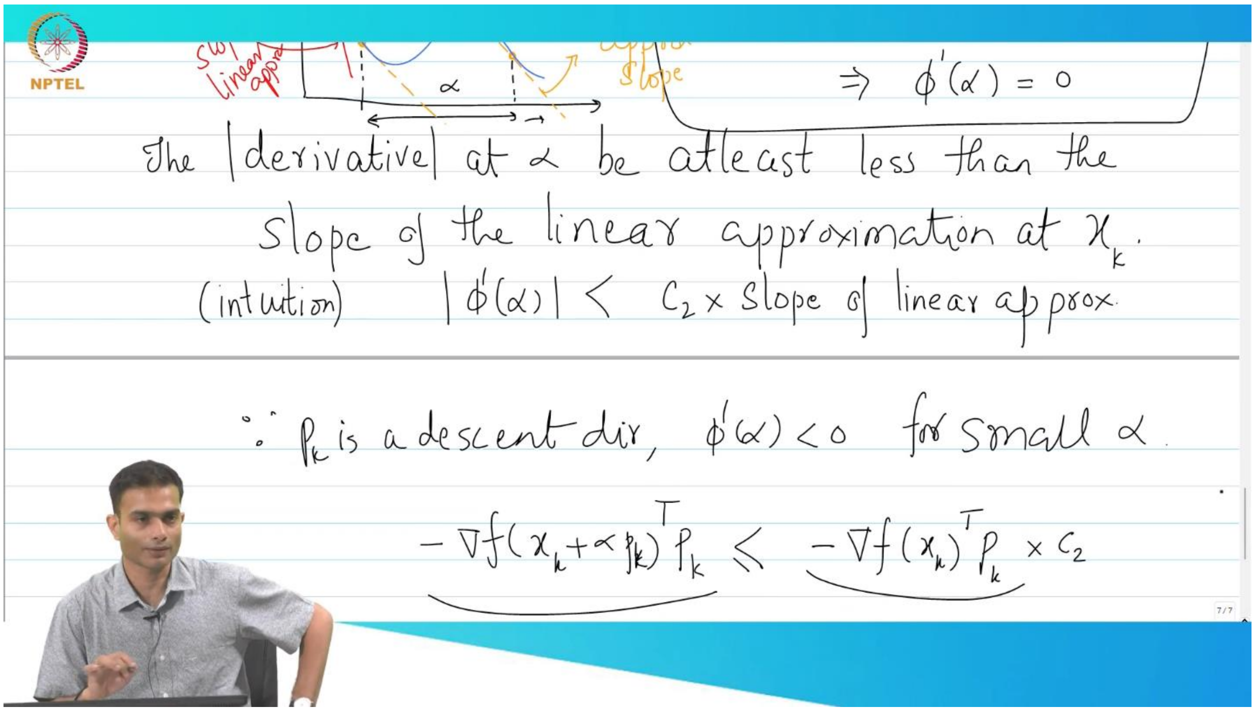
Course Name: Optimization Theory and Algorithms
Professor Name: Dr. Uday K. Khankhoje
Department Name: Electrical Engineering
Institute Name: Indian Institute of Technology Madras
Week - 04
Lecture - 25

Strong Wolfe Conditions

The **Strong Wolfe Conditions** are an enhancement over the basic Wolfe conditions, aimed at providing more robust control in line search methods for optimization. Let’s break down the key points discussed here:



1. Slope and Descent Direction

The function’s slope, $\phi'(\alpha)$, is negative for small values of α when searching in a descent direction, as $\phi'(\alpha)$ relates to the gradient:

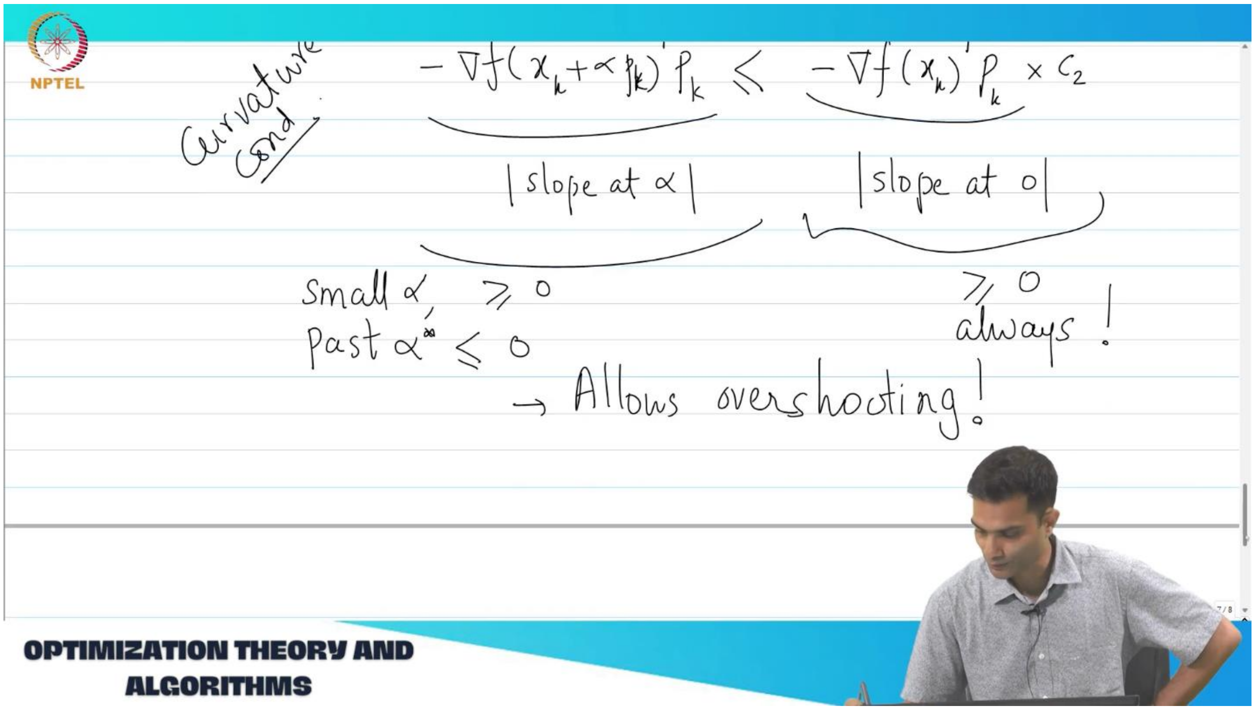
$$\phi'(\alpha) < 0 \quad \text{for small } \alpha \text{ since } p_k \text{ is a descent direction.}$$

2. Need for Strong Wolfe Condition

The basic Wolfe condition imposes restrictions on the slope, but a problem arises once α surpasses a stationary point. Beyond this point, $\phi'(\alpha)$ changes sign, allowing the basic Wolfe condition to be trivially satisfied (which could lead to overshooting).

The solution is to replace the negative slope constraint with a modulus of the slope, leading to the **Strong Wolfe Condition**, which ensures that the slope at any step does not overshoot beyond acceptable bounds.

3. Curvature Condition



The curvature condition is modified by applying a modulus to the slope, thereby creating a constraint that better manages step sizes and prevents overshooting, particularly past local maxima or stationary points.

4. Range of Acceptable Alpha

The **Strong Wolfe Condition** reduces the acceptable range of α values compared to the basic Wolfe conditions. This is necessary to avoid overshooting, especially in non-convex functions with multiple local maxima.

5. Computational Cost

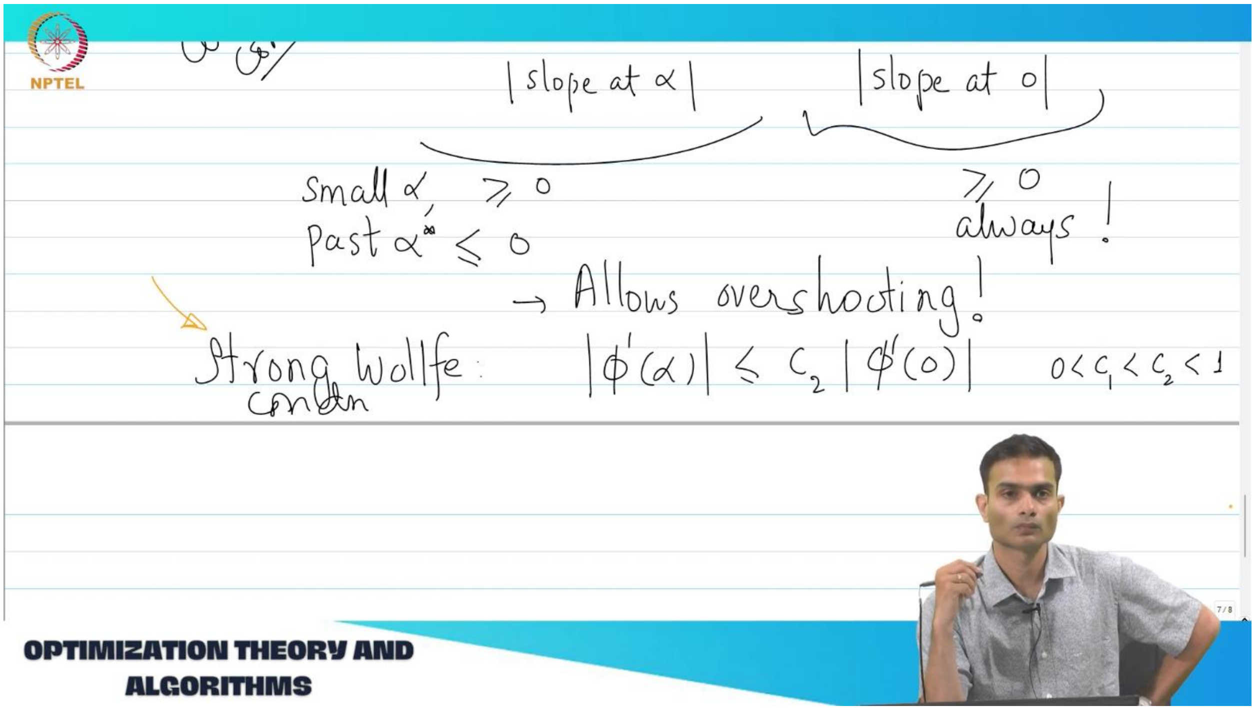
The main cost of implementing the **Strong Wolfe Condition** lies in the evaluation of $\phi'(\alpha)$, which involves gradient calculations.

For each step in the line search, $\phi'(\alpha)$ needs to be computed, which can be done using finite differences. For an n-dimensional function, this requires $n+1$ function evaluations, making it computationally expensive, especially in high-dimensional spaces:

$$\text{Function evaluations needed: } n+1.$$

However, parallelization (using multiple cores) can reduce computation time by distributing function evaluations across different processors.

6. Preventing Overshooting

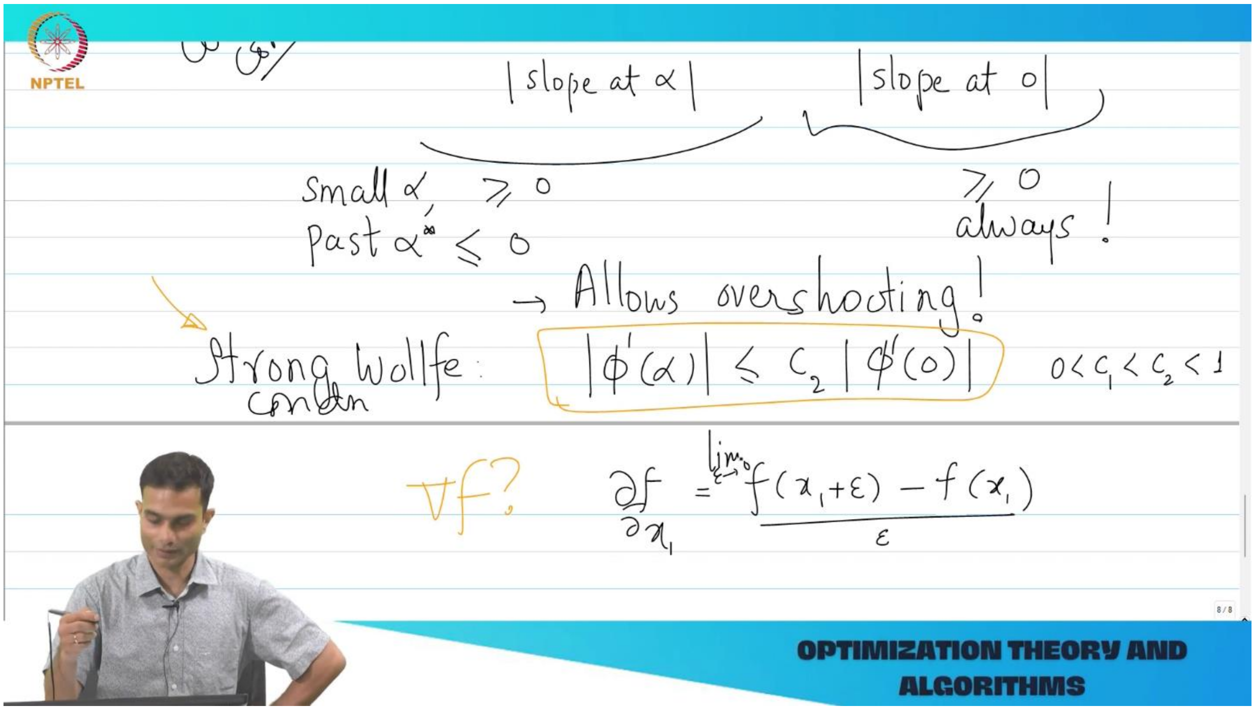


By applying the **Strong Wolfe Condition**, the algorithm prevents large steps that might take it past a local maximum, ensuring that the solution stays within a "bracketed" range that balances convergence speed with stability.

7. Convex Functions

In the case of convex functions, the regular Wolfe conditions (with minor adjustments) are sufficient because the function has a single global minimum. The **Strong Wolfe Condition** is more useful in non-convex cases with multiple local maxima.

8. Trade-offs in Line Search



While more aggressive methods can be used to deal with flat curvature, the **Strong Wolfe Condition** provides a balance by rejecting very small α values and preventing overshooting without the high computational cost of second-order methods. Nonetheless, gradient evaluations are necessary, and in high-dimensional problems, these evaluations can be expensive.

$$\text{Gradient evaluation cost: } \mathcal{O}(n).$$

By using the **Strong Wolfe Conditions**, you ensure that your optimization method carefully brackets the solution without overshooting, while also controlling step sizes more effectively than the basic Wolfe conditions alone.