Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 12

Hello, everyone. This is the second lecture of Week 3. Recall that in the previous lecture, we began with the motivation for optimization algorithms and presented a framework for a typical optimization algorithm.
Note: $x_k = x^k$, $d_k = d^k$, $\alpha_k = \alpha^k$
The basic algorithmic structure is as follows:

1. Initialize a point x_0.
2. For each iteration $k = 0, 1, 2, \ldots$:
   * Choose a descent direction d_k.
   * Choose an appropriate step size $\alpha_k > 0$.
   * Update: $x_{k+1} = x_k + \alpha_k d_k$.
3. Stop when a stopping criterion is met (e.g., $\|\nabla f(x_k)\| \leq \varepsilon$).

This process is analogous to using a navigation app: it provides a direction (d_k) and a distance to travel (α_k) in each step, and finally confirms when you have reached your destination (the stopping criterion).

Different optimization methods specify how to choose the descent direction d_k and the step size α_k. This lecture will focus on strategies for selecting the step size α_k.

The Challenge of Choosing the Step Size

Merely instructing an algorithm to "choose α_k" is inadequate. For a given point x_k, there are often infinitely many possible descent directions d_k.
Similarly, for a given x_k and d_k, there is often an interval of α values that satisfy the descent condition
$f(x_k + \alpha d_k) < f(x_k)$.
We need a specific, automated way to choose a single value α_k.

Algorithms that search for this step size are called line search algorithms. The core idea is that the chosen step size should satisfy:
$f(x_k + \alpha_k d_k) < f(x_k)$

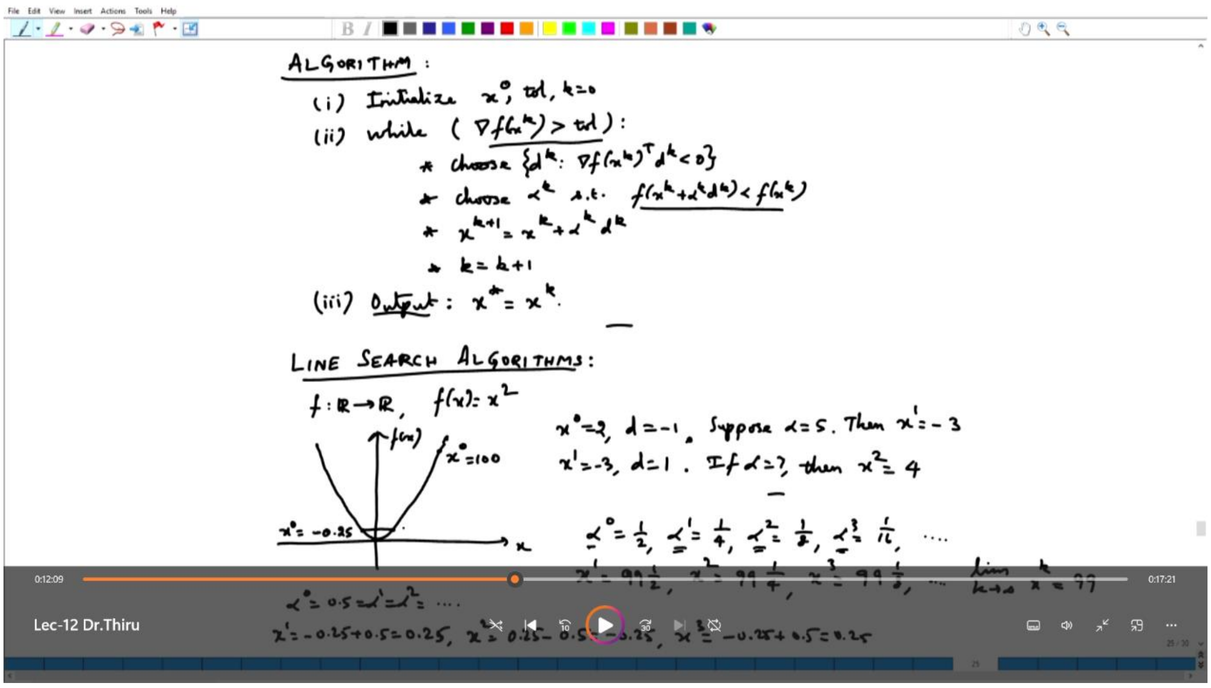
Let's consider what happens with poor choices of α:

1. Arbitrarily Large α:
   Consider $f(x) = x^2$, $x_0 = 2$, $d_0 = -1$ (the correct descent direction).
   If we choose $\alpha = 5$, then:
   $x_1 = x_0 + \alpha d_0 = 2 + 5\times(-1) = -3$.
   Now, from $x_1 = -3$, the descent direction is $d_1 = +1$.
   If we again choose a large step, say $\alpha = 7$:
   $x_2 = x_1 + \alpha d_1 = -3 + 7\times1 = 4$.
   The algorithm oscillates away from the optimum ($x=0$).

2. Arbitrarily Small α:
   Consider $f(x) = x^2$, $x_0 = 100$, $d_0 = -1$.
   If we choose a sequence of diminishing steps, e.g., $\alpha_0 = 1/2$, $\alpha_1 = 1/4$, $\alpha_2 = 1/8$, ..., then:
   $x_1 = 100 - 1/2 = 99.5$
   $x_2 = 99.5 - 1/4 = 99.25$
   $x_3 = 99.25 - 1/8 = 99.125$
   ...
   The sequence converges to $x_k \to 99$, not to the optimum (0). Progress is too slow.

3. Constant Step Size:
   Consider $f(x) = x^2$ and a constant step size $\alpha = 0.5$ for all iterations.
   If we start at $x_0 = -0.25$, the descent direction is $d_0 = +1$ (toward 0).
   $x_1 = -0.25 + 0.5\times1 = 0.25$.

(Refer Slide Time 12:09)



ALGORITHM :

```
(i)  Initialize  $x^0$, tol, $k=0$
(ii) while ( $\nabla f(x^k) > tol$ ):
          * choose $\{d^k : \nabla f(x^k)^T d^k < 0\}$
          * choose $\alpha^k$ s.t. $f(x^k + \alpha^k d^k) < f(x^k)$
          * $x^{k+1} = x^k + \alpha^k d^k$
          * $k = k+1$
(iii) Output : $x^* = x^k$.
```

LINE SEARCH ALGORITHMS :

$f : \mathbb{R} \to \mathbb{R}$, $f(x) = x^2$

$x^0 = 2$, $d = -1$. Suppose $\alpha = 5$. Then $x^1 = -3$

$x^1 = -3$, $d = 1$. If $\alpha = 7$, then $x^2 = 4$

$\alpha^0 = \frac{1}{2}$, $\alpha^1 = \frac{1}{4}$, $\alpha^2 = \frac{1}{8}$, $\alpha^3 = \frac{1}{16}$, ...

$x^1 = 99\frac{1}{2}$, $x^2 = 99\frac{1}{4}$, $x^3 = 99\frac{1}{8}$, ... $\lim_{k\to\infty} x^k = 99$

$\alpha^0 = 0.5 = \alpha^1 = \alpha^2 = \dots$

$x^1 = -0.25 + 0.5 = 0.25$, $x^2 = 0.25 - 0.5 = -0.25$, $x^3 = -0.25 + 0.5 = 0.25$

Now, from $x_1 = 0.25$, the descent direction is $d_1 = -1$.
$x_2 = 0.25 + 0.5\times(-1) = -0.25$.
The algorithm oscillates indefinitely between -0.25 and 0.25, never converging.

These examples demonstrate that the step size cannot be too large, too small, or constant across all iterations. We need a principled method.

Two Line Search Algorithms

1. Exact Line Search

Definition: For a given x_k and d_k, define the function $h(\alpha) = f(x_k + \alpha d_k)$. The exact line search chooses α_k as the minimizer of this univariate function:

$\alpha_k = \text{argmin}\_\{\alpha>0\}\ h(\alpha)$

Example: Let $f(x) = x_1^2 + x_2^2$, $x_0 = (5, 4)$, $d_0 = (-1, 1)$.

$h(\alpha) = f(x_0 + \alpha d_0) = (5 - \alpha)^2 + (4 + \alpha)^2$
$= 25 -10\alpha + \alpha^2 + 16 + 8\alpha + \alpha^2$
$= 41 - 2\alpha + 2\alpha^2$

To find the minimum, set the derivative to zero:

$h'(\alpha) = -2 + 4\alpha = 0 \Rightarrow \alpha = 1/2$

Check second derivative: $h''(\alpha) = 4 > 0$, confirming a minimum.
Therefore, $\alpha_0 = 1/2$.

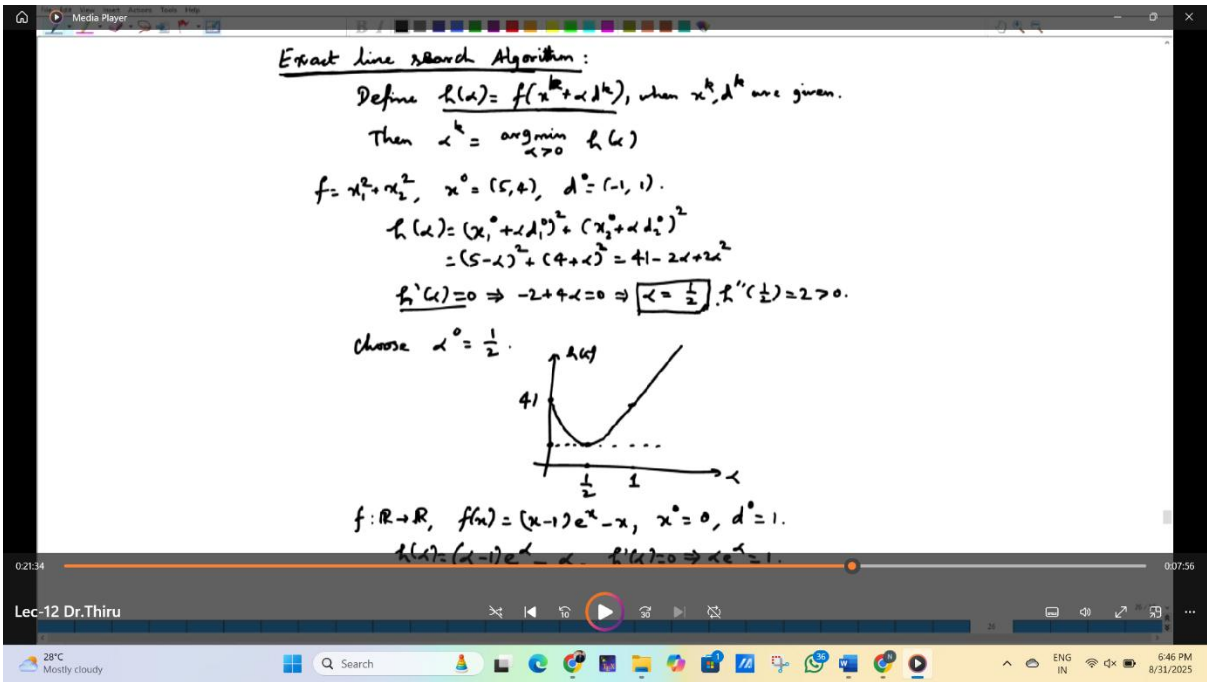
Advantage: It provides the best possible step size for the given direction, maximizing reduction in f.
Disadvantage: It requires solving an optimization subproblem in every iteration, which can be computationally expensive or analytically intractable.

Example of intractability: Consider $f(x) = (x-1)e^x - x$, $x_0 = 0$, $d_0 = 1$.

$h(\alpha) = f(x_0 + \alpha d_0) = f(\alpha) = (\alpha-1)e^\alpha - \alpha$

Finding the minimum requires solving $h'(\alpha) = \alpha e^\alpha = 1$. The solution is $\alpha=1$. In practice, more complex functions lead to equations that cannot be solved analytically.

(Refer Slide Time 21:04)



Exact line search Algorithm:

Define $h(\alpha) = f(x^k + \alpha d^k)$, when x^k, d^k are given.

Then $\alpha^k = \underset{\alpha > 0}{\arg\min}\ h(\alpha)$

$f = x_1^2 + x_2^2$, $x^0 = (5,4)$, $d^0 = (-1,1)$.

$h(\alpha) = (x_1^0 + \alpha d_1^0)^2 + (x_2^0 + \alpha d_2^0)^2$

$= (5-\alpha)^2 + (4+\alpha)^2 = 41 - 2\alpha + 2\alpha^2$

$h'(\alpha) = 0 \Rightarrow -2 + 4\alpha = 0 \Rightarrow \boxed{\alpha = \frac{1}{2}}.\ h''(\tfrac{1}{2}) = 2 > 0.$

Choose $\alpha^0 = \frac{1}{2}$.

$f: \mathbb{R} \to \mathbb{R}$, $f(x) = (x-1)e^x - x$, $x^0 = 0$, $d^0 = 1$.

$h(\alpha) = (\alpha - 1)e^\alpha - \alpha$. $h'(\alpha) = 0 \Rightarrow \alpha e^\alpha = 1$.

2. Backtracking Line Search (Armijo Line Search)

This is an inexact method that ensures sufficient decrease in f without finding the exact minimum.

Algorithm:

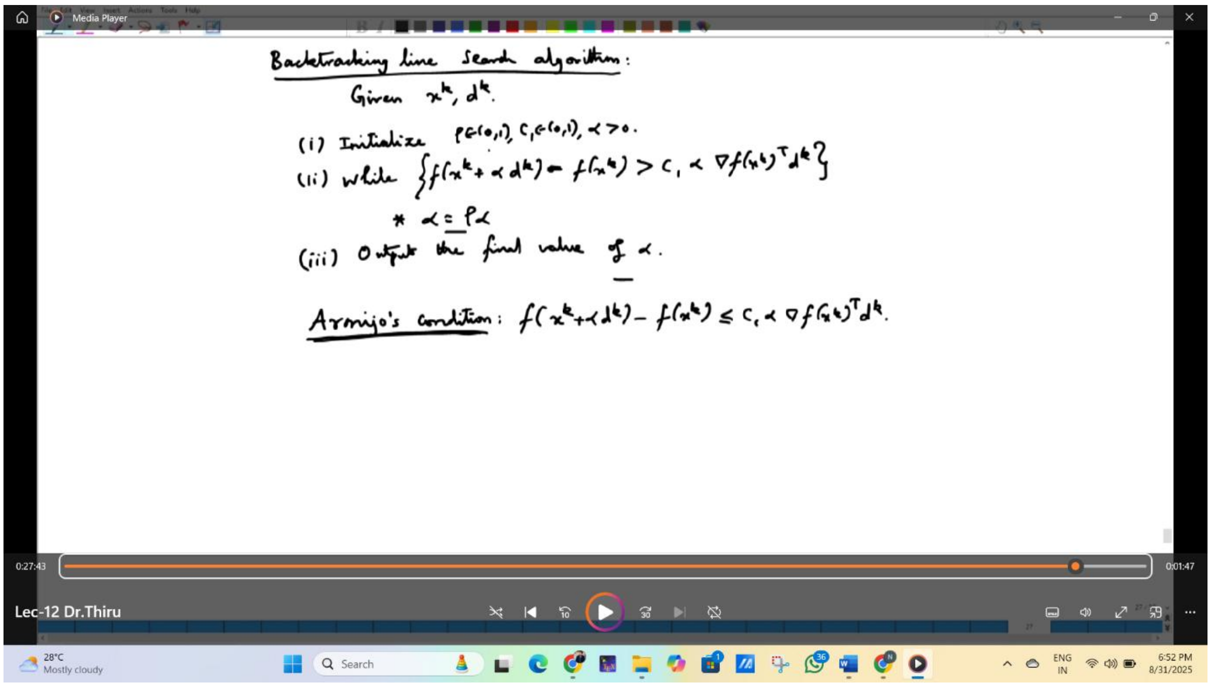
Input: Current point x_k, descent direction d_k, parameters $\rho \in (0,1)$, $c_1 \in (0,1)$, and initial step size α^s (often $\alpha^s=1$).

```
Set $\alpha = \alpha^s$.
While the Armijo condition is NOT satisfied:
    $f(x_k + \alpha d_k) > f(x_k) + c_1 \alpha \nabla f(x_k)^T d_k$
    Reduce the step size: $\alpha = \rho \alpha$
End While
Output: $\alpha_k = \alpha$
```

The Armijo condition is:

$$f(x_k + \alpha d_k) \le f(x_k) + c_1 \alpha \nabla f(x_k)^T d_k$$

(Refer Slide Time 27:43)



Backtracking line Search algorithm:

Given x^k, d^k.

(i) Initialize $\rho \in (0,1)$, $c_1 \in (0,1)$, $\alpha > 0$.

(ii) while $\{f(x^k + \alpha d^k) - f(x^k) > c_1 \alpha \nabla f(x^k)^T d^k\}$

$* \ \alpha = \rho \alpha$

(iii) Output the final value of α.

Armijo's condition: $f(x^k + \alpha d^k) - f(x^k) \leq c_1 \alpha \nabla f(x^k)^T d^k$.

This condition ensures that the decrease in f is proportional to the step size and the directional derivative. The parameter ρ is the reduction factor (e.g., 0.5 or 0.9), and c_1 controls the required decrease (e.g., 10^{-4}).

How it works: Start with a trial step size (α=1). If it yields sufficient decrease (satisfies Armijo), accept it. If not, reduce the step size by multiplying by ρ and check again. Repeat until the condition is met.

Example: Let ρ=0.5, c_1=0.4, initial α=1.
Iteration 1: Check if $f(x_k + d_k) \leq f(x_k) + 0.4\ \nabla f(x_k)^T\ d_k$. If not, set $\alpha = 0.5$.
Iteration 2: Check if $f(x_k + 0.5\ d_k) \leq f(x_k) + 0.2\ \nabla f(x_k)^T\ d_k$. If not, set $\alpha = 0.25$.
Iteration 3: Check condition for α=0.25. If satisfied, output α_k=0.25.

Advantage: Computationally cheap per iteration. It only requires function evaluations, not derivatives for the check (though the derivative $\nabla f(x_k)$ is needed for the condition itself).
Disadvantage: The step size may be conservative, potentially leading to more iterations than exact line search.

Why the Armijo condition works and how to choose parameters will be discussed in the next lecture.

Thank you.