathbf{b})$, so

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k = \mathbf{A}^{-1}\mathbf{b},$$

i.e. it reaches the minimizer in one iteration, independently of the condition number of $\mathbf{A}$. This is a key contrast with gradient descent, whose speed on quadratics strongly depends on the curvature (see the diagonal example in Chapter 2).

3.3.3 Invariance to linear scaling

One of the most pleasant properties of Newton's method is its invariance to invertible linear changes of variables. Let $\mathbf{x} = \mathbf{A}\hat{\mathbf{x}}$ with an invertible matrix $\mathbf{A}$ and define

$$\hat{f}(\hat{\mathbf{x}}) := f(\mathbf{A}\hat{\mathbf{x}}).$$

Then

$$\nabla \hat{f}(\hat{\mathbf{x}}) = \mathbf{A}^\top \nabla f(\mathbf{A}\hat{\mathbf{x}}), \qquad \nabla^2 \hat{f}(\hat{\mathbf{x}}) = \mathbf{A}^\top \nabla^2 f(\mathbf{A}\hat{\mathbf{x}})\mathbf{A}.$$

Consider one Newton step for $\hat{f}$:

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k - \alpha_k (\nabla^2 \hat{f}(\hat{\mathbf{x}}_k))^{-1} \nabla \hat{f}(\hat{\mathbf{x}}_k).$$

Plugging the transformation rules above yields

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k - \alpha_k \mathbf{A}^{-1} (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k), \qquad \mathbf{x}_k := \mathbf{A}\hat{\mathbf{x}}_k.$$

Multiplying by $\mathbf{A}$, we recover the Newton update in the original coordinates:

$$\mathbf{x}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_{k+1} = \mathbf{x}_k - \alpha_k (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k).$$

Therefore, the trajectories in $\mathbf{x}$-space are identical: the same iterates (just expressed in different coordinates) and the same number of iterations, no matter how we linearly rescale the variables.

Newton's method is "scale-free" under linear reparameterizations: it automatically adapts to elliptical level sets. This explains why Newton does not suffer from the same zig-zag behavior as gradient descent on ill-conditioned quadratics, but it also makes each step computationally heavier.

3.4 Global and local convergence

3.4.1 Global convergence via line search

The "pure" Newton iteration with $\alpha_k = 1$ is a *local* method: it can diverge if started far from a solution. A standard globalization strategy is to use a line search:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k, \qquad \nabla^2 f(\mathbf{x}_k)\mathbf{d}_k = -\nabla f(\mathbf{x}_k), \tag{3.2}$$

where α_k is chosen by backtracking or Armijo–Wolfe conditions as in Chapter 2 (Algorithms 2.1 to 2.3).

Recall from Theorem 2.9 that if f is L-smooth and α_k satisfies Armijo–Wolfe conditions, then the method achieves a decrease bound of the form

$$f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) - \frac{c_1(1-c_2)}{L} \cos^2\theta_k \, \|\nabla f(\mathbf{x}_k)\|^2, \qquad \cos\theta_k := \frac{\nabla f(\mathbf{x}_k)^\top \mathbf{d}_k}{\|\nabla f(\mathbf{x}_k)\| \, \|\mathbf{d}_k\|}.$$

Therefore, to inherit the same global convergence guarantees as generic descent methods, we want $\cos\theta_k$ to be bounded away from 0 (i.e. the direction should not become nearly orthogonal to the negative gradient).

Claim 3.1 (Angle bound for the Newton direction)**.** Assume $\nabla^2 f(\mathbf{x}_k) \succ 0$ and let $\kappa_k := \frac{\lambda_{\max}(\nabla^2 f(\mathbf{x}_k))}{\lambda_{\min}(\nabla^2 f(\mathbf{x}_k))}$. Then the Newton direction satisfies

$$\cos\theta_k \le -\frac{1}{\kappa_k} < 0.$$

Proof: Let $\mathbf{g}_k := \nabla f(\mathbf{x}_k)$ and $\mathbf{H}_k := \nabla^2 f(\mathbf{x}_k)$. Since $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$, we have

$$\cos\theta_k = -\frac{\mathbf{g}_k^\top \mathbf{H}_k^{-1}\mathbf{g}_k}{\|\mathbf{g}_k\| \, \|\mathbf{H}_k^{-1}\mathbf{g}_k\|}.$$

Using $\mathbf{g}_k^\top \mathbf{H}_k^{-1}\mathbf{g}_k \ge \lambda_{\min}(\mathbf{H}_k^{-1})\|\mathbf{g}_k\|^2$ and $\|\mathbf{H}_k^{-1}\mathbf{g}_k\| \le \lambda_{\max}(\mathbf{H}_k^{-1})\|\mathbf{g}_k\|$, we obtain

$$\cos\theta_k \le -\frac{\lambda_{\min}(\mathbf{H}_k^{-1})}{\lambda_{\max}(\mathbf{H}_k^{-1})} = -\frac{1/\lambda_{\max}(\mathbf{H}_k)}{1/\lambda_{\min}(\mathbf{H}_k)} = -\frac{1}{\kappa_k}.$$

$\square$

In particular, if κ_k stays uniformly bounded along the iterates, then $\cos\theta_k \le -\delta$ for some $\delta > 0$, and the global convergence mechanism from Theorem 2.9 applies to Newton as well (e.g. $\|\nabla f(\mathbf{x}_k)\| \to 0$ under the same assumptions).

3.4.2 Local quadratic convergence of the full Newton step

We now explain why Newton's method is so fast near a nondegenerate minimizer.

Theorem 3.1 (Local quadratic convergence of Newton)**.** Let $f \in C_M^{2,2}$ in a neighborhood of $\mathbf{x}^\star$ and $\nabla^2 f(\mathbf{x}^\star) \succeq \mu\mathbf{I}$ for some $\mu > 0$. Then there $\exists r > 0$ such that if $\|\mathbf{x}_0 - \mathbf{x}^\star\| \le r$, the full-step Newton iterates[a]

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\nabla^2 f(\mathbf{x}_k))^{-1}\nabla f(\mathbf{x}_k)$$

are well-defined and satisfy

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \leq \frac{M}{\mu} \|\mathbf{x}_k - \mathbf{x}^\star\|^2.$$

Therefore, the method has local quadratic convergence.

[a]The choice $\alpha_k = 1$ is crucial for the proof, and it also explains why line-search implementations typically try $\alpha = 1$ first: it preserves fast local convergence of Newton's method.

Proof: Denote $\mathbf{r}_k := \mathbf{x}_k - \mathbf{x}^\star$ and $\mathbf{H}_k := \nabla^2 f(\mathbf{x}_k)$. Since $\nabla f(\mathbf{x}^\star) = 0$,

$$\mathbf{r}_{k+1} = \mathbf{x}_k - \mathbf{H}_k^{-1} \nabla f(\mathbf{x}_k) - \mathbf{x}^\star = \mathbf{H}_k^{-1}\Big(\mathbf{H}_k \mathbf{r}_k - \big(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star)\big)\Big).$$

Using the integral form of the mean value theorem,

$$\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star) = \int_0^1 \nabla^2 f(\mathbf{x}^\star + t\mathbf{r}_k)\, \mathbf{r}_k \, \mathrm{d}t,$$

so

$$\mathbf{H}_k \mathbf{r}_k - \big(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star)\big) = \int_0^1 \big(\mathbf{H}_k - \nabla^2 f(\mathbf{x}^\star + t\mathbf{r}_k)\big)\, \mathbf{r}_k \, \mathrm{d}t.$$

Taking norms and applying the Hessian Lipschitz bound,

$$\begin{aligned}
\|\mathbf{r}_{k+1}\| &\leq \|\mathbf{H}_k^{-1}\| \int_0^1 \|\mathbf{H}_k - \nabla^2 f(\mathbf{x}^\star + t\mathbf{r}_k)\| \, \|\mathbf{r}_k\| \, \mathrm{d}t \\
&\leq \|\mathbf{H}_k^{-1}\| \int_0^1 M \|\mathbf{x}_k - (\mathbf{x}^\star + t\mathbf{r}_k)\| \, \|\mathbf{r}_k\| \, \mathrm{d}t \\
&= \|\mathbf{H}_k^{-1}\| \, M \|\mathbf{r}_k\|^2 \int_0^1 (1-t)\, \mathrm{d}t = \frac{M}{2} \|\mathbf{H}_k^{-1}\| \, \|\mathbf{r}_k\|^2.
\end{aligned}$$

By continuity of $\nabla^2 f$, there exists a radius $r > 0$ such that $\|\mathbf{x}_k - \mathbf{x}^\star\| \leq r$ implies $\nabla^2 f(\mathbf{x}_k) \succeq \frac{\mu}{2}\mathbf{I}$ and hence $\|\mathbf{H}_k^{-1}\| \leq \frac{2}{\mu}$. Therefore, for such k we obtain

$$\|\mathbf{r}_{k+1}\| \leq \frac{M}{\mu} \|\mathbf{r}_k\|^2.$$

□

Why only local? The proof heavily relies on the fact that if we have $r_{k+1} \leq C r_k^2$, then the sequence converges to 0 provided $C r_0 < 1$. This means that r_0 must be small; in particular, $\mathbf{x}_0$ must lie in a neighborhood of $\mathbf{x}^\star$.

3.4.3 Computational cost

Even when Newton is theoretically attractive, it can be impractical for large problems. Let n be the dimension of $\mathbf{x} \in \mathbb{R}^n$, and let q denote the cost of one *first-order oracle call* (evaluating $f(\mathbf{x})$ and $\nabla f(\mathbf{x})$ at a given point).

Method	Memory	Per-iteration compute
Gradient descent (GD)	$O(n)$	$O(q+n)$
Newton (dense)	$O(n^2)$	$O(qn+n^3)$

Table 3.1: Rough per-iteration memory and computational cost. Here n is the dimension and q is the cost of one first-order oracle call. The Newton row assumes a dense Hessian and a direct solve via a matrix factorization.

Why these numbers? (see Table 3.1)

For GD, the dominant cost is the gradient evaluation (one oracle call), while vector operations such as $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$ are only $O(n)$. The memory footprint is just a few vectors in $\mathbb{R}^n$.

For Newton, storing the Hessian requires $O(n^2)$ memory. To obtain a *dense* Hessian one needs, in the worst case, n pieces of first-order information, which leads to the $O(qn)$ term. Finally, solving $\nabla^2 f(\mathbf{x}_k)\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ by Cholesky/LU/LDL factorization costs $O(n^3)$ flops (see Section 3.2). This cubic term is the main reason why, in large-scale settings, one often prefers quasi-Newton or Hessian-free approaches.

3.5 Newton-type methods for nonconvex objectives

When f is nonconvex, $\nabla^2 f(\mathbf{x}_k)$ need not be positive definite, and the raw Newton direction may point uphill. A standard remedy is to replace the Hessian by a *corrected* positive definite matrix

$$\mathbf{B}_k = \nabla^2 f(\mathbf{x}_k) + \mathbf{E}_k \succ 0,$$

and take the Newton-like direction

$$\mathbf{d}_k = -\mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k).$$

If $\mathbf{B}_k \succ 0$, then $\mathbf{d}_k$ is a descent direction and can be combined with line search exactly as in Section 3.4.1. Near a (strongly) convex minimizer where $\nabla^2 f(\mathbf{x}^\star) \succeq \mu \mathbf{I}$, the correction becomes unnecessary.

Below are three common correction strategies.

3.5.1 Spectral correction (eigenvalue modification)

Assume $\nabla^2 f(\mathbf{x}_k)$ is symmetric and compute its spectral decomposition

$$\nabla^2 f(\mathbf{x}_k) = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top.$$

Fix a small threshold $\delta > 0$ and define

$$\tilde{\lambda}_i := \begin{cases} \lambda_i, & \lambda_i \geq \delta, \\ \delta, & \lambda_i < \delta, \end{cases} \qquad \tilde{\mathbf{\Lambda}} := \operatorname{diag}(\tilde{\lambda}_1, \dots, \tilde{\lambda}_n), \qquad \mathbf{B}_k := \mathbf{Q}\tilde{\mathbf{\Lambda}}\mathbf{Q}^\top \succ 0.$$

The resulting direction is

$$\mathbf{d}_k = -\mathbf{Q}\tilde{\mathbf{\Lambda}}^{-1}\mathbf{Q}^\top \nabla f(\mathbf{x}_k).$$

This is conceptually clean (we directly “fix” negative or tiny curvature for global convergence Claim 3.1), but it is expensive: a full eigendecomposition is typically among the most costly dense factorizations.

3.5.2 Diagonal shift: $\nabla^2 f + \tau \mathbf{I}$

Instead of an eigendecomposition, one can enforce positive definiteness by adding a multiple of the identity:

$$\mathbf{B}_k = \nabla^2 f(\mathbf{x}_k) + \tau_k \mathbf{I}.$$

For sufficiently large τ_k, the matrix becomes positive definite and one can compute a Cholesky factorization to solve for the step. A practical scheme is to increase τ_k until Cholesky succeeds.

Algorithm 3.1 DIAGONAL SHIFT for a nonconvex Newton step

```
Require: $\mathbf{x}_k$, $\nabla f(\mathbf{x}_k)$, $\nabla^2 f(\mathbf{x}_k)$; initial shift $\tau > 0$; increase factor $\gamma > 1$; decrease factor $\beta \in (0,1)$
  $\mathbf{H} \leftarrow \nabla^2 f(\mathbf{x}_k)$, $\mathbf{g} \leftarrow \nabla f(\mathbf{x}_k)$
  while Cholesky factorization of $\mathbf{H} + \tau\mathbf{I}$ fails do
    $\tau \leftarrow \gamma\tau$
  end while
  Compute $\mathbf{L}$ such that $\mathbf{H} + \tau\mathbf{I} = \mathbf{L}\mathbf{L}^\top$
  Solve $(\mathbf{H} + \tau\mathbf{I})\mathbf{d}_k = -\mathbf{g}$ via triangular solves with $\mathbf{L}$
  $\tau_{k+1} \leftarrow \beta\tau$
  return $\mathbf{d}_k$, $\tau_{k+1}$
```

If $\tau_k \gg 1$, then $\mathbf{B}_k \approx \tau_k \mathbf{I}$ and $\mathbf{d}_k \approx -\frac{1}{\tau_k}\nabla f(\mathbf{x}_k)$, i.e. the method behaves like a gradient step with step size $1/\tau_k$. If $\tau_k \ll 1$, then $\mathbf{B}_k \approx \nabla^2 f(\mathbf{x}_k)$ and we recover Newton. In this sense, the diagonal shift method interpolates between gradient descent and Newton.

3.5.3 LDL-based correction

Another approach is to compute a pivoted LDL factorization

$$\mathbf{P}^\top \nabla^2 f(\mathbf{x}_k)\mathbf{P} = \mathbf{L}\mathbf{D}\mathbf{L}^\top,$$

and then "repair" $\mathbf{D}$ to a positive definite $\tilde{\mathbf{D}}$. Since $\mathbf{D}$ is block diagonal with only 1×1 and 2×2 blocks, one can modify each block cheaply (e.g. by eigenvalue thresholding on the 2×2 blocks). The corrected matrix is then

$$\mathbf{B}_k = \mathbf{P}\mathbf{L}\tilde{\mathbf{D}}\mathbf{L}^\top\mathbf{P}^\top \succ 0.$$

The potential advantage is that we perform one "expensive" factorization per iteration (LDL)[4], while the diagonal-shift method may attempt multiple factorizations if many τ updates are needed.

3.6 Examples

3.6.1 Rosenbrock function: why curvature matters

Example 3.1 (see fig. 3.2)**.** Consider the classic Rosenbrock function

$$f(x,y) = (1-x)^2 + 100\,(y - x^2)^2.$$

[4]This is typically less expensive than a full spectral decomposition.

Its level sets form a long, narrow curved valley (a "banana" shape). Gradient descent tends to zig-zag across the valley and may require very small step sizes to stay stable, which can lead to hundreds or thousands of iterations. Newton's method, on the other hand, uses curvature information to align steps with the valley and can converge in tens of iterations when combined with a reasonable line search.

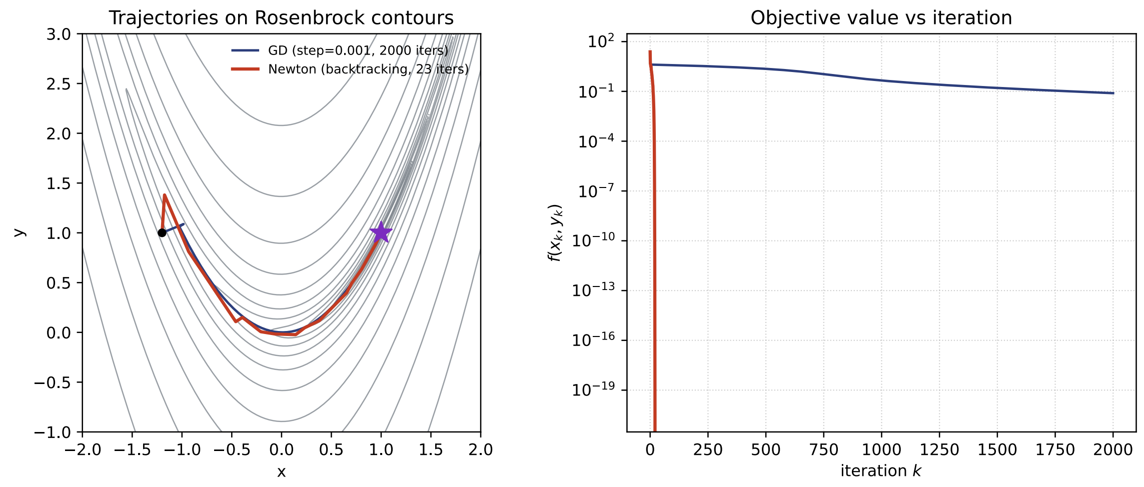

Figure 3.2: Rosenbrock function: a typical comparison between gradient descent (fixed small step) and damped Newton (Armijo backtracking), starting from $(-1.2, 1)$. Newton follows the curved valley and reaches the minimizer in a few dozen steps, while gradient descent progresses slowly due to ill-conditioning.

3.6.2 A convex function where Newton is only linear

The local quadratic convergence theorem (Theorem 3.1) assumes that the Hessian at the solution is positive definite. If the Hessian degenerates at the minimizer, Newton can lose quadratic convergence even for a convex function.

Example 3.2 ($f(x) = \frac{1}{3}|x|^3$)**.** Let $f(x) = \frac{1}{3}|x|^3$ on $\mathbb{R}$. Then (for $x \neq 0$)

$$f'(x) = |x|x, \qquad f''(x) = 2|x|.$$

The minimizer is $x^\star = 0$, but $f''(0) = 0$, so f is not strongly convex near the solution. Newton's method with step size α gives, for $x_k \neq 0$,

$$x_{k+1} = x_k - \alpha \frac{f'(x_k)}{f''(x_k)} = x_k - \alpha \frac{|x_k| x_k}{2|x_k|} = \left(1 - \frac{\alpha}{2}\right) x_k.$$

Thus the convergence is linear rather than quadratic.