IEOR 4500 Applications Programming for FE

Week 7-2: Gradient Descent

Anran Hu

Introduction to Optimization

- **What is Optimization?**
  - The process of finding the best solution from a set of feasible solutions.
  - Involves maximizing or minimizing an objective function.
- **Optimization Problems**
  - **Objective Function**: The function to be optimized $f(x)$ (e.g., profit, cost).
  - **Variables**: Parameters that can be adjusted to optimize the objective x.
  - **Constraints**: Conditions that the variables must satisfy e.g., $g(x) \leq 0, g(x) = 0$.

Introduction to Optimization

- **Different Types of Optimization**
  - **Linear vs. Nonlinear**: Linear functions vs. functions with nonlinear relationships.
  - **Convex vs. Non-Convex**: Convex problems have one global optimum; non-convex may have multiple local optima.
  - **Constrained vs. Unconstrained**: Constrained problems include restrictions on the variables.
- **Applications of Optimization**
  - Finance: Portfolio optimization, risk management.
  - Engineering: Resource allocation, design optimization.
  - Machine Learning: Training algorithms, hyperparameter tuning.

Unconstrained Optimization

Say we want to minimize some function $f(x)$, without constraints

- Objective: Find the minimum (or maximum) of a function $f(x)$.
- Formally:

$$\min_{x \in \mathbb{R}^n} f(x)$$

- **Common Methods**
  - **Gradient Descent**: Iteratively moves in the direction of the negative gradient.
  - **Newton's Method**: Uses second-order derivatives (Hessian) to refine updates.
  - **Quasi-Newton Methods**: Approximates the Hessian for faster convergence (e.g., BFGS).

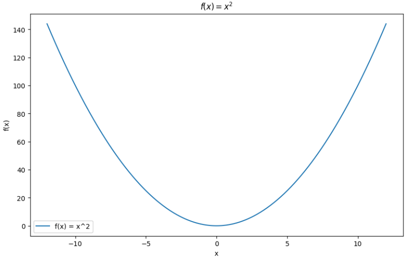
Descent Directions and Optimal Points

- Goal: Minimize a differentiable convex function $f(x)$ over $\mathbb{R}^n$.
- Scheme:
  - Start at an initial point x_0.
  - At each step, move to a new point $x + \Delta x$ such that $f(x + \Delta x) < f(x)$.
- Descent Directions: Δx is called a *descent direction* if it reduces $f(x)$: $f(x + \Delta x) < f(x)$.
- First-order condition for convex functions: For all $x, y \in \mathbb{R}^n$,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

- Stopping Criterion: Stop when $\nabla f(x) = 0$, indicating no further descent directions.

Gradient Descent in One Dimension



- In 1D, only two directions: forward or backward.
- For convex $f(x)$:
  - If gradient $f'(x) < 0$: Moving forward decreases $f(x)$.
  - If gradient $f'(x) > 0$: Moving backward decreases $f(x)$.
- The step direction should be opposite to the sign of the gradient.

Gradient Descent for Multivariate Functions

- In multiple dimensions, there are many descent directions.
- A natural choice: $\Delta x = -t\nabla f(x)$ (negative gradient).
- For small step size $t > 0$, Taylor expansion shows

$$f(x + \Delta x) \approx f(x) - t\nabla f(x)^T \nabla f(x) < f(x).$$

- This shows $\Delta x = -t\nabla f(x)$ is a descent direction.
- This approach is the foundation of the **Gradient Descent Method**.

Gradient Descent

$$\min_{x \in \mathbb{R}^n} f(x)$$

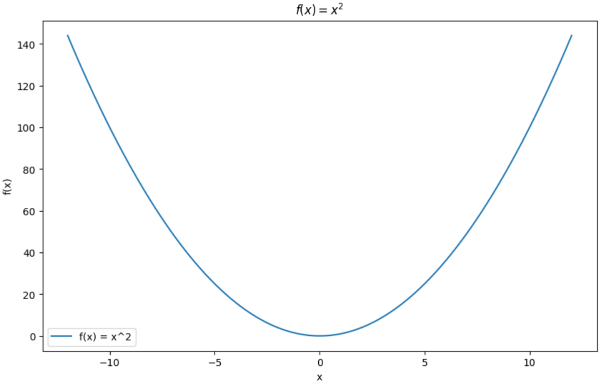
- An iterative optimization algorithm to minimize a differentiable function.
- Updates x by moving in the direction opposite to the gradient.
- Update rule: $x \leftarrow x - \alpha \nabla f(x)$.
- The learning rate α controls the step size.
- Repeat until convergence: parameters change minimally or objective stabilizes.

Why Gradient Descent is Popular

- **Efficient for Large-Scale Problems**
  - Only requires the first-order derivative, making it scalable to high-dimensional data.
  - Stochastic and mini-batch variants allow for handling large datasets effectively.
- **Flexible with Different Objective Functions**
  - Applicable to both convex and non-convex functions.
  - Widely used in machine learning, from linear regression to deep neural networks.
- **Simple and Easy to Implement**
  - Iterative update rule is straightforward and adaptable to various optimization tasks.
  - Can be customized with different learning rates and regularization terms.

Example

- Minimizing $f(x) = x^2$, with derivative $f'(x) = 2x$.

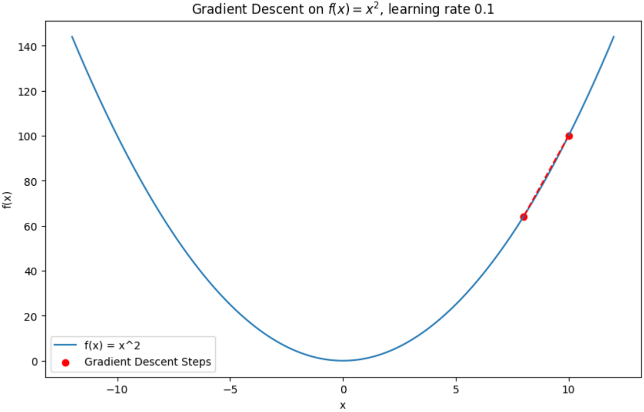


- Start at $x = 10$, with $\alpha = 0.1$.

Example: Step 1

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

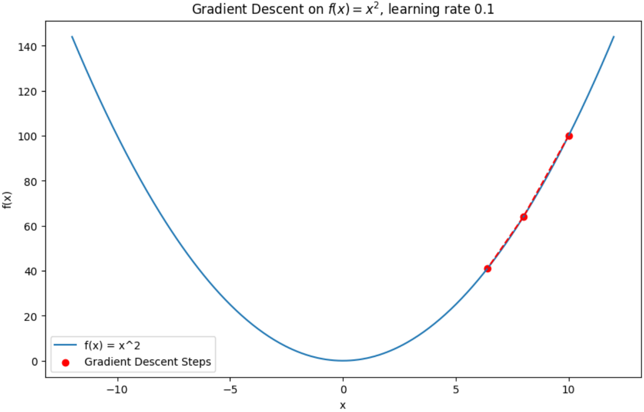
- Start at $x_0 = 10$, with $\alpha = 0.1$.
- $x_1 = x_0 - \alpha f'(x_0) = x_0 - \alpha \cdot 2x_0 = 0.8x_0 = 8$



Example: Step 2

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

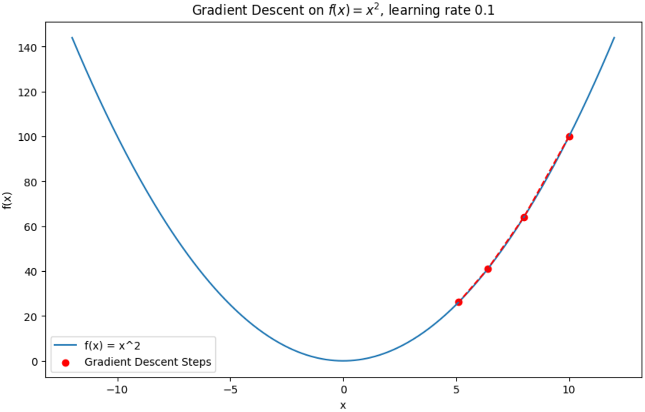
- Second iteration
- $x_2 = x_1 - \alpha f'(x_1) = x_1 - \alpha \cdot 2x_1 = 0.8x_1 = 6.4$



Example: Step 3

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

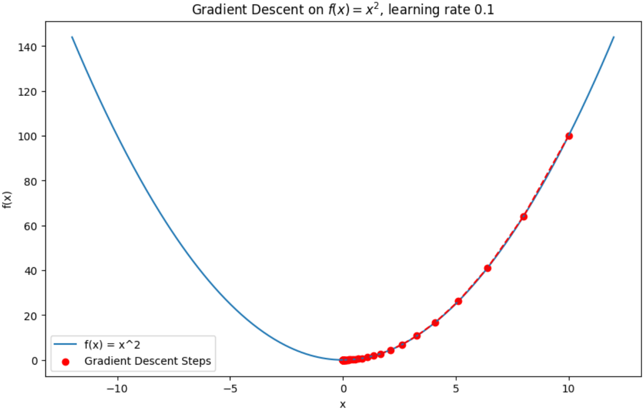
- Third iteration
- $x_3 = x_2 - \alpha f'(x_2) = x_2 - \alpha \cdot 2x_2 = 0.8x_2 = 5.12$



Example: Step 200

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

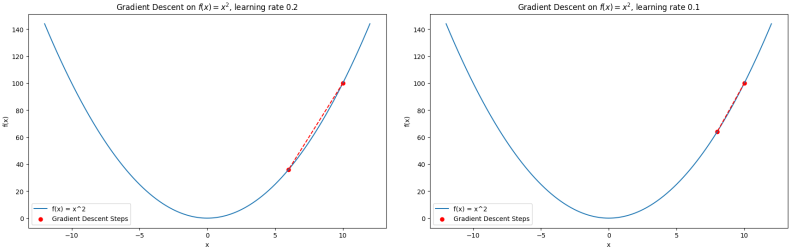
- After 200 iteration
- $x_{200} = 10 \times 0.8^{200} \approx 4e^{-19}$



Effect of Learning Rate

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

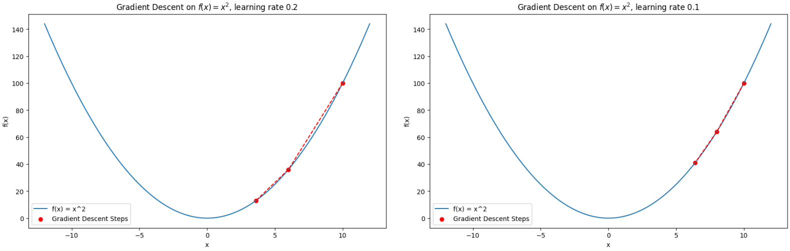
- If we choose a different learning rate, $\alpha = 0.2$, $x_0 = 10$.
- Comparing with $\alpha = 0.1$, step size is larger
- First iteration:



Effect of Learning Rate

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

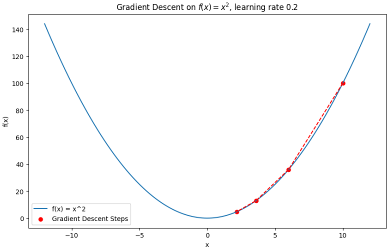
- If we choose a different learning rate, $\alpha = 0.2$, $x_0 = 10$.
- Comparing with $\alpha = 0.1$, step size is larger
- Second iteration:

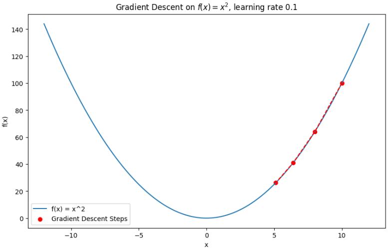


Effect of Learning Rate

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

- If we choose a different learning rate, $\alpha = 0.2$, $x_0 = 10$.
- Comparing with $\alpha = 0.1$, step size is larger
- Third iteration:

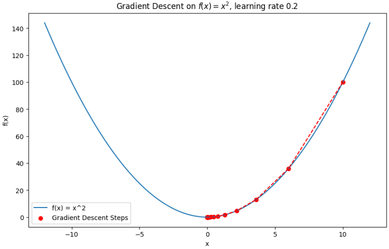


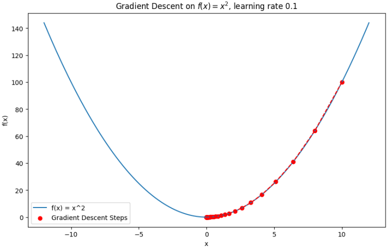


Effect of Learning Rate

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

- If we choose a different learning rate, $\alpha = 0.2$, $x_0 = 10$.
- Comparing with $\alpha = 0.1$, step size is larger
- In the end

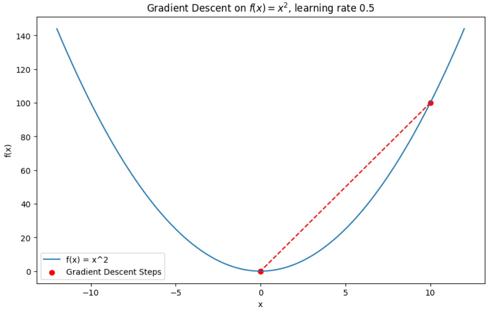




Effect of Learning Rate

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

- It seems that increasing learning rate α can make the algorithm converge faster. Is this always true?
- Let's try a bigger learning rate $\alpha = 0.5$
- $x_1 = x_0 - \alpha f'(x_0) = x_0 - x_0 = 0$. Converge in one step!



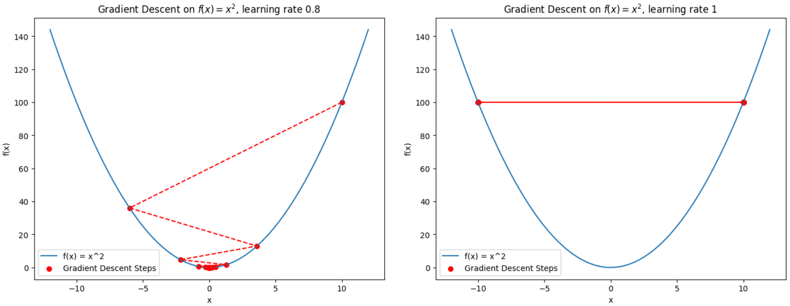
Effect of Learning Rate

- Let's try bigger α
- When $\alpha = 0.8$,

$$x_{k+1} = x_k - 0.8 \cdot 2x_k = -0.6x_k$$

- When $\alpha = 1$,

$$x_{k+1} = x_k - 2x_k = -x_k$$



Importance of Learning Rate (α)

- Controls step size.
- Too large: Overshooting, possible divergence.
- Too small: Slow convergence.
- How to choose learning rate?

- Constant learning rate
  - Fixed α requires tuning.
  - Common methods: Grid search, manual adjustment.
- Adaptive learning rate: backtracking line search
  - Dynamically adjusts α until sufficient decrease in the objective.
  - More computationally intensive but robust.

Stopping Criteria

- Stop when gradient norm is small.
- Set a maximum iteration.
- Monitor change in objective function.

Convergence Theorem for Gradient Descent

- Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex and differentiable with a Lipschitz continuous gradient.
- L-smooth: there exists $L > 0$ such that:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y \in \mathbb{R}^n.$$

- L-smoothness limits how quickly the gradient can change, ensuring f is not too steep or irregular.
- The parameter L is known as the *smoothness constant*.

Convergence Theorem for Gradient Descent

- Gradient descent update:

$$x_{k+1} = x_k - \alpha \nabla f(x_k).$$

- **Convergence for General Convex Functions**
  - If $\alpha \leq \frac{1}{L}$:

$$f(x_k) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\alpha k}.$$

  - Sublinear rate: $O\left(\frac{1}{k}\right)$.

Convergence Theorem for Gradient Descent

- A function $f : \mathbb{R}^n \to \mathbb{R}$ is *strongly convex* if there exists a constant $\mu > 0$ such that:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2}\|y - x\|^2$$

  for all $x, y \in \mathbb{R}^n$.
- The parameter μ is called the *strong convexity constant*.
- Strong convexity implies that the function f is *more curved* than a regular convex function, providing a quadratic lower bound.
- When $\mu > 0$, the function has a unique global minimum.

Convergence Theorem for Gradient Descent

- **Convergence for Strongly Convex Functions**
  - If f is strongly convex with μ and $\alpha \leq \frac{1}{L}$:

$$\|x_k - x^*\| \leq (1 - \mu\alpha)^k \|x_0 - x^*\|.$$

  - Linear rate: $O\left((1 - \mu\alpha)^k\right)$.

Key Points

- **Lipschitz Gradient**: Ensures the gradient does not change too quickly, stabilizing gradient descent.
- **Step Size**: α must satisfy $\alpha \leq \frac{1}{L}$.
- **Rates**: Sublinear for convex; linear for strongly convex, indicating faster convergence.

IEOR 4500 Applications Programming for FE

Week 8-2: Gradient Descent

Anran Hu

Descent Directions and Optimal Points

- Goal: Minimize a differentiable convex function $f(x)$ over $\mathbb{R}^n$.
- Scheme:
  - Start at an initial point x_0.
  - At each step, move to a new point $x + \Delta x$ such that $f(x + \Delta x) < f(x)$.
- Descent Directions: Δx is called a *descent direction* if it reduces $f(x)$: $f(x + \Delta x) < f(x)$.
- First-order condition for convex functions: For all $x, y \in \mathbb{R}^n$,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x).$$

- Stopping Criterion: Stop when $\nabla f(x) = 0$, indicating no further descent directions.

Gradient Descent

- In multiple dimensions, there are many descent directions.
- A natural choice: $\Delta x = -t\nabla f(x)$ (negative gradient).
- For small step size $t > 0$, Taylor expansion shows

$$f(x + \Delta x) \approx f(x) - t\nabla f(x)^T \nabla f(x) < f(x).$$

- This shows $\Delta x = -t\nabla f(x)$ is a descent direction.
- This approach is the foundation of the **Gradient Descent Method**.

Gradient Descent

$$\min_{x \in \mathbb{R}^n} f(x)$$

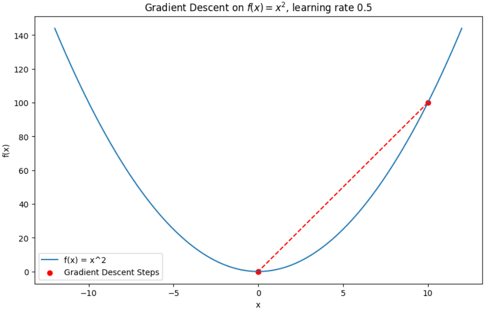
- An iterative optimization algorithm to minimize a differentiable function.
- Updates x by moving in the direction opposite to the gradient.
- Update rule: $x \leftarrow x - \alpha \nabla f(x)$.
- The learning rate α controls the step size.
- Repeat until convergence: parameters change minimally or objective stabilizes. (e.g.,$\|\nabla f(x)\| \leq \epsilon$)

Important: The choice of step size α is crucial. A larger step size can help explore faster but can also result in an increase in function value or insufficient decrease.

Effect of Learning Rate

$$\min_{x \in \mathbb{R}} f(x) = x^2$$

- Let's try learning rate $\alpha = 0.5$
- $x_1 = x_0 - \alpha f'(x_0) = x_0 - x_0 = 0$. Converge in one step!



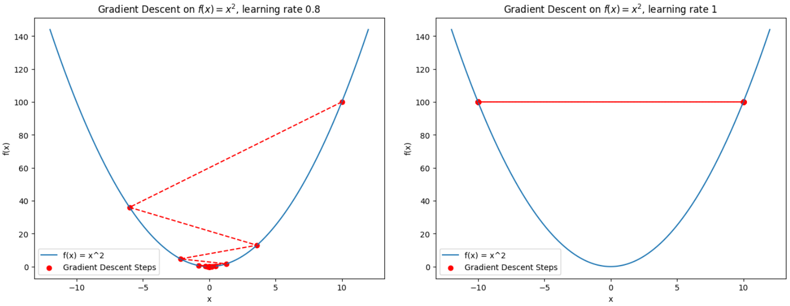
Effect of Learning Rate

- Let's try bigger α
- When $\alpha = 0.8$,

$$x_{k+1} = x_k - 0.8 \cdot 2x_k = -0.6x_k$$

- When $\alpha = 1$,

$$x_{k+1} = x_k - 2x_k = -x_k$$



Importance of Learning Rate (α)

- Controls step size.
- Too large: Overshooting, possible divergence.
- Too small: Slow convergence.
- How to choose learning rate?

- Constant learning rate
  - Fixed α requires tuning.
  - Common methods: Grid search, manual adjustment.
- Adaptive learning rate: backtracking line search
  - Dynamically adjusts α until sufficient decrease in the objective.
  - More computationally intensive but robust.

Convergence Theorem for Gradient Descent

- Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex and differentiable with a Lipschitz continuous gradient.
- L-smooth: there exists $L > 0$ such that:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \forall x, y \in \mathbb{R}^n.$$

- L-smoothness limits how quickly the gradient can change, ensuring f is not too steep or irregular.
- The parameter L is known as the *smoothness constant*.

Convergence Theorem for Gradient Descent

- Gradient descent update:

$$x_{k+1} = x_k - \alpha \nabla f(x_k).$$

- **Convergence for General Convex Functions**
  - If $\alpha \leq \frac{1}{L}$:

$$f(x_k) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\alpha k}.$$

  - Sublinear rate: $O\left(\frac{1}{k}\right)$.

Convergence Theorem for Gradient Descent

- A function $f : \mathbb{R}^n \to \mathbb{R}$ is *strongly convex* if there exists a constant $\mu > 0$ such that:

  $$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2}\|y - x\|^2$$

  for all $x, y \in \mathbb{R}^n$.
- The parameter μ is called the *strong convexity constant*.
- Strong convexity implies that the function f is *more curved* than a regular convex function, providing a quadratic lower bound.
- When $\mu > 0$, the function has a unique global minimum.

Convergence Theorem for Gradient Descent

- **Convergence for Strongly Convex Functions**
  - If f is strongly convex with μ and $\alpha \leq \frac{1}{L}$:

$$\|x_k - x^*\| \leq (1 - \mu\alpha)^k \|x_0 - x^*\|.$$

  - Linear rate: $O\left((1 - \mu\alpha)^k\right)$.

Key Points

- **Lipschitz Gradient**: Ensures the gradient does not change too quickly, stabilizing gradient descent.
- **Step Size**: α must satisfy $\alpha \leq \frac{1}{L}$.
- **Rates**: Sublinear for convex; linear for strongly convex, indicating faster convergence.

Constant Learning Rate: When It May Not Work

- **Constant Learning Rate in Gradient Descent**:
  - A fixed step size α is used to update the parameters.
  - Works well for simple, well-conditioned functions.
- **Problems with Constant Learning Rate**:
  - **Ill-conditioned problems**: Large condition number leads to slow convergence.
  - **Non-convex functions**: A constant step size may oscillate near saddle points or get stuck in local minima.
  - **Variable curvature**: A single step size may be too large for steep regions and too small for shallow regions.

Condition Number and Ill-Conditioned Problems

- **Condition Number (κ)**:
  - Ratio of the largest eigenvalue to the smallest eigenvalue of the Hessian matrix.
  - $\kappa \approx 1$: Function is well-conditioned, easier to optimize.
  - $\kappa \gg 1$: Function is ill-conditioned, difficult to optimize.
- **Convergence for Strongly Convex Functions**
  - If f is strongly convex with μ and $\alpha \leq \frac{1}{L}$:

$$\|x_k - x^*\| \leq (1 - \mu\alpha)^k \|x_0 - x^*\|.$$

  - μ and L are the smallest/largest eigenvalues of the Hessian
- **Effect on Gradient Descent**:
  - High κ: Zigzagging behavior, slow convergence.
  - A constant learning rate may overshoot in some directions and lead to inefficient updates in others.

An example

Ill-Conditioned Quadratic Function:

$$f(x_1, x_2) = x_1^2 + \gamma x_2^2 \quad \text{(with } \gamma \gg 1\text{)}$$

- Condition number κ is large due to $\gamma \gg 1$, resulting in elongated contours.
- Gradient descent with constant learning rate will struggle, leading to:
  - **Zigzagging** in steep directions.
  - **Slow progress** in shallow directions.
- See notebook

Step Size Selection

Exact Line Search:

$$t = \arg\min_{s \geq 0} f(x + s\Delta x)$$

An exact line search is used when the minimization cost with one variable is lower than computing the search direction.

Inexact Methods: Backtracking line search is a popular heuristic for selecting t. It ensures that the step size is reduced enough to guarantee sufficient reduction in $f(x)$.

Backtracking Line Search

Adaptive Step Size: Adjusts t dynamically based on the Armijo condition:

$$f(x - t\nabla f(x)) \leq f(x) - \alpha t\|\nabla f(x)\|^2$$

- Set $t = 1$.
- While $f(x - t\nabla f(x)) > f(x) - \alpha t\|\nabla f(x)\|^2$:
  - Update $t := \beta t$, where $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$.
- Allows larger steps in shallow directions and smaller steps in steep directions.
- Reduces **zigzagging** and ensures stable convergence.

Taylor Expansion Insight: There always exists a small enough t ($t < 1/L$ for strongly smooth function) that satisfies the condition due to the Taylor expansion.

$$f(x - t\nabla f(x)) \approx f(x) - t\nabla f(x)^\top \nabla f(x) + \cdots$$

Benefits of Backtracking Line Search

Benefits:

- **Improved Stability:** Step size is dynamically reduced, preventing overshooting.
- **Faster Convergence:** Larger steps can be taken in shallow directions, speeding up convergence.
- **No Need for Condition Number Knowledge:** Backtracking adapts based on local curvature without needing explicit knowledge of the condition number.

Example: Unconstrained Minimization in $\mathbb{R}^2$

Consider the convex function:

$$f(x) = x_1^2 + \gamma x_2^2$$

The gradient is:

$$\nabla f(x) = \begin{bmatrix} 2x_1 \\ 2\gamma x_2 \end{bmatrix}$$

After the update step, the next point in the gradient descent algorithm is:

$$x'(t) = ((1 - 2t)x_1, (1 - 2\gamma t)x_2)$$

Exact Line Search

Exact Line Search:

$$t = \arg\min_{t \geq 0} f(x'(t))$$

Solving this yields:

$$t = \frac{x_1^2 + \gamma^2 x_2^2}{2(x_1^2 + \gamma^3 x_2^2)}$$

Comparison:

- **Backtracking Line Search:** Slower convergence but faster step size computation.
- **Exact Line Search:** Faster convergence but more computationally expensive for each step size calculation.

IEOR 4500 Applications Programming for FE

Week 9-1: Constrained Optimization

Anran Hu

Gradient Descent

$$\min_{x \in \mathbb{R}^n} f(x)$$

- An iterative optimization algorithm to minimize a differentiable function.
- Updates x by moving in the direction opposite to the gradient.
- Update rule: $x \leftarrow x - \alpha \nabla f(x)$.
- The learning rate α controls the step size.
- Repeat until convergence: parameters change minimally or objective stabilizes. (e.g.,$\|\nabla f(x)\| \leq \epsilon$)

Important: The choice of step size α is crucial. A larger step size can help explore faster but can also result in an increase in function value or insufficient decrease.

Importance of Learning Rate (α)

- Controls step size.
- Too large: Overshooting, possible divergence.
- Too small: Slow convergence.
- How to choose learning rate?

- Constant learning rate
  - Fixed α requires tuning.
  - Common methods: Grid search, manual adjustment.
  - smaller than $1/L$, where L is the smoothness parameter, the Lipschitz constant for gradient of f.
- Adaptive learning rate: backtracking line search
  - Dynamically adjusts α until sufficient decrease in the objective.
  - More computationally intensive but robust.

Step Size Selection

Exact Line Search:

$$t = \arg\min_{s \geq 0} f(x + s\Delta x)$$

An exact line search is used when the minimization cost with one variable is lower than computing the search direction.

Inexact Methods: Backtracking line search is a popular heuristic for selecting t. It ensures that the step size is reduced enough to guarantee sufficient reduction in $f(x)$.

Backtracking Line Search

Adaptive Step Size: Adjusts t dynamically based on the Armijo condition:

$$f(x - t\nabla f(x)) \leq f(x) - \alpha t \|\nabla f(x)\|^2$$

- Set $t = 1$.
- While $f(x - t\nabla f(x)) > f(x) - \alpha t \|\nabla f(x)\|^2$:
  - Update $t := \beta t$, where $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$.
- Allows larger steps in shallow directions and smaller steps in steep directions.
- Reduces **zigzagging** and ensures stable convergence.

Accelerated Gradient Descent

Nesterov's Accelerated Gradient Descent improves standard gradient descent by introducing momentum. The update rule becomes:

$$\begin{aligned} y^{(k+1)} &= x^{(k)} - t^{(k)} \nabla f(x^{(k)}) \quad (1) \\ x^{(k+1)} &= y^{(k+1)} + s^{(k)}(y^{(k+1)} - y^{(k)}) \quad (2) \end{aligned}$$

Momentum correction accelerates convergence by reducing "zigzagging."

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD):

- Useful for minimizing functions that are the sum of many terms, such as in statistical estimation.

$$\text{minimize} \sum_i f_i(x)$$

- At each iteration, a random sample $f_i(x)$ is used to compute the gradient, reducing computation time:

$$x^{(k+1)} = x^{(k)} - t^{(k)} \nabla f_i(x^{(k)})$$

Introduction to Constrained Optimization

Constrained Optimization Problem:

- Given an objective function $f(x)$ to minimize over $x \in \mathbb{R}^n$.
- Subject to constraints:

$$
\begin{aligned}
g_i(x) &\leq 0 \quad \text{(inequality constraints)} \\
h_j(x) &= 0 \quad \text{(equality constraints)}
\end{aligned}
$$

Goal: Find x^* that minimizes $f(x)$ while satisfying all constraints.

- Constraints restrict the feasible region for the solution.
- Common methods: Projected Gradient Descent, Penalty Method, and Barrier Method.

Projected Gradient Descent (PGD)

Projected Gradient Descent:

- Suitable for problems with simple convex constraints, where $x \in \mathcal{C}$.
- Update rule:
$$x_{k+1} = \Pi_{\mathcal{C}}(x_k - \alpha \nabla f(x_k))$$
where $\Pi_{\mathcal{C}}$ denotes the projection onto the set $\mathcal{C}$.
- Each gradient step is followed by a projection back onto the feasible set.

Example Applications:

- Problems with box constraints, e.g., $x_i \in [a_i, b_i]$.
- Problems where $\mathcal{C}$ is defined by linear inequalities.

Projected Gradient Descent (PGD)

Objective: Minimize $f(x_1, x_2) = (x_1 + 1)^2 + 2x_2^2$

Constraints:

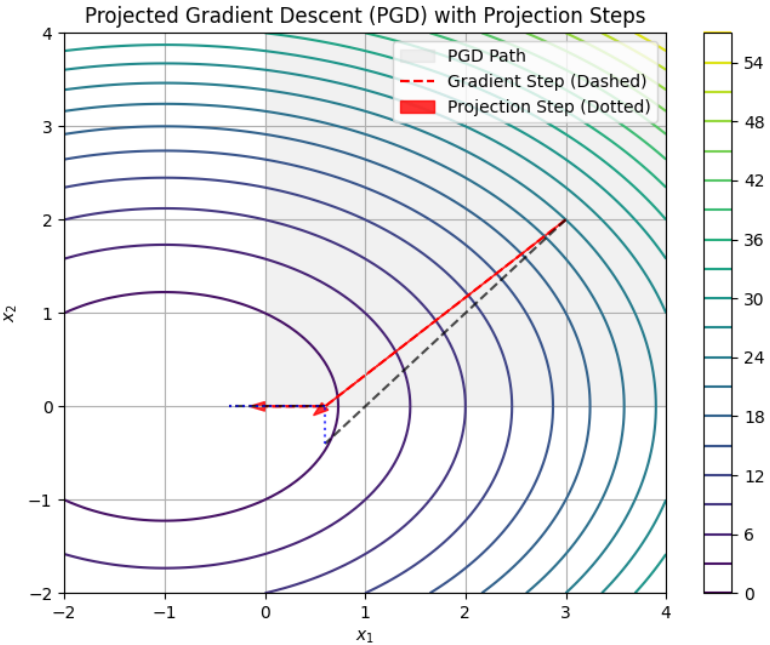
- $x_1 \geq 0$, $x_2 \geq 0$ (first quadrant only)

PGD Steps:

1. Take a gradient descent step: $x_{\text{new}} = x - \alpha \nabla f(x)$.
2. *Project* the new point x_{new} back onto the feasible region if it lies outside.

- Initialization: $x_1 = 3, x_2 = 2$, learning rate $\alpha = 0.3$
- First update $x_{1,\text{new}} = 3 - 0.3 \times 2 \times (3 + 1) = 0.6$, $x_{2,\text{new}} = 2 - 0.3 \times 4 \times 2 = -0.4$
- After projection: $x_{\text{new}} = (0.6, 0)$
- Second update $x_{1,\text{new}} = 0.6 - 0.3 \times 2 \times (0.6 + 1) = -0.36$, $x_{2,\text{new}} = 0$
- After projection: $x_{\text{new}} = (0, 0)$

PGD Illustration



PGD for Different Constraints

- Box constraints: $x_i \in [a_i, b_i]$
- Single linear constraint: $a^\top x = b$ or $a^\top x \leq b$
  - Given a point $x_0 \in \mathbb{R}^n$.
  - The projection of x_0 onto $a^T x = b$ is given by:

$$x_{\text{proj}} = x_0 - \frac{a^T x_0 - b}{\|a\|^2} a$$

  - This formula adjusts x_0 along a to satisfy the constraint $a^T x = b$.
- Multiple constraints: could be tricky $\Rightarrow$ alternating projection

PGD for Different Constraints

- Box constraints: $x_i \in [a_i, b_i]$
- Single linear constraint: $a^\top x = b$ or $a^\top x \leq b$
  - Given a point $x_0 \in \mathbb{R}^n$.
  - The projection of x_0 onto $a^T x = b$ is given by:

$$x_{\text{proj}} = x_0 - \frac{a^T x_0 - b}{\|a\|^2} a$$

  - This formula adjusts x_0 along a to satisfy the constraint $a^T x = b$.
- Multiple constraints: could be tricky $\Rightarrow$ alternating projection

- Works well for simple, convex feasible sets.
- Efficient for box or linear constraints.
- Less effective for complex or non-convex constraints.

Using QP for Projection onto Linear Constraints

Problem Setup:

- Given a point $x_0 \in \mathbb{R}^n$.
- Project x_0 onto a feasible region defined by:
  - **Equality constraints:** $A_{eq}x = b_{eq}$
  - **Inequality constraints:** $A_{ineq}x \leq b_{ineq}$

QP Formulation:

- Objective: Minimize the distance to x_0.
- Formulate as:

$$\min_x \quad \|x - x_0\|^2$$

subject to:

$$A_{eq}x = b_{eq} \quad \text{and} \quad A_{ineq}x \leq b_{ineq}$$

Penalty Method

- Given an objective function $f(x)$ with equality and inequality constraints:

$$\min_x f(x)$$

subject to:

$$g_i(x) \leq 0, \quad i = 1, \ldots, m \quad \text{(inequality constraints)}$$

$$h_j(x) = 0, \quad j = 1, \ldots, p \quad \text{(equality constraints)}$$

Penalty Method:

- Converts the constrained problem into an unconstrained one by adding a penalty term to the objective.
- New objective function:

$$f_{\text{penalty}}(x, \rho) = f(x) + \rho \sum_i \max(0, g_i(x))^2 + \rho \sum_j h_j(x)^2$$

- ρ is the penalty parameter; larger values enforce constraints more strictly.

Penalty Method

$$f_{\text{penalty}}(x, \rho) = f(x) + \rho \sum_i \max(0, g_i(x))^2 + \rho \sum_j h_j(x)^2$$

The above new objective function is not differentiable. Can we do something to make it differentiable?

Penalty Method

$$f_{\text{penalty}}(x, \rho) = f(x) + \rho \sum_i \max(0, g_i(x))^2 + \rho \sum_j h_j(x)^2$$

The above new objective function is not differentiable. Can we do something to make it differentiable?

- Introduce new variables s_i
- Consider constrained optimization problem

$$\text{minimize} \quad f_{\text{penalty}}(x, s, \rho) = f(x) + \rho \sum_i (g_i(x) - s_i)^2 + \rho \sum_j h_j(x)^2$$

subjecto to $s_i \leq 0$ for all i

- Now we have an optimization problem with simple box constraints!

Penalty Method

Algorithm:

1. Start with an initial ρ.
2. Solve the unconstrained problem with $f_{\text{penalty}}(x, \rho)$, or problem with simple box constraints .
3. Increase ρ and repeat until constraints are approximately satisfied.

- Converts the problem into a series of unconstrained problems or problems with simple box constraints.
- Suitable for both equality and inequality constraints.
- Large ρ can lead to numerical issues; requires careful tuning.

Barrier Method

Barrier Method:

- Suitable for inequality constraints: $g_i(x) \leq 0$.
- Adds a barrier term to prevent the solution from approaching the boundary of the feasible region.
- New objective function:

$$f_{\text{barrier}}(x, \mu) = f(x) - \frac{1}{\mu} \sum_i \ln(-g_i(x))$$

- As $\mu \to \infty$, the barrier prevents x from violating constraints.

Algorithm:

1. Start with a small μ.
2. Solve the unconstrained problem with $f_{\text{barrier}}(x, \mu)$.
3. Gradually increase μ for stricter enforcement of constraints.

Projecting onto the Feasible Region in Log Barrier Method

Problem: When $g(x) > 0$, the point x is outside the feasible region defined by $g(x) \leq 0$.

Solution: Projecting along the Gradient Direction

- The gradient $\nabla g(x)$ points in the direction of the steepest increase in $g(x)$.
- Moving along $-\nabla g(x)$ reduces $g(x)$ and pushes x back toward the feasible region.
- Projection step for small adjustment:

$$x_{\text{new}} = x - \frac{\nabla g(x)}{\|\nabla g(x)\|^2} g(x)$$

  where $\frac{g(x)}{\|\nabla g(x)\|^2}$ scales the adjustment to ensure feasibility.

Intuition:

- $g(x_{\text{new}}) \approx g(x) - \nabla g(x)^\top \frac{\nabla g(x)}{\|\nabla g(x)\|^2} g(x) = 0$

Barrier Method

- Ideal for inequality constraints.
- Prevents boundary solutions; becomes more strict as μ increases.
- Not suitable for equality constraints; requires carefully increasing μ.
- Needs to choose stepsizes carefully: easy to fall into region $g_i(x) > 0$

IEOR 4500 Applications Programming for FE

Week 9-2: Constrained Optimization

Anran Hu

Introduction to Constrained Optimization

Constrained Optimization Problem:

- Given an objective function $f(x)$ to minimize over $x \in \mathbb{R}^n$.
- Subject to constraints:

$$\begin{aligned} g_i(x) &\leq 0 \quad \text{(inequality constraints)} \\ h_j(x) &= 0 \quad \text{(equality constraints)} \end{aligned}$$

Goal: Find x^* that minimizes $f(x)$ while satisfying all constraints.

- Constraints restrict the feasible region for the solution.
- Common methods: Projected Gradient Descent, Penalty Method, and Barrier Method.

Projected Gradient Descent (PGD)

Projected Gradient Descent:

- Suitable for problems with simple convex constraints, where $x \in \mathcal{C}$.
- Update rule:
$$x_{k+1} = \Pi_{\mathcal{C}}(x_k - \alpha \nabla f(x_k))$$
where $\Pi_{\mathcal{C}}$ denotes the projection onto the set $\mathcal{C}$.
- Each gradient step is followed by a projection back onto the feasible set.

Example Applications:

- Problems with box constraints, e.g., $x_i \in [a_i, b_i]$.
- Problems where $\mathcal{C}$ is defined by linear inequalities.

Projected Gradient Descent (PGD)

Objective: Minimize $f(x_1, x_2) = (x_1 + 1)^2 + 2x_2^2$

Constraints:

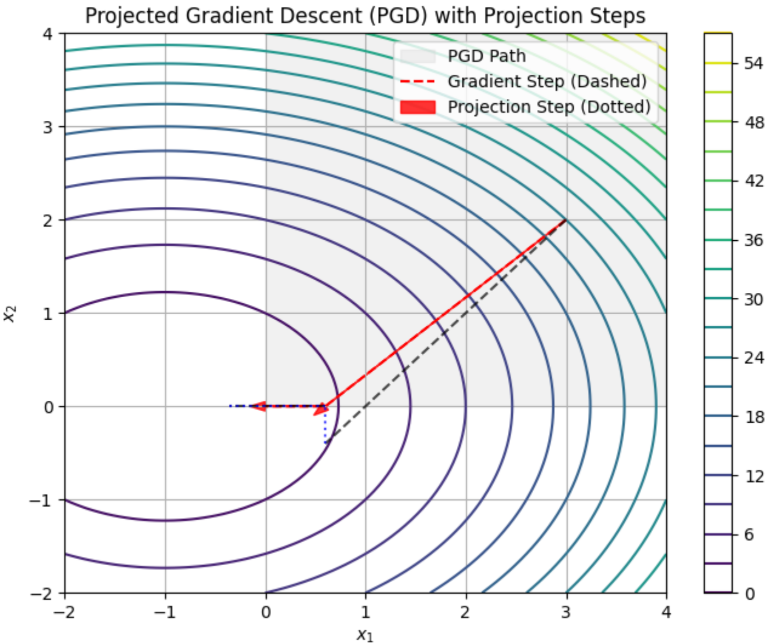
- $x_1 \geq 0$, $x_2 \geq 0$ (first quadrant only)

PGD Steps:

1. Take a gradient descent step: $x_{\text{new}} = x - \alpha \nabla f(x)$.
2. *Project* the new point x_{new} back onto the feasible region if it lies outside.

- Initialization: $x_1 = 3$,$x_2 = 2$, learning rate $\alpha = 0.3$
- First update $x_{1,\text{new}} = 3 - 0.3 \times 2 \times (3 + 1) = 0.6$, $x_{2,\text{new}} = 2 - 0.3 \times 4 \times 2 = -0.4$
- After projection: $x_{\text{new}} = (0.6, 0)$
- Second update $x_{1,\text{new}} = 0.6 - 0.3 \times 2 \times (0.6 + 1) = -0.36$, $x_{2,\text{new}} = 0$
- After projection: $x_{\text{new}} = (0, 0)$

PGD Illustration



PGD for Different Constraints

- Box constraints: $x_i \in [a_i, b_i]$
- Single linear constraint: $a^\top x = b$ or $a^\top x \leq b$
  - Given a point $x_0 \in \mathbb{R}^n$.
  - The projection of x_0 onto $a^T x = b$ is given by:

$$x_{\text{proj}} = x_0 - \frac{a^T x_0 - b}{\|a\|^2} a$$

  - This formula adjusts x_0 along a to satisfy the constraint $a^T x = b$.
- Multiple constraints: could be tricky $\Rightarrow$ alternating projection

PGD for Different Constraints

- Box constraints: $x_i \in [a_i, b_i]$
- Single linear constraint: $a^\top x = b$ or $a^\top x \leq b$
  - Given a point $x_0 \in \mathbb{R}^n$.
  - The projection of x_0 onto $a^T x = b$ is given by:

$$x_{\text{proj}} = x_0 - \frac{a^T x_0 - b}{\|a\|^2} a$$

  - This formula adjusts x_0 along a to satisfy the constraint $a^T x = b$.
- Multiple constraints: could be tricky $\Rightarrow$ alternating projection

- Works well for simple, convex feasible sets.
- Efficient for box or linear constraints.
- Less effective for complex or non-convex constraints.

Using QP for Projection onto Linear Constraints

Problem Setup:

- Given a point $x_0 \in \mathbb{R}^n$.
- Project x_0 onto a feasible region defined by:
  - **Equality constraints:** $A_{eq}x = b_{eq}$
  - **Inequality constraints:** $A_{ineq}x \leq b_{ineq}$

QP Formulation:

- Objective: Minimize the distance to x_0.
- Formulate as:

$$\min_{x} \quad \|x - x_0\|^2$$

subject to:

$$A_{eq}x = b_{eq} \quad \text{and} \quad A_{ineq}x \leq b_{ineq}$$

Penalty Method

- Given an objective function $f(x)$ with equality and inequality constraints:

$$\min_{x} f(x)$$

subject to:

$$g_i(x) \leq 0, \quad i = 1, \ldots, m \quad \text{(inequality constraints)}$$

$$h_j(x) = 0, \quad j = 1, \ldots, p \quad \text{(equality constraints)}$$

Penalty Method:

- Converts the constrained problem into an unconstrained one by adding a penalty term to the objective.
- New objective function:

$$f_{\text{penalty}}(x, \rho) = f(x) + \rho \sum_{i} \max(0, g_i(x))^2 + \rho \sum_{j} h_j(x)^2$$

- ρ is the penalty parameter; larger values enforce constraints more strictly.

Penalty Method

$$f_{\text{penalty}}(x, \rho) = f(x) + \rho \sum_i \max(0, g_i(x))^2 + \rho \sum_j h_j(x)^2$$

The above new objective function is not differentiable. Can we do something to make it differentiable?

Penalty Method

$$f_{\text{penalty}}(x, \rho) = f(x) + \rho \sum_i \max(0, g_i(x))^2 + \rho \sum_j h_j(x)^2$$

The above new objective function is not differentiable. Can we do something to make it differentiable?

- Introduce new variables s_i
- Consider constrained optimization problem

$$\text{minimize} \quad f_{\text{penalty}}(x, s, \rho) = f(x) + \rho \sum_i (g_i(x) - s_i)^2 + \rho \sum_j h_j(x)^2$$

subjecto to $s_i \leq 0$ for all i

- Now we have an optimization problem with simple box constraints!

Penalty Method

Algorithm:

1. Start with an initial ρ.
2. Solve the unconstrained problem with $f_{\text{penalty}}(x, \rho)$, or problem with simple box constraints .
3. Increase ρ and repeat until constraints are approximately satisfied.

- Converts the problem into a series of unconstrained problems or problems with simple box constraints.
- Suitable for both equality and inequality constraints.
- Large ρ can lead to numerical issues; requires careful tuning.

Subgradient Descent

Subgradient Descent Overview

- Used for non-differentiable functions where a standard gradient does not exist.
- In subgradient descent, we use a subgradient g at point x such that:
$$f(y) \geq f(x) + g^T(y - x) \quad \forall y$$
- Subgradient is usually not unique.
- The subgradient g generalizes the concept of a gradient, providing a direction for descent even if the function is not smooth.

Examples of Subgradients

1. Absolute Value Function

For $f(x) = |x|$:

$$\partial f(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ s \in [-1, 1] & \text{if } x = 0 \end{cases}$$

2. Maximum Function

For $f(x) = \max(g(x), h(x))$:

$$\partial f(x) = \begin{cases} \nabla g(x) & \text{if } g(x) > h(x) \\ \nabla h(x) & \text{if } h(x) > g(x) \\ \alpha \nabla g(x) + (1 - \alpha) \nabla h(x) & \text{if } g(x) = h(x), \alpha \in [0, 1] \end{cases}$$

Subgradient Descent

Subgradient Descent Algorithm

1. Start with an initial point $x^{(0)}$.
2. For each iteration k:
   - Compute a subgradient $g^{(k)}$ of f at $x^{(k)}$.
   - Update x using:

     $$x^{(k+1)} = x^{(k)} - \alpha^{(k)} g^{(k)}$$

     where $\alpha^{(k)}$ is a step size, typically diminishing over iterations.
3. Repeat until convergence.

Barrier Method

Barrier Method:

- Suitable for inequality constraints: $g_i(x) \leq 0$.
- Adds a barrier term to prevent the solution from approaching the boundary of the feasible region.
- New objective function:

$$f_{\text{barrier}}(x, \mu) = f(x) - \frac{1}{\mu} \sum_i \ln(-g_i(x))$$

- As $\mu \to \infty$, the barrier prevents x from violating constraints.

Algorithm:

1. Start with a small μ.
2. Solve the unconstrained problem with $f_{\text{barrier}}(x, \mu)$.
3. Gradually increase μ for retrieving original solution.

Projecting onto the Feasible Region in Log Barrier Method

Problem: When $g(x) > 0$, the point x is outside the feasible region defined by $g(x) \leq 0$.

Solution: Projecting along the Gradient Direction

- The gradient $\nabla g(x)$ points in the direction of the steepest increase in $g(x)$.
- Moving along $-\nabla g(x)$ reduces $g(x)$ and pushes x back toward the feasible region.
- Projection step for small adjustment:

$$x_{\text{new}} = x - \frac{\nabla g(x)}{\|\nabla g(x)\|^2} g(x)$$

  where $\frac{g(x)}{\|\nabla g(x)\|^2}$ scales the adjustment to ensure feasibility.

Intuition:

- $g(x_{\text{new}}) \approx g(x) - \nabla g(x)^\top \frac{\nabla g(x)}{\|\nabla g(x)\|^2} g(x) = 0$

Barrier Method

- Ideal for inequality constraints.
- Prevents boundary solutions; becomes more strict as μ increases.
- Not suitable for equality constraints; requires carefully increasing μ.
- Needs to choose stepsizes carefully: easy to fall into region $g_i(x) > 0$