Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 44

Hello everyone, this is Lecture 4 of Week 9. In the previous lectures, we were learning about the simplex method. Given a linear programming problem, we first worked on finding the vertices of its feasible polytope. This involved choosing m linearly independent columns from the n columns of the matrix A. The variables corresponding to these columns are called basic variables, and the remaining variables are called non-basic variables. The basic variables have the solution $x_B = B^{-1}b$, and the non-basic variables are set to zero, $x_N = 0$.

After starting at a vertex, we determine if it is the minimizer.

We compute the vector $c_N^T - c_B^T B^{-1} N$ and verify if every component is non-negative. If it is, the vertex is the minimizer. If it is not, we must exchange one variable between the set of basic and non-basic variables.

We already decided which variable to bring into the basis: the non-basic variable x_q corresponding to the most negative component of the vector $c_N^T - c_B^T B^{-1} N$.

Now, we must determine which current basic variable to remove.

Recall the equation:

$$x_B = B^{-1}b - B^{-1}N x_N$$

Since we are only making x_q non-zero (the other non-basic variables remain zero), this simplifies to:

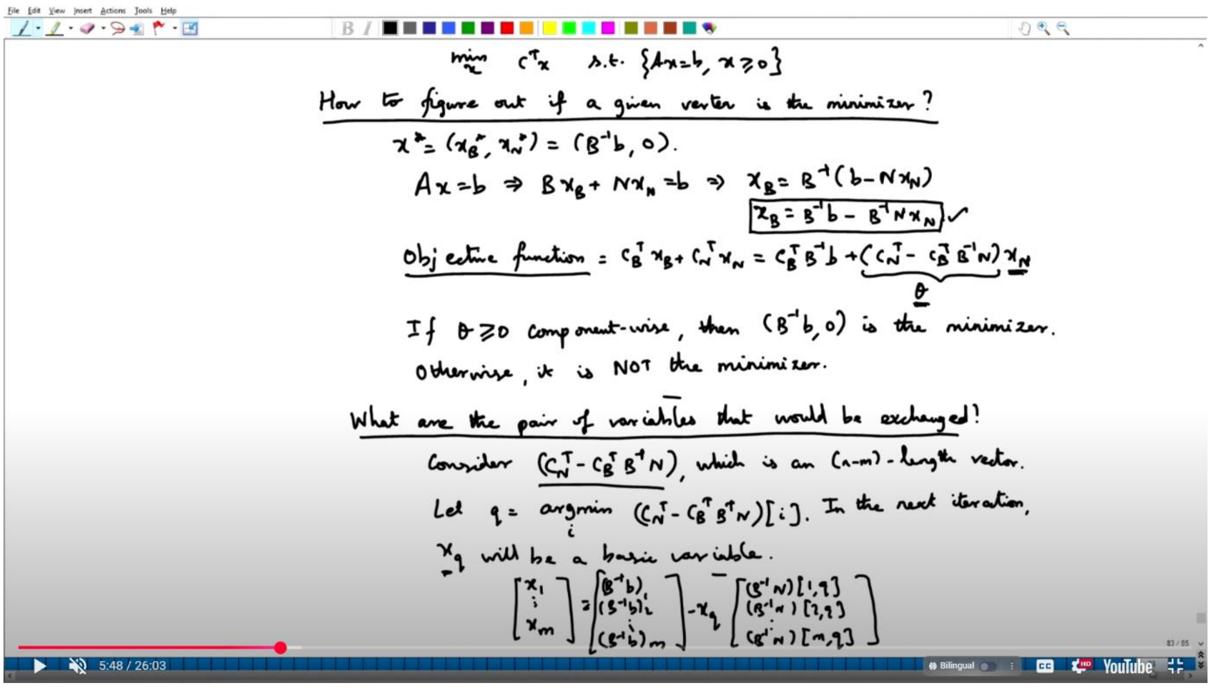
$$x_B = B^{-1}b - x_q * (B^{-1}N)_{:,q}$$

where $(B^{-1}N)_{:,q}$ is the q-th column of the matrix $B^{-1}N$.

We require that all components of x_B remain non-negative to stay feasible. This gives us the constraint for each component i:

$$(B^{-1}b)_i - x_q * (B^{-1}N)_{i,q} \geq 0.$$

(Refer Slide Time 5:48)



$$\min_{x} \; c^T x \quad \text{s.t.} \; \{Ax = b, \; x \geq 0\}$$

How to figure out if a given vertex is the minimizer?

$$x^* = (x_B^*, x_N^*) = (B^{-1}b, 0).$$

$$Ax = b \Rightarrow Bx_B + Nx_N = b \Rightarrow x_B = B^{-1}(b - Nx_N)$$

$$\boxed{x_B = B^{-1}b - B^{-1}Nx_N} \checkmark$$

Objective function $= c_B^T x_B + c_N^T x_N = c_B^T B^{-1} b + \underbrace{(c_N^T - c_B^T B^{-1} N)}_{\theta} x_N$

If $\theta \geq 0$ component-wise, then $(B^{-1}b, 0)$ is the minimizer.

Otherwise, it is NOT the minimizer.

What are the pair of variables that would be exchanged?

Consider $(c_N^T - c_B^T B^{-1} N)$, which is an $(n-m)$-length vector.

Let $q = \underset{i}{\text{argmin}} \; ((c_N^T - c_B^T B^{-1} N)[i])$. In the next iteration, x_q will be a basic variable.

$$\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} (B^{-1}b)_1 \\ (B^{-1}b)_2 \\ \vdots \\ (B^{-1}b)_m \end{bmatrix} - x_q \begin{bmatrix} (B^{-1}N)[1,q] \\ (B^{-1}N)[2,q] \\ \vdots \\ (B^{-1}N)[m,q] \end{bmatrix}$$

We must find a constraint on x_q from this.

* If $(B^{-1}N)_{i,q} > 0$, we rearrange the inequality to get:

$$x_q \leq (B^{-1}b)_i / (B^{-1}N)_{i,q}$$

* If $(B^{-1}N)_{i,q} \leq 0$, the inequality is automatically satisfied for any $x_q \geq 0$ because the term $- x_q * (B^{-1}N)_{i,q}$ will be non-negative, thus increasing or not affecting the value of $(B^{-1}b)_i$. Therefore, these indices impose no upper bound on x_q.

We only need to consider indices i where $(B^{-1}N)_{i,q} > 0$. Let us define the index set:

$$I = \{ i \mid (B^{-1}N)_{i,q} > 0 \}$$

For all i in I, we require:

$$x_q \leq (B^{-1}b)_i / (B^{-1}N)_{i,q}$$

To ensure all constraints are satisfied, x_q must be less than or equal to the smallest of these ratios. The variable that will be forced to zero first as we increase x_q is the one for which this ratio is the smallest. Therefore, the variable to be removed from the basis is the one corresponding to:

$$j = \text{argmin}_{i \in I} \left[(B^{-1}b)_i / (B^{-1}N)_{i,q} \right]$$

The variable x_j will become non-basic in the next iteration.

We now have the full algorithm. To summarize:

The Simplex Algorithm

1. Convert to Standard Form: Convert the given LP into the standard form:

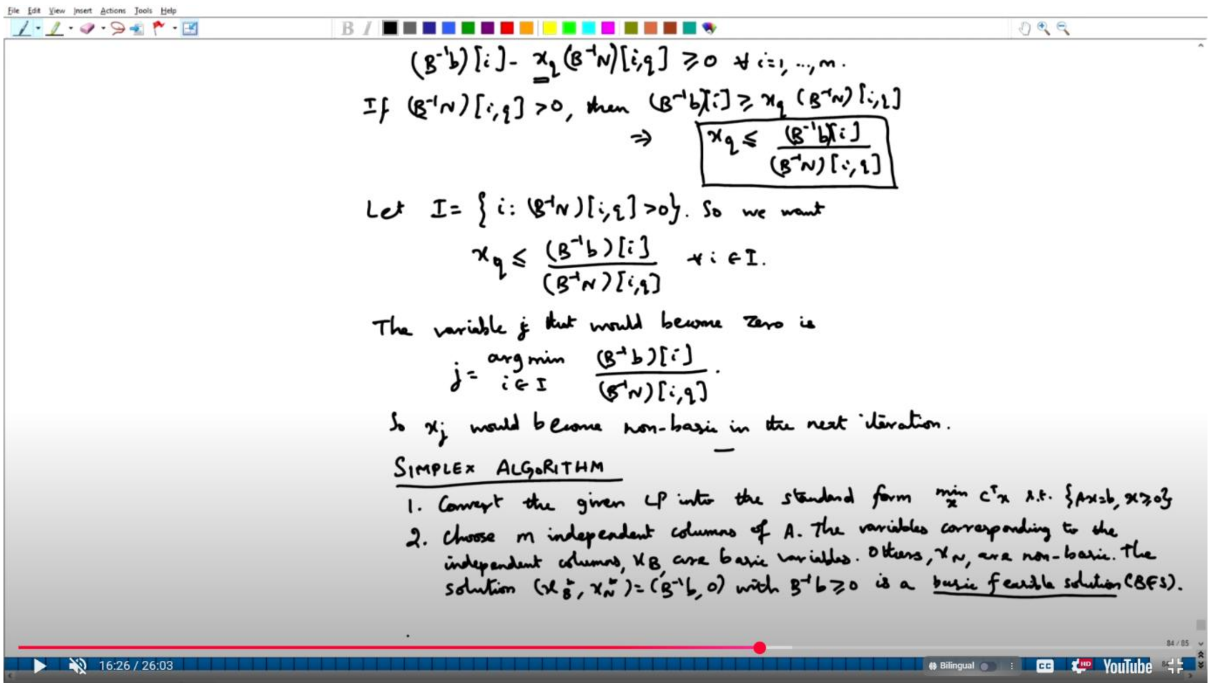
   Minimize $c^T x$ subject to $Ax = b$, $x \geq 0$

where A is an $m \times n$ matrix with full row rank. (This step is often done manually before coding).

2. Find an Initial Basic Feasible Solution (BFS): Choose m linearly independent columns of A to form the basis matrix B.

The variables x_B are basic, and x_N are non-basic. The solution $(x_B, x_N) = (B^{-1}b, 0)$ is a vertex, or Basic Feasible Solution, provided $B^{-1}b \geq 0$.

(Refer Slide Time 16:26)


$$(B^{-1}b)[i] - \underline{x_q}(B^{-1}N)[i,q] \geq 0 \quad \forall i=1,\dots,m.$$

If $(B^{-1}N)[i,q] > 0$, then $(B^{-1}b)[i] \geq x_q (B^{-1}N)[i,q]$

$$\Rightarrow \boxed{x_q \leq \frac{(B^{-1}b)[i]}{(B^{-1}N)[i,q]}}$$

Let $I = \{ i : (B^{-1}N)[i,q] > 0 \}$. So we want

$$x_q \leq \frac{(B^{-1}b)[i]}{(B^{-1}N)[i,q]} \quad \forall i \in I.$$

The variable j that would become zero is

$$j = \underset{i \in I}{\arg\min} \frac{(B^{-1}b)[i]}{(B^{-1}N)[i,q]}.$$

So x_j would become non-basic in the next iteration.

SIMPLEX ALGORITHM

1. Convert the given LP into the standard form $\min_x c^T x$ s.t. $\{Ax=b, x \geq 0\}$
2. Choose m independent columns of A. The variables corresponding to the independent columns, x_B, are basic variables. Others, x_N, are non-basic. The solution $(x_B^*, x_N^*) = (B^{-1}b, 0)$ with $B^{-1}b \geq 0$ is a basic feasible solution (BFS).

3. Optimality Check: Compute the vector $\theta = c_N^T - c_B^T B^{-1} N$.

 * If $\theta \geq 0$ (all components are non-negative), stop . The current BFS is optimal.

 * Otherwise, continue.

4. Select Entering Variable: Let $q = \text{argmin}_i\ \theta[i]$ (choose the non-basic variable with the most negative reduced cost).

5. Unboundedness Check: Consider the column $d = (B^{-1}N)_{\{:,q\}}$.

 * If $d \leq 0$ (all components are non-positive), then stop. The problem is unbounded .

 * Otherwise, continue.

6. Select Leaving Variable: Define the index set $I = \{ i \mid d_i > 0 \}$.

 Let $j = \text{argmin}_{\{i \in I\}} [(B^{-1}b)_i / d_i]$.

7. Update Basis: Exchange variables x_q (entering) and x_j (leaving). Update the sets of basic and non-basic variables, and the matrices B and N accordingly.

8. Loop: Go back to Step 3.

This process iterates, moving from vertex to adjacent vertex, improving the objective function each time, until an optimal solution is found or unboundedness is detected.

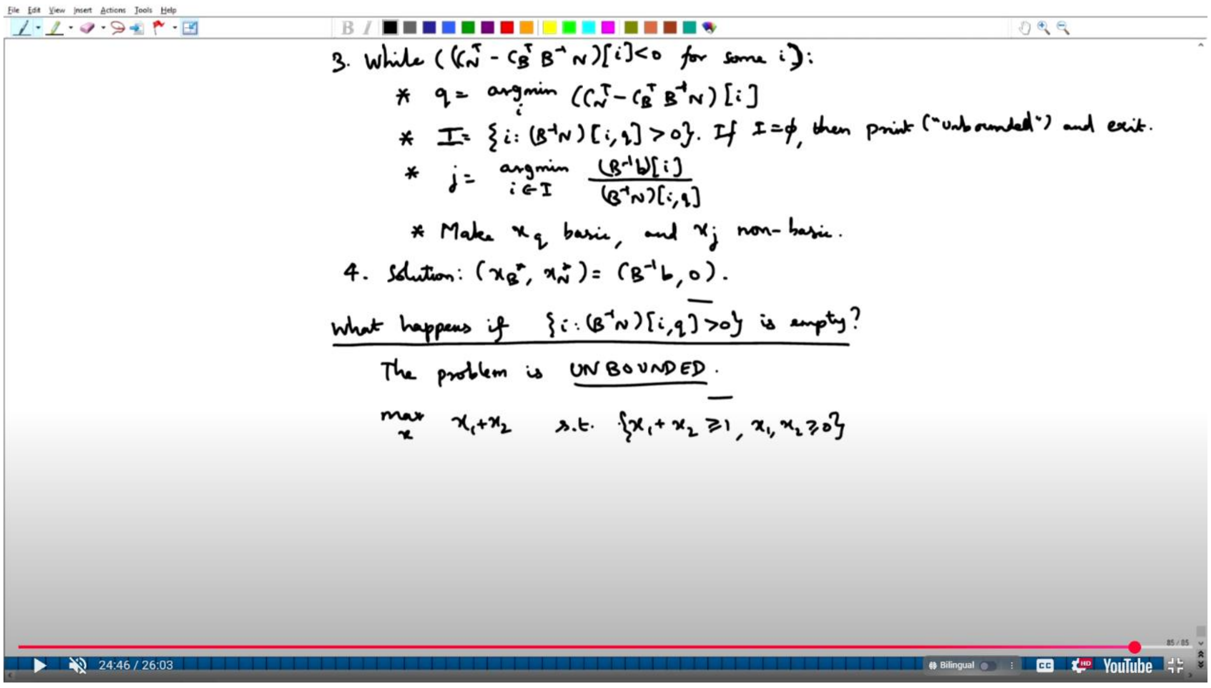
Of course. Here is the cleaned-up explanation of the unboundedness example.

If you want an immediate example, consider this problem:

Maximize: $x_1 + x_2$

Subject to: $x_1 + x_2 \geq 1$, $x_1 \geq 0$, $x_2 \geq 0$

(Refer Slide Time 24:26)



```
3. While ((c_N^T - c_B^T B^{-1} N)[i] < 0 for some i):
      * q = argmin_i ((c_N^T - c_B^T B^{-1} N)[i])
      * I = {i : (B^{-1} N)[i, q] > 0}. If I = φ, then print ("unbounded") and exit.
      * j = argmin_{i ∈ I} (B^{-1} b)[i] / (B^{-1} N)[i, q]
      * Make x_q basic, and x_j non-basic.
4. Solution: (x_B^*, x_N^*) = (B^{-1} b, 0).
```

<u>What happens if $\{i : (B^{-1}N)[i,q] > 0\}$ is empty?</u>

The problem is <u>UNBOUNDED</u>.

$\max_x \; x_1 + x_2 \quad \text{s.t.} \; \{x_1 + x_2 \geq 1, \; x_1, x_2 \geq 0\}$

Just by looking at the problem, you can determine that the maximum value of $x_1 + x_2$ is infinity. The constraint is $x_1 + x_2 \geq 1$, meaning any pair (x_1, x_2) where their sum is greater than or equal to 1 is feasible. For instance, the points (100, 100), (1000, 1000), (10^9, 10^9), and (10^{100}, 10^{100}) all belong to the feasible set.

Since you can increase the values of x_1 and x_2 without any upper bound, the value of the objective function $x_1 + x_2$ can also be increased indefinitely towards infinity. Therefore, there is no finite maximum value, and the problem is unbounded .

This is a clear example of an unbounded linear programming problem. The simplex algorithm is designed to detect such a condition. If you apply the algorithm's steps to this problem (after converting it to standard form), it will correctly identify and report that the problem is unbounded.

We will learn how to code this algorithm in the next lecture. Thank you.