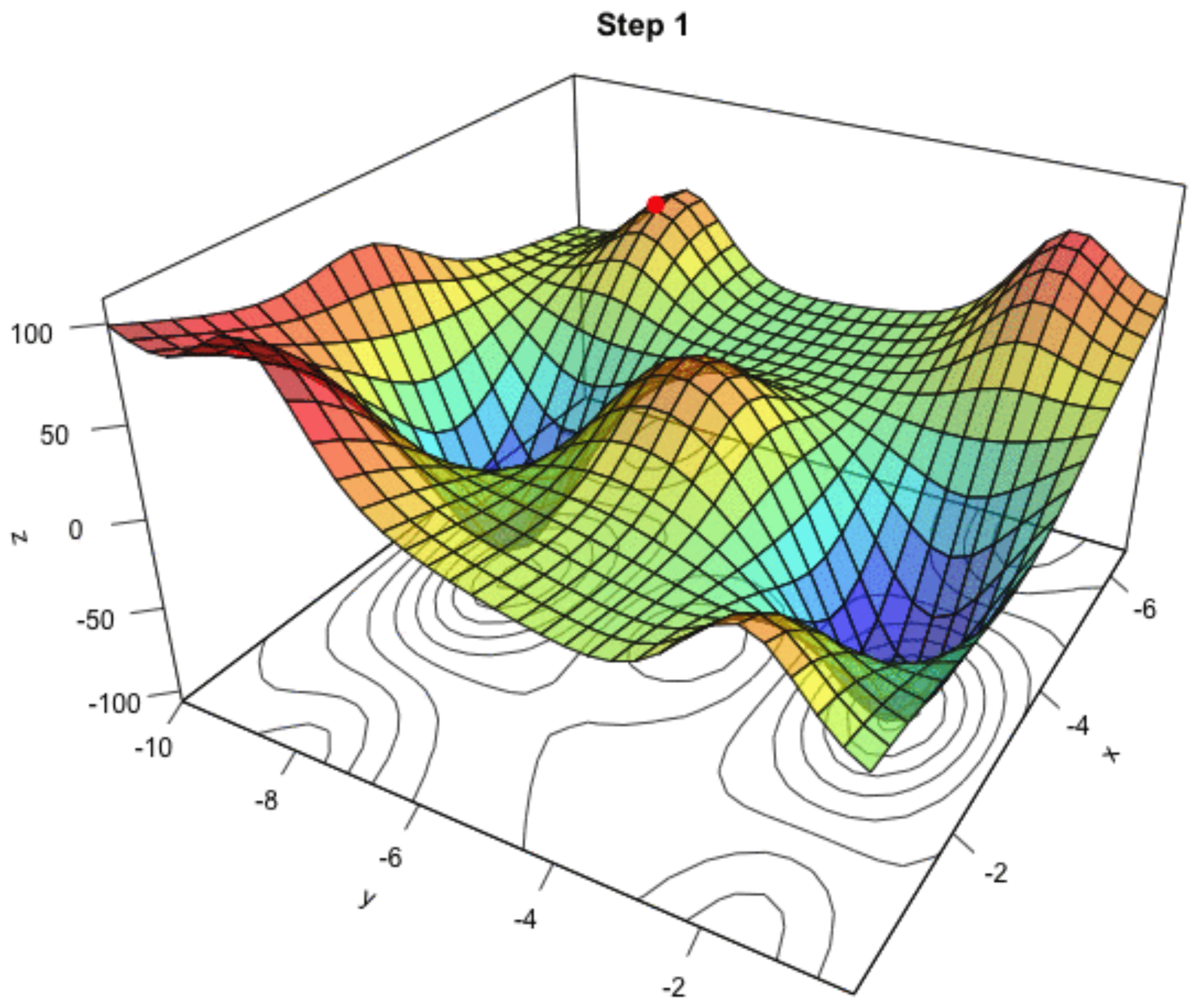
Lec 11: Unconstrained Optimization II

15-369/669/769: Numerical Computing

Instructor: Minchen Li

Multivariable Strategies

Intuition



- In each step:
  - Pick a descent direction
  - Decide a step size

Image source: https://medium.com/@pau.moskwa/eli5-ml-optmization-algorithms-9ba6305b1190

Table of Content

Table of Content

Gradient Descent

Gradient Descent with Line Search

- **Goal:** Minimize differentiable $f : \mathbb{R}^n \to \mathbb{R}$ by iteratively updating:

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

  where t_k is determined by a **line search**.

- **Descent direction:** $-\nabla f(\mathbf{x}_k)$ ensures $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$ for small enough $t_k > 0$.

- **Line search formulation:** define one-dimensional function

$$g(t) = f(\mathbf{x}_k - t\nabla f(\mathbf{x}_k)),$$

  and choose t_k to approximately minimize $g(t)$.

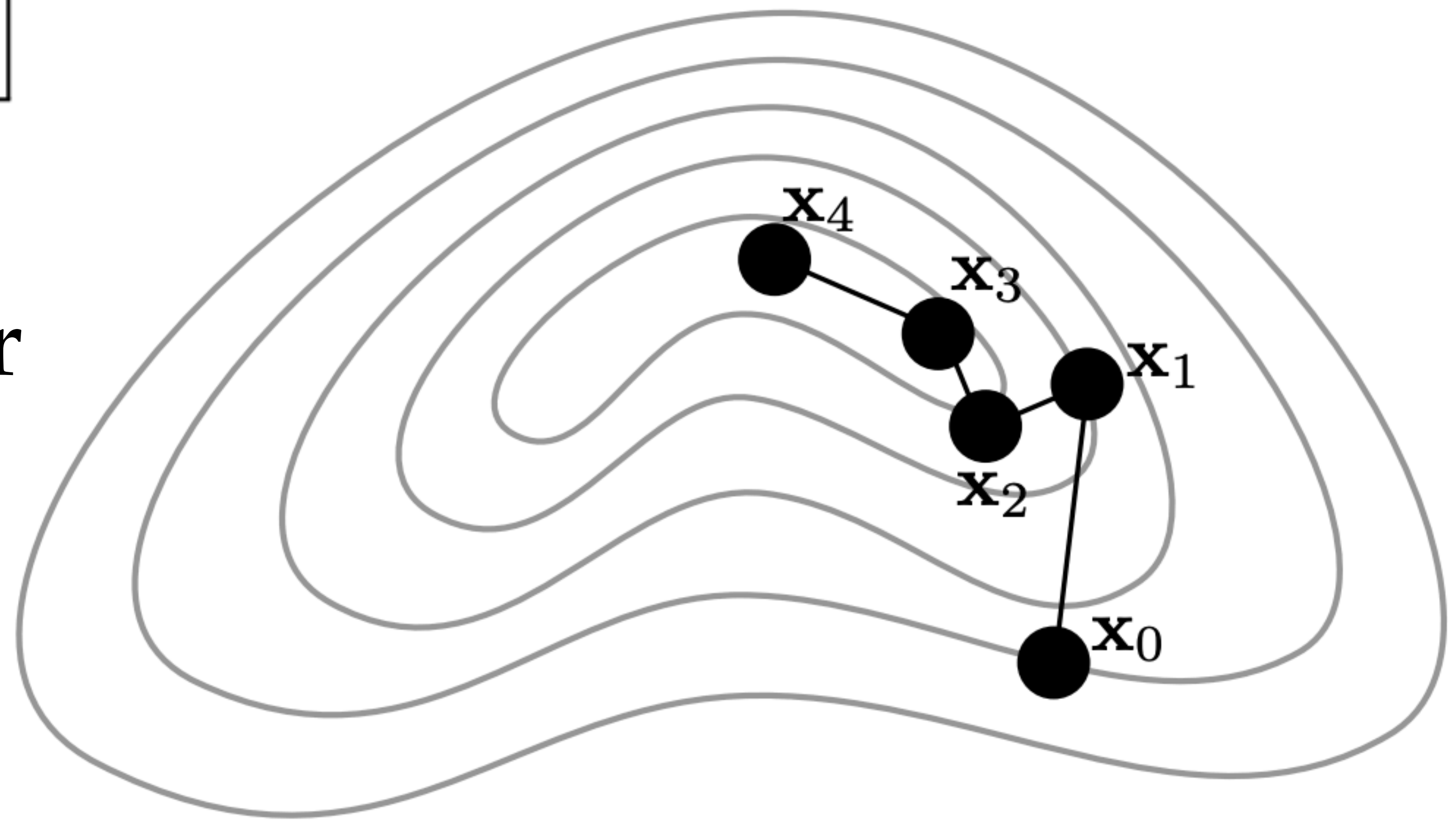
- **Convergence:** If f bounded below and differentiable, gradient descent decreases $f(\mathbf{x}_k)$.

Gradient Descent

Pseudo-Code

function GRADIENT-DESCENT($f(\mathbf{x}), \mathbf{x}_0$)
  $\mathbf{x} \leftarrow \mathbf{x}_0$
  for $k \leftarrow 1, 2, 3, \ldots$
    DEFINE-FUNCTION($g(t) := f(\mathbf{x} - t\nabla f(\mathbf{x}))$)
    $t^* \leftarrow$ LINE-SEARCH($g(t), t \geq 0$)
    $\mathbf{x} \leftarrow \mathbf{x} - t^* \nabla f(\mathbf{x})$ ▷ Update estimate of minimum
    if $\|\nabla f(\mathbf{x})\|_2 < \varepsilon$ **then**
      return $x^* = \mathbf{x}$

- The gradient points perpendicular to the level sets of the function:



Gradient Descent

Inexact Line Search

- **Inexact line search:** Instead of exact minimization, can iteratively halve t_k starting from 1 until it satisfy the **Wolfe conditions:**

$$
\begin{aligned}
f(\mathbf{x}_k + t_k \mathbf{d}_k) &\leq f(\mathbf{x}_k) + c_1 t_k \mathbf{d}_k^T \nabla f(\mathbf{x}_k), \\
-\mathbf{d}_k^T \nabla f(\mathbf{x}_k + t_k \mathbf{d}_k) &\geq -c_2 \mathbf{d}_k^T \nabla f(\mathbf{x}_k),
\end{aligned}
\qquad \text{where } 0 < c_1 < c_2 < 1.
$$

- Again, these ensure that the gradient norm converges to zero starting from **arbitrary $\mathbf{x}_0$** – a property called **global convergence**.

- In practice, can use a parameter-free, simplified version to just ensure energy decrease:

$$f(\mathbf{x}_k + t_k \mathbf{d}_k) \leq f(\mathbf{x}_k),$$

  which still guarantees global convergence.

Gradient Descent

Example: Violation of Wolfe Conditions

- Consider $f(x, y) = \frac{1}{2}(x^2 + y^2)$, starting from $(x_0, y_0) = (1, 0)$.
- Gradient: $\nabla f(x, y) = (x, y)$.
- Update rule with step size t_k:

$$x_{k+1} = (1 - t_k)x_k,$$
$$y_{k+1} = (1 - t_k)y_k.$$

- If we take $t_k = 2$, gradient descent diverges: $(x_k, y_k) = (1, 0), (-1, 0), (1, 0), \; ....$
- If we take $t_k = (1/2)^k$, $(x_k, y_k) \to (0.288788..., 0)$, which is not the minimum at $(0, 0)$.
- Poor choices of t_k can lead to convergence problems for gradient descent.

Gradient Descent

Gradient Descent without Line Search

- **Motivation:** Line search can be computationally expensive – each iteration requires multiple function (and gradient) evaluations.
- **Constant step size:** fix $t_k = t$, also called the *learning rate* in machine learning.
- **Trade-off:**
  - Large t: fast progress but possible oscillation or divergence.
  - Small t: guaranteed descent but slow convergence.
- If f is convex, twice differentiable, and ∇f is L-Lipschitz: $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\|_2$, and $t \leq 1/L$, then $f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2}{2tk}$.
  - **Implication:** Constant step size yields $O(1/k)$ convergence rate for convex functions.

Gradient Descent

Learning Rate Schedule

- For more general f, choose t_k as a function of iteration k, not of current gradient.
- **Two key desiderata:**
  - Diminishing step size: $t_k \to 0$ as $k \to \infty$ (avoids overshooting).
  - Infinite travel: $\sum_{i=0}^{k} t_i \to \infty$ as $k \to \infty$ (ensures progress can be made).
- Example: *Harmonic sequence* $t_k = 1/k$ satisfies both.
- Practical schedules:
  - Step decay: reduce t_k by factor every few epochs.
  - Cyclic or warm restarts: alternate between large and small steps.
- Used widely in machine learning to balance convergence speed and stability.

Table of Content

Newton's Method

Newton's Method in Multiple Variables

- **Goal:** Extend Newton's method to multivariable functions $f : \mathbb{R}^n \to \mathbb{R}$.
- **Idea:** Use both first- and second-order derivatives:
  - Gradient: $\nabla f(\mathbf{x})$; Hessian: $H_f(\mathbf{x}) = \nabla^2 f(\mathbf{x})$
- **Taylor expansion near** $\mathbf{x}_k$: $f(\mathbf{x}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\top H_f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$.
- **Optimization step:** Set gradient of RHS to zero:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [H_f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k).$$

- Converges **quadratically** when $\mathbf{x}_0$ is close to a local minimum.
- Cost: computing and storing H_f requires $O(n^2)$ memory; best for small or medium n.

Newton's Method

Practical Considerations

- **Geometric intuition:**
  - Gradient descent uses local slope – "walk downhill."
  - Newton's method uses curvature – "estimate where the valley floor is."
- **When H_f is nearly singular:**
  - Large, unstable steps possible.
  - Use a damping factor $\gamma > 0$: $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma [H_f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$.
  - Or perform a line search along direction $\mathbf{d}_k = -[H_f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$.
- **Computational cost:** Each iteration requires forming and inverting H_f. Often more expensive than gradient descent, but faster convergence.

Newton's Method

Handling Indefinite Hessians

- If H_f is **not positive definite**, local region may resemble a saddle or peak rather than a minimum, possibly leading to a non-descent search direction.

- **Remedies:**

  - Check if H_f is positive definite before using it. If not, revert to gradient descent step: $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \nabla f(\mathbf{x}_k)$.

  - Similar to Tikhonov regularization, compute $\mathbf{d}_k = -[H_f(\mathbf{x_k}) + \mu I]^{-1} \nabla f(\mathbf{x}_k)$ instead.

  - Modify H_f, e.g., project it to the nearest positive definite matrix (the **projected Newton's method**): Given the Eigendecomposition $H_f(\mathbf{x}_k) = Q \Lambda Q^T$, compute $\mathbf{d}_k = -[H_f^+(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)$, where $H_f^+(\mathbf{x}_k) = Q \Lambda^+ Q$ and $\Lambda_{ij}^+ = \max(\Lambda_{ij}, \epsilon)$, with $\epsilon > 0$.

    In practice, directly projecting H_f can be expensive. If $f = \sum_i f_i$, can separately project $\nabla^2 f_i$ (possibly with smaller sizes, e.g. a spring in the mass-spring system only associates with 2 nodes).

Table of Content

Quasi-Newton Methods

Overview

- **Motivation:** Newton's method converges rapidly but requires computing and inverting the full Hessian $H_f(\mathbf{x}_k)$, which is $O(n^2)$ in size and thus expensive.

- **Idea:** Approximate H_f using cheaper computations, leading to **quasi-Newton methods**.

- **Approximation:** near current iterate $\mathbf{x}_k$, $f(\mathbf{x}_k + \delta\mathbf{x}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\top B_k \delta\mathbf{x}$, where $B_k \approx H_f(\mathbf{x}_k)$ is a symmetric positive-definite approximation.

- **Update step:** $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k B_k^{-1} \nabla f(\mathbf{x}_k)$, with α_k determined via line search.

- **Special cases:**

  - $B_k = H_f(\mathbf{x}_k)$: Newton's method.

  - $B_k = I$: Gradient descent.

Quasi-Newton Methods

Secant Condition

- To update B_k without computing a new Hessian, impose the **secant condition:**

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

- Define:

$$\mathbf{s}_k := \mathbf{x}_{k+1} - \mathbf{x}_k, \quad \mathbf{y}_k := \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k),$$

  so that $B_{k+1}\mathbf{s}_k = \mathbf{y}_k$.

- This ensures B_k behaves like the local Hessian along the line connecting $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$.
- Additional constraints (e.g., symmetry) narrow the possible B_{k+1}.

Quasi-Newton Methods

Symmetric Quasi-Newton Update

- We seek the symmetric matrix B_{k+1} that:

$$\min_{B_{k+1}} \|B_{k+1} - B_k\| \quad \text{s.t. } B_{k+1}^\top = B_{k+1},\ B_{k+1}\mathbf{s}_k = \mathbf{y}_k.$$

- This "lazy" update preserves as much of the old approximation as possible while enforcing the secant condition.

- Using the Frobenius norm $\|A\|_F^2 = \mathrm{tr}(A^\top A)$ gives a least-squares-type solution.

- But since B_k models curvature, mismatched units across variables can distort the update.

- Remedy: use a **weighted Frobenius norm** with a positive-definite weighting matrix W:

$$\|A\|_W^2 := \mathrm{tr}(A^\top W^\top A W).$$

Quasi-Newton Methods

DFP Update

- Using the weighted Frobenius norm yields the **Davidon–Fletcher–Powell (DFP)** update.
- **Proposition:** For symmetric $B_0 \in \mathbb{R}^{n \times n}$ and vectors $\mathbf{s}, \mathbf{y} \in \mathbb{R}^n$, if $W\mathbf{y} = \mathbf{s}$ and W is positive definite, then:

$$\min_B \|B - B_0\|_W \quad \text{s.t. } B^\top = B,\ B\mathbf{s} = \mathbf{y}$$

  has solution

$$B = (I - \rho \mathbf{y}\mathbf{s}^\top) B_0 (I - \rho \mathbf{s}\mathbf{y}^\top) + \rho \mathbf{y}\mathbf{y}^\top,$$

  where $\rho = 1/(\mathbf{s}^\top \mathbf{y})$.
- This formula defines the DFP quasi-Newton update.
- Ensures symmetry and secant condition while improving numerical stability.

Quasi-Newton Methods
DFP Derivation Outline

- Assume W is positive definite, $W = LL^\top$ (Cholesky factorization).
- Substitute $B = B_0 + L^{-\top} D L^{-1}$ and reduce problem to:

$$\min_D \frac{1}{2}\,\mathrm{tr}(D^\top D) \quad \text{s.t. } D^\top = D,\ D\mathbf{w} = \mathbf{z},$$

  where $\mathbf{w} = L^{-1}\mathbf{s}$, $\mathbf{z} = L^\top(\mathbf{y} - B_0\mathbf{s})$.
- Apply Lagrange multipliers: $\Lambda(D; A, \lambda) = \frac{1}{2}\,\mathrm{tr}(D^\top D) + \mathrm{tr}(A^\top(D - D^\top)) + \lambda^\top(\mathbf{z} - D\mathbf{w})$.
- Optimize to find: $D = \frac{1}{2}(\lambda\mathbf{w}^\top + \mathbf{w}\lambda^\top), \quad \lambda = \frac{2\mathbf{z}}{\|\mathbf{w}\|_2^2} - \frac{\mathbf{z}\cdot\mathbf{w}}{\|\mathbf{w}\|_2^4}\mathbf{w}$.
- Substitute back to obtain final DFP formula.

Quasi-Newton Methods

DFP Algorithm Implementation

- The **Davidon–Fletcher–Powell (DFP)** update is: $B \leftarrow (I - \rho \mathbf{y}\mathbf{s}^\top) B (I - \rho \mathbf{s}\mathbf{y}^\top) + \rho \mathbf{y}\mathbf{y}^\top$, where $\rho = 1/(\mathbf{s}^\top \mathbf{y})$.
- This resembles Newton's method but replaces the Hessian with the updated approximation B.
- The main cost per iteration: solving $B\mathbf{d} = -\nabla f(\mathbf{x})$.
- Using the Sherman–Morrison formula, we can update the **inverse** B^{-1} efficiently without explicit inversion:

$$B^{-1} = B_0^{-1} - \frac{\mathbf{q}\mathbf{q}^\top}{\mathbf{y}^\top \mathbf{q}} + \rho \mathbf{s}\mathbf{s}^\top, \quad \text{where } \mathbf{q} = B_0^{-1}\mathbf{y}.$$

- Runtime: reduces from $O(n^3)$ to $O(n^2)$ for dense problems.

Quasi-Newton Methods

From DFP to BFGS

- DFP maintains B^{-1}, an approximation of the inverse Hessian.
- Directly storing and updating B^{-1} is computationally efficient but may introduce numerical instability.
- To mitigate instability, a more stable alternative – the **BFGS** (Broyden–Fletcher–Goldfarb–Shanno) method – updates the inverse Hessian directly.
- **BFGS update constraint:** $C_{k+1}\mathbf{y}_k = \mathbf{s}_k, \quad \mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \quad \mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$
- Optimization formulation: $\min_{C_{k+1}} \|C_{k+1} - C_k\|_{W'} \quad \text{s.t. } C_{k+1}^\top = C_{k+1},\ C_{k+1}\mathbf{y}_k = \mathbf{s}_k.$
- The result (flipping the roles of $\mathbf{s}$ and $\mathbf{y}$) gives: $C \leftarrow (I - \rho\mathbf{s}\mathbf{y}^\top)C(I - \rho\mathbf{y}\mathbf{s}^\top) + \rho\mathbf{s}\mathbf{s}^\top.$

Quasi-Newton Methods

L-BFGS

- **BFGS update:** $C_{k+1} = (I - \rho_k \mathbf{s}_k \mathbf{y}_k^\top) C_k (I - \rho_k \mathbf{y}_k \mathbf{s}_k^\top) + \rho_k \mathbf{s}_k \mathbf{s}_k^\top$. with $C_k \approx H_f(\mathbf{x}_k)^{-1}$ an approximation of the inverse Hessian.
- **Advantages:**
  - Avoids explicit computation/inversion of H_f.
  - Empirically converges faster and more stably than DFP.
  - Guarantees positive definiteness if C_k is SPD and $\mathbf{s}_k^\top \mathbf{y}_k > 0$.
- **Cost:** still requires $O(n^2)$ storage for C_k.
- **L-BFGS (Limited-memory BFGS):** Stores only a few recent $(\mathbf{s}_i, \mathbf{y}_i)$ pairs. Applies updates recursively to approximate C_k. Greatly reduces memory cost and is widely used in large-scale ML optimization.

Quasi-Newton Methods

Summary

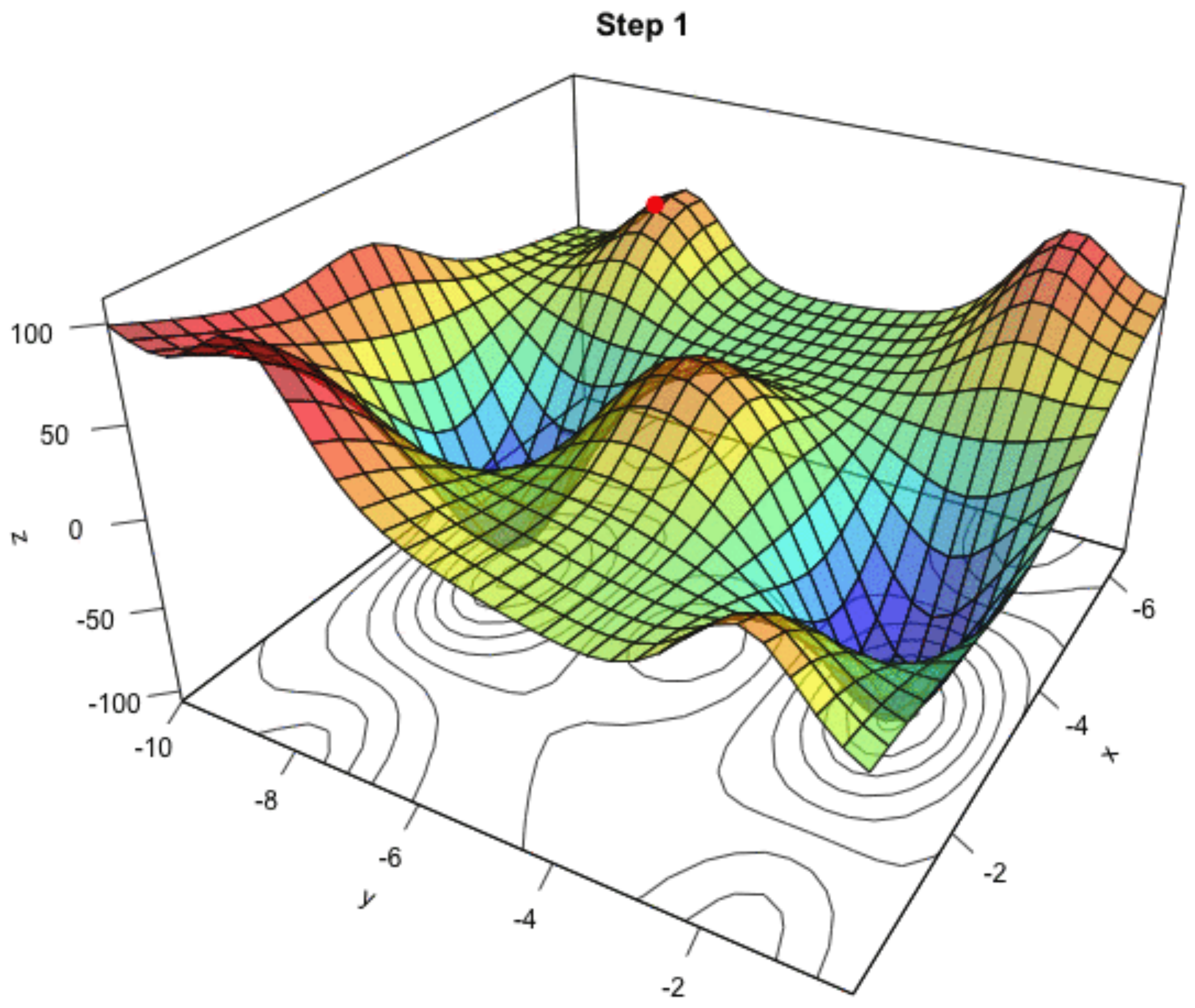
- Both DFP and BFGS are **quasi-Newton methods**.
- **DFP:** updates approximation of Hessian B_k.
- **BFGS:** updates approximation of inverse Hessian $C_k = B_k^{-1}$.
- Their updates are structurally similar but inversely related:

$$B_{k+1} = (I - \rho y s^\top) B_k (I - \rho s y^\top) + \rho y y^\top,$$
$$C_{k+1} = (I - \rho s y^\top) C_k (I - \rho y s^\top) + \rho s s^\top.$$

- BFGS tends to outperform DFP in practice due to better numerical stability.
- L-BFGS remains one of the most popular large-scale optimization algorithms today.

Table of Content

Gradient-Based Optimization Methods



- In each step:
  - Pick a descent direction
    - Gradient Descent/Newton's Method/Quasi-Newton Methods/…
  - Decide a step size
    - Line Search/Scheduled Step Sizes/…

Image source: https://medium.com/@pau.moskwa/eli5-ml-optmization-algorithms-9ba6305b1190