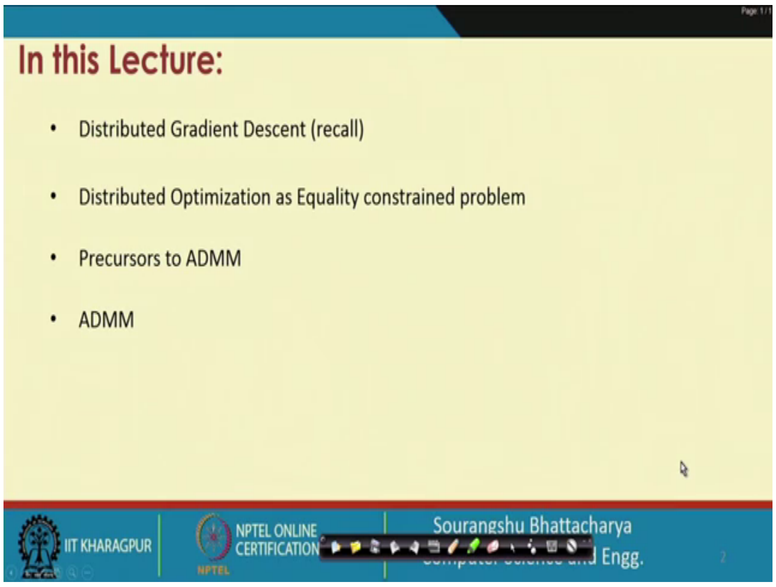
Scalable Data Science

Prof. Sourangshu Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 22 a
Alternating Direction Method of Multipliers

Hello everyone, welcome to the NPTEL course 22 lecture of NPTEL course on scalable data mining or scalable data science. And I am Prof. Sourangshu Bhattacharya from IIT, Kharagpur. And this is today's lecture is going to be an Alternating Direction Method of Multipliers.

(Refer Slide Time: 00:38)



In this Lecture:

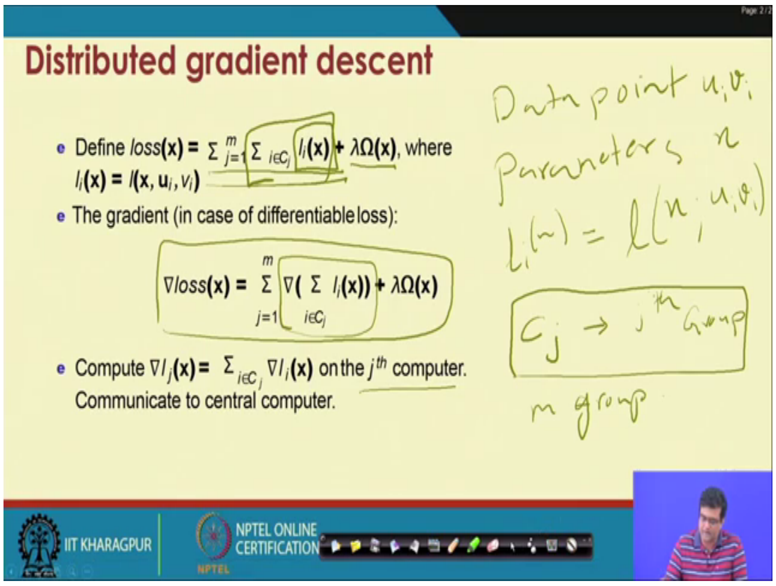
- Distributed Gradient Descent (recall)
- Distributed Optimization as Equality constrained problem
- Precursors to ADMM
- ADMM

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION
Sourangshu Bhattacharya
Engg.

So, today we are first going to discuss the distributed gradient descent or rather we are going to recall the distributed gradient descent, and distributed optimization problem. And importantly we are going to discuss distributed optimization as an equality constraint problem and then followed by that we are going to discuss the precursors to ADMM, which are some of the methods that we will discuss for solving the equality constraint optimization problems.

And we will discuss their limitation. So, what are the limitations of this method? And then finally we will discuss ADMM and which basically overcome some of these limitations. And then we will also discuss an example the toy example of how to use ADMM for a practical machine learning distributed optimization problem ok.

(Refer Slide Time: 01:45)



Distributed gradient descent

- Define $loss(\mathbf{x}) = \sum_{j=1}^{m} \boxed{\sum_{i \in C_j} \boxed{l_i(\mathbf{x})}} + \underline{\lambda\Omega(\mathbf{x})}$, where $l_i(\mathbf{x}) = l(\mathbf{x}, u_i, v_i)$
- The gradient (in case of differentiable loss):

$$\boxed{\nabla loss(\mathbf{x}) = \sum_{j=1}^{m} \boxed{\nabla(\sum_{i \in C_j} l_i(\mathbf{x}))} + \lambda\Omega(\mathbf{x})}$$

- Compute $\nabla l_j(\mathbf{x}) = \sum_{i \in C_j} \nabla l_i(\mathbf{x})$ on the j^{th} computer. Communicate to central computer.

Data point $u_i v_i$

Parameters x

$l_i(x) = l(x; u_i, v_i)$

$$\boxed{C_j \rightarrow j^{th} \text{ Group}}$$

m group

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION

So, distributed optimization. So, so in the context of machine learning the distribute distributed loss minimization problem can be written like this. So, basically we have so we have the data point x. So, our data point is x or rather x i. So, x i has all the data or rather sorry the data point is u i and v i. So, our data point is u i and v i. And the parameters with respect to which we are going to optimize are x. So, basically our l i of x l i of x is the loss function, which is the function of basically x given u i and v i. So, our loss can be decomposed like this where i i rather break down the data into clusters C j ok. So, jth cluster or jth group we can say jth group.

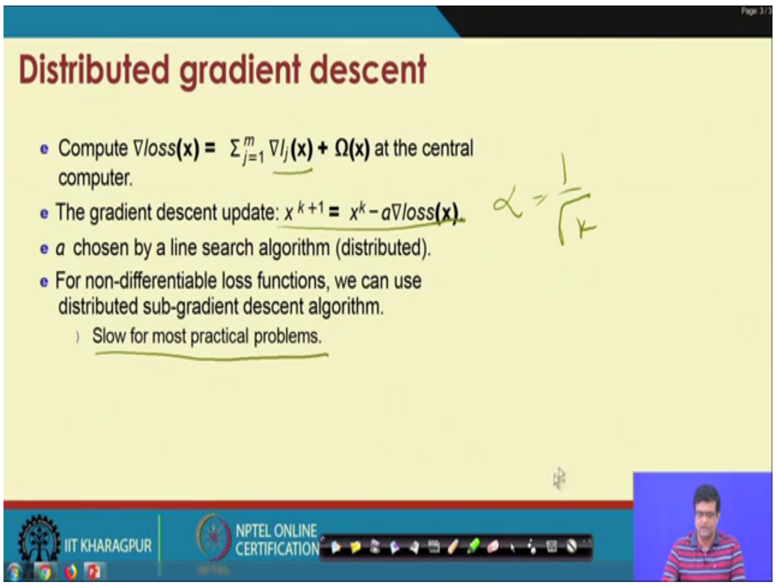
And and for all the examples i in the jth group. This inner term it sums up the loss for those data points. And then there are m groups, so there are m groups. So, the outer sum basically sum the over the groups. Now, the thing is that maybe each of these groups or clusters are distributed on two different machines so, then our total loss function look something like this.

So, the first part is the loss, which is dependent on the data. And it can be broken down in a hierarchical manner into the loss over the a data points in a group. And then also the total sum over of the loss over all groups. And then we add the regularize that which is not dependent on any data point ok. So, this is a typical structure of an optimization problem that we get in machine learning.

Now, given this structure, what will the gradient descent algorithm look like ok. So, the gradient descent algorithm will look like this, where just like your loss is hierarchically broken down into data points within a group. And also the sum over all data points within the group. The gradient can also be similarly broken down ok. So, you can calculate the gradient over a group ok.

This can further be actually pushed inside ok, but for the time you can just say you want to calculate the gradient of the loss of data points in a group. And then sum them over and then you can get the total gradient. So, the advantage of these strategies that, you can compute this in a gradient by just independently on each computer. And then you can communicate it to a central computer ok.

(Refer Slide Time: 05:46)



Distributed gradient descent

- Compute $\nabla loss(\mathbf{x}) = \sum_{j=1}^{m} \underline{\nabla l_j(\mathbf{x})} + \Omega(\mathbf{x})$ at the central computer.
- The gradient descent update: $x^{k+1} = x^k - \alpha \nabla loss(\mathbf{x})$
- α chosen by a line search algorithm (distributed).
- For non-differentiable loss functions, we can use distributed sub-gradient descent algorithm.
  - Slow for most practical problems.

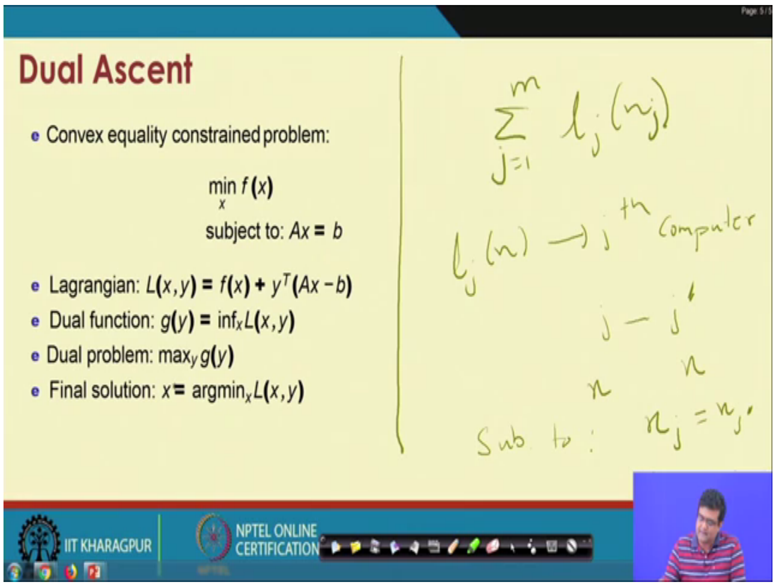
$\alpha \to \frac{1}{\sqrt{k}}$

So, then you can compute the total loss on gradient on the central computer by adding over the individual gradients of the jth groups ok. And then you can do this update, where your alpha can in case of gradient descent be chosen by line search or if you are using something like mini bus stochastic gradient descent, you can use the alpha, which is you can use alpha is equal to something like 1 by root k. So, alpha can vary with the number of iterations. So, the problem with this is basically that this is kind of slow for most practical problems ok.

So, and basically the reason it is slow is that is that you are not doing much computation on the nodes. So, what you are doing is, you are just calculating the gradients on the nodes. And then you are transferring the gradient gradient for each iteration on to the central computers. So, you are doing lot of transferring of data onto the central computer, and doing less amount of relatively less amount of computation in the central node ok. So, sometimes it is also called that it is communication wise it is not very efficient ok. So, we want to solve this problem.

So, the general strategy for solving this problem will be that we want to do some optimization also on the nodes that is a few iterations maybe on each of the nodes. And then somehow exchange parameters with the central computer, so that I have to exchange the parameters with the central computer a less number of times, so that is the general idea behind ADMM ok, so but before going into ADMM we look at some precursors of ADMM ok.

(Refer Slide Time: 08:24)



Dual Ascent

- Convex equality constrained problem:

$$\min_x f(x)$$
$$\text{subject to: } Ax = b$$

- Lagrangian: $L(x,y) = f(x) + y^T(Ax - b)$
- Dual function: $g(y) = \inf_x L(x,y)$
- Dual problem: $\max_y g(y)$
- Final solution: $x = \text{argmin}_x L(x,y)$

$$\sum_{j=1}^{m} l_j(x_j)$$

$l_j(x) \rightarrow j^{th}$ Computer

$j - j'$

$x \quad x$

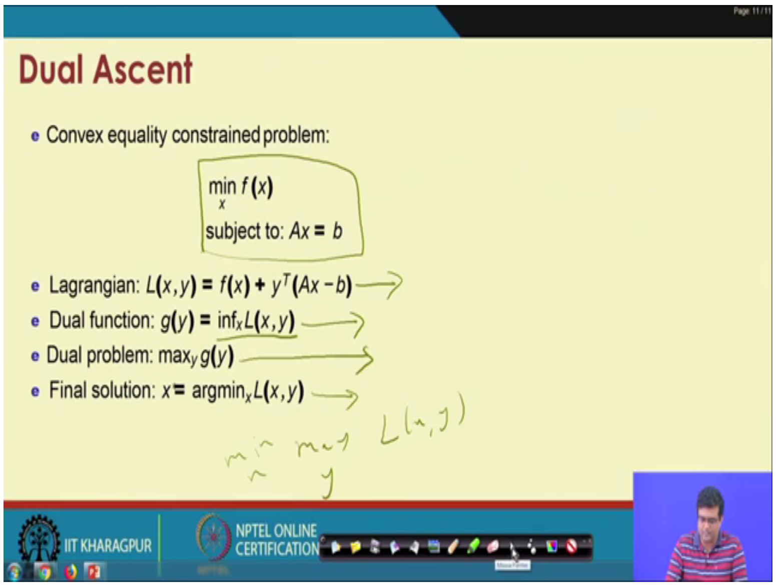
Sub to: $x_j = x_{j'}$

So, why do we have to go into these precursors of ADMM? The reason is that you can understand. So, so before looking at this precursor of ADMM you can understand that if you have if you have decomposable function, which is as we have written down, which is some over j is equal to 1 to m let us say and then l j of x.

Suppose this is our this is our this is a function plus there is some regularize the which I am not so much bothered about at this point in time. If this is the function ok, so we want to minimize this l j of x on the jth computer, but then the x let us say there are two computers let us say j and j dash.

So, then the x variable on the jth computer and the x variable on the j dashth computer are couple together ok. So, how do you express this coupling, you can express this coupling by slightly modifying this optimization problem in this manner that you instead of writing x here you write x j, but then you add the condition or you add the constraint rather subject to the constraint that x j equal to x j dash something like this ok. So, some equality constraint. So, this constraint rather sorry ok, this equality constraint is what we want to tackle. So, this is going to be our idea we will see how exactly we do it as we go forward.

(Refer Slide Time: 10:45)



Dual Ascent

- Convex equality constrained problem:

$$\boxed{\begin{aligned}&\min_x f(x)\\ &\text{subject to: } Ax = b\end{aligned}}$$

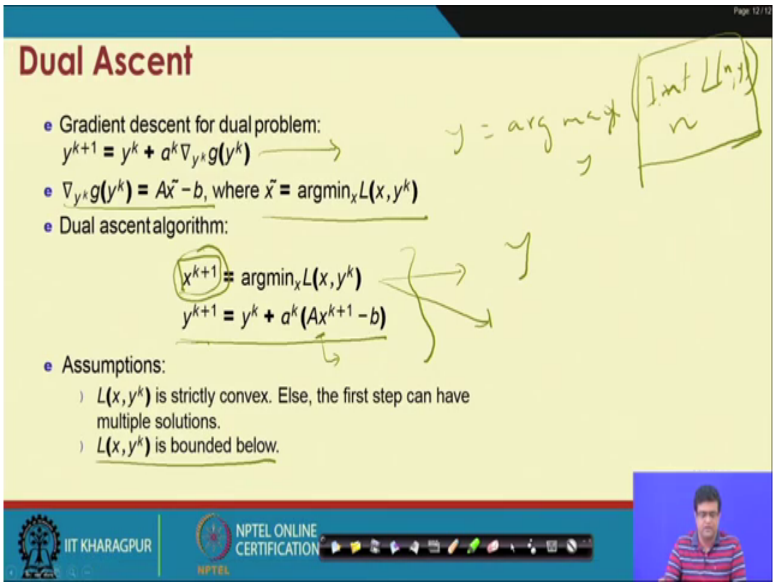
- Lagrangian: $L(x,y) = f(x) + y^T(Ax-b) \longrightarrow$
- Dual function: $g(y) = \inf_x \underline{L(x,y)} \longrightarrow$
- Dual problem: $\max_y g(y) \longrightarrow$
- Final solution: $x = \operatorname{argmin}_x L(x,y) \longrightarrow$

$\min_x \max_y L(x,y)$

So, basically what this brings us to is this equality constraint problem ok. So, the equality constraint problem basically looks like this that you have to minimize f of x subject to the constraint that A of x equal to b. And now from optimization theory as you can write the Lagrangian, so this is the Lagrangian for this particular problem. And this is the dual function ok, so, the dual function is basically the infimum. So, the dual function is the Lagrangian function is, of course a function of the primal variable, and the Lagrange multipliers, which is y.

And, now the dual function is only a function of the Lagrange multipliers. And it is the infimum over x a Lagrangian of x comma y. And then you can also solve the dual problem, which is maximization problem. If the primal problem is minimization problem and the final solution is of course, is you can you can show that you can either solve it you can solve it basically in a minimax manner. So, you can compute the you can take the max over y l of x comma y, and you can compute the mean over x or you can do the other way round. And in case of convex optimization problem the two are the same.

(Refer Slide Time: 12:33)



Dual Ascent

- Gradient descent for dual problem:
  $y^{k+1} = y^k + a^k \nabla_{y^k} g(y^k) \longrightarrow$ $y = \arg\max_y \boxed{\inf_x L(x,y)}$
- $\underline{\nabla_{y^k} g(y^k) = A\tilde{x} - b}$, where $\underline{\tilde{x} = \text{argmin}_x L(x, y^k)}$
- Dual ascent algorithm:

$$\underline{\begin{aligned} \boxed{x^{k+1}} &= \text{argmin}_x L(x, y^k) \\ y^{k+1} &= y^k + a^k (Ax^{k+1} - b) \end{aligned}}$$

y

- Assumptions:
  - $L(x, y^k)$ is strictly convex. Else, the first step can have multiple solutions.
  - $\underline{L(x, y^k) \text{ is bounded below.}}$

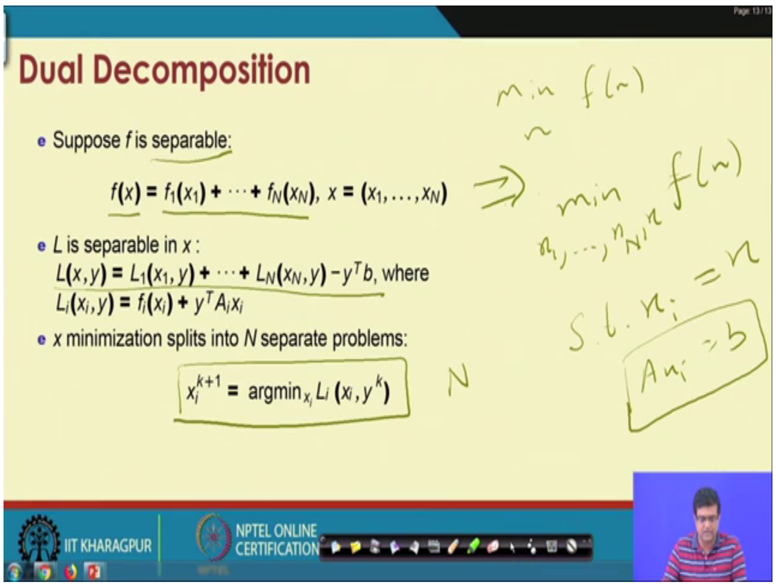
So, the dual ascent method tries to solve this problem in this manner ok. So, it tries it tries to do a gradient decent or it tries to do a gradient descent step for the dual problem. So, if you if you recall the dual problem is that you have to find y, which is arg max over y and then l of x comma y or rather sorry you this is this is rather if I were to write it like this it is arg max over y and then infimum over or minimum over x L of x comma y.

So, this is the dual problem and. So, basically you have to compute this function minimum over x L of x comma y. Another way of saying it is that you compute the x tilde, which is the minimum over x L of x comma y k. And then you, so you once you have computed the x tilde you substitute the x tilde in the gradient in this gradient and then you compute the next iterate ok.

So, this results in this algorithm. So, at each step so first you randomly initialize sum y. And then you compute the next iterate of x by solving this problem the inner problem. And then once you have solved this inner problem you do a gradient descent steps, so this is the gradient descent step here for the dual variable. Now, the problem with this method is that it works only if this Lagrangian is strictly convex. In other words this solution x k plus 1 for this problem should have a unique solution. So, otherwise what happens is this step can have multiple solutions.

And even though the Lagrangian value will decrease, because this step is having a multiple solution the, so the and you have no control over which solution gets chosen. The update to the dual variable will oscillate ok. So, this is the problem with the dual ascent method ok. And also another problem is that this should be bounded below, so it cannot be unbounded.

(Refer Slide Time: 15:54)



Dual Decomposition

- Suppose f is separable:

$$f(x) = f_1(x_1) + \cdots + f_N(x_N),\ x = (x_1, \ldots, x_N) \Rightarrow \min_{x_1,\ldots,x_N,x} f(x)$$

- L is separable in x:

$L(x,y) = L_1(x_1,y) + \cdots + L_N(x_N,y) - y^T b$, where
$L_i(x_i,y) = f_i(x_i) + y^T A_i x_i$

- x minimization splits into N separate problems:

$$\boxed{x_i^{k+1} = \text{argmin}_{x_i} L_i(x_i, y^k)} \quad N$$

$\min_x f(x)$

s.t. $x_i = x$

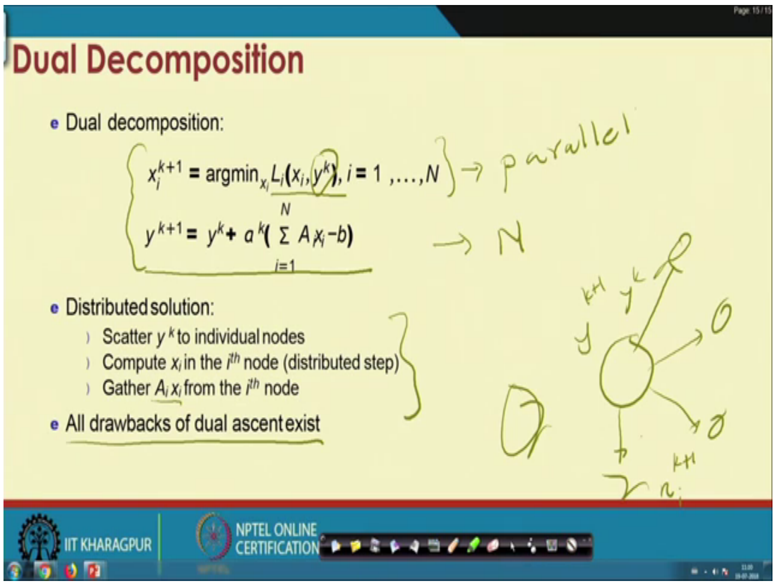
$\boxed{A x_i = b}$

So, a generalization of this a dual ascent method is called the dual decomposition method, where the method is exactly the same except that instead of f of x as one function you can now have f of x as sum of many functions ok. So, your objective function f of x is now is called separable ok. So, your objective function is separable.

Now, if this is the case if your objective function is separable, and then what you can do is you can as mentioned earlier, you can write. So, you can write the problem of minimizing. So, we can write the problem of minimizing over x f of x as that of minimizing. So, these two problems become equivalent you minimize over x 1 till x N. So, you take N variables and then you write this whole f of x again which is a sum over f 1 X 1, f 2 X 2 till f N XN.

Subject to the constraint that x 1 and also of course x is variable and subject to the constraint that x i is equal to x ok. So, if you have a constraint like this, then you can write the Lagrangian function as this ok. So, this can be written as A of x i is equal to b. So, this is in this form that A of x i is equal to b. So, you are you are L is now separable in x. Your Lagrangian function as you can see is separable in x. So, X are the primary primal variables and y is the dual variable. So, if that is the case then your earlier X minimization step, it now splits into this capital N many minimization problems.

(Refer Slide Time: 18:51)



Dual Decomposition

- Dual decomposition:

$$\begin{cases} x_i^{k+1} = \text{argmin}_{x_i} L_i(x_i, y^k), i = 1, \ldots, N \\ y^{k+1} = y^k + \alpha^k \left(\sum_{i=1}^{N} A x_i - b \right) \end{cases}$$

- Distributed solution:
  - Scatter y^k to individual nodes
  - Compute x_i in the i^{th} node (distributed step)
  - Gather $A_i x_i$ from the i^{th} node
- All drawbacks of dual ascent exist

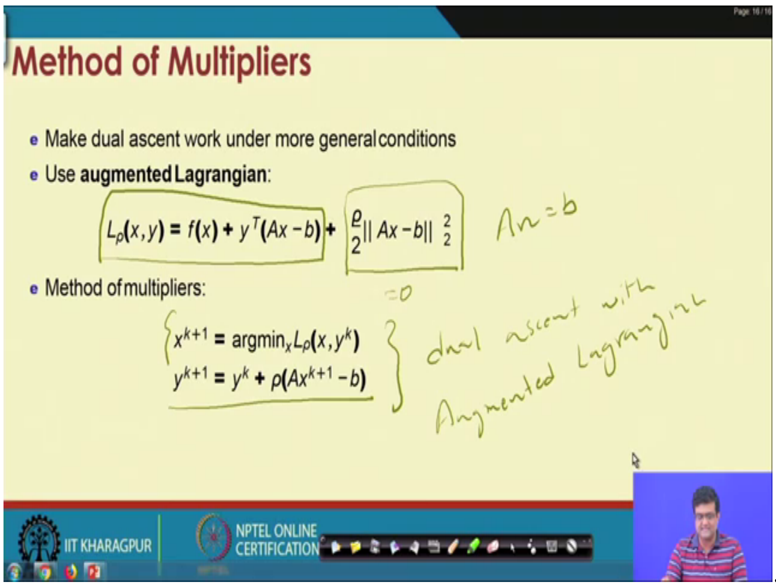
parallel
N
y^{k+1}
y^k
x_i^{k+1}
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION

So, your algorithm, now changes slightly. So, it is same as dual ascent except that the first X minimization step. It is now split into N optimization steps. So, see that here what we are doing is we are of course iterating over this whole thing, but within one iteration we are trying to minimize this Lagrangian function ok on each computer over the local variable X i. So, there is a local variable X i on each computer. So, I am assuming here that i stands each i is put on one computer. And, now we are minimizing this L i of X with respect to the previous Lagrange multiplier iterate on each computer.

So, this can be done in parallel ok. So, once we solve this problem in parallel we can do this update to the Lagrange multipliers. So, in this case the Lagrange number of Lagrange multiplier is exactly N. So, we can update these Lagrange multipliers by this step, which is the dual ascent step or which is rather the gradient descent in the dual step ok. So, the distributed solution is something like this that you scatter; so from a central node. So, if this is your central node, and these are your compute nodes, then you sent y k to all of these nodes. Then you optimize you optimize and obtained x i k plus 1 in each of these nodes, which is again send back to the central node.

And then you compute this A i x i and you some and out update and obtained y k plus 1 in central node. And you keep going back and forth. Now, this is a very neat solution, but all the drawbacks of dual ascent exist. So, what are the drawback drawbacks is basically that if your Lagrangian is does not have a unique solution. Then this algorithm with oscillate ok.

(Refer Slide Time: 21:34)



Method of Multipliers

- Make dual ascent work under more general conditions
- Use **augmented Lagrangian**:

$$\boxed{L_\rho(x,y) = f(x) + y^T(Ax-b)} + \boxed{\underbrace{\frac{\rho}{2}\|Ax-b\|_2^2}_{=0}} \quad Ax=b$$

- Method of multipliers:

$$\left\{\begin{aligned} x^{k+1} &= \operatorname{argmin}_x L_\rho(x, y^k) \\ y^{k+1} &= y^k + \rho(Ax^{k+1}-b) \end{aligned}\right\} \text{ dual ascent with Augmented Lagrangian}$$

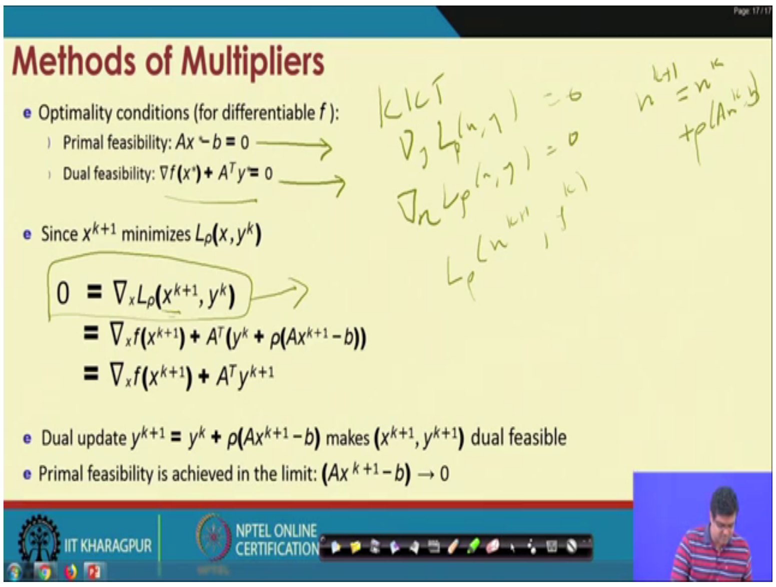
Now, here is a solution for that oscillation problem. So, the solution for the oscillation problem is that we add what is called the second order proximal term to the Lagrangian. So, earlier if you remember, the Lagrangian was just this portion ok. So, the first portion is the objective function, and the second portion is the Lagrange multiplier, which is the penalty for the constraint violation.

Now, we add another penalty for the constraint violation, but this penalty is always positive. And this is like this is A X minus b whole square. So, this will always if the constraints are violated, then this will always be positive. So, this will always drive the solution towards the solution, where A X is equal to b, which is the original constraint.

So, in other words we are not changing things at optimality so at optimality, this is going to be 0, because A X is equal to b and whatever the solution for this is that will stay here. So, minimizing in other words if we were to minimize Lagrangian what is if we were to minimize the augmented Lagrangian, the solution will not change ok.

Now, if this is the case ok, and we may want to do the dual ascent now, so method of multipliers is nothing but dual ascent with this augmented Lagrangian ok. So, this is dual ascent with augmented Lagrangian ok. So, your gradient, now changes to something like this ok. And your x update remains of course this of course same, but your gradient, now changes to something like this or rather the dual updates step changes to some something like this ok.

(Refer Slide Time: 24:16)



Methods of Multipliers

- Optimality conditions (for differentiable f):
  - Primal feasibility: $Ax - b = 0 \longrightarrow$
  - Dual feasibility: $\nabla f(x^*) + A^T y^* = 0 \longrightarrow$
- Since x^{k+1} minimizes $L_\rho(x, y^k)$

$$\boxed{0 = \nabla_x L_\rho(x^{k+1}, y^k)} \rightarrow$$
$$= \nabla_x f(x^{k+1}) + A^T(y^k + \rho(Ax^{k+1} - b))$$
$$= \nabla_x f(x^{k+1}) + A^T y^{k+1}$$

- Dual update $y^{k+1} = y^k + \rho(Ax^{k+1} - b)$ makes (x^{k+1}, y^{k+1}) dual feasible
- Primal feasibility is achieved in the limit: $(Ax^{k+1} - b) \rightarrow 0$

KKT

$\nabla_y L_\rho(x, y) = 0$

$\nabla_x L_\rho(x, y) = 0$

$L_\rho(x^{k+1}, y^k)$

$x^{k+1} = x^k + \rho(Ax^k - b)$

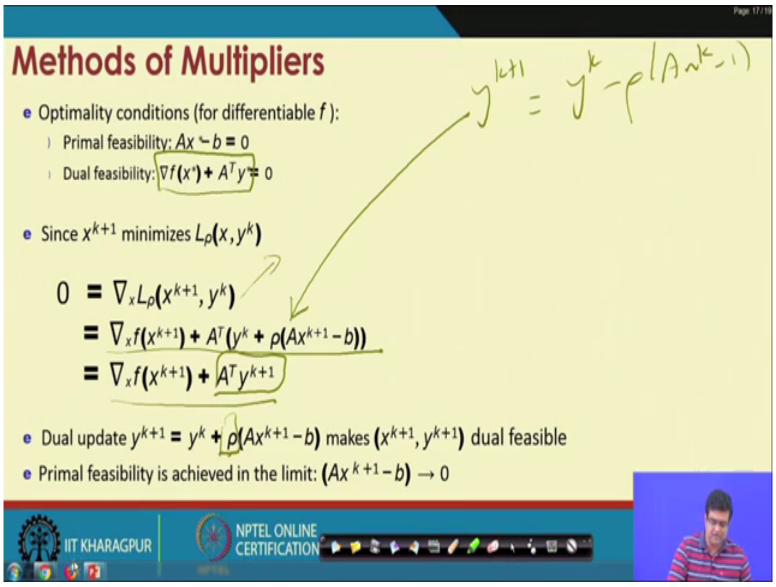
Why this is the case. So, you see when we are trying to solve this problem ok, we basically want two conditions to hold ok. So, this is this is coming from let us say something like we have discuss the KKT conditions Karush Kuhn Tucker conditions for optimality. So, the first condition that should be satisfied is the primal feasibility ok. So, primal feasibility basically means that the constant should be satisfied which is that A X is equal to b.

The second condition that should be satisfied is called the dual feasibility ok. Dual feasibility is basically that you take the gradient with respect to the dual variable ok. And then you say that that gradient should be 0. So, if you take the gradient with rather sorry you take the gradient with respect to primal variable that should be 0. So, the dual feasibility condition comes. When the gradient with respect to dual variable is 0, or so this is like saying gradient with respect to y L of x comma y L rho of x comma y is equal to 0.

And this is saying gradient with respect to x L rho of x comma y equal to 0. Now, if you put this condition here ok, so you put this condition on this that you know, so at X k plus 1 your so X k plus 1 is the solution that you get or rather y k plus 1 is the solution that you get after doing the dual updates step.

So, let us say the problem was a dual feasible before the update, so after the updates also it should be dual feasible ok. So, so in other words this should hold ok. And you see if you write, so if you write if you expand L rho of x k plus 1 with respect, so basically you know that the update is so you know that x k plus 1 is nothing but x k plus this rho of A x k minus b. So, or let me just yeah. So, sorry this is the update for y variable.

(Refer Slide Time: 27:44)



Methods of Multipliers

$$y^{k+1} = y^k - \rho(Ax^k - 1)$$

- Optimality conditions (for differentiable f):
  - Primal feasibility: $Ax - b = 0$
  - Dual feasibility: $\boxed{\nabla f(x^*) + A^T y^*} = 0$
- Since x^{k+1} minimizes $L_\rho(x, y^k)$

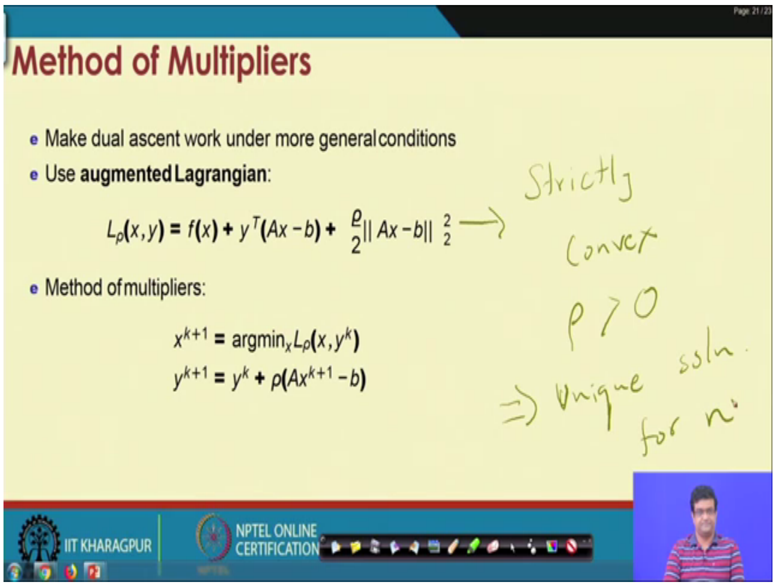
$$\begin{aligned} 0 &= \nabla_x L_\rho(x^{k+1}, y^k) \\ &= \underline{\nabla_x f(x^{k+1}) + A^T(y^k + \rho(Ax^{k+1} - b))} \\ &= \underline{\nabla_x f(x^{k+1}) + \boxed{A^T y^{k+1}}} \end{aligned}$$

- Dual update $y^{k+1} = y^k + \boxed{\rho}(Ax^{k+1} - b)$ makes (x^{k+1}, y^{k+1}) dual feasible
- Primal feasibility is achieved in the limit: $(Ax^{k+1} - b) \rightarrow 0$

So, you should have the you should have the update like. So, so you are update is basically that your y k plus 1 is is y k minus rho of A x k minus 1. So, if you plugin y k from here ok, so you get so so rather if you expand the L rho and then compute the so, so basically this is your y k plus 1 if you expand the L rho, and compute this gradient this becomes your y k plus 1. And then, so you have this a transverse y k plus 1, which is in the same form as your dual feasibility condition here.

So, you start with the optimality with respect to x ok. You get the dual feasibility for free ok. So, basically that is why instead of. So, if you remember, there was a step size update here. So actually this is a special property of method of multipliers that the step size, if you want to ensure dual feasibility, turns out to be rho ok. So, here rho is the parameter that you have used for this is the rho parameter that you have used for the augmented Lagrangian.

(Refer Slide Time: 29:28)



Method of Multipliers

- Make dual ascent work under more general conditions
- Use **augmented Lagrangian**:

$$L_\rho(x,y) = f(x) + y^T(Ax-b) + \frac{\rho}{2}\|Ax-b\|_2^2 \longrightarrow \text{Strictly convex}$$

$\rho > 0$

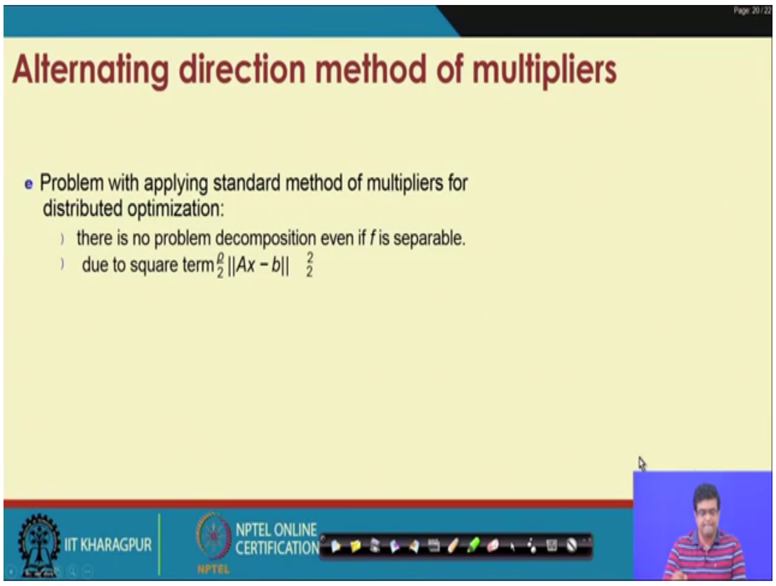
$\Rightarrow$ Unique soln. for x

- Method of multipliers:

$$x^{k+1} = \text{argmin}_x L_\rho(x, y^k)$$
$$y^{k+1} = y^k + \rho(Ax^{k+1} - b)$$

So, basically that is why instead of using an arbitrary step size here you use like the rho as the step size ok. So, this describes the method of multiplier ok.

(Refer Slide Time: 30:04)



Alternating direction method of multipliers

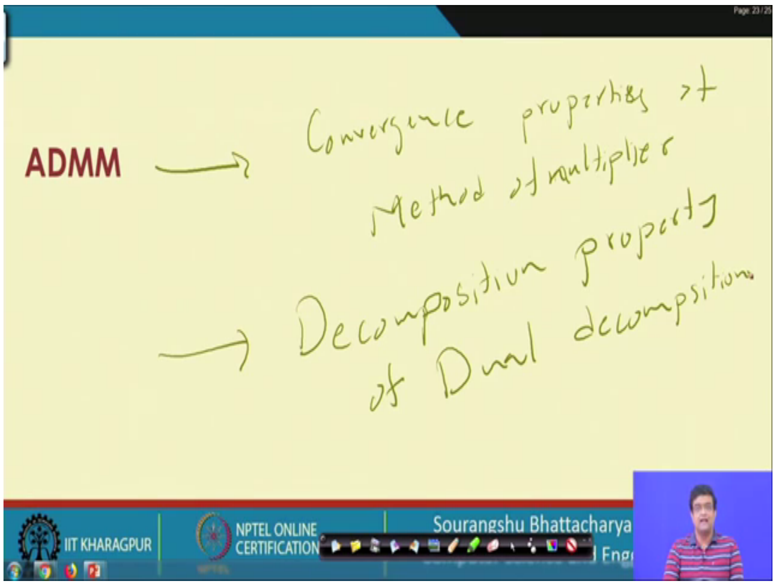
- Problem with applying standard method of multipliers for distributed optimization:
  - there is no problem decomposition even if f is separable.
  - due to square term $\frac{\rho}{2}||Ax - b||_2^2$

So, now, we come to ADMM. So, basically ADMM combines both of these. So, so if we want to step back, so now we want so method of multipliers have the advantage. So, why do we use method of multipliers, because method of multipliers has the advantage that your augmented Lagrangian. If we were to go back to this augmented Lagrangian, this is always strictly convex. So, this is always strictly convex for any positive rho.

So, if you have a positive rho, this is always strictly convex, which implies that it has a unique solution for x ok. So, if you try to minimize the augmented Lagrangian, the solution is always unique. So, method of multipliers does not suffer from the problem of multiple solutions ok.

And on the other hand the dual decomposition method. It actually very nicely splits into multiple optimization problems, which can be solved in a distributed manner on different machines as we have already shown. And the solution can be gathered and then the update can be made. So these are the two properties of two methods. And our objective is to combine the good properties of both these methods.

(Refer Slide Time: 32:18)

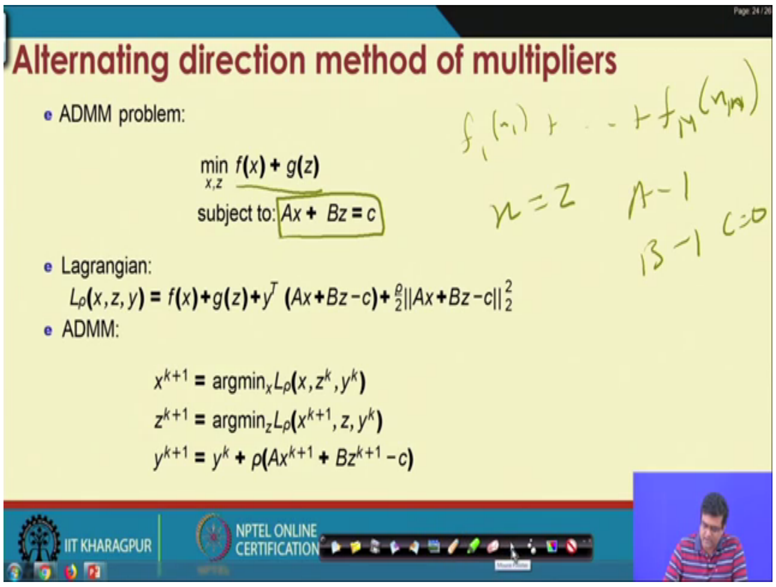


ADMM $\longrightarrow$ Convergence properties of Method of multiplier

$\longrightarrow$ Decomposition property of Dual decomposition

So, combine, so again for ok, so in some sense we should have convergence properties of method of multiplier ok and decomposition property of dual decomposition method ok. So, both of these should be there and that is where ADMM comes ADMM comes it. So, ADMM has both this properties. So, how does it work?

(Refer Slide Time: 33:37)



Alternating direction method of multipliers

- ADMM problem:

$$\min_{x,z} f(x) + g(z)$$
$$\text{subject to: } \boxed{Ax + Bz = c}$$

$f_1(x_1) + \dots + f_N(x_N)$

$x = z \quad A = 1 \quad B = -1 \quad C = 0$

- Lagrangian:

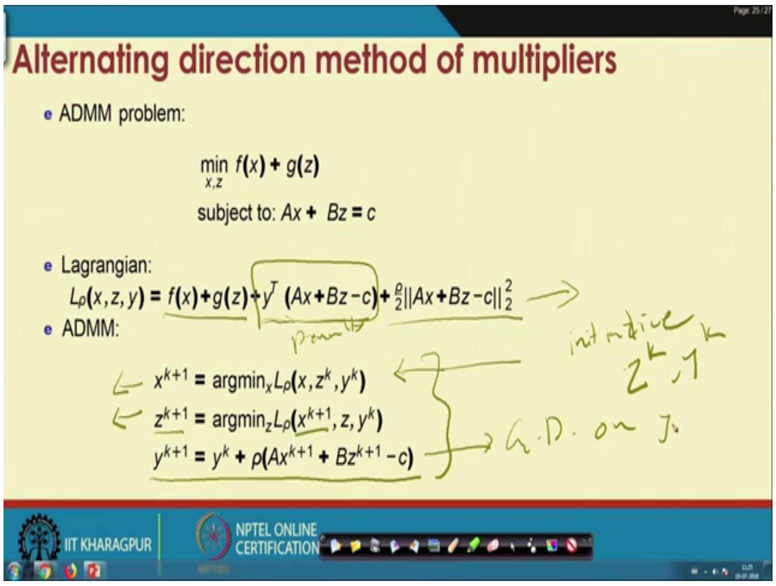
$$L_\rho(x,z,y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2$$

- ADMM:

$$x^{k+1} = \text{argmin}_x L_\rho(x, z^k, y^k)$$
$$z^{k+1} = \text{argmin}_z L_\rho(x^{k+1}, z, y^k)$$
$$y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$$

So, this is the updates. So, suppose so for simplicity here. So, we have described two functions. So, your actual your actual thing maybe some over N function. So, f N x N plus still f n x n instead we are just adding over two function for simplicity and this is the equality constraint. So, for some values of a you can think that this will be x is equal to Z. So, you can take A is equal to 1, B is equal to minus 1 and c is equal to 0. And this constraint will be x is equal to z as a special case of that. So, in general ADMM can solve both these problems ok. So, how does it work ok?

(Refer Slide Time: 34:52)



Alternating direction method of multipliers

- ADMM problem:

$$\min_{x,z} f(x) + g(z)$$

subject to: $Ax + Bz = c$

- Lagrangian:

$$L_\rho(x,z,y) = \underline{f(x)+g(z)} + \underbrace{\boxed{y^T(Ax+Bz-c)}}_{\text{penalty}} + \underline{\frac{\rho}{2}\|Ax+Bz-c\|_2^2} \rightarrow$$

- ADMM:

$$x^{k+1} = \text{argmin}_x L_\rho(x, z^k, y^k)$$
$$z^{k+1} = \text{argmin}_z L_\rho(x^{k+1}, z, y^k)$$
$$y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \rightarrow \text{G.D. on } y.$$

initialize z^k, y^k

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION

So, as you can see even in this case the Lagrange multiplier rather yeah. So, so the Lagrange multiplier is the same. So, you have the constraint. So, this is the penalty of the or rather the Lagrangian function is the same. So, you have the penalty, so this is the penalty term ok. This is the objective function. And this is the augmentation term. So, this is the augmented penalty function ok. Now, the updates go like this. So, you first update the X variable ok. So, you first update the X. So, you randomly initialize, so initialize randomly initialize Z k and y k ok. So, you can for the first step, you can initialize them randomly.

So, you first solve for X ok. Then once you have solved for x you solve for Z ok. So, there is an update ordering here ok. So, first to solve for X ok, you get X k plus 1. You plugin X k plus 1 in this Lagrangian now, and you solve for Z all the while you keep the y same. And then you do basically gradient descent gradient descent on Lagrange multipliers on y which are the Lagrange multipliers. So, this is your ADMM algorithm or algorithm for that is called alternating direction method of multipliers.

So, we will see how this can be used. So, so we will see why it is not trivial to derive this algorithm. So, you have to actually prove the convergence of this algorithm. And we will also see how this algorithm can be actually used to solve all right a machine learning problem, which is for or for which we will there is a optimization problem, of course. And this problem can be solved using this ADMM method in a distributed manner in the second part of this talk.

Thank you.