CS/MATH 3414 — Numerical Methods

Lecture 11
Optimization, Part 2

Virginia Tech Department of Computer Science, Fall 2026
Instructor: Sumeet Khatri
Date: October 29, 2026

1 Optimization recap **2**
1.1 Optimality tests . . . 2
1.2 Convex functions . . . 3
1.3 Gradient descent . . . 5
2 Numerical differentiation **6**
2.1 Forward, backward, and central differences . . . 7
2.2 Second derivatives . . . 10
2.3 How small should the step be? . . . 12
2.4 From derivatives to gradients and Hessians . . . 13
3 The BFGS algorithm: using curvature information **15**
4 The Nelder–Mead algorithm: optimization without derivatives **22**
5 Simulated annealing: allowing uphill moves **31**

Learning Objectives

- Understand the concept of numerical differentiation for first and second derivatives, and explain the finite-difference method.
- Explain the BFGS and Nelder–Mead algorithms, including their safeguards and limitations; understand how to use these algorithms by calling existing implementations in NumPy and SciPy.

Primary References

[1] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer Series in Operations Research and Financial Engineering, Chapters 3 and 11: line searches and nonlinear equations (Springer, New York, 2006).

[2] S. Boyd and L. Vandenberghe, *Convex Optimization*, Sections 3.1 and 9.1–9.4: convexity, unconstrained minimization, and descent methods (Cambridge University Press, Cambridge, 2004), https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf (visited on 10/23/2026).

[3] L. Vandenberghe, *Gradient Method*, ECE236C: Optimization Methods for Large-Scale Systems, Lecture 1, Spring 2022, University of California, Los Angeles, (2022) https://www.seas.ucla.edu/~vandenbe/236C/lectures/gradient.pdf (visited on 10/26/2026).

[4] T. A. Driscoll and R. J. Braun, *Fundamentals of Numerical Computation*, https://fncbook.com.

[5] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, 7th ed., Section 13.2: Multivariate Case, pp. 573–586 (Brooks/Cole, Cengage Learning, Boston, 2013).

[6] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms: 2. The new algorithm", Journal of the Institute of Mathematics and its Applications **6**, 222 (1970).

Primary References (continued)

[7] R. Fletcher, "A new approach to variable metric algorithms", The Computer Journal **13**, 317 (1970).

[8] D. Goldfarb, "A family of variable-metric methods derived by variational means", Mathematics of Computation **24**, 23 (1970).

[9] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization", Mathematics of Computation **24**, 647 (1970).

[10] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization", The Computer Journal **7**, 308 (1965).

[11] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions", SIAM Journal on Optimization **9**, 112 (1998).

[12] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by Simulated Annealing", Science **220**, 671 (1983).

[13] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, et al., "Equation of State Calculations by Fast Computing Machines", The Journal of Chemical Physics **21**, 1087 (1953).

1 Optimization recap

Given a function $f : \mathbb{R}^n \to \mathbb{R}$, with $n \in \{1, 2, \dots\}$, an (unconstrained) optimization problem involves the finding the minimum (or maximum) of the function over all inputs $\boldsymbol{x} \in \mathbb{R}^n$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}). \tag{1.1}$$

The function $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, and $\boldsymbol{x}$ contains the decision variables. The case $n = 1$ concerns a function of a single variable. "Unconstrained" means that we impose no additional equations or inequalities restricting the variables. (We will consider constrained optimization problems in the next lecture.) We will principally consider differentiable objectives; whenever second derivatives are used, the required smoothness will be stated.

For $f : \mathbb{R}^n \to \mathbb{R}$, the *gradient* vector and *Hessian* matrix collect its first and second derivatives:

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \partial f/\partial x_1 \\ \vdots \\ \partial f/\partial x_n \end{pmatrix}, \qquad H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x}) = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}) \right)_{i,j=1}^{n}. \tag{1.2}$$

For twice continuously differentiable f, the Hessian is symmetric. A point $\boldsymbol{x}_* \in \mathbb{R}^n$ satisfying $\nabla f(\boldsymbol{x}_*) = \boldsymbol{0}$ is called a *stationary point*.

1.1 Optimality tests

We are particularly interested in tests that certify whether the function has a local or global maximum or minimum.

Single-variable optimality tests

Let f be twice continuously differentiable near x_*.

1. If x_* is a local minimum, then $f'(x_*) = 0$ and $f''(x_*) \geq 0$.

Key Idea (continued)

2. If $f'(x_*) = 0$ and $f''(x_*) > 0$, then x_* is a strict local minimum.
3. If $f'(x_*) = 0$ and $f''(x_*) < 0$, then x_* is a strict local maximum.

Multi-variable optimality tests

Suppose f is twice continuously differentiable near $\boldsymbol{x}_* \in \mathbb{R}^n$.

1. If $\boldsymbol{x}_*$ is a local minimum, then $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$ and $H(\boldsymbol{x}_*)$ positive semi-definite.
2. If $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$ and $H(\boldsymbol{x}_*)$ is positive definite, then $\boldsymbol{x}_*$ is a strict local minimum.
3. If $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$ and $H(\boldsymbol{x}_*)$ is negative definite, then $\boldsymbol{x}_*$ is a strict local maximum.
4. If $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$ and $H(\boldsymbol{x}_*)$ is indefinite, then $\boldsymbol{x}_*$ is a saddle point: arbitrarily nearby points have both larger and smaller objective values.

Proofs of the necessary and sufficient minimum conditions appear in [1, Theorems 2.2–2.4, pp. 14–16]. The maximum condition follows by applying these results to $-f$. The saddle condition follows from Taylor's expansion along eigenvectors with eigenvalues of opposite signs, as shown below.

Saddle points

For a differentiable objective, a *saddle point* is a stationary point that is neither a local minimum nor a local maximum. More precisely, every neighborhood of $\boldsymbol{x}_*$ contains both a point with a smaller objective value and a point with a larger objective value. For every $r > 0$, there are points $\boldsymbol{x}_-$ and $\boldsymbol{x}_+$ such that

$$\|\boldsymbol{x}_- - \boldsymbol{x}_*\| < r, \qquad \|\boldsymbol{x}_+ - \boldsymbol{x}_*\| < r, \qquad f(\boldsymbol{x}_-) < f(\boldsymbol{x}_*) < f(\boldsymbol{x}_+). \tag{1.3}$$

The condition $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$ means that all first-order directional slopes vanish. It does not mean that the function is locally constant: second-order or higher-order changes can have different signs in different directions.

1.2 Convex functions

There are three separate questions about minimization: does a global minimizer exist, is it unique, and must a local minimizer be global? For arbitrary functions, there is very little we can say in general. On the other hand, for special classes of function, such as *convex functions*, we can make definitive statements.

Definition 1. A function f is *convex* if, for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $0 \le t \le 1$,

$$f((1-t)\boldsymbol{x} + t\boldsymbol{y}) \le (1-t)f(\boldsymbol{x}) + tf(\boldsymbol{y}). \tag{1.4}$$

Geometrically, the graph lies on or below every chord joining two points on the graph. The function is *strictly convex* if this inequality is strict whenever $\boldsymbol{x} \neq \boldsymbol{y}$ and $0 < t < 1$. Thus, strict convexity excludes a straight segment in the graph between distinct input points.

For a differentiable function, convexity is equivalent to the *supporting-plane inequality*

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) \qquad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n. \tag{1.5}$$

This says that every tangent plane is a global lower bound for the function; in one variable, the tangent line lies below the graph. A proof of this equivalence appears in [2, Section 3.1.3, pp. 69–70].

Optimality under convexity and strict convexity

Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex.

1. Every local minimizer of f is a global minimizer.
2. If f is differentiable, then $\boldsymbol{x}_*$ is a global minimizer if and only if $\nabla f(\boldsymbol{x}_*) = \mathbf{0}$.
3. If f is strictly convex, then f has at most one global minimizer: if a minimizer exists, it is unique.

Neither convexity nor strict convexity alone guarantees that a global minimizer exists.

Definition 2 (Strongly convex function)**.** A differentiable function f is *μ-strongly convex*, for some fixed $\mu > 0$, if

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\mu}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 \qquad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n. \tag{1.6}$$

The same positive constant μ must work at every point and for every displacement. Ordinary convexity places the graph above its tangent planes; strong convexity places it above an upward-opening quadratic surface based at each point. The extra quadratic term prevents the function from becoming arbitrarily flat. Strong convexity implies strict convexity, and hence convexity; the converse implications do not hold.

Hessian criteria for convexity

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, with Hessian $H(\boldsymbol{x})$.

1. The function is convex if and only if $H(\boldsymbol{x})$ is positive semidefinite at every $\boldsymbol{x}$.
2. If $H(\boldsymbol{x})$ is positive definite at every $\boldsymbol{x}$, then f is strictly convex. This condition is sufficient but not necessary for strict convexity.
3. For a fixed $\mu > 0$, the function is μ-strongly convex if and only if $H(\boldsymbol{x}) - \mu\mathbb{1}$ is positive semidefinite at every $\boldsymbol{x}$. Equivalently, every eigenvalue of every Hessian is at least μ.

Existence and uniqueness under strong convexity

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and μ-strongly convex for some $\mu > 0$. Then f attains a unique global minimum at a point $\boldsymbol{x}_*$. Moreover,

$$f(\boldsymbol{y}) - f(\boldsymbol{x}_*) \geq \frac{\mu}{2}\|\boldsymbol{y} - \boldsymbol{x}_*\|_2^2 \qquad \text{for every } \boldsymbol{y} \in \mathbb{R}^n. \tag{1.7}$$

1.3 Gradient descent

The optimality conditions describe what a solution must look like, but they do not yet tell us how to find it. In a nonlinear problem, solving $\nabla f(\boldsymbol{x}) = \mathbf{0}$ symbolically can be just as difficult as the original minimization. We therefore construct a sequence of approximations. At each iteration we ask two separate questions: in which direction should we move, and how far should we move? The gradient descent algorithm moves in the direction of steepest descent at every point — which we now know is given by the gradient at that point.

The gradient points in the steepest direction

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. Its gradient at $\boldsymbol{x} \in \mathbb{R}$, i.e., $\nabla f(\boldsymbol{x})$, points in the direction of steepest ascent at $\boldsymbol{x}$; the negative gradient, $-\nabla f(\boldsymbol{x})$, points in the direction of steepest descent.

Algorithm: Basic gradient descent

1. Choose $\boldsymbol{x}_0$, a maximum iteration count, and a tolerance for the gradient norm.
2. At iteration k, evaluate $f(\boldsymbol{x}_k)$ and $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$.
3. If $\|\boldsymbol{g}_k\|_2$ is below the tolerance, report approximate stationarity and stop.
4. Choose a positive step size α_k.
5. Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k$ and repeat.

Choosing the negative gradient settles the direction of motion, but it does not settle how far to move and how fast the algorithm will converge to a (local) minimum. The gradient describes the slope at the current point; after a long move, the slope may have changed substantially. A step can therefore point initially downhill and still end at a point with a larger function value. Conversely, an unnecessarily small step may produce very little progress. We first examine a quadratic, where every iterate can be calculated exactly, and then identify assumptions that give useful guarantees for more general functions.

Definition 3 ($\boldsymbol{L}$-Lipschitz continuity)**.** We say that the gradient is *L-Lipschitz continuous* if there is a constant $L > 0$ such that

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \le L\|\boldsymbol{x} - \boldsymbol{y}\|_2 \qquad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n. \tag{1.8}$$

A function with this property is called *L-smooth*. The condition limits the change in slope over a given displacement. It does not require convexity: a function can have bounded positive and negative curvature and still satisfy (1.8). For a twice continuously differentiable function, a sufficient condition is $\|H(\boldsymbol{x})\|_2 \le L$ everywhere, where the matrix norm is defined by $\|H\|_2 = \max_{\|\boldsymbol{p}\|_2=1} \|H\boldsymbol{p}\|_2$.

Gradient descent guarantee for L-Lipschitz gradient

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and has an L-Lipschitz gradient. Then, for every point $\boldsymbol{x}$ and displacement $\boldsymbol{p}$,

$$f(\boldsymbol{x} + \boldsymbol{p}) \leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathsf{T}} \boldsymbol{p} + \frac{L}{2} \|\boldsymbol{p}\|_2^2. \tag{1.9}$$

In particular, for every $\alpha > 0$,

$$f\big(\boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})\big) \leq f(\boldsymbol{x}) - \alpha \left(1 - \frac{L\alpha}{2}\right) \|\nabla f(\boldsymbol{x})\|_2^2. \tag{1.10}$$

Consequently, $0 < \alpha < 2/L$ guarantees a strict decrease whenever $\nabla f(\boldsymbol{x}) \neq \boldsymbol{0}$.

Convergence of objective values for a convex function

Suppose f is convex, has an L-Lipschitz gradient on $\mathbb{R}^n$, and attains a global minimum at $\boldsymbol{x}_\star$. Gradient descent with step size $1/L$ satisfies

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_\star) \leq \frac{L\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{2k}, \qquad k \geq 1. \tag{1.11}$$

The bound is of order $1/k$: doubling the iteration count halves the stated upper bound on the objective error. This describes a guarantee, not an exact formula for every run; a particular objective may converge faster. The proof uses both convexity and an attained minimum, whereas the stationarity theorem required neither convexity nor attainment. Further discussion of this estimate appears in [3, pp. 1.24–1.26].

Strong convexity provides a stronger connection between the gradient and the objective error. If the function value is still substantially above its minimum, the gradient cannot already be arbitrarily small. Combining that fact with the decrease obtained at each step produces a fixed fractional reduction of the remaining error.

Convergence under strong convexity

Suppose f is μ-strongly convex and has an L-Lipschitz gradient on $\mathbb{R}^n$, where $\mu > 0$. Then $\mu \leq L$, and f has a unique global minimizer $\boldsymbol{x}_\star$. Gradient descent with step size $1/L$ satisfies

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_\star) \leq \left(1 - \frac{\mu}{L}\right)^k \big(f(\boldsymbol{x}_0) - f(\boldsymbol{x}_\star)\big), \qquad k \geq 1. \tag{1.12}$$

The points $\boldsymbol{x}_k$ also converge to $\boldsymbol{x}_\star$.

2 Numerical differentiation

Recall the definition of a derivative. For a real-valued function f defined on an open interval containing x, the derivative at x is

$$f'(x) = \lim_{s \to 0} \frac{f(x + s) - f(x)}{s}, \tag{2.1}$$

provided this limit exists as a finite number. For a nonzero increment s, the numerator is the change in height between the graph points $(x, f(x))$ and $(x+s, f(x+s))$, while the denominator is their horizontal separation. Their quotient is the slope of the secant line through the two points. When f is differentiable, these secant slopes approach the tangent slope $f'(x)$ as the second point approaches the first.

Numerical differentiation replaces this limiting process by a calculation with a small, nonzero increment. An optimization method may need a derivative even when evaluating f requires a simulation and no usable derivative formula is available. A finite difference estimates the derivative from a few nearby function values; we then ask how its error depends on the spacing and on the smoothness of f [4, Sections 5.4–5.5]. A smaller spacing brings the sample points closer to x, but on a computer it can also amplify errors in the function values.

2.1 Forward, backward, and central differences

Choose a step size $h > 0$. Taking $s = h$ in the difference quotient in (2.1) gives the *forward difference*, using x and $x + h$. Taking $s = -h$ and cancelling the two minus signs gives the *backward difference*, using $x - h$ and x:

$$D_h^+ f(x) = \frac{f(x+h) - f(x)}{h}, \qquad D_h^- f(x) = \frac{f(x) - f(x-h)}{h}. \tag{2.2}$$

Both estimate $f'(x)$ using a secant line that has x as an endpoint. For a differentiable function, both quotients tend to $f'(x)$ as $h \to 0$, so their average does as well. This average is the *central difference*, which places x halfway between the two evaluation points:

$$D_h^0 f(x) = \frac{D_h^+ f(x) + D_h^- f(x)}{2} = \frac{f(x+h) - f(x-h)}{2h}. \tag{2.3}$$

Its denominator is $2h$ because that is the distance between the points. Symmetry will give this formula better accuracy for sufficiently smooth functions. A one-sided formula remains useful when evaluations on one side of x are unavailable, as can happen near the boundary of the domain.

The *truncation error* is the difference between the exact derivative and the finite-difference approximation, assuming exact function values and exact arithmetic. To bound this error uniformly as h decreases, fix $h_0 > 0$ such that f is defined on $I = [x - 2h_0, x + 2h_0]$, and use $0 < h \le h_0$. Whenever the relevant derivative is continuous, write

$$M_j = \max_{t \in I} |f^{(j)}(t)|, \tag{2.4}$$

where $f^{(j)}$ denotes the jth derivative of f. These constants do not depend on h.

Theorem 4 (First-derivative error bounds)**.** If $f \in C^2(I)$, then

$$|D_h^+ f(x) - f'(x)| \le \frac{M_2 h}{2}, \qquad |D_h^- f(x) - f'(x)| \le \frac{M_2 h}{2}. \tag{2.5}$$

If $f \in C^3(I)$, then

$$|D_h^0 f(x) - f'(x)| \le \frac{M_3 h^2}{6}. \tag{2.6}$$

Proof. Taylor's theorem gives $f(x \pm h) = f(x) \pm hf'(x) + R_\pm$, where $|R_\pm| \le M_2 h^2/2$. Substitution into the one-sided formulas and division by h prove (2.5). For the central formula, retain one more term:

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{h^2}{2} f''(x) + R_\pm, \qquad |R_\pm| \le \frac{M_3 h^3}{6}. \tag{2.7}$$

Subtracting the two expansions cancels both $f(x)$ and the quadratic term. Consequently,

$$D_h^0 f(x) - f'(x) = \frac{R_+ - R_-}{2h}, \qquad \left|\frac{R_+ - R_-}{2h}\right| \le \frac{M_3 h^2}{6}. \tag{2.8}$$

This proves the central-difference bound. ∎

An approximation is *convergent* if its truncation error tends to zero as $h \to 0$. It has error of order $O(h^p)$ if there is a constant C, independent of sufficiently small h, such that the absolute error is at most Ch^p [4, Section 5.5]. The theorem therefore proves convergence, with first-order error for forward and backward differences and second-order error for the central difference. Halving h divides the corresponding bounds by 2 and 4. Actual errors exhibit these ratios asymptotically when the leading error coefficient is nonzero; special functions or evaluation points can give smaller errors.

Example 5 (Comparing the three finite-difference methods)**.** For $f(x) = x^3$ at $x = 1$, the exact derivative is $f'(1) = 3$. Expanding the cubes gives

$$\begin{aligned} D_h^+ f(1) &= 3 + 3h + h^2, \\ D_h^- f(1) &= 3 - 3h + h^2, \\ D_h^0 f(1) &= 3 + h^2. \end{aligned} \tag{2.9}$$

The forward and backward errors have opposite linear terms. Averaging the two formulas cancels those terms and gives the central difference. The following absolute errors illustrate the different rates:

h	Forward error	Backward error	Central error
0.1	0.31	0.29	0.01
0.05	0.1525	0.1475	0.0025
0.025	0.075625	0.074375	0.000625

Each halving of h reduces the central error by exactly 4 here, while the one-sided error ratios approach 2. Figure 1 compares the three secants with the tangent at $x = 1$, using the larger step $h = 1/4$ to make the geometry visible. The central secant joins the two surrounding graph points; it need not pass through $(1, f(1))$ itself.

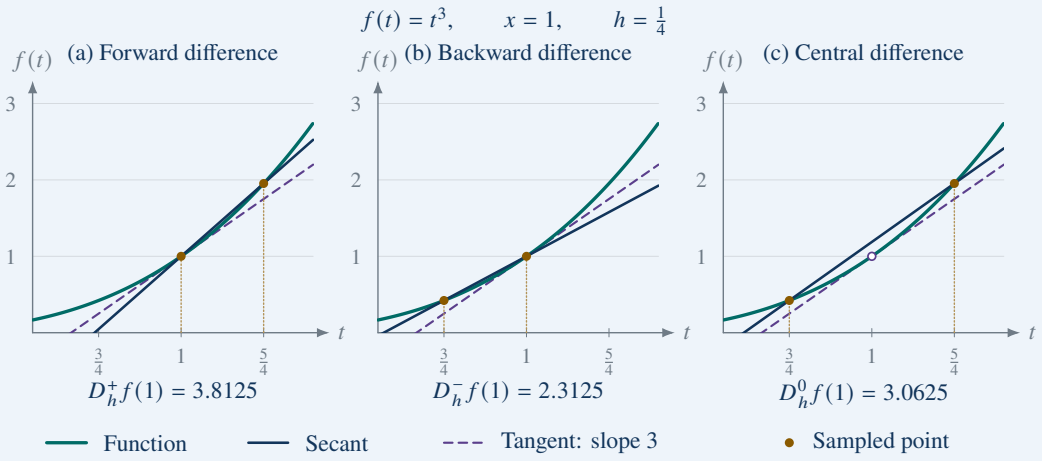


FIGURE 1: Forward, backward, and central secants for $f(t) = t^3$ at $x = 1$, with $h = 1/4$. Gold points supply the sampled function values; the dashed tangent has slope 3. In panel (c), the open point marks $(1, f(1))$, which the central formula does not need to sample.

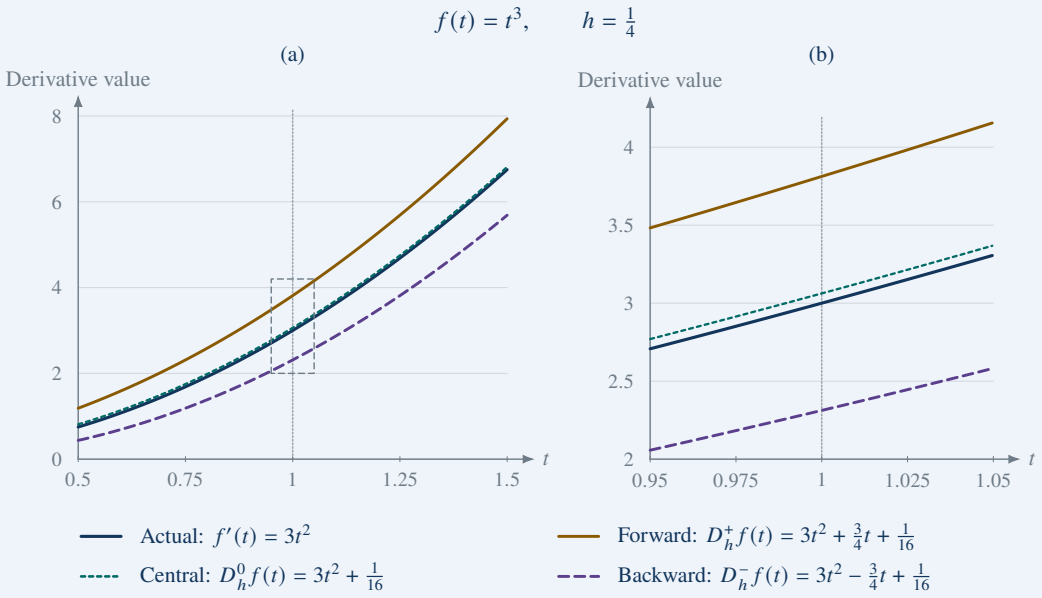


FIGURE 2: Forward, backward, and central finite-difference estimates of $f'(t) = 3t^2$ for $f(t) = t^3$, using $h = 1/4$. (a) The estimates as functions of t on $[0.5, 1.5]$. (b) A magnified view of the outlined region near $t = 1$. The central estimate lies $h^2 = 1/16$ above the exact derivative and is the closest of the three estimates throughout the interval shown in (a).

2.2 Second derivatives

A second derivative measures the rate of change of the first derivative. We therefore approximate $f''(x)$ by taking a difference of two nearby secant slopes and dividing by their separation. Each secant slope is already a first-difference approximation, so this amounts to applying finite differences twice.

Let $h > 0$. For the forward formula, compare the slopes on $[x, x+h]$ and $[x+h, x+2h]$. For the backward formula, compare the slopes on $[x-2h, x-h]$ and $[x-h, x]$. For the central formula, compare the slopes on $[x-h, x]$ and $[x, x+h]$. In each case, the two intervals have midpoints separated by h. Using the first-difference formulas from (2.2), we define the following approximations to $f''(x)$ [4, Section 5.4]:

$$\begin{aligned} D^+_{2,h} f(x) &= \frac{D^+_h f(x+h) - D^+_h f(x)}{h} \\ &= \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}, \\ D^-_{2,h} f(x) &= \frac{D^-_h f(x) - D^-_h f(x-h)}{h} \\ &= \frac{f(x) - 2f(x-h) + f(x-2h)}{h^2}, \\ D^0_{2,h} f(x) &= \frac{D^+_h f(x) - D^-_h f(x)}{h} \\ &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \end{aligned} \tag{2.10}$$

The subscript 2 indicates that these formulas approximate the second derivative; it does not denote a square. The superscripts $+$, $-$, and 0 indicate forward, backward, and central placement of the evaluation points.

Each denominator contains h^2 because one factor of h comes from calculating the secant slopes, and the other comes from measuring how those slopes change. In the central formula, the left and right secant intervals have midpoints $x - h/2$ and $x + h/2$, which are h apart. This explains why their slope difference is divided by h, even though the outer evaluation points $x - h$ and $x + h$ are $2h$ apart. Associating a secant slope with its interval midpoint is an approximation; the secant slope need not equal the derivative at that midpoint.

The forward and backward formulas use three points on one side of x. The central formula uses three points arranged symmetrically around x, allowing leading error terms to cancel.

Theorem 6 (Second-derivative error bounds)**.** If $f \in C^3(I)$, then

$$|D^+_{2,h} f(x) - f''(x)| \le M_3 h, \qquad |D^-_{2,h} f(x) - f''(x)| \le M_3 h. \tag{2.11}$$

If $f \in C^4(I)$, then

$$|D^0_{2,h} f(x) - f''(x)| \le \frac{M_4 h^2}{12}. \tag{2.12}$$

Thus the one-sided formulas converge with first-order error, and the central formula converges with second-order error.

Proof. Applying the fundamental theorem of calculus twice gives

$$D^{+}_{2,h} f(x) = \frac{1}{h^2} \int_0^h \int_0^h f''(x + s + t)\, \mathrm{d}s\, \mathrm{d}t. \tag{2.13}$$

Since $|f''(x + s + t) - f''(x)| \le M_3(s + t)$, the error is at most $M_3 h^{-2} \int_0^h \int_0^h (s + t)\, \mathrm{d}s\, \mathrm{d}t = M_3 h$. Replacing $x + s + t$ by $x - s - t$ proves the backward bound. For the central formula, Taylor expansions through the cubic term give

$$f(x + h) - 2f(x) + f(x - h) = h^2 f''(x) + R_+ + R_-, \qquad |R_\pm| \le \frac{M_4 h^4}{24}. \tag{2.14}$$

The odd-degree terms cancel. Dividing by h^2 proves the central bound. ∎

Example 7 (A central second difference)**.** For $f(x) = x^4$, we have $f''(1) = 12$, whereas $D^0_{2,h} f(1) = 12 + 2h^2$. The error is exactly $2h^2$. Since $f^{(4)}(x) = 24$, the bound $M_4 h^2/12 = 2h^2$ is attained in this example.

The central second difference can also be read directly as a change in two adjacent secant slopes:

$$D^0_{2,h} f(1) = \frac{D^+_h f(1) - D^-_h f(1)}{h}. \tag{2.15}$$

At $h = 1/4$, the backward slope is 2.734375 and the forward slope is 5.765625. Their difference divided by $1/4$ is 12.125, giving error $0.125 = 2(1/4)^2$. Figure 3 shows these slopes and how the error decreases as h is reduced.

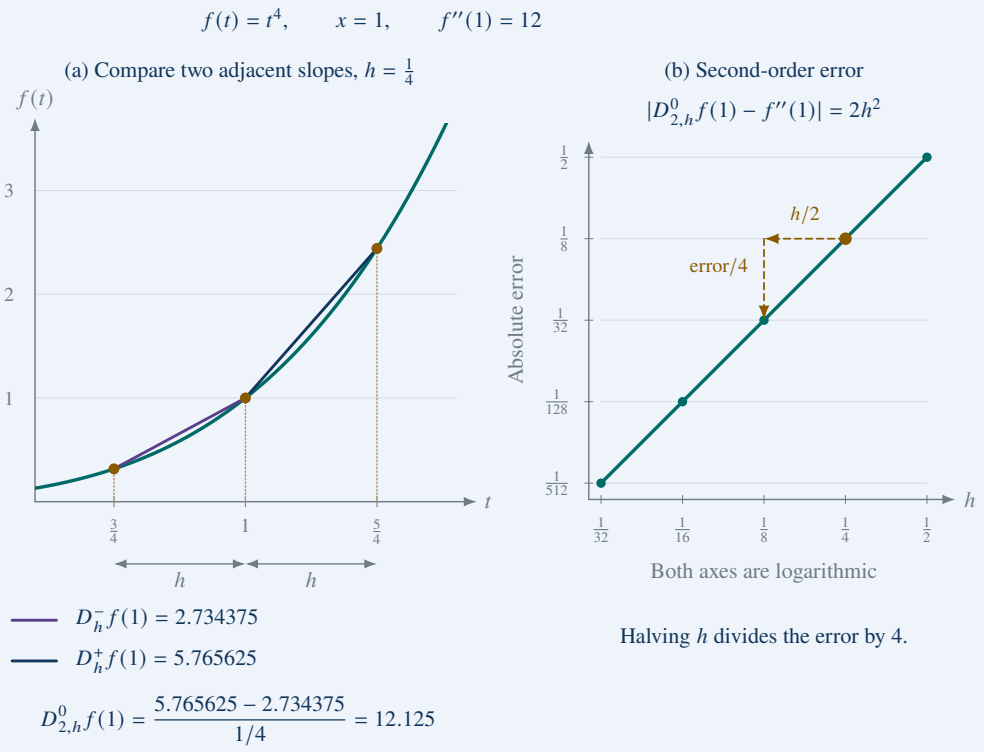


FIGURE 3: The central second difference for $f(t) = t^4$ compares the right and left secant slopes at $x = 1$. With $h = 1/4$, it gives 12.125, while $f''(1) = 12$. The log–log plot shows the exact truncation error $2h^2$: halving h divides the error by four.

2.3 How small should the step be?

The preceding convergence results assume exact function values and arithmetic. On a computer, nearby function values can have nearly identical leading digits, which disappear on subtraction. Dividing the remaining error by a small step can then magnify it. This competition between truncation error and errors in computation limits the benefit of reducing h [4, Section 5.5].

To make the issue precise, suppose each available function value is $\widetilde{f}(t) = f(t) + e(t)$ with $|e(t)| \le \delta$. In a simplified model that otherwise performs arithmetic exactly, the triangle inequality gives

$$\begin{aligned} |D_h^{\pm} \widetilde{f}(x) - f'(x)| &\le \frac{M_2 h}{2} + \frac{2\delta}{h}, \\ |D_h^0 \widetilde{f}(x) - f'(x)| &\le \frac{M_3 h^2}{6} + \frac{\delta}{h}. \end{aligned} \tag{2.16}$$

The first term decreases with h, while the second increases. All three second-derivative formulas have evaluation-error contributions bounded by $4\delta/h^2$, because the absolute values of their numerator coefficients sum to 4. For example,

$$|D_{2,h}^0 \widetilde{f}(x) - f''(x)| \le \frac{M_4 h^2}{12} + \frac{4\delta}{h^2}. \tag{2.17}$$

Thus exact-arithmetic convergence does not imply convergence as $h \to 0$ with a fixed evaluation accuracy. A simulation with numerical error, as well as floating-point arithmetic, can impose such a limit.

Balancing the two terms in (2.16) suggests h proportional to $\delta^{1/2}$ for one-sided first differences and to $\delta^{1/3}$ for central first differences, with proportionality constants depending on derivative bounds. For quantities scaled to have typical magnitudes near 1, this motivates the familiar starting choices $h \approx \epsilon_{\text{mach}}^{1/2}$ and $h \approx \epsilon_{\text{mach}}^{1/3}$, respectively. These are guidelines, not universal step sizes: the scales of the variable, function values, derivatives, and evaluation errors all matter.

2.4 From derivatives to gradients and Hessians

For $f : \mathbb{R}^n \to \mathbb{R}$, a partial derivative measures the rate of change when one coordinate varies and the others are held fixed. Let $\boldsymbol{e}_j$ be the jth coordinate vector. The definition is the ordinary derivative definition applied to $g_j(t) = f(\boldsymbol{x} + t\boldsymbol{e}_j)$:

$$\frac{\partial f}{\partial x_j}(\boldsymbol{x}) = \lim_{t \to 0} \frac{f(\boldsymbol{x} + t\boldsymbol{e}_j) - f(\boldsymbol{x})}{t}. \tag{2.18}$$

The gradient $\nabla f(\boldsymbol{x})$ collects these n partial derivatives into a column vector. We can approximate each component by choosing a step $h_j > 0$ and using a forward, backward, or central difference along that coordinate:

$$\begin{aligned} \frac{\partial f}{\partial x_j}(\boldsymbol{x}) &\approx \frac{f(\boldsymbol{x} + h_j\boldsymbol{e}_j) - f(\boldsymbol{x})}{h_j}, \\ \frac{\partial f}{\partial x_j}(\boldsymbol{x}) &\approx \frac{f(\boldsymbol{x}) - f(\boldsymbol{x} - h_j\boldsymbol{e}_j)}{h_j}, \\ \frac{\partial f}{\partial x_j}(\boldsymbol{x}) &\approx \frac{f(\boldsymbol{x} + h_j\boldsymbol{e}_j) - f(\boldsymbol{x} - h_j\boldsymbol{e}_j)}{2h_j}. \end{aligned} \tag{2.19}$$

The corresponding componentwise errors are $O(h_j)$, $O(h_j)$, and $O(h_j^2)$ under the same smoothness assumptions as in one variable. Reusing $f(\boldsymbol{x})$, a forward- or backward-difference gradient needs $n + 1$ function evaluations; a central-difference gradient needs $2n$. This applies the preceding one-variable formulas along each coordinate direction [4, Sections 5.4–5.5].

The Hessian records how these partial slopes change:

$$H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x}), \qquad H_{ij}(\boldsymbol{x}) = \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}), \qquad 1 \le i, j \le n. \tag{2.20}$$

A diagonal entry H_{ii} measures curvature along a single coordinate. An off-diagonal entry H_{ij} measures how the slope in coordinate j changes as coordinate i varies. For a twice continuously differentiable function, $H_{ij} = H_{ji}$, so the Hessian is symmetric. Both types of entries can be approximated using only function values [1, Section 8.1].

For simplicity, use the same step $h > 0$ in every coordinate, and write $\widehat{H}_h$ for the approximation. For a diagonal entry, apply the central second difference to $t \mapsto f(\boldsymbol{x} + t\boldsymbol{e}_i)$:

$$(\widehat{H}_h)_{ii}(\boldsymbol{x}) = \frac{f(\boldsymbol{x} + h\boldsymbol{e}_i) - 2f(\boldsymbol{x}) + f(\boldsymbol{x} - h\boldsymbol{e}_i)}{h^2}. \tag{2.21}$$

For a mixed entry with $i \neq j$, first estimate the jth partial derivative at each of $\boldsymbol{x} + h\boldsymbol{e}_i$ and $\boldsymbol{x} - h\boldsymbol{e}_i$. Taking the difference of these two estimated slopes and dividing by $2h$ gives

$$(\widehat{H}_h)_{ij}(\boldsymbol{x}) = \frac{1}{4h^2}\Big[f(\boldsymbol{x} + h\boldsymbol{e}_i + h\boldsymbol{e}_j) - f(\boldsymbol{x} + h\boldsymbol{e}_i - h\boldsymbol{e}_j) \\ - f(\boldsymbol{x} - h\boldsymbol{e}_i + h\boldsymbol{e}_j) + f(\boldsymbol{x} - h\boldsymbol{e}_i - h\boldsymbol{e}_j)\Big]. \tag{2.22}$$

The four evaluations lie at the corners of a square in the i, j coordinate plane. Opposite corners have the same sign. These signs compare the slope in one direction on opposite sides of the square. Figure 4 shows the sampling points and weights for the diagonal and mixed formulas. Compute each mixed entry once and set $(\widehat{H}_h)_{ji} = (\widehat{H}_h)_{ij}$ to preserve symmetry.

Suppose $f \in C^4$ on a fixed closed box around $\boldsymbol{x}$, and let L_4 be a bound on the absolute values of all its fourth-order partial derivatives there. For sufficiently small h that all sampling points lie in the box, the errors satisfy

$$\left|(\widehat{H}_h)_{ii} - H_{ii}\right| \leq \frac{L_4 h^2}{12}, \qquad \left|(\widehat{H}_h)_{ij} - H_{ij}\right| \leq \frac{L_4 h^2}{3} \quad (i \neq j). \tag{2.23}$$

The diagonal bound is the one-variable central second-derivative bound. To understand the mixed bound, applying the fundamental theorem of calculus in both directions expresses the four-corner formula as an average:

$$(\widehat{H}_h)_{ij}(\boldsymbol{x}) = \frac{1}{4h^2}\int_{-h}^{h}\int_{-h}^{h} H_{ij}(\boldsymbol{x} + s\boldsymbol{e}_i + t\boldsymbol{e}_j)\,\mathrm{d}s\,\mathrm{d}t. \tag{2.24}$$

For a one-variable function q with $|q''| \leq L_4$, Taylor's theorem and cancellation of the odd term give $|(2h)^{-1}\int_{-h}^{h} q(t)\,\mathrm{d}t - q(0)| \leq L_4 h^2/6$. Apply this estimate to the two successive coordinate averages above and add their errors to obtain $L_4 h^2/3$. Thus every Hessian entry converges with error $O(h^2)$, and for fixed n the entire approximate Hessian converges as $h \to 0$ in exact arithmetic.

Evaluation error is again amplified. If each function value has absolute error at most δ, its contribution to a diagonal-entry error is at most $4\delta/h^2$, while its contribution to a mixed-entry error is at most δ/h^2. These terms must be added to the respective truncation bounds in (2.23). Making h arbitrarily small is therefore not a reliable way to improve a computed Hessian. With reuse of function values, the central gradient and the full symmetric Hessian together use $1 + 2n + 4\binom{n}{2} = 2n^2 + 1$ sampling points: the center, two points on each coordinate axis, and four points for each pair of distinct coordinates.

Example 8 (A gradient and Hessian from nearby values)**.** Consider $f(x, y) = x^4 + x^2y^2 + y^4$ at $(x, y) = (1, 1)$. Direct differentiation gives

$$\nabla f(1, 1) = \begin{bmatrix} 6 \\ 6 \end{bmatrix}, \qquad H(1, 1) = \begin{bmatrix} 14 & 4 \\ 4 & 14 \end{bmatrix}. \tag{2.25}$$

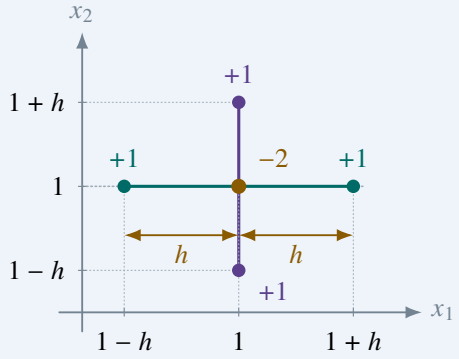
Holding $y = 1$ fixed gives $f(x, 1) = x^4 + x^2 + 1$. Its central first difference at $x = 1$ is $6 + 4h^2$, and its central second difference is $14 + 2h^2$. The same expressions hold with x and y interchanged. In the mixed formula, the terms depending only on x or only on y cancel; the remaining numerator is $[(1 + h)^2 - (1 - h)^2]^2 = 16h^2$. Hence the central approximations are

$$\widehat{\boldsymbol{g}}_h(1, 1) = \begin{bmatrix} 6 + 4h^2 \\ 6 + 4h^2 \end{bmatrix}, \qquad \widehat{H}_h(1, 1) = \begin{bmatrix} 14 + 2h^2 & 4 \\ 4 & 14 + 2h^2 \end{bmatrix}. \tag{2.26}$$

For $h = 1/4$, this gives the approximate gradient $(6.25, 6.25)^\mathsf{T}$ and Hessian diagonal entries 14.125. Halving h divides each nonzero error by 4. The mixed entry is exact for this polynomial; second-order error bounds allow such cancellation. Figure 4 identifies the nine points used when $n = 2$: five points supply the two diagonal entries and gradient components, and the four corners supply the mixed entry.

(a) Diagonal entries: $H_{11}(1,1)$ and $H_{22}(1,1)$

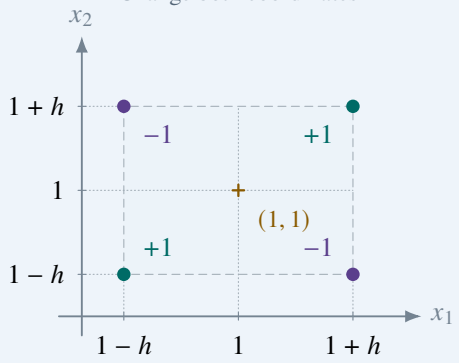
Horizontal: H_{11}; vertical: H_{22}



Use each three-point line separately; divide by h^2.

(b) A mixed entry: $H_{12}(1,1)$

Change both coordinates



Use the signed sum of the four corner values; divide by $4h^2$.

The displayed numbers are formula weights, not function values. The step is $h = 1/4$ in the worked example.

FIGURE 4: Function-evaluation points for the gradient and Hessian of $f(x, y) = x^4 + x^2y^2 + y^4$ at $(1, 1)$. The numbers next to the points are finite-difference weights. On the left, apply the horizontal and vertical three-point formulas separately, dividing each weighted sum by h^2; the four outer points also supply the central gradient. On the right, divide the signed sum of the four corner values by $4h^2$ to approximate the mixed entry. The center and eight neighboring points supply the entire central gradient and Hessian.

3 The BFGS algorithm: using curvature information

Gradient descent uses the local slope to choose a direction, but it uses the same scalar step size in every coordinate. In an elongated valley, the curvature can be much larger across the valley than along it: a step that is safe across the valley can make little progress toward the minimizer. Newton's method addresses this mismatch by using the Hessian to change both the length and the direction of the gradient step. Quasi-Newton methods seek a similar benefit while avoiding the computation of a Hessian at every iteration. They use changes in the gradient to update an approximation to the curvature; see [5, p. 580] and [1, Section 6.1] for more information.

What can we learn from one step?

A gradient tells us the current slope. The change in the gradient after a step tells us how quickly that slope is changing. If a small move causes a large slope change, the function has substantial curvature along that move, and future steps should account for it. BFGS repeatedly uses this information to adjust the matrix that turns a gradient into a proposed displacement. Each iteration therefore does two jobs: it seeks a lower function value and gathers information for choosing the next direction.

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) [6–9] method maintains a matrix H_k that approximates the *inverse* Hessian. This matrix changes the scale and direction of the gradient step, helping the method account for unequal curvature in different directions. Starting with $H_0 = \mathbb{1}$ gives the gradient-descent direction on the first iteration. At iteration k, use

$$\boldsymbol{p}_k = -H_k \nabla f(\boldsymbol{x}_k), \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k. \tag{3.1}$$

A line search chooses the step length $\alpha_k > 0$: the matrix determines a useful direction, but a full step need not decrease the objective.

After taking the step, record how far we moved and how much the gradient changed:

$$\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k, \qquad \boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k). \tag{3.2}$$

In one variable, y_k/s_k measures the change in slope per unit displacement, so its reciprocal s_k/y_k estimates inverse curvature when the ratios are defined. In several variables, the corresponding requirement is that the new matrix map the observed gradient change back to the displacement:

$$H_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k. \tag{3.3}$$

This is the *secant equation*. One step supplies information in only one direction, so it does not determine the whole matrix. BFGS combines this new information with the previous approximation H_k, using gradients rather than evaluating second derivatives [1, Section 6.1]. It forms

$$\begin{aligned} \rho_k &= \frac{1}{\boldsymbol{y}_k^{\mathsf{T}} \boldsymbol{s}_k}, \\ H_{k+1} &= (\mathbb{1} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^{\mathsf{T}}) H_k (\mathbb{1} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^{\mathsf{T}}) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^{\mathsf{T}}. \end{aligned} \tag{3.4}$$

The update adjusts the old matrix using the latest displacement and gradient change, so that $H_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k$. Indeed, since $\rho_k = 1/(\boldsymbol{y}_k^{\mathsf{T}}\boldsymbol{s}_k)$, we have $\rho_k \boldsymbol{s}_k^{\mathsf{T}}\boldsymbol{y}_k = 1$. Consequently,

$$\begin{aligned} (\mathbb{1} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^{\mathsf{T}})\boldsymbol{y}_k &= \boldsymbol{y}_k(1 - \rho_k \boldsymbol{s}_k^{\mathsf{T}}\boldsymbol{y}_k) = \boldsymbol{0}, \\ H_{k+1}\boldsymbol{y}_k &= (\mathbb{1} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^{\mathsf{T}}) H_k \boldsymbol{0} + \rho_k \boldsymbol{s}_k (\boldsymbol{s}_k^{\mathsf{T}}\boldsymbol{y}_k) = \boldsymbol{s}_k. \end{aligned} \tag{3.5}$$

If $\boldsymbol{s}_k^{\mathsf{T}}\boldsymbol{y}_k > 0$, it also preserves symmetry and positive definiteness, which keeps the next direction downhill whenever the gradient is nonzero [1, Section 6.1]. The algorithm checks this curvature information before using it.

Algorithm: Basic BFGS

Inputs. An initial point $\boldsymbol{x}_0$, a gradient tolerance $\varepsilon_g > 0$, an iteration limit K, and a limit on the number of trial step lengths.

1. **Initialize.** Set $H_0 = \mathbb{1}$, evaluate $f(\boldsymbol{x}_0)$ and $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0)$, and set $k = 0$.
2. **Check for termination.** If $\|\boldsymbol{g}_k\|_2 \le \varepsilon_g$, return $\boldsymbol{x}_k$ as an approximate stationary point. If $k = K$, stop and report the iteration limit.
3. **Choose a direction.** Set $\boldsymbol{p}_k = -H_k \boldsymbol{g}_k$.

4. **Choose and take a step.** Choose $\alpha_k > 0$ so that

$$f(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) < f(\boldsymbol{x}_k). \tag{3.6}$$

   A simple option is to try $\alpha = 1$ and repeatedly halve it until this decrease occurs. If no acceptable step is found within the trial limit, stop and report failure. Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$ and evaluate $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{x}_{k+1})$. If $\|\boldsymbol{g}_{k+1}\|_2 \le \varepsilon_g$, return $\boldsymbol{x}_{k+1}$.

5. **Update the curvature estimate.** Compute $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{y}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$. If $\boldsymbol{s}_k^{\mathsf{T}} \boldsymbol{y}_k > 0$, use the BFGS formula (3.4) to obtain H_{k+1}. Otherwise keep $H_{k+1} = H_k$. In either case, retain the new point.

6. **Repeat.** Increase k by 1 and return to Step 2.

The method alternates between taking a downhill step and learning curvature from the resulting gradient change. Halving the step length is called *backtracking*; other choices, such as the exact line searches in Example 11, can also be used. The positive-curvature check keeps H_k positive definite in exact arithmetic, so the proposed direction remains downhill whenever the gradient is nonzero. No Hessian needs to be evaluated or inverted.

This is a basic version of BFGS. Decreasing the objective and preserving positive definiteness do not by themselves guarantee convergence to a minimizer; stronger convergence guarantees require additional assumptions. Reaching an iteration or trial limit is reported separately from satisfying the gradient tolerance.

Example 9 (Updating the curvature as the function changes)**.** Consider the nonquadratic function

$$f(x) = (x-1)^4 + (x-1)^2, \qquad f'(x) = 4(x-1)^3 + 2(x-1), \qquad f''(x) = 12(x-1)^2 + 2. \tag{3.7}$$

Its unique minimizer is $x_\star = 1$. Start with $x_0 = 2$ and $H_0 = 1$. In one variable, the two factors surrounding H_k in (3.4) vanish because $\rho_k s_k y_k = 1$. Thus BFGS simply uses $H_{k+1} = s_k / y_k$ after a step with $s_k y_k > 0$.

Initially $f'(x_0) = 6$ and $p_0 = -6$. A full step would land at -4, increasing f from 2 to 650. Choose $\alpha_0 = 1/4$ instead. This gives

$$\begin{aligned} x_1 &= 2 + \tfrac{1}{4}(-6) = \tfrac{1}{2}, \qquad f'(x_1) = -\tfrac{3}{2}, \\ s_0 &= -\tfrac{3}{2}, \qquad y_0 = -\tfrac{3}{2} - 6 = -\tfrac{15}{2}, \qquad H_1 = \frac{s_0}{y_0} = \frac{1}{5}. \end{aligned} \tag{3.8}$$

The next direction is $p_1 = -(1/5)(-3/2) = 3/10$. Accepting $\alpha_1 = 1$ gives $x_2 = 4/5$, where $f'(x_2) = -54/125$. The new information changes the inverse-curvature estimate again:

$$s_1 = \frac{3}{10}, \qquad y_1 = -\frac{54}{125} + \frac{3}{2} = \frac{267}{250}, \qquad H_2 = \frac{s_1}{y_1} = \frac{25}{89}. \tag{3.9}$$

Notice that $H_2 \neq 1/f''(x_2) = 25/62$. The estimate records curvature measured over the previous displacement; it need not equal the inverse Hessian at the new endpoint.

Trying a unit step and halving until the objective decreases gives $\alpha_0 = 1/4$ initially and unit steps thereafter. Every nonzero step has $s_k y_k > 0$ because $f''(x) > 0$ everywhere, so no matrix update is skipped. The resulting values are shown below, rounded for display.

k	x_k	$f(x_k)$	$\lvert f'(x_k)\rvert$	H_k
0	2	2	6	1
1	0.5	0.3125	1.5	0.2
2	0.8	0.0416	0.432	0.280899
3	0.921348	0.00622436	0.159250	0.444906
4	0.992199	6.08531×10^{-5}	0.0156031	0.493232
5	0.999895	1.09509×10^{-8}	2.09293×10^{-4}	0.499938
6	0.999999987	1.66528×10^{-16}	2.58092×10^{-8}	—

With gradient tolerance 10^{-6}, the algorithm stops at x_6 before computing H_6. The estimates approach $1/f''(1) = 1/2$, and the final errors decrease rapidly. Unlike a quadratic with constant curvature, this objective requires the algorithm to keep revising its scale as it moves; no single early measurement determines the curvature everywhere.

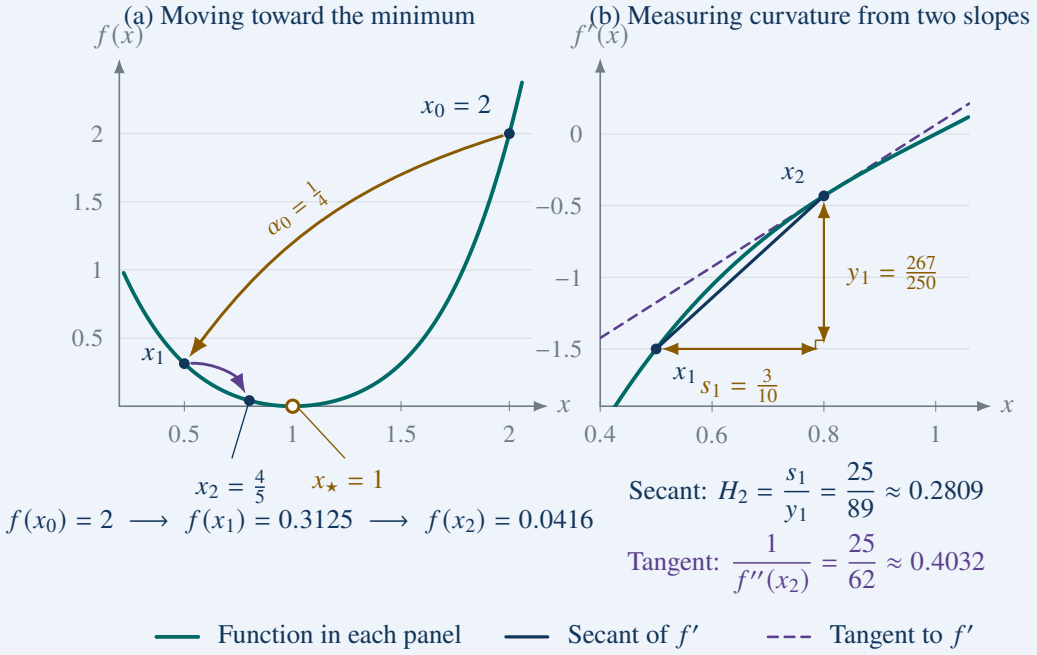


FIGURE 5: BFGS applied to $f(x) = (x-1)^4 + (x-1)^2$. Left: the arrows connect the first three iterates, with $\alpha_0 = 1/4$ and $\alpha_1 = 1$; the later iterates approach the marked minimizer. Right: the secant of f' between x_1 and x_2 has slope y_1/s_1, so its reciprocal gives the updated inverse-curvature estimate $H_2 = s_1/y_1$. The dashed tangent has slope $f''(x_2)$, showing why the measured curvature need not equal the curvature at the new endpoint. The panels use different coordinate scales.

Theorem 10 (Secant equation and descent for BFGS)**.** Suppose that H_k is symmetric positive definite and that $\boldsymbol{s}_k^{\mathsf{T}}\boldsymbol{y}_k > 0$. Then the matrix H_{k+1} in (3.4) is symmetric positive definite and satisfies $H_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k$. Consequently, if the gradient at the next iterate is nonzero, then $-H_{k+1}\nabla f(\boldsymbol{x}_{k+1})$ is a descent direction.

Proof. The transpose of the left factor surrounding H_k is the right factor, and H_k is symmetric. Both terms in (3.4) are therefore symmetric. To verify the secant equation, first observe that

$$(\mathbb{1} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^\top)\boldsymbol{y}_k = \boldsymbol{y}_k - \frac{\boldsymbol{y}_k(\boldsymbol{s}_k^\top \boldsymbol{y}_k)}{\boldsymbol{y}_k^\top \boldsymbol{s}_k} = \boldsymbol{0}. \tag{3.10}$$

Multiplying (3.4) by $\boldsymbol{y}_k$ thus makes its first term zero, while its second term gives

$$H_{k+1}\boldsymbol{y}_k = \rho_k \boldsymbol{s}_k(\boldsymbol{s}_k^\top \boldsymbol{y}_k) = \boldsymbol{s}_k. \tag{3.11}$$

To prove positive definiteness, take any nonzero z and define $\boldsymbol{v} = (\mathbb{1} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^\top)z$. Substitution into the quadratic form gives

$$z^\top H_{k+1} z = \boldsymbol{v}^\top H_k \boldsymbol{v} + \rho_k(\boldsymbol{s}_k^\top z)^2. \tag{3.12}$$

Both terms are nonnegative. If $\boldsymbol{s}_k^\top z \neq 0$, the second term is strictly positive because $\rho_k > 0$. If $\boldsymbol{s}_k^\top z = 0$, then $\boldsymbol{v} = z \neq \boldsymbol{0}$, and the first term is strictly positive because H_k is positive definite. Hence the sum is positive for every nonzero z.

Finally, the directional derivative at $\boldsymbol{x}_{k+1}$ in the proposed direction is

$$\begin{aligned} \nabla f(\boldsymbol{x}_{k+1})^\top \left[-H_{k+1}\nabla f(\boldsymbol{x}_{k+1})\right] &= -\nabla f(\boldsymbol{x}_{k+1})^\top H_{k+1} \nabla f(\boldsymbol{x}_{k+1}) \\ &< 0 \end{aligned} \tag{3.13}$$

when the gradient is nonzero. A sufficiently small positive move in this direction therefore decreases f. ∎

At any iteration with a positive definite H_k, the same argument gives

$$\begin{aligned} \boldsymbol{p}_k &= -H_k \nabla f(\boldsymbol{x}_k), \\ \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}_k &= -\nabla f(\boldsymbol{x}_k)^\top H_k \nabla f(\boldsymbol{x}_k) < 0 \qquad \text{if } \nabla f(\boldsymbol{x}_k) \neq \boldsymbol{0}. \end{aligned} \tag{3.14}$$

Gradient descent uses the fixed matrix $H_k = \mathbb{1}$; Newton's method uses the exact inverse Hessian when it is positive definite; BFGS changes its approximation after each accepted step. Multiplication by H_k can both rescale and rotate the negative gradient. The resulting direction need not be the direction of steepest decrease per unit Euclidean displacement, but it can account for the different curvatures that restrict a gradient-descent step.

Example 11 (Gradient descent, Newton, and BFGS on the same valley)**.** Consider the strongly convex quadratic

$$f(x_1, x_2) = \frac{1}{2}(9x_1^2 + x_2^2), \qquad \nabla f(\boldsymbol{x}) = \begin{bmatrix} 9x_1 \\ x_2 \end{bmatrix}, \qquad A = \nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}. \tag{3.15}$$

Figure 6 shows the objective as a surface whose height is $f(x_1, x_2)$. The larger curvature in the x_1 direction produces a narrow valley extending along the x_2 direction. The marked points identify the common starting point and the minimum before we compare the methods' paths across this surface.

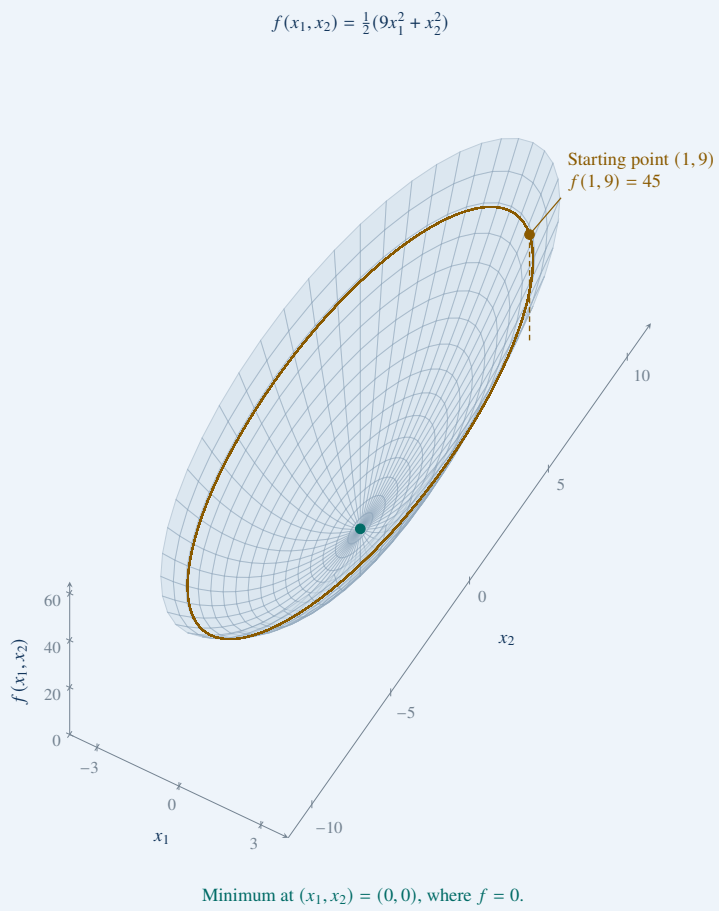


FIGURE 6: The objective surface $f(x_1, x_2) = \frac{1}{2}(9x_1^2 + x_2^2)$ for the gradient-descent, Newton, and BFGS comparison. The gold point is the starting point $(1, 9)^\top$ lifted to height 45; the gold curve is the level $f = 45$. The teal point is the unique minimizer $(0, 0)^\top$, with value 0. The horizontal coordinates use equal unit scales, while the vertical scale is compressed to display the valley.

All three methods start at $\boldsymbol{x}_0 = (1, 9)^\top$, where $f(\boldsymbol{x}_0) = 45$, and the unique minimizer is $\boldsymbol{x}_\star = \boldsymbol{0}$. To isolate the effect of the search direction, use exact line searches. Along any nonzero descent direction $\boldsymbol{p}$, differentiating the scalar quadratic $f(\boldsymbol{x} + \alpha \boldsymbol{p})$ gives

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\alpha} f(\boldsymbol{x} + \alpha \boldsymbol{p}) &= \nabla f(\boldsymbol{x})^\top \boldsymbol{p} + \alpha \boldsymbol{p}^\top A \boldsymbol{p}, \\ \alpha &= -\frac{\nabla f(\boldsymbol{x})^\top \boldsymbol{p}}{\boldsymbol{p}^\top A \boldsymbol{p}}. \end{aligned} \tag{3.16}$$

The denominator is positive because A is positive definite, and the numerator makes $\alpha > 0$ for a descent direction. This formula is available because the objective is quadratic; an exact line search for a general nonlinear function usually requires additional work.

Gradient descent. Initially $\nabla f(\boldsymbol{x}_0) = (9, 9)^\top$, so $\boldsymbol{p}_0 = (-9, -9)^\top$ and $\alpha_0 = 162/810 = 1/5$. The first iterate is $\boldsymbol{x}_1 = (-4/5, 36/5)^\top$. At subsequent iterates the two components of the gradient continue to have equal magnitudes, so the exact line-search formula continues to give $\alpha_k = 1/5$. The update therefore

multiplies the first coordinate by $-4/5$ and the second by $4/5$. Induction gives

$$\boldsymbol{x}_k = \begin{bmatrix} (-4/5)^k \\ 9(4/5)^k \end{bmatrix}, \qquad f(\boldsymbol{x}_k) = 45\left(\frac{16}{25}\right)^k. \tag{3.17}$$

The first coordinate repeatedly changes sign as the method moves across the valley. The objective decreases at every iteration, but the method does not reach the minimizer in any finite number of exact-arithmetic steps. It first reduces $f(\boldsymbol{x}_k)/f(\boldsymbol{x}_0)$ below 10^{-6} at $k = 31$.

Newton's method. The exact inverse Hessian cancels the unequal coordinate curvatures:

$$\boldsymbol{p}_0 = -A^{-1}\nabla f(\boldsymbol{x}_0) = -\begin{bmatrix} 1/9 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 9 \\ 9 \end{bmatrix} = \begin{bmatrix} -1 \\ -9 \end{bmatrix} = -\boldsymbol{x}_0. \tag{3.18}$$

The exact line search chooses $\alpha_0 = 1$, so Newton's method reaches the minimizer in one step. Here the quadratic model is the objective itself; this one-step conclusion is special to the quadratic problem.

BFGS. Start with $H_0 = \mathbb{1}$. Its first direction and first iterate are the same as those of gradient descent, but it then uses that move to update its inverse-Hessian approximation:

$$\begin{aligned} \boldsymbol{s}_0 &= \begin{bmatrix} -9/5 \\ -9/5 \end{bmatrix}, & \boldsymbol{y}_0 &= \begin{bmatrix} -81/5 \\ -9/5 \end{bmatrix}, \quad \rho_0 = \frac{5}{162}, \\ H_1 &= \frac{1}{25}\begin{bmatrix} 3 & -2 \\ -2 & 43 \end{bmatrix}, & \nabla f(\boldsymbol{x}_1) &= \begin{bmatrix} -36/5 \\ 36/5 \end{bmatrix}. \end{aligned} \tag{3.19}$$

Substitution into the direction formula now gives

$$\boldsymbol{p}_1 = -H_1\nabla f(\boldsymbol{x}_1) = \begin{bmatrix} 36/25 \\ -324/25 \end{bmatrix} = -\frac{9}{5}\boldsymbol{x}_1, \qquad \alpha_1 = \frac{5}{9}, \qquad \boldsymbol{x}_2 = \boldsymbol{x}_1 + \alpha_1\boldsymbol{p}_1 = \boldsymbol{0}. \tag{3.20}$$

The learned curvature turns the second direction toward the minimizer, and BFGS reaches it after two steps. Notice that $H_1 \neq A^{-1}$: the method did not need to recover the whole inverse Hessian before obtaining the required direction. These are exact-arithmetic results for the stated quadratic and line searches, not iteration guarantees for arbitrary objectives.

Figure 7 makes the comparison visible. Gradient descent repeats its zigzag, whereas BFGS shares the first step and then changes direction using the measured gradient difference. Newton's method uses the exact curvature from the start. The iteration counts describe progress on this example; they do not measure equal computational effort, since Newton uses second derivatives, BFGS stores and updates a matrix, and gradient descent does neither.

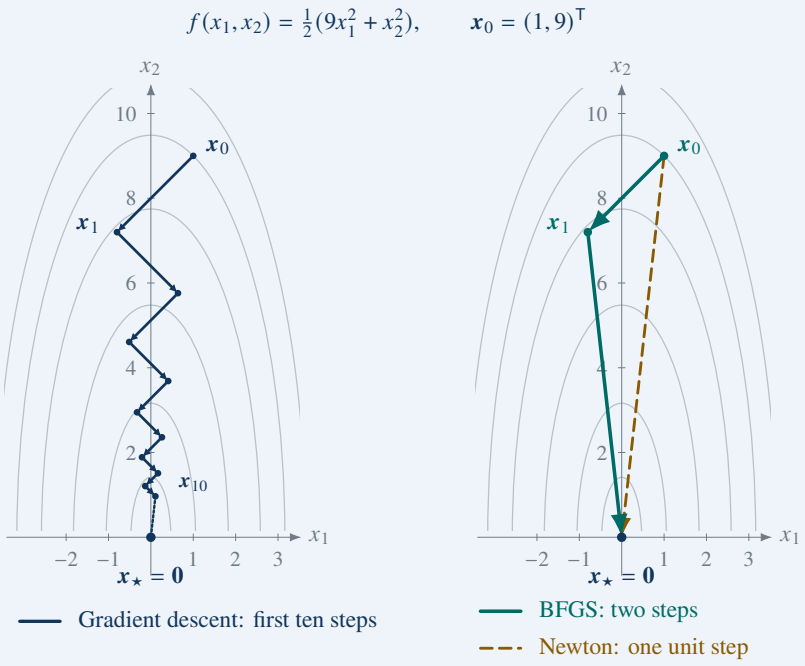


FIGURE 7: Gradient descent, Newton's method, and BFGS applied to $f(x_1, x_2) = \frac{1}{2}(9x_1^2 + x_2^2)$ from $(1, 9)^\mathsf{T}$ with exact line searches and $H_0 = \mathbb{1}$ for BFGS. The contours show equal objective values. BFGS shares the first gradient-descent step and then reaches the minimizer on its second step; Newton reaches it on its first. Gradient descent repeatedly crosses the valley, reducing the objective by a factor of 16/25 per iteration.

4 The Nelder–Mead algorithm: optimization without derivatives

Gradient descent and BFGS use derivatives to decide how to move. When reliable derivatives are unavailable, Nelder–Mead [10] instead compares function values at a collection of points and uses their geometry to choose new points; see [5, pp. 580–581] for more information and more references. For a problem with n variables, it maintains $n + 1$ points called the *vertices* of a simplex. In one dimension, a simplex is a line segment; in two dimensions, it is a filled triangle; in three dimensions, it is a solid tetrahedron.

A simplex includes the points between its vertices, not just the vertices or its boundary. For example, a triangular simplex includes its three edges and its entire interior. The general construction uses weighted averages of the vertices, with nonnegative weights that add to one. The vertices must also be positioned so that the resulting shape occupies all n dimensions.

Definition 12 (***n*-dimensional simplex**)**.** Let $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be $n + 1$ points in $\mathbb{R}^n$. Suppose that the n displacement vectors

$$\boldsymbol{v}_1 - \boldsymbol{v}_0, \quad \boldsymbol{v}_2 - \boldsymbol{v}_0, \quad \ldots, \quad \boldsymbol{v}_n - \boldsymbol{v}_0 \tag{4.1}$$

are linearly independent. The points are then called *affinely independent*. The simplex with these vertices is

$$\mathcal{S} = \left\{ \sum_{i=0}^{n} \lambda_i \boldsymbol{v}_i \ : \ \lambda_i \geq 0 \text{ for every } i, \quad \sum_{i=0}^{n} \lambda_i = 1 \right\}. \tag{4.2}$$

Each point in $\mathcal{S}$ is therefore a weighted average of the vertices. Such an average is called a *convex combination*, and the set of all these combinations is called the *convex hull* of the vertices.

Affine independence describes the geometry of the vertices relative to one another. Starting at $\boldsymbol{v}_0$, the vectors pointing toward the other vertices must provide n independent directions. In two dimensions, this means that the three vertices must not lie on one straight line. In three dimensions, the four vertices must not all lie in one plane. Without this condition, the construction collapses into a lower-dimensional shape.

The weights also explain the distinction between the interior and the boundary. If every weight λ_i is positive, the resulting point lies in the interior of the simplex. If one or more weights vanish, the point lies on its boundary. For example, setting one weight to zero in a triangle leaves a weighted average of the other two vertices, which lies on the edge joining them.

Example 13 (Simplex examples)**.** Figure 8 illustrates the following constructions, including a degenerate choice that fails to produce a two-dimensional simplex.

One variable: a segment. Choose the two vertices $v_0 = 0$ and $v_1 = 2$ in $\mathbb{R}$. Their convex combinations have the form

$$x = (1 - \lambda)0 + \lambda 2 = 2\lambda, \qquad 0 \leq \lambda \leq 1. \tag{4.3}$$

These points fill the interval $[0, 2]$. The choices $\lambda = 0$ and $\lambda = 1$ give the endpoints, while $0 < \lambda < 1$ gives points strictly between them. Two distinct vertices are needed to obtain a one-dimensional simplex.

Two variables: a triangle. Choose

$$\boldsymbol{v}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \boldsymbol{v}_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \qquad \boldsymbol{v}_2 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}. \tag{4.4}$$

The two displacement vectors from $\boldsymbol{v}_0$ point along the two coordinate axes and are linearly independent. A convex combination has coordinates $x_1 = 2\lambda_1$ and $x_2 = 2\lambda_2$, with $\lambda_0 = 1 - \lambda_1 - \lambda_2 \geq 0$. Consequently, the simplex is the filled triangular region

$$\mathcal{S} = \left\{ (x_1, x_2)^\mathsf{T} \in \mathbb{R}^2 \ : \ x_1 \geq 0, \quad x_2 \geq 0, \quad x_1 + x_2 \leq 2 \right\}. \tag{4.5}$$

For instance, $(1/2, 1/2)^\mathsf{T} = \frac{1}{2}\boldsymbol{v}_0 + \frac{1}{4}\boldsymbol{v}_1 + \frac{1}{4}\boldsymbol{v}_2$ lies in the interior because all three weights are positive. The point $(1, 1)^\mathsf{T} = \frac{1}{2}\boldsymbol{v}_1 + \frac{1}{2}\boldsymbol{v}_2$ lies on the edge opposite $\boldsymbol{v}_0$ because the weight of $\boldsymbol{v}_0$ is zero.

Three variables: a tetrahedron. Choose

$$\boldsymbol{v}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \qquad \boldsymbol{v}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \qquad \boldsymbol{v}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \qquad \boldsymbol{v}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{4.6}$$

The three displacement vectors from $\boldsymbol{v}_0$ are linearly independent. Their convex hull is the solid tetrahedron

$$\mathcal{S} = \left\{ (x_1, x_2, x_3)^\mathsf{T} \in \mathbb{R}^3 \ : \ x_1, x_2, x_3 \geq 0, \quad x_1 + x_2 + x_3 \leq 1 \right\}. \tag{4.7}$$

It has four vertices, six edges, and four triangular faces. The inequality includes the entire solid region enclosed by those faces.

A degenerate choice. The three points $(0,0)^\mathsf{T}$, $(1,0)^\mathsf{T}$, and $(2,0)^\mathsf{T}$ do not form a nondegenerate two-dimensional simplex. Their displacement vectors from $(0,0)^\mathsf{T}$ are linearly dependent, and every convex combination has second coordinate zero. Their convex hull is only a line segment, with no triangular interior. Merely choosing three distinct points in $\mathbb{R}^2$ is therefore insufficient: they must also be noncollinear.

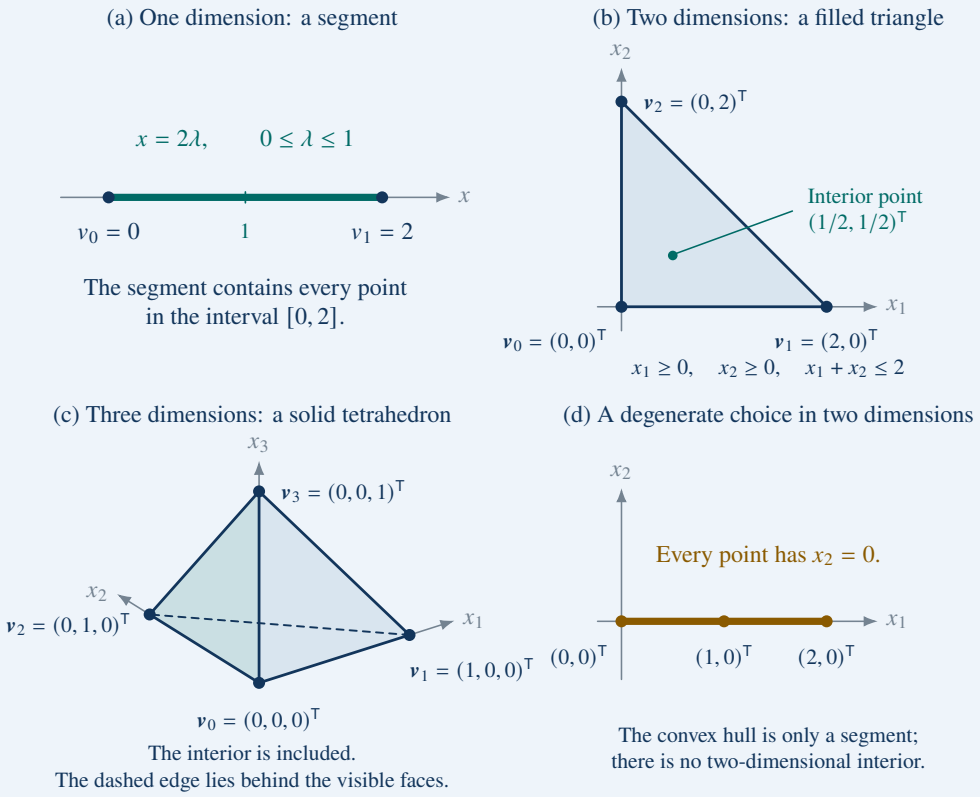


FIGURE 8: Simplices and a degenerate configuration. (a) Two distinct points determine a line segment. (b) Three noncollinear points determine a filled triangle, including the illustrated interior point. (c) Four noncoplanar points determine a solid tetrahedron; the dashed edge is hidden behind the visible faces. (d) Three collinear points have a line-segment convex hull and do not form a nondegenerate two-dimensional simplex.

Nelder–Mead stores the vertices and their function values; it does not evaluate the objective at every point inside the simplex. In two variables, for example, the algorithm begins with three noncollinear sample points, compares their objective values, and changes the triangle by replacing a vertex or moving several vertices. These operations allow the simplex to change its location, size, and shape as the search proceeds.

Starting with a non-degenerate simplex is important because the algorithm constructs new points by affine combinations of the existing vertices. If all initial points lie on one line, these operations keep every subsequent point on that same line, preventing exploration in other directions. The Nelder–Mead method described here is distinct from the simplex algorithm for linear programming.

The geometric operations. At the start of one iteration, order the current vertices so that $f(\boldsymbol{x}_1) \le \cdots \le f(\boldsymbol{x}_{n+1})$. These subscripts indicate the vertices' ranking within the current simplex, not the iteration number. Write $f_i = f(\boldsymbol{x}_i)$, so that $\boldsymbol{x}_1$ is a best vertex and $\boldsymbol{x}_{n+1}$ is a worst vertex. The method first tries to replace the worst vertex while keeping the other n vertices. Their centroid $\boldsymbol{c}$ is their average; in two dimensions, it is the midpoint of the edge joining the two retained vertices. With the usual coefficients, the trial points are

$$\begin{aligned} \boldsymbol{c} &= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i, & \boldsymbol{x}_r &= 2\boldsymbol{c} - \boldsymbol{x}_{n+1}, \\ \boldsymbol{x}_e &= \boldsymbol{c} + 2(\boldsymbol{x}_r - \boldsymbol{c}), & \boldsymbol{x}_o &= \boldsymbol{c} + \tfrac{1}{2}(\boldsymbol{x}_r - \boldsymbol{c}), \\ \boldsymbol{x}_{\text{in}} &= \boldsymbol{c} + \tfrac{1}{2}(\boldsymbol{x}_{n+1} - \boldsymbol{c}). \end{aligned} \tag{4.8}$$

The first trial is a *reflection*. Since $\boldsymbol{x}_r - \boldsymbol{c} = -(\boldsymbol{x}_{n+1} - \boldsymbol{c})$, it lies the same distance from the centroid as the worst vertex, but on the opposite side. The reasoning is that the retained vertices have lower values, so continuing from the worst vertex through their centroid is a promising direction to test. This reasoning uses comparisons of function values; it does not establish that the reflected point must improve the objective.

If reflection is better than even the current best vertex, the method tests an *expansion*: $\boldsymbol{x}_e$ lies twice as far from the centroid in the reflected direction. If reflection is disappointing, the method tests a smaller replacement. An *outside contraction* lies halfway between the centroid and the reflected point, while an *inside contraction* lies halfway between the centroid and the old worst point. If the appropriate contraction also fails its comparison test, the method *shrinks* the entire simplex toward the best vertex. Thus a successful exploration can enlarge the region being sampled, while repeated unsuccessful trials make the search more local.

Algorithm: Nelder–Mead minimization

The following algorithm uses reflection coefficient 1, expansion coefficient 2, contraction coefficient $1/2$, and shrinkage coefficient $1/2$. Its acceptance comparisons follow [11].

Inputs. An objective $f : \mathbb{R}^n \to \mathbb{R}$, affinely independent initial vertices $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}$, positive tolerances ε_x and ε_f, and a maximum number $N_{\max} \ge n + 1$ of function evaluations.

Output. The best point evaluated, its objective value, the number of function evaluations, and the reason for stopping.

Function-evaluation rule. Let N_f count function evaluations. Before every new evaluation, check whether $N_f = N_{\max}$. If so, stop and return the best point evaluated so far, with termination reason "evaluation budget exhausted." Otherwise evaluate the objective, increase N_f by one, and update the stored best point if the new value is smaller. The first evaluation initializes this record. This rule applies to every trial point and every point evaluated during shrinkage.

1. **Initialize.** Set $N_f = 0$ and evaluate $f_i = f(\boldsymbol{x}_i)$ for $i = 1, \ldots, n+1$, following the function-evaluation rule above. Store these values so that unchanged vertices do not require reevaluation.
2. **Order the vertices and test for termination.** Reorder the vertices and their stored values so that

$$f_1 \le f_2 \le \cdots \le f_{n+1}. \tag{4.9}$$

   Thus $\boldsymbol{x}_1$ is a best vertex, $\boldsymbol{x}_n$ is a second-worst vertex, and $\boldsymbol{x}_{n+1}$ is a worst vertex. Preserve the current relative order of vertices with equal function values. An accepted replacement occupies the old worst vertex's position before this sorting step, so it follows surviving vertices with the same value. Initially, ties follow the input order.

Compute the simplex size and objective spread:

$$d = \max_{2 \le j \le n+1} \|\boldsymbol{x}_j - \boldsymbol{x}_1\|_2, \qquad \delta = \max_{2 \le j \le n+1} |f_j - f_1|. \tag{4.10}$$

If both $d \le \varepsilon_x$ and $\delta \le \varepsilon_f$, stop and return $\boldsymbol{x}_1$, f_1, and N_f, with termination reason "simplex tolerances satisfied." Otherwise continue.

3. **Compute the centroid and evaluate reflection.** Average all vertices except the worst:

$$\boldsymbol{c} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i. \tag{4.11}$$

Reflect the worst vertex through this centroid:

$$\boldsymbol{x}_r = \boldsymbol{c} + (\boldsymbol{c} - \boldsymbol{x}_{n+1}) = 2\boldsymbol{c} - \boldsymbol{x}_{n+1}. \tag{4.12}$$

Evaluate $f_r = f(\boldsymbol{x}_r)$.

4. **Choose exactly one of the following cases.**

   (a) **Reflection improves on the best vertex: $\boldsymbol{f_r < f_1}$.** Try moving farther in the reflected direction:

$$\boldsymbol{x}_e = \boldsymbol{c} + 2(\boldsymbol{x}_r - \boldsymbol{c}), \qquad f_e = f(\boldsymbol{x}_e). \tag{4.13}$$

   If $f_e < f_r$, replace the worst vertex and its stored value by $\boldsymbol{x}_e$ and f_e. Otherwise replace them by $\boldsymbol{x}_r$ and f_r. Return to Step 2.

   (b) **Reflection lies between the best and second-worst values: $\boldsymbol{f_1 \le f_r < f_n}$.** Replace the worst vertex and its stored value by $\boldsymbol{x}_r$ and f_r. Return to Step 2.

   (c) **Reflection improves on the worst value but not the second-worst: $\boldsymbol{f_n \le f_r < f_{n+1}}$.** Try an outside contraction, halfway from the centroid toward the reflected point:

$$\boldsymbol{x}_o = \boldsymbol{c} + \frac{1}{2}(\boldsymbol{x}_r - \boldsymbol{c}), \qquad f_o = f(\boldsymbol{x}_o). \tag{4.14}$$

   If $f_o \le f_r$, replace the worst vertex and its stored value by $\boldsymbol{x}_o$ and f_o, and return to Step 2. Otherwise proceed to Step 5.

   (d) **Reflection does not improve on the worst value: $\boldsymbol{f_r \ge f_{n+1}}$.** Try an inside contraction, halfway from the centroid toward the old worst vertex:

$$\boldsymbol{x}_{\text{in}} = \boldsymbol{c} + \frac{1}{2}(\boldsymbol{x}_{n+1} - \boldsymbol{c}), \qquad f_{\text{in}} = f(\boldsymbol{x}_{\text{in}}). \tag{4.15}$$

   If $f_{\text{in}} < f_{n+1}$, replace the worst vertex and its stored value by $\boldsymbol{x}_{\text{in}}$ and f_{in}, and return to Step 2. Otherwise proceed to Step 5.

5. **Shrink after a failed contraction.** Keep the best vertex $\boldsymbol{x}_1$ and its stored value f_1 unchanged. For each $j = 2, \ldots, n+1$, replace the vertex by

$$\boldsymbol{x}_j \leftarrow \boldsymbol{x}_1 + \frac{1}{2}(\boldsymbol{x}_j - \boldsymbol{x}_1), \tag{4.16}$$

and evaluate its new value $f_j = f(\boldsymbol{x}_j)$. Keep the old best vertex in the first position until sorting, so it remains first if tied for the smallest value. Return to Step 2.

Every accepted reflection, expansion, or contraction replaces exactly one vertex. Shrinkage occurs only after a failed contraction and moves all vertices except the best. Satisfying the simplex tolerances reports a small simplex with similar function values; it does not, by itself, prove stationarity.

For the cited variant's tie rule, preserve the relative order of surviving vertices and insert a newly accepted point after all surviving vertices with the same objective value. After a shrink, keep the old best vertex first if it is tied for the smallest value. This detail is relevant in symmetric examples, where several trial points can have exactly the same value.

Nelder–Mead variants can use different acceptance thresholds. One variant accepts an expansion when it beats the *old best* value and accepts a contraction when it beats the *old worst* value [5, pp. 580–581]. The algorithm above instead requires an expansion to beat the *reflection*, and its outside-contraction rule also compares to the reflection. These choices matter when reproducing a numerical trajectory. Different coefficient conventions can nevertheless describe the same geometric expansion point.

The initial simplex requires $n + 1$ function evaluations. With stored vertex values, an iteration that accepts reflection immediately needs one new evaluation. Testing expansion or contraction adds a second evaluation, even if a tested expansion is rejected and the reflection is accepted instead. A failed contraction followed by shrinkage needs those two evaluations and another n evaluations for the moved vertices. No gradients are evaluated. Therefore, a count of simplex iterations is not a count of function evaluations, and it is not directly comparable to a count of gradient-descent or BFGS iterations.

Proposition 14 (The best sampled value does not increase)**.** Let $f_{\text{best}}^{(k)}$ be the smallest function value among the simplex vertices after k Nelder–Mead iterations. For the algorithm above,

$$f_{\text{best}}^{(k+1)} \leq f_{\text{best}}^{(k)}. \tag{4.17}$$

If these values are bounded below, they converge to a finite limit. This statement alone does not identify that limit as a minimum of f.

Proof. In every accepted reflection, expansion, or contraction, only the worst vertex is replaced. The previous best vertex remains in the simplex, so taking the minimum over the new simplex cannot give a larger value. A shrink also retains the previous best vertex exactly, giving the same conclusion. Thus the sequence of best values is nonincreasing. If it is bounded below, the monotone convergence property for real sequences gives a finite limit. Nothing in this argument says that the retained best vertex has zero gradient or that every nearby direction has been explored. ■

Example 15 (One triangle update compared with gradient descent)**.** Consider

$$f(x, y) = (x - 1)^2 + (y + 1)^2, \qquad \boldsymbol{x}_\star = (1, -1)^\mathsf{T}, \qquad f(\boldsymbol{x}_\star) = 0. \tag{4.18}$$

Its circular level curves make it possible to see exactly what information each method uses. Start Nelder–Mead with the vertices

$$\boldsymbol{x}_1 = (0, 0)^\mathsf{T}, \quad \boldsymbol{x}_2 = (2, 0)^\mathsf{T}, \quad \boldsymbol{x}_3 = (0, 2)^\mathsf{T}, \qquad (f_1, f_2, f_3) = (2, 2, 10). \tag{4.19}$$

Figure 9 shows the objective surface and the heights of the three initial vertices. The two vertices with value 2 lie on the same horizontal level curve, while the worst vertex lies higher, at value 10. The circular level curves surround the minimum at $(1, -1)^\mathsf{T}$; this geometry will explain why the gradient-descent comparison is particularly simple.

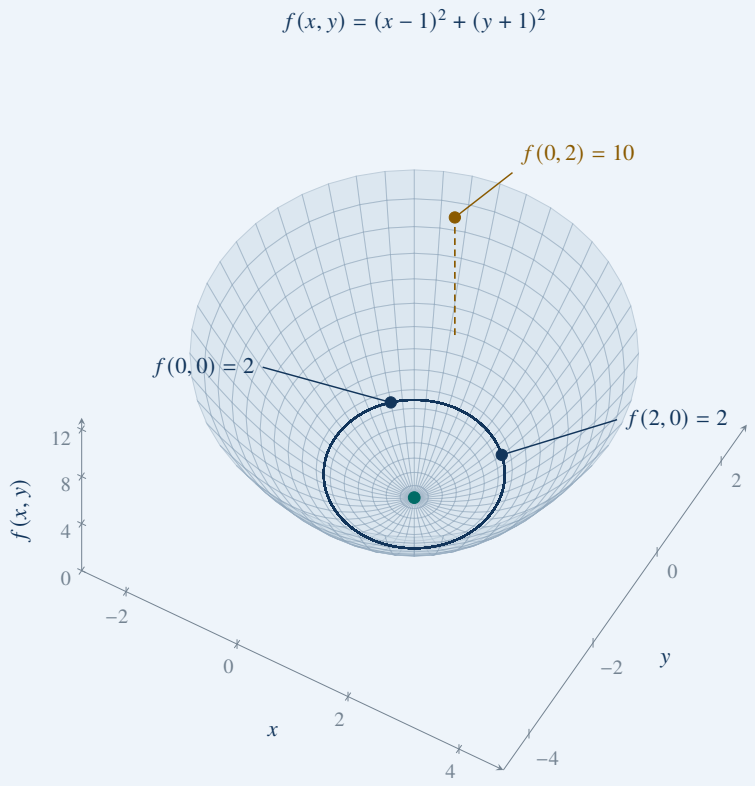


FIGURE 9: The objective surface $f(x, y) = (x - 1)^2 + (y + 1)^2$ for the triangle-update example. The initial Nelder–Mead vertices are lifted to their function values: $f(0, 0) = f(2, 0) = 2$ and $f(0, 2) = 10$. The blue curve is the level $f = 2$, and the teal point marks the minimum $(1, -1)^\mathsf{T}$, with value 0. The horizontal coordinates use equal unit scales; the vertical scale is compressed.

Keep the listed order to break the tie. The centroid of the two retained vertices is $\boldsymbol{c} = (1, 0)^\mathsf{T}$. Reflection gives $\boldsymbol{x}_r = (2, -2)^\mathsf{T}$, with $f_r = 2$. Although this improves on the worst value 10, it ties the second-worst value 2. Hence case 3, rather than case 1, applies: the method evaluates the outside contraction

$$\boldsymbol{x}_o = (1, 0)^\mathsf{T} + \tfrac{1}{2}\left[(2, -2)^\mathsf{T} - (1, 0)^\mathsf{T}\right] = (3/2, -1)^\mathsf{T}, \qquad f_o = 1/4. \tag{4.20}$$

Because $1/4 \le 2 = f_r$, the contraction is accepted. The new simplex consists of $(0, 0)^\mathsf{T}$, $(2, 0)^\mathsf{T}$, and $(3/2, -1)^\mathsf{T}$, and its best function value has decreased from 2 to $1/4$. The iteration needed two new function values after the three initialization values; no derivatives were used.

Now start gradient descent at the same best point $\boldsymbol{x}_0 = (0, 0)^\mathsf{T}$. Since

$$\nabla f(x, y) = \begin{pmatrix} 2(x - 1) \\ 2(y + 1) \end{pmatrix}, \qquad -\nabla f(\boldsymbol{x}_0) = \begin{pmatrix} 2 \\ -2 \end{pmatrix}, \tag{4.21}$$

the points on its search line have coordinates $(2\alpha, -2\alpha)$. Along this line,

$$f(2\alpha, -2\alpha) = 2(2\alpha - 1)^2, \tag{4.22}$$

which is minimized at $\alpha = 1/2$. The resulting gradient-descent point is exactly $(1, -1)^\mathsf{T}$. BFGS initialized with $H_0 = \mathbb{1}$ takes the same first direction and also reaches the minimizer with this exact line search. Newton's method uses $\nabla^2 f = 2\mathbb{1}$ and reaches the same point with its unit step.
Figure 10 shows the distinction: the simplex method finds an improved point by comparing trial values, while the gradient points directly toward the center of these circular contours. Derivative information is especially effective for this particular objective. Nelder–Mead remains useful when that information is unavailable, but giving it only function values does not give it the same information as an analytic gradient and an exact line minimization.

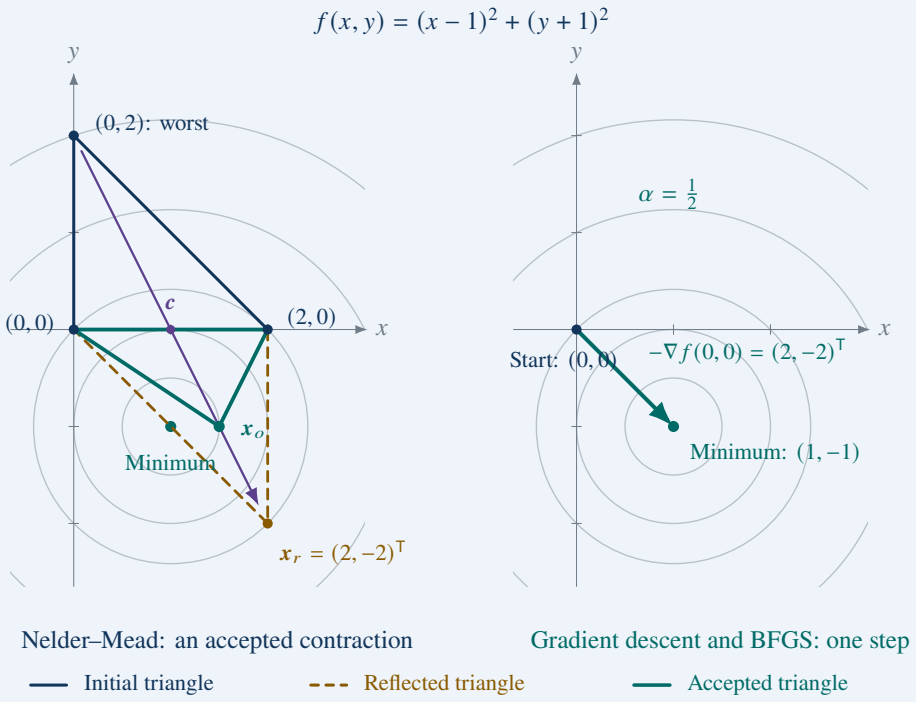


FIGURE 10: A Nelder–Mead update and a gradient-descent step for $f(x, y) = (x - 1)^2 + (y + 1)^2$. The reflected point has value 2, so the accepted outside contraction has value $1/4$. From the same best starting point, gradient descent with exact line search reaches the minimizer; BFGS with $H_0 = \mathbb{1}$ takes the same step.

Circular contours make a gradient step particularly simple. Elongated contours create a different comparison: the gradient can point across a valley, BFGS can learn how to correct for unequal curvature, and Nelder–Mead can change its shape using only value comparisons. Using the same objective and starting region lets us compare these mechanisms directly.

Example 16 (Three methods using different information)**.** Return to the quadratic

$$f(x_1, x_2) = \tfrac{1}{2}(9x_1^2 + x_2^2), \qquad \boldsymbol{x}_\star = (0, 0)^\mathsf{T}, \qquad \boldsymbol{x}_0 = (1, 9)^\mathsf{T}. \tag{4.23}$$

This is the same objective surface shown in Figure 6. The three methods below explore this elongated bowl using different information.

Use the gradient-descent and BFGS iterations from Section 3, with exact line searches and $H_0 = \mathbb{1}$ for BFGS. For Nelder–Mead, use the stated algorithm and the initial simplex

$$(1, 9)^\mathsf{T}, \qquad (3/2, 9)^\mathsf{T}, \qquad (1, 19/2)^\mathsf{T}. \tag{4.24}$$

The first vertex is the best initial vertex, so all three methods begin with best value $f(\boldsymbol{x}_0) = 45$. The two additional simplex vertices specify the region initially sampled by Nelder–Mead; changing this initial size or orientation can change its trajectory.

For gradient descent, the exact line-search parameter is $1/5$ at every iteration, and the iterates are

$$\boldsymbol{x}_k = \begin{pmatrix} (-4/5)^k \\ 9(4/5)^k \end{pmatrix}, \qquad f(\boldsymbol{x}_k) = 45(16/25)^k. \tag{4.25}$$

The first coordinate alternates in sign, producing the zigzag across the valley. BFGS takes the same first step but uses its gradient difference to change the next direction; as calculated in Section 3, its second exact line search reaches the minimizer. These two steps are a consequence of this quadratic example and its exact line searches, not a general two-iteration guarantee for BFGS.

To make the numerical comparison precise, record the first iteration at which the current value, or best simplex value, is at most 10^{-6} times the initial value 45. The target is therefore 4.5×10^{-5}. The specified runs give

Method	Iterations to the target	Value when first attained
Gradient descent	31	4.413587×10^{-5}
BFGS	2	0
Nelder–Mead	23	2.987560×10^{-5}

(4.26)

For gradient descent, the count follows directly from $45(16/25)^k \leq 4.5 \times 10^{-5}$. For Nelder–Mead, applying the stated reflection, contraction, expansion, and tie rules gives the first successful best vertex approximately $(-0.001909633, -0.005189499)^\mathsf{T}$. This run uses 47 objective evaluations including the three initialization evaluations, and no gradient evaluations. Its best value after 22 updates is approximately 9.282917×10^{-5}, so the target has not yet been met at that point. Here the known minimum $f(\boldsymbol{x}_\star) = 0$ lets us use function error to compare progress; an unknown optimization problem would not normally provide that stopping certificate in advance.

Figure 11 follows the best sampled vertex for Nelder–Mead and the current iterate for the derivative-based methods. A horizontal portion of the Nelder–Mead curve means that the simplex changed without producing a new best vertex; it does not mean that the algorithm did no work. Nelder–Mead adjusts its simplex to the valley without evaluating gradients, but it does not use gradient differences to build an inverse-Hessian approximation.

The comparison concerns progress per iteration for these specified runs. Nelder–Mead uses one or more new function values per iteration; gradient descent and BFGS use gradients, and their exact line searches here use the known quadratic formula. On a general objective those line searches also have a computational cost. Consequently, the plot explains the different search mechanisms; it does not establish a universal ordering of run times or function-evaluation costs.

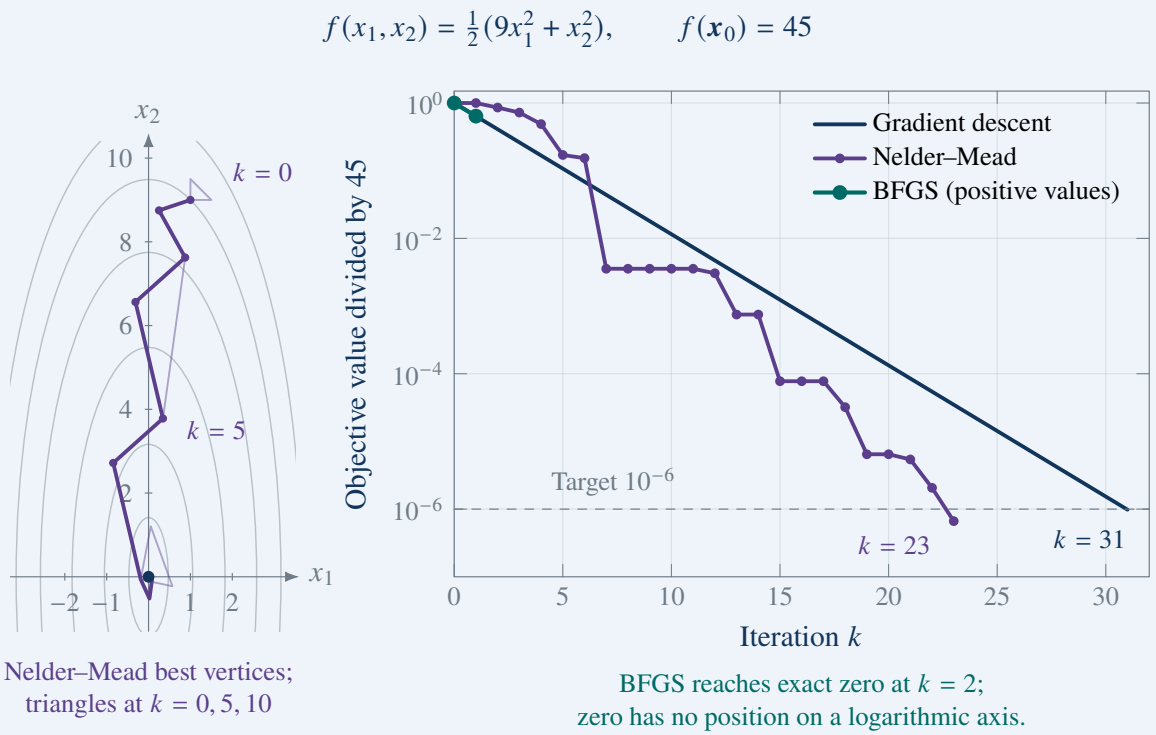


FIGURE 11: Gradient descent, BFGS, and Nelder–Mead on $f(x_1, x_2) = \frac{1}{2}(9x_1^2 + x_2^2)$, starting at $(1, 9)^\mathsf{T}$ with the additional Nelder–Mead vertices specified in the example. The derivative-based methods use exact line searches, and BFGS begins with $H_0 = \mathbb{1}$. Paths show current iterates or best simplex vertices; the value comparison counts algorithm iterations, whose computational costs differ between methods.

5 Simulated annealing: allowing uphill moves

At a strict local minimum, every sufficiently small nonzero displacement increases the objective, even when a better minimum exists elsewhere. A method that insists on decreasing the objective can therefore become confined to the region around that minimum. Randomized search permits some uphill moves to explore other regions. Simulated annealing gradually reduces their acceptance probability by lowering a *temperature*; the name comes from controlled cooling in a physical system, with objective value playing the role of energy [12].

The search distinguishes its *current point*, which determines where the next proposal is generated and may become worse, from the *best point found*, which is retained for the final answer. The two rules below share these ideas and use exponential preferences for lower objective values. We present them as branches of one algorithm: normalized selection chooses among several candidates, whereas Metropolis accepts or rejects one proposal.

For *normalized candidate selection*, generate m candidates $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$ near the current point $\boldsymbol{x}$ [5, pp. 581–582]. Write $b = \min_i f(\boldsymbol{u}_i)$. If $b < f(\boldsymbol{x})$, choose a best candidate. Otherwise select candidate i with

probability

$$p_i = \frac{\exp(-\alpha[f(\boldsymbol{u}_i) - b])}{\sum_{j=1}^{m} \exp(-\alpha[f(\boldsymbol{u}_j) - b])}, \qquad \alpha > 0. \tag{5.1}$$

These are the normalized weights $\exp(\alpha[f(\boldsymbol{x}) - f(\boldsymbol{u}_i)])$, after canceling a common factor. Subtracting b keeps all exponentials at most one and at least one equal to one, avoiding an all-zero denominator from underflow. Larger α favors lower-valued candidates more strongly. The parameter α is fixed, and a candidate is always selected, even if all candidates are worse than the current point.

For *Metropolis acceptance*, generate one proposal $\boldsymbol{y}$ and write $\Delta = f(\boldsymbol{y}) - f(\boldsymbol{x})$ for the proposed change in objective value. At temperature $T > 0$, accept with probability [13]

$$A_T(\boldsymbol{x}, \boldsymbol{y}) = \min\{1, \exp(-\Delta/T)\}. \tag{5.2}$$

Every improving or equal-valued proposal is accepted. An uphill proposal is accepted with probability $\exp(-\Delta/T)$; otherwise the current point is retained. Larger increases and lower temperatures make acceptance less likely. The temperature has the same units as the objective, so Δ/T is dimensionless.

Both branches below generate candidates by adding independent increments uniformly distributed in $[-\sigma, \sigma]^n$, where $\sigma > 0$ controls the proposal size. This gives the symmetric proposal distribution used with Metropolis: the forward and reverse proposal densities are equal. In the Metropolis branch only, hold the temperature fixed for M trials, then multiply it by $r \in (0, 1)$. Stage j therefore uses $T_j = T_0 r^j$, where j counts completed blocks of M trials [12].

Algorithm: Randomized search with a choice of uphill-move rule

Inputs. An objective $f : \mathbb{R}^n \to \mathbb{R}$, an initial point $\boldsymbol{x}_0$, a proposal scale $\sigma > 0$, a positive integer objective-evaluation budget $N_{\max}$, and one rule: **normalized selection**, with batch size $m \geq 1$ and fixed $\alpha > 0$; or **Metropolis acceptance**, with $T_0 > 0$, $r \in (0, 1)$, and $M \geq 1$ trials per temperature. The counts m and M are integers. Fix the random seed for a reproducible run.

Output. The best point evaluated, its objective value, the evaluation count, and the termination reason.

1. **Initialize.** Set $\boldsymbol{x} \leftarrow \boldsymbol{x}_0$, evaluate $f_{\text{current}} \leftarrow f(\boldsymbol{x}_0)$, and set $N_f \leftarrow 1$. Store $\boldsymbol{x}_{\text{best}} \leftarrow \boldsymbol{x}$ and $f_{\text{best}} \leftarrow f_{\text{current}}$. Set the number of candidates per iteration to $q = m$ for normalized selection or $q = 1$ for Metropolis. For Metropolis, also set $T \leftarrow T_0$ and $\ell \leftarrow 0$.

2. **Check the budget.** If $N_f + q > N_{\max}$, return $\boldsymbol{x}_{\text{best}}$, f_{best}, and N_f, with termination reason "insufficient budget for the next iteration." Otherwise continue.

3. **Propose and evaluate.** Draw independent V_{ij} uniformly from $(0, 1)$ and form

$$\boldsymbol{u}_i = \boldsymbol{x} + \sigma(2V_{i1} - 1, \ldots, 2V_{in} - 1)^{\mathsf{T}}, \qquad i = 1, \ldots, q. \tag{5.3}$$

   Evaluate and store all $f(\boldsymbol{u}_i)$, and set $N_f \leftarrow N_f + q$. Let $b = \min_i f(\boldsymbol{u}_i)$, and let $i_\star$ be the first index attaining b.

4. **Retain the best point.** If $b < f_{\text{best}}$, set $\boldsymbol{x}_{\text{best}} \leftarrow \boldsymbol{u}_{i_\star}$ and $f_{\text{best}} \leftarrow b$.

5. **Branch according to the chosen rule.**

   Normalized selection: If $b < f_{\text{current}}$, set $J = i_\star$. Otherwise compute p_i from (5.1), draw an independent U uniformly from $(0, 1)$, and choose the first J with $U \leq \sum_{i=1}^{J} p_i$. Set $\boldsymbol{x} \leftarrow \boldsymbol{u}_J$ and $f_{\text{current}} \leftarrow f(\boldsymbol{u}_J)$, using the stored value.

Metropolis acceptance: Set $\Delta = f(\boldsymbol{u}_1) - f_{\text{current}}$ and draw an independent U uniformly from $(0, 1)$ on every trial. If $\Delta \le 0$, or if $\Delta > 0$ and $\log U < -\Delta/T$, set $\boldsymbol{x} \leftarrow \boldsymbol{u}_1$ and $f_{\text{current}} \leftarrow f(\boldsymbol{u}_1)$. Otherwise retain the current point and value.

6. **Record, cool if applicable, and repeat.** Record the current and best values if a history is needed. For Metropolis, increment $\ell \leftarrow \ell + 1$ after every trial, including rejection; if $\ell = M$, set $T \leftarrow rT$ and $\ell \leftarrow 0$. For normalized selection, keep α fixed. Return to Step 2.

Stored objective values are reused: normalized selection costs m new evaluations per iteration, whereas Metropolis costs one, even on rejection. The best value never increases, although the current value may. For normalized selection, set the last cumulative probability to one in floating-point arithmetic. For Metropolis, the logarithmic acceptance test is equivalent to $U < \exp(-\Delta/T)$ when $\Delta > 0$ and avoids exponential underflow. Drawing U on every Metropolis trial keeps the random-draw sequence consistent across runs. The budget is a stopping condition, not a certificate of global optimality.

For the Metropolis branch, choose a temperature scale by specifying that a representative uphill increase $\Delta_{\text{typ}} > 0$ should initially be accepted with probability $a \in (0, 1)$. Then

$$T_0 = -\frac{\Delta_{\text{typ}}}{\log a}. \tag{5.4}$$

For example, an increase of 2 is accepted with probability approximately 0.819 at $T = 10$, 0.135 at $T = 1$, and 0.0000454 at $T = 0.2$. Adding a constant to f changes no acceptance probabilities; multiplying f by a positive constant requires multiplying all temperatures by the same constant.

The proposal scale σ and cooling schedule control different aspects of exploration. Small proposals may require many moves to reach another basin; large proposals may produce increases that are rarely accepted. Taking r closer to one or increasing M gives more trials at higher temperatures. Cooling too quickly can suppress uphill moves before the search leaves its initial basin. These choices must be balanced against the evaluation budget; a geometric cooling schedule alone does not guarantee a global minimum in a finite run.

Example 17. Suppose the current objective value is 3 and the candidate values are 4 and 6. Normalized selection assigns them probabilities $1/(1+\mathrm{e}^{-2\alpha})$ and $\mathrm{e}^{-2\alpha}/(1+\mathrm{e}^{-2\alpha})$. If Metropolis instead proposes either candidate with probability $1/2$, the probabilities of moving to them are $\frac{1}{2}\,\mathrm{e}^{-1/T}$ and $\frac{1}{2}\,\mathrm{e}^{-3/T}$, with the remainder assigned to staying at value 3. Figure 12 compares the exponential preferences using $\alpha = 1/T$: normalized selection must move, whereas Metropolis can reject. This common plotting scale does not change the algorithm's fixed α or cooling schedule.

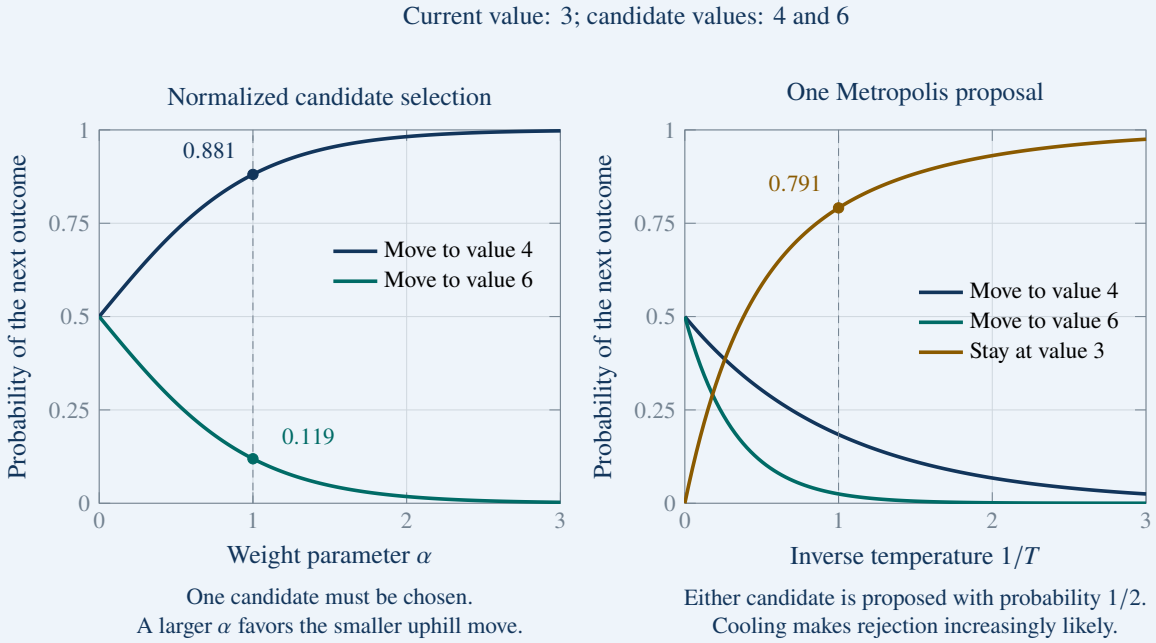


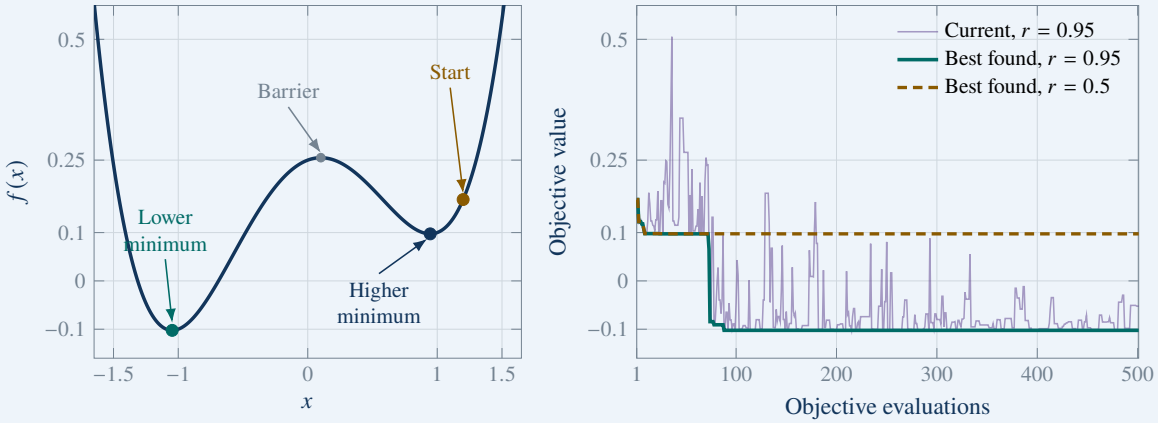
FIGURE 12: Normalized selection always chooses a candidate; Metropolis can retain the current point. Each candidate is proposed with probability $1/2$ in the right panel. The horizontal parameters are identified by $\alpha = 1/T$; zero denotes the limit $\alpha \downarrow 0$ or $T \to \infty$.

Example 18. Consider the tilted double-well objective

$$f(x) = \frac{1}{4}(x^2 - 1)^2 + \frac{1}{10}x. \tag{5.5}$$

The left minimum is lower, but the starting point $x_0 = 1.2$ lies near the right well. Figure 13 compares two annealing runs with $\sigma = 0.6$, $T_0 = 0.2$, $M = 10$, and 500 proposals (501 objective evaluations including initialization). Both use Python's `random.Random(0)`, drawing the proposal uniform first and the acceptance uniform second on each trial; only the cooling factor changes, from $r = 0.95$ to $r = 0.5$. The local runs shown for comparison also start at 1.2: gradient descent uses step size 0.2; BFGS uses $H_0 = 1$ and Armijo backtracking with $c_1 = 10^{-4}$, trial step 1, and halving; Nelder–Mead starts from $[1.2, 1.4]$. BFGS updates H only when $sy > 10^{-12}|s||y|$. The derivative-based runs stop at $|f'(x)| \le 10^{-8}$, and Nelder–Mead uses segment-length and value-spread tolerances 10^{-8} and 10^{-12}, respectively. Each has a cap of 501 objective evaluations; the cap does not imply equal work.
For these runs, the local methods and rapid cooling remain in the higher well, whereas slower cooling discovers the lower well. The best-found value is retained even when the current value subsequently increases. This illustrates the role of exploration and cooling, not a guarantee that every random run finds the lower minimum.

$$f(x) = \tfrac{1}{4}(x^2 - 1)^2 + 0.1x, \qquad x_0 = 1.2$$



The specified gradient-descent, BFGS, and Nelder–Mead runs finish at the higher minimum.

Same random seed, two cooling schedules. An uphill move changes the current value, not the best found.

FIGURE 13: Left: the objective, its two minima, and the intervening barrier. Right: slower cooling finds the lower well in the displayed run; rapid cooling remains in the right well. The current value can increase, while the best-found value cannot. The horizontal axis counts objective evaluations, including initialization.