Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Indian Institute of Technology Kanpur

Lecture: 11

Hello everyone. This is the third week of the course, and we will begin studying optimization algorithms.

Recall that optimization problems are broadly classified into unconstrained and constrained problems. For an unconstrained optimization problem, we consider a twice-differentiable function f: $\mathbb{R}^n \rightarrow \mathbb{R}$.
The problem is to minimize f(x) subject to $x \in \mathbb{R}^n$.
Since we know how to solve unconstrained problems analytically, a natural question arises: why do we need optimization algorithms?

There are two main reasons.

The first reason is that not all problems can be solved analytically. We use the term "analytically intractable." For example, consider the univariate function
$f(x) = (x - 1)e^x - x$.
To solve this analytically, you differentiate and set the derivative to zero. The derivative is $f'(x) = (x - 1)e^x + e^x - 1 = x e^x - 1$.
Setting this to zero gives $x e^x = 1$.
The solution for $x e^x = 1$ is not known and cannot be solved analytically. Some might know this as the Lambert W function of 1, but this is not a usual function. Therefore, this problem cannot be solved analytically. You cannot provide an x for which $x e^x = 1$.

In a similar way, consider another function from R to R: $f(x) = x^2 + e^x$.
In this case, $f'(x) = 2x + e^x$,
which needs to be zero, and this again cannot be solved analytically. Note that these are all univariate functions. Imagine such problems in two or three dimensions.
For example, consider a function f: $\mathbb{R}^2 \rightarrow \mathbb{R}$ where $f(x_1, x_2) = x_1 e^{x_2} + x_2 e^{x_1}$.
To solve this analytically, you must solve the following equations simultaneously:
$e^{x_2} + x_2 e^{x_1} = 0$ and $e^{x_1} + x_1 e^{x_2} = 0$.

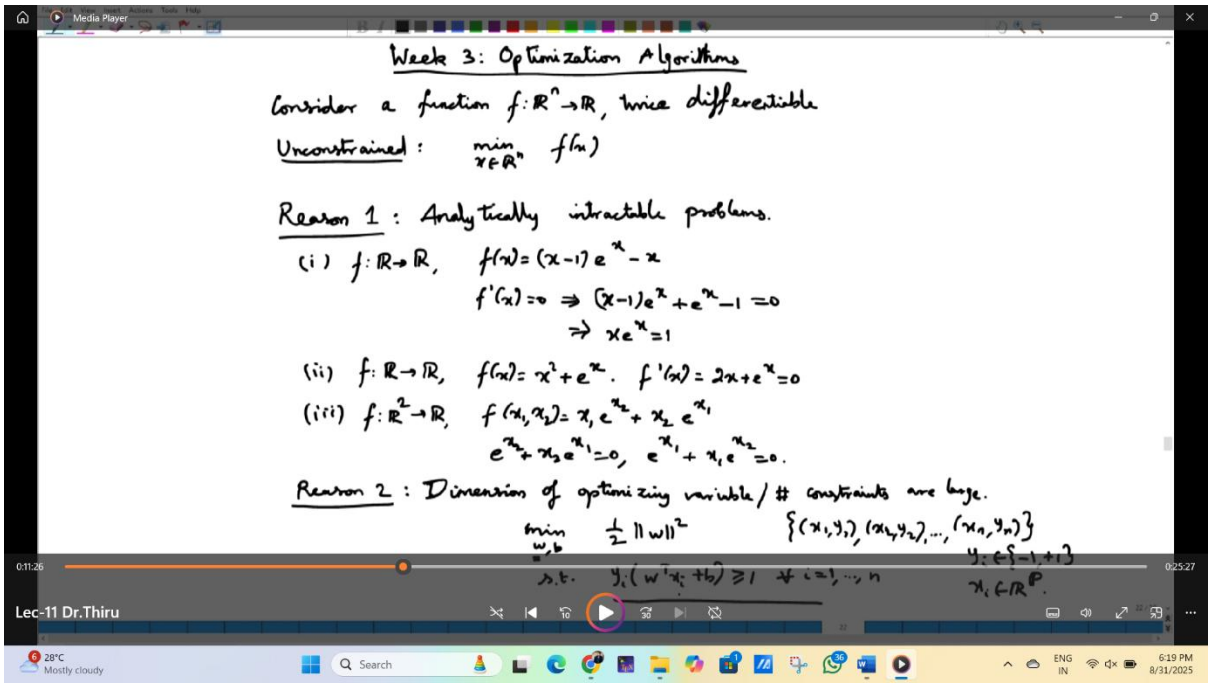
Solving even one of these equations is difficult. Solving two simultaneously cannot be done analytically. These problems are called analytically intractable. They can be addressed using other methods, such as computational techniques, but not analytically. In practice, we have many problems of this sort.

That is one main reason why we use optimization algorithms and do not stop at analytical solutions.

There is another important reason more suited for today's scenario. Even if problems are analytically tractable, they could be large. For example, the dimension of the optimizing variable or the number of constraints could be large. The problems we solve analytically are often small, like two or three-dimensional examples.
We never take a ten-dimensional problem and try to solve it analytically. But in today's scenario, especially after the advent of machine learning, we have problems with hundreds of features. The starting problem often has a dimension of 100 or much more. The number of constraints is also huge.
In big data, the number of data points is massive. Solving these analytically is out of the question. We want to give all optimization problems to a computer.

For example, consider the support vector machine problem from the first lecture.
You minimize $\frac{1}{2}\|w\|^2$ with respect to w and b, subject to $y_i(w^T x_i + b) \geq 1$ for i = 1 to n.
You have n data points: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), where y_i is either -1 or +1, and x_i is in $\mathbb{R}^p$.

(Refer Slide Time 11:26)



The dimension of w is p, which is usually tens or hundreds, and b is one-dimensional. If p is 10 and n is 1000, you have 1000 data points, each of 10 dimensions. The dimension of the optimizing variable is 11, and the number of constraints is 1000.
Imagine solving a problem of this scale with 11 variables and 1000 constraints. This is what we face in the current day. Therefore, we want to outsource this work to a computer because solving it by pen and paper is impossible.

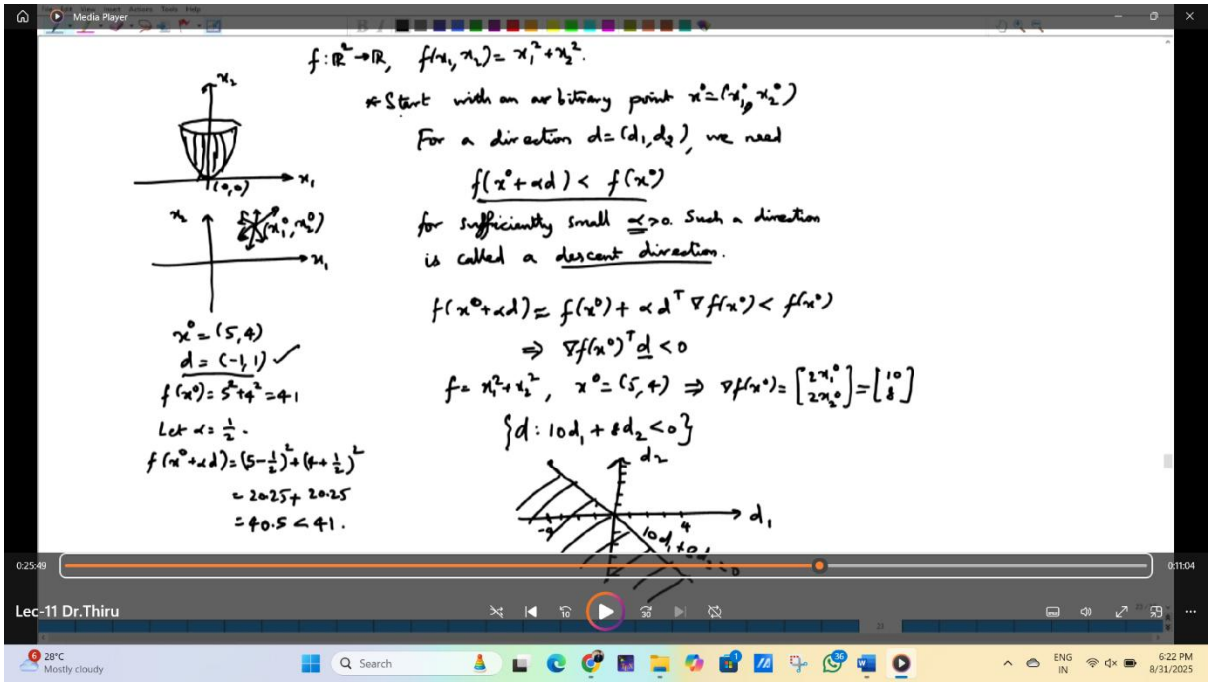
These are two basic reasons for moving to an algorithmic framework for optimization problems, leaving out the mathematical or analytical framework. This is sufficient motivation.

We will now try to understand how to construct an algorithm. Let us start with a simple example.
Let f be a function from $\mathbb{R}^2$ to $\mathbb{R}$, and $f(x_1, x_2) = x_1^2 + x_2^2$.
This is a simple problem because the minimizer is at (0, 0).
We use this example to explain how the algorithm works.
The function looks like an open bowl facing upwards in three dimensions, with the minimum at (0, 0).
For now, assume we do not know where the minimum is. How do we frame an algorithm to find it?

Start with an arbitrary point, $x^0 = (x_1^0, x_2^0)$.
Since we are in two dimensions, (x_1^0, x_2^0) is a point in the plane.
From here, we can move in any direction.
The first way to choose a direction is to see if the function value decreases.
Let us choose a direction $d = (d_1, d_2)$.
We need $f(x^0 + \alpha d) < f(x^0)$ for sufficiently small $\alpha > 0$.

For example, let $x^0 = (5, 4)$.
We can choose any direction, such as $d = (-1, 1)$.
First, find $f(x^0) = 5^2 + 4^2 = 41$.
If we move from (5, 4) in the direction (-1, 1), the function value must decrease for some small α.
It may not decrease for all α, but for some small α it must. We do not want to increase the function value because we want to minimize f.

(Refer Slide Time 25:50)



Take $\alpha = 0.5$.
Then $f(x^0 + \alpha d) = (5 - 0.5)^2 + (4 + 0.5)^2 = 4.5^2 + 4.5^2 = 20.25 + 20.25 = 40.5$,
which is less than 41. So, this direction satisfies
$f(x^0 + \alpha d) < f(x^0)$
for sufficiently small α. Such a direction is called a descent direction.
The function value decreases when moving from x^0 along d.

Our first task is to find a descent direction for f at x^0.
How do we find it? We have $f(x^0 + \alpha d) < f(x^0)$.
Using the Taylor expansion, $f(x^0 + \alpha d) \approx f(x^0) + \alpha \nabla f(x^0)^T d$.
We want this to be less than $f(x^0)$.
Since α is positive, we need $\nabla f(x^0)^T d < 0$.
If the dot product of the direction and the gradient is negative, then it is a descent direction.

In our example, $f(x_1, x_2) = x_1^2 + x_2^2$ and $x^0 = (5, 4)$.
Then $\nabla f(x^0) = (10, 8)$.
Any d that satisfies $10d_1 + 8d_2 < 0$ is a descent direction.
For instance, $d = (-1, 1)$ gives $-10 + 8 = -2 < 0$.
If we plot the line $10d_1 + 8d_2 = 0$,
any direction on one side of this line is a descent direction.
For example, (-1, 1) lies on the descent side.

Now, we need to choose a step size α.
For the same example, with $d = (-1, 1)$, we have
$f(x^0 + \alpha d) - f(x^0) = (5 - \alpha)^2 + (4 + \alpha)^2 - 41 = 25 - 10\alpha + \alpha^2 + 16 + 8\alpha + \alpha^2 - 41 = 2\alpha^2 - 2\alpha = 2\alpha(\alpha - 1)$.

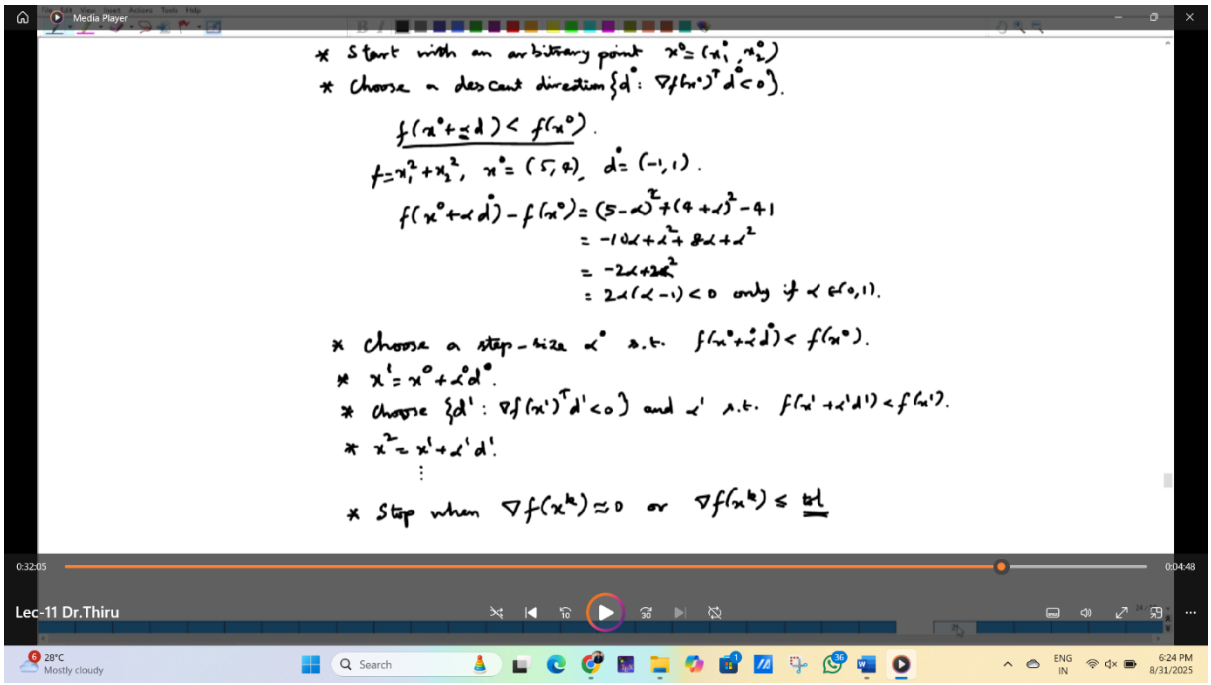
This is less than zero only when α is in (0, 1).
So we must choose α strictly less than 1.

This shows that it is not enough to choose a descent direction; we must also choose a step size α such that $f(x^0 + \alpha d) < f(x^0)$.
So, we start with an arbitrary point, choose a descent direction, choose a step size, and then update $x^1 = x^0 + \alpha d$.
We can denote the direction and step as d^0 and α^0.

This gives a new point x^1, but it is not the final answer.
We repeat the procedure: choose d^1 such that $\nabla f(x^1)^T d^1 < 0$, choose α^1 such that $f(x^1 + \alpha^1 d^1) < f(x^1)$, then set $x^2 = x^1 + \alpha^1 d^1$, and so on.

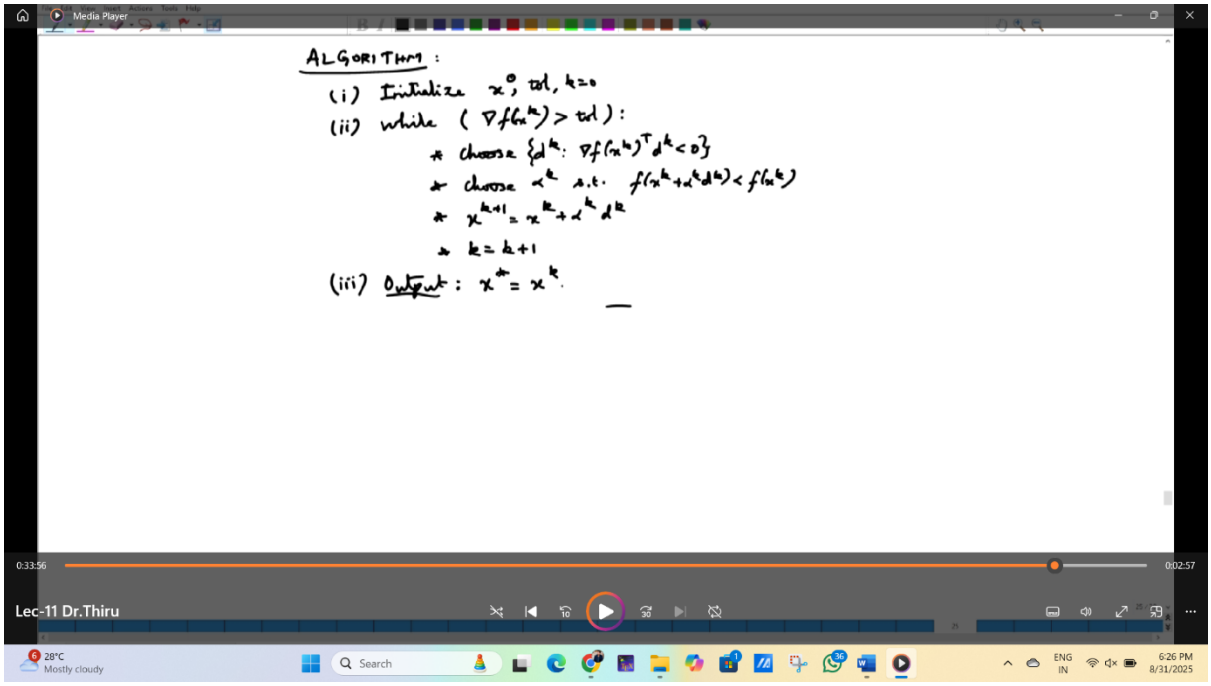
When do we stop? For this example, we know the answer is (0, 0), but in general, we do not.
At the critical point, the gradient is zero.
So we stop when $\nabla f(x^k)$ is close enough to zero.
Practically, we choose a tolerance, say 10^{-6} or 10^{-8}.
When $\|\nabla f(x^k)\|$ is less than or equal to the tolerance, we stop.

(Refer Slide Time 32:05)



Putting this together, the algorithm is as follows.
Initialize x^0. Set $k = 0$.
While $\|\nabla f(x^k)\| >$ tolerance, choose d^k such that $\nabla f(x^k)^T d^k < 0$, choose α^k such that $f(x^k + \alpha^k d^k) < f(x^k)$, set $x^{k+1} = x^k + \alpha^k d^k$, and then set $k = k + 1$.
The output is $x^* = x^k$.
When the gradient is less than or equal to the tolerance, we exit the loop and output the result.

(Refer Slide Time 33:56)



This is a formal algorithm.

To explain what this means practically, imagine you are starting from a village and want to go to another village 100 kilometers away without knowing the route. You ask for directions. Someone tells you to go in a certain direction for 2 kilometers.

You go there and ask again. This repeats until you are very near the destination, and someone tells you to go 100 meters, and you see a signboard.

The last person gives you the stopping criterion. Similarly, at each step, the algorithm gives a direction and a step size. After many iterations, you stop when the gradient is within tolerance. This is the basis of any optimization algorithm we will study in this course.

We will continue in the next lecture. Thank you.