Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 31

Over the past six weeks, we have studied various algorithms for solving unconstrained optimization problems, including the gradient descent method, conjugate gradient method, Fletcher-Reeves algorithm, Newton's method, and several quasi-Newton methods. Starting this week, we will transition to constrained optimization. While we will leverage some concepts from unconstrained optimization, it is important to understand that the algorithms for constrained problems differ significantly in their approach.

Let's begin by recalling the definition of a constrained optimization problem. The goal is to minimize a function f(x), where x is the decision variable, subject to the condition that x belongs to a constraint set S. Crucially, S must be a strict subset of R^n. If S were the entire space R^n, the problem would be unconstrained.

We focus on problems with a specific structure. Namely, we assume the constraint set S is defined by:

* p inequality constraints: $g_i(x) \leq 0$ for i = 1, 2, ..., p
* m equality constraints: $h_j(x) = 0$ for j = 1, 2, ..., m

Therefore, the standard form of the problem we will address is:

Minimize f(x)

subject to:

$g_i(x) \leq 0$ for all i = 1 to p

$h_j(x) = 0$ for all j = 1 to m

This structured form, introduced at the beginning of the course, encompasses a wide range of problems. Any problem whose constraints can be expressed in this way falls within the scope of our study.

A point (x^*, λ^*, μ^*) is a critical point of the constrained optimization problem if it satisfies the following set of conditions, known as the Karush-Kuhn-Tucker (KKT) conditions:

1. Stationarity Condition:

The gradient of the Lagrangian with respect to x must be zero:

$$\nabla f(x^*) + \sum \lambda_i^* \nabla g_i(x^*) + \sum \mu_j^* \nabla h_j(x^*) = 0$$

2. Primal Feasibility Conditions:

- The original inequality constraints must hold: $g_i(x^*) \leq 0$ for all $i = 1$ to p
- The original equality constraints must hold: $h_j(x^*) = 0$ for all $j = 1$ to m

3. Dual Feasibility Condition:

The multipliers for the inequality constraints must be non-negative:

$\lambda_i^* \geq 0$ for all $i = 1$ to p

4. Complementary Slackness Condition:

For each inequality constraint, the product of the multiplier and constraint must be zero:

$\lambda_i^* g_i(x^*) = 0$ for all $i = 1$ to p

A point (x^*, λ^*, μ^*) that satisfies all the KKT conditions is called a critical point or KKT point of the constrained optimization problem. This is the same concept we studied earlier in the course.

This is a brief recap of constrained optimization. This is how we find critical points analytically. If we can find solutions analytically, you might wonder why we need algorithms. As with unconstrained optimization, there are two main reasons:

1. Analytically Intractable Problems: Some problems are too complex to solve by hand. For example, consider minimizing:

$f(x_1, x_2) = x_1e^{x_2} + x_2e^{x_1}$

subject to: $x_1^2 + x_2^2 = 1$

The Lagrangian is:

$L = x_1e^{x_2} + x_2e^{x_1} + \lambda (x_1^2 + x_2^2 - 1)$

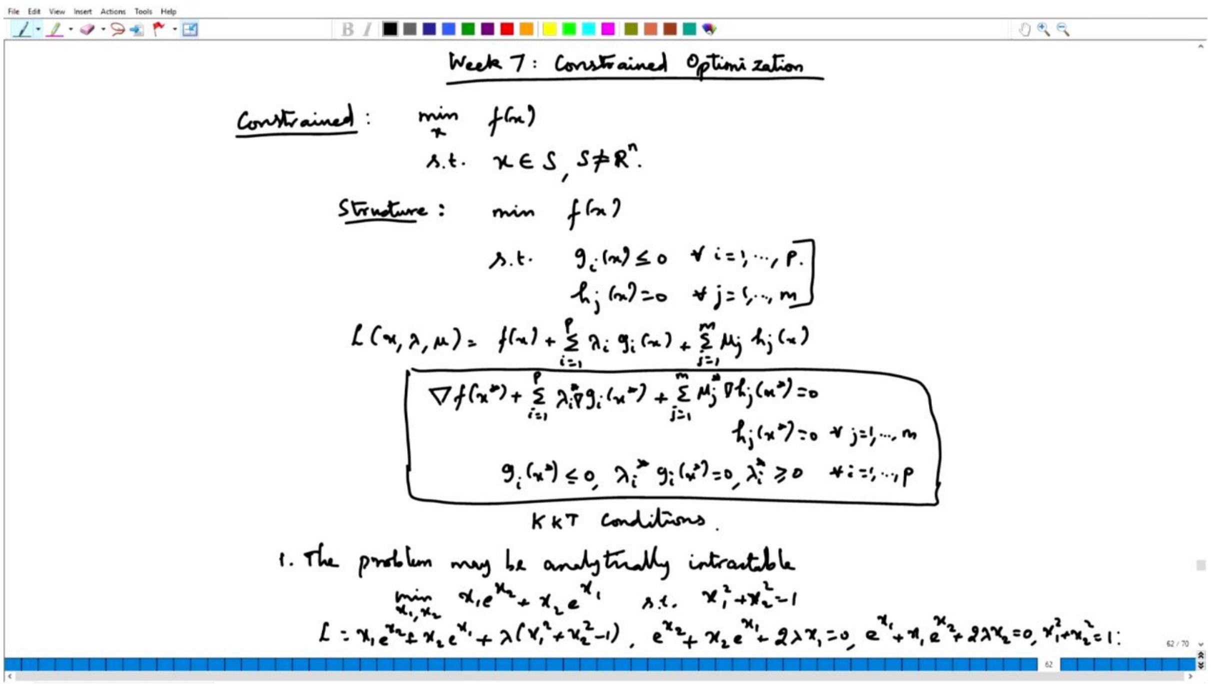
The KKT conditions lead to the system:

$e^{x_2} + x_2e^{x_1} + 2\lambda x_1 = 0$

$e^{x_1} + x_1e^{x_2} + 2\lambda x_2 = 0$

$x_1^2 + x_2^2 = 1$

(Refer Slide Time 10:20)


Week 7: Constrained Optimization

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

s.t. $x \in S$, $S \neq \mathbb{R}^n$.

Structure: $\min f(x)$

$$\left.\begin{array}{ll} \text{s.t.} \quad g_i(x) \leq 0 & \forall i = 1, \dots, p. \\ h_j(x) = 0 & \forall j = 1, \dots, m \end{array}\right]$$

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{p} \lambda_i g_i(x) + \sum_{j=1}^{m} \mu_j h_j(x)$$

$$\boxed{\begin{array}{r} \nabla f(x^*) + \sum_{i=1}^{p} \lambda_i^* \nabla g_i(x^*) + \sum_{j=1}^{m} \mu_j^* \nabla h_j(x^*) = 0 \\ h_j(x^*) = 0 \quad \forall j = 1, \dots, m \\ g_i(x^*) \leq 0, \ \lambda_i^* g_i(x^*) = 0, \ \lambda_i^* \geq 0 \quad \forall i = 1, \dots, p \end{array}}$$

KKT conditions.

1. The problem may be analytically intractable

$$\min_{x_1, x_2} x_1 e^{x_2} + x_2 e^{x_1} \quad \text{s.t.} \quad x_1^2 + x_2^2 = 1$$

$$L = x_1 e^{x_2} + x_2 e^{x_1} + \lambda(x_1^2 + x_2^2 - 1), \quad e^{x_2} + x_2 e^{x_1} + 2\lambda x_1 = 0, \quad e^{x_1} + x_1 e^{x_2} + 2\lambda x_2 = 0, \quad x_1^2 + x_2^2 = 1:$$

62 / 70

This system of equations is too complex to solve analytically, requiring iterative numerical methods.

2. Large-Scale Problems: In many real-world applications, the size of the problem makes analytical solution impossible. A prime example is the Support Vector Machine (SVM) problem in machine learning.

Given labeled data $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ where $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$, the separable SVM problem is:

Minimize: $\frac{1}{2}\|w\|^2$

Subject to: $y_i(w^T x_i + b) \geq 1$ for all $i = 1$ to n

Due to the complexity and scale of real-world problems, we must develop computational algorithms for constrained optimization. Before discussing specific algorithms, it's important to understand the two main classifications of these problems:

1. Linear Programming (LP)

A problem is called linear if all of the following are linear functions of the variable x:

* Objective Function: $f(x) = c^T x$ (e.g., $c_1 x_1 + c_2 x_2 + \dots$)

* Inequality Constraints: $g_i(x) = a_i^T x - b_i \leq 0$

* Equality Constraints: $h_j(x) = \bar{a}_j^T x - \bar{b}_j = 0$

The standard form of a Linear Programming problem is:

Minimize: $c^T x$

Subject to: $a_i^T x - b_i \leq 0$ for all $i = 1$ to p

$\bar{a}_j^T x - \bar{b}_j = 0$ for all $j = 1$ to m

2. Nonlinear Programming (NLP)

A problem is called nonlinear if any one of the following is a nonlinear function:

* The objective function $f(x)$
* Any inequality constraint $g_i(x)$
* Any equality constraint $h_j(x)$

This classification is crucial for two reasons:

1. Solution Structure: Linear problems have a very well-defined structure and properties that lead to efficient solution methods (like the Simplex method).

2. Algorithm Applicability: The algorithms designed for linear problems (LP) do not work for nonlinear problems (NLP), and vice-versa. They require completely different approaches.

Therefore, we will study algorithms for these two classes separately. We will begin with nonlinear programming problems. We will focus on two key algorithms:

* The Quadratic Penalty Method
* The Augmented Lagrangian Method

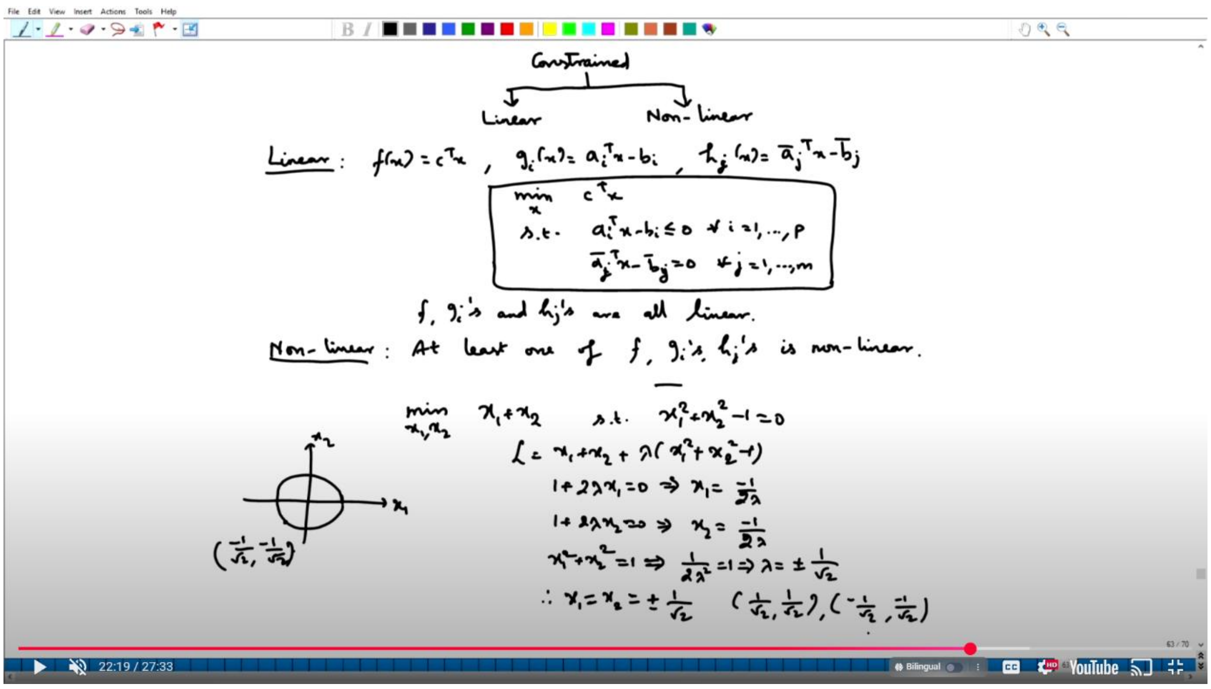
We will begin our study of algorithms with nonlinear programming problems, where at least one function (objective or constraint) is nonlinear.

Recall that for unconstrained optimization, we had a straightforward iterative method: start at a point, find a descent direction and step size, move to the new point, and repeat. This approach does not work directly for constrained optimization. The fundamental challenge is the presence of constraints.

Consider a simple example:

Minimize $f(x_1, x_2) = x_1 + x_2$ subject to the constraint $x_1^2 + x_2^2 - 1 = 0$ (i.e., the solution must lie on the unit circle). The solution to this problem is the point $(-1/\sqrt{2}, -1/\sqrt{2})$.

(Refer Slide Time 22:19)


$$\begin{tikzcd} & \text{Constrained} \arrow[dl] \arrow[dr] & \\ \text{Linear} & & \text{Non-linear} \end{tikzcd}$$

Linear: $f(x) = c^T x$, $g_i(x) = a_i^T x - b_i$, $h_j(x) = \bar{a}_j^T x - \bar{b}_j$

$$\boxed{\begin{aligned} \min_x \quad & c^T x \\ \text{s.t.} \quad & a_i^T x - b_i \leq 0 \quad \forall i = 1, \ldots, p \\ & \bar{a}_j^T x - \bar{b}_j = 0 \quad \forall j = 1, \ldots, m \end{aligned}}$$

f, g_i's and h_j's are all linear.

Non-linear: At least one of f, g_i's, h_j's is non-linear.

$$\min_{x_1, x_2} \quad x_1 + x_2 \quad \text{s.t.} \quad x_1^2 + x_2^2 - 1 = 0$$

$$\mathcal{L} = x_1 + x_2 + \lambda(x_1^2 + x_2^2 - 1)$$

$$1 + 2\lambda x_1 = 0 \Rightarrow x_1 = \frac{-1}{2\lambda}$$

$$1 + 2\lambda x_2 = 0 \Rightarrow x_2 = \frac{-1}{2\lambda}$$

$$x_1^2 + x_2^2 = 1 \Rightarrow \frac{1}{2\lambda^2} = 1 \Rightarrow \lambda = \pm \frac{1}{\sqrt{2}}$$

$$\therefore x_1 = x_2 = \pm \frac{1}{\sqrt{2}} \quad \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right), \left(-\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}\right)$$

If we try to apply an unconstrained method:

1. Start at a point on the circle.
2. Calculate a descent direction (e.g., the negative gradient of f(x), which is (-1, -1)).
3. Take a step in that direction.
4. You will immediately move off the circle, violating the constraint.

The core challenge in constrained optimization is that the straightforward approach from unconstrained problems fails. In an unconstrained method, you:

1. Start at a point.
2. Find a descent direction (e.g., the negative gradient).
3. Choose a step size.
4. Move to the new point.

However, in a constrained problem, taking a step in a descent direction will almost certainly move you outside the feasible region defined by the constraints. For example, if your constraint is a circle, a step in the descent direction will leave the circle's boundary.

The problem becomes exponentially harder in higher dimensions. In a 5-dimensional space, you cannot visualize the constraint set. There is no simple way to "see" where you are or how to project a point outside the feasible region back onto it.

Therefore, we need a new strategy. The goal is to somehow incorporate the constraints g_i(x) and $h_j(x)$ directly into the optimization process. We need to modify the problem so that we can use a similar iterative approach (finding a direction and step size) but in a way that automatically respects the constraints or penalizes violations. This is the fundamental idea behind algorithms for constrained optimization.

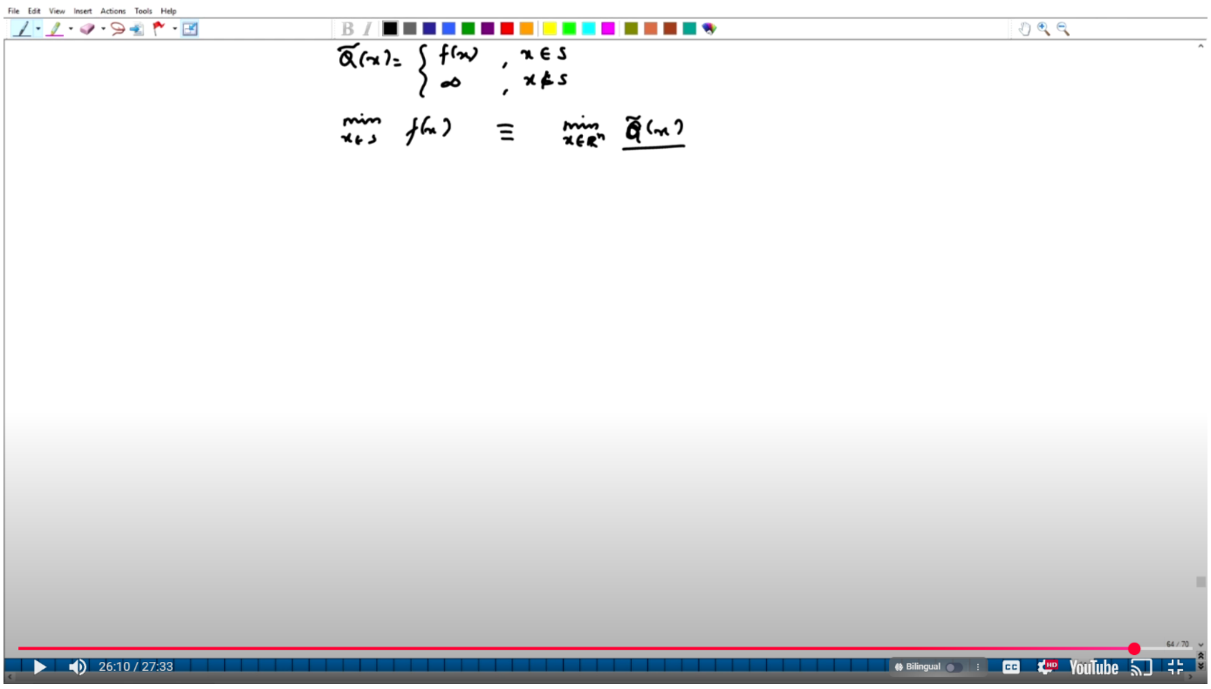
One intuitive idea to handle constraints is to define a new function, let's call it Q̃(x). This function is defined as:

Q̃(x) = f(x) if x is within the feasible constraint set S

Q̃(x) = infinity if x is outside the feasible set S

This definition creates an equivalence: the original problem of minimizing f(x) subject to x in S is now identical to the new problem of minimizing Q̃(x) without any constraints. This is because the moment an algorithm tries to step outside S, the function value becomes infinitely bad, forcing it to stay within the feasible region.

(Refer Slide Time 26:20)



$$\tilde{Q}(x) = \begin{cases} f(x) & , x \in S \\ \infty & , x \notin S \end{cases}$$

$$\min_{x \in S} f(x) \equiv \min_{x \in \mathbb{R}^n} \underline{\tilde{Q}(x)}$$

However, this conceptual solution has a major practical flaw: differentiability. The function Q̃(x) is not differentiable—its value jumps to infinity abruptly at the boundary of the feasible set S.

This is a critical problem because all the powerful unconstrained optimization algorithms we've studied (like gradient descent, Newton's method) rely heavily on the function being smooth and differentiable. They use gradients and Hessians to find descent directions and determine step sizes. A non-differentiable function like Q̃(x) breaks these methods.

Therefore, the key challenge is to transform this idea into a practical algorithm. We need to find a way to approximate this ideal but non-differentiable function Q̃(x) with a smooth and differentiable function. This new function should effectively penalize constraint violations but in a gradual, differentiable way, allowing us to use our familiar unconstrained optimization techniques.

We will explore how to create such a smooth, tractable function in the next lecture. Thank You