Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

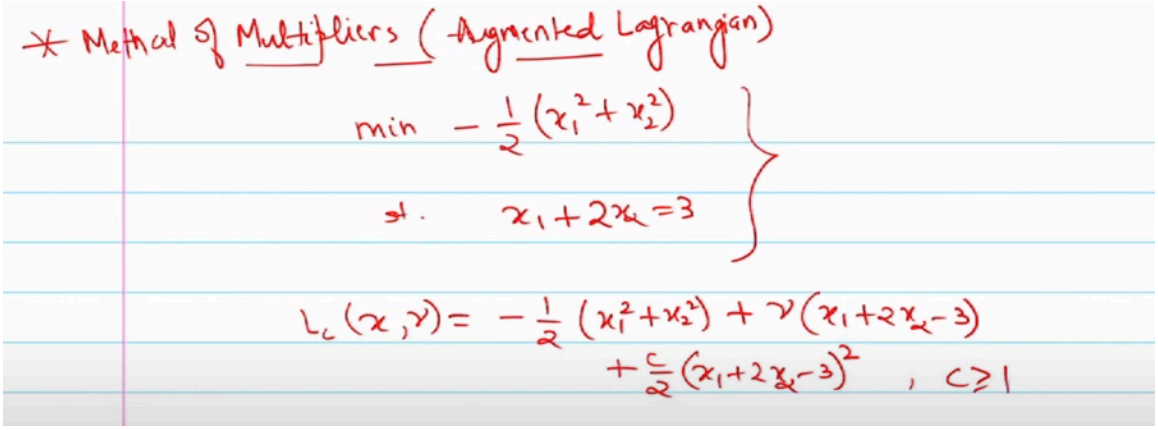
Computer Science & Engineering, Electrical Engineering, Mathematics

Indian Institute of Technology Bombay

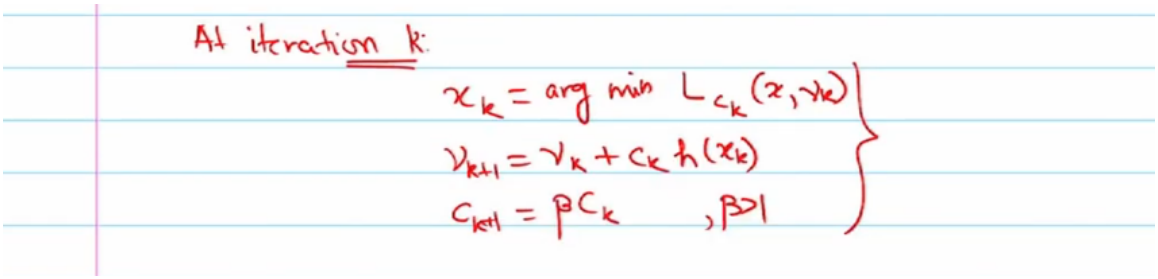
Week-7

Lecture - 25: ADMM Algorithm

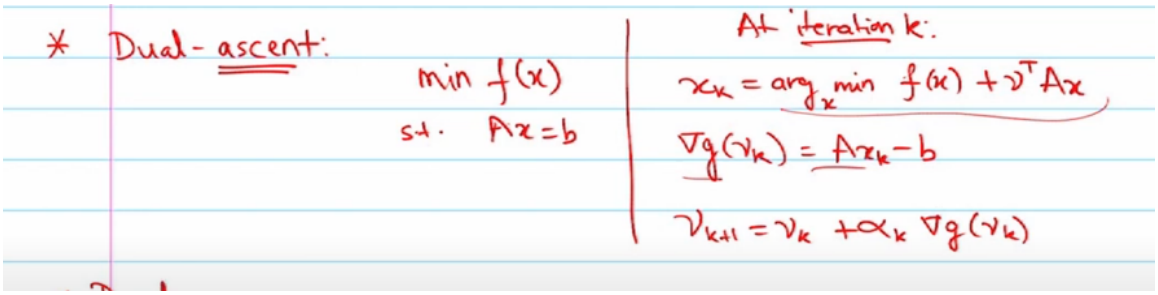
In the last few lectures we have looked at method of multipliers using augmented Lagrangian And an example could be let us say you are trying to minimize this function something like this and what is method of multipliers do or how do you augment the Lagrangian in this case. So, you define which turns out to be or rather scalar here. So, we can just and for c we know that for c greater than equal to 1, this in fact starts acting like a strongly convex objective right, even though your problem is non-convex to start with. The objective function here is basically concave, but using this augmentation you can potentially convert this to a strongly convex and that is a nice thing about augmented Lagrangian method. So, it does a nice regularization of loss or nice regularization of of optimization landscape and you can convert non-convex problems to potentially strongly So, non-convex optimization problem we can potentially convert them into a strongly convex of problems. The other approach that we looked at was dual ascent.



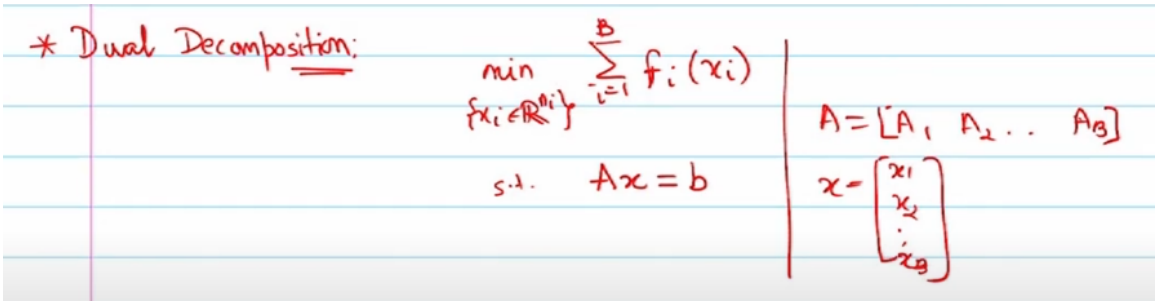
So, by the way how does the algorithm for method of multiplier works? So, for a at any iteration k we define x k to be argmin of Then you update mu k plus 1 using this value of x k and finally, you have c k plus 1 which is some with beta greater than 1. This is how the method of multiplier works. Then we looked at something called dual ascent. and we are looking at problems of the form subject to Ax equal to b.



And what did we have in terms of like as an algorithmic implementation of dual ascent? So, at iteration k, so xk is defined to be argmin of f of x plus nu transpose A x right and then you define nu k plus or let us say let us do it in two steps nu k plus 1 it is an ascent on your dual variable nu k right. So, what was particularly attractive about dual ascent? Something that we looked at in the last class, why did we look at dual ascent? What can we potentially achieve with dual ascent? That is one thing. So, when this closed form expression is not known, I mean you can obtain as obtain it as Ax minus b, where x happens to be the solution to this particular problem,

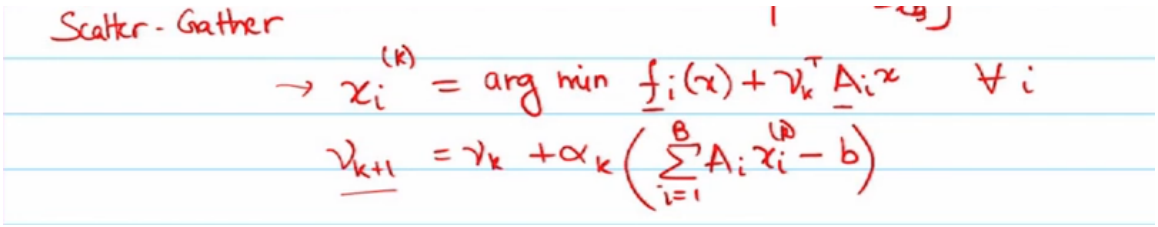


but what else? we looked at something called dual decomposition right, when you have objective function that can be decomposed into several block variables right. And the idea is now we are trying to minimize with respect to x i in let us say R sub n i summation i equal 1 through capital N or rather capital B, let me use capital B. right subject to A x or let us write it even better.



So, A x equal to b where you think of A as a block matrix of this form A 1 A 2 A b right and x is basically x 1 it is a concatenated sort of and we arrived at a scatter gather kind of

algorithm for this right and the idea was scatter or the broadcast step. So, each agent or each block is going to be run by it basically they are going to run a dual dual ascent on their own variables right. if the cost function is decomposable into these block variables, then at any iteration k x i k is going to be defined as arg min of f i nu k transpose right. So, every that is true for. So, every agent knows its own A i and knows its own f i and they are going to be running this particular step right.

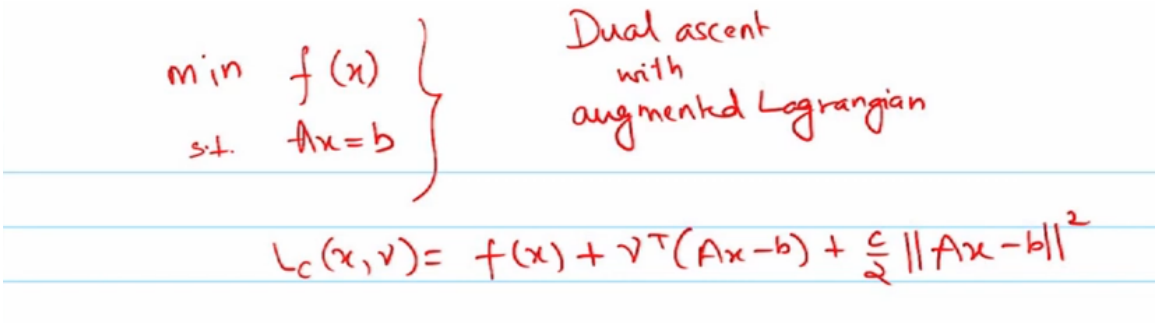


And then there is a gather step which is basically there is going to be a centralized aggregator that is going to aggregate. So, I mean if I were to combine these two steps nu k plus 1 is nothing but nu k plus alpha k times A xk minus b right. So, in the gather step a centralized aggregator would gather such that you can update nu k plus 1 So, every agent is going to broadcast A times Ai times xi to a centralized aggregator, they would be summing that up subtracting b from it. By the way this quantity is called residual, it is called residual because I mean it should be satisfied with equality technically. So, if it is not satisfied with equality, this is called residual.

So, the centralized aggregator would be computing this residual and once this new new new k plus 1 or the new value of this dual variable is available that is again going to be broadcasted back to individual agents where they would again run this particular step. So, dual ascent if the objective function is let us say you can decompose the objective function into several block variables, dual ascent in some sense allows for parallelizability right. So, you can make the problem parallelizable and that is one particular attractive thing about dual ascent. But what happens if I, but then dual ascent does not have this nice property of method of multipliers right where you can convert a potentially non-convex. Why? Because let us say I am trying to again let us consider the case when I am trying to minimize f of x right.

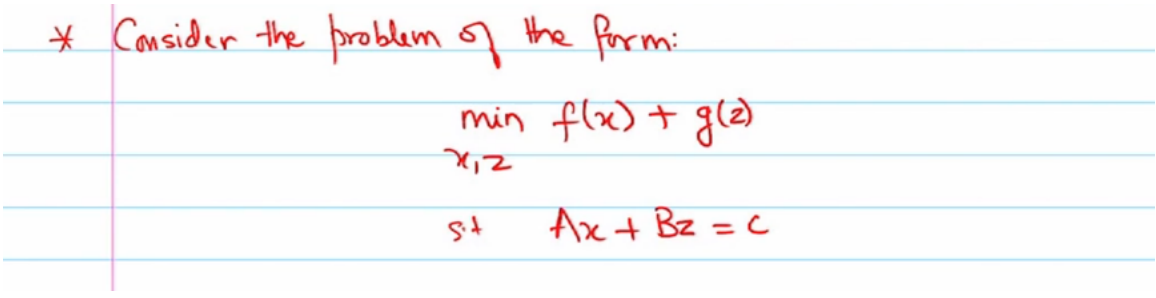
let us say subject to Ax equal to b ok. And if I let us say on top of it if I try to run, if I try to combine dual ascent with augmented Lagrangian. So, the augmented Lagrangian if I look at for any c for this particular objective, it gives, it looks like f of x plus nu transpose Ax minus b. So, even if your original f of x may have been decomposable into several block variables, this one would not be decomposable anymore, because you will get cross terms of the form x 1, x 2, x 2, x 3 and so on. So you cannot enjoy the nicer properties

that you had with the augmented Lagrangian if you try to combine dual ascent with augmented Lagrangian.



Is the premise clear to everyone? So, what do we want to do? We want to retain the sort of nice properties that we have with augmented Lagrangian. One thing that we know is the algorithm is pretty robust and it basically regularizes the optimization landscape. So, you can basically work with strongly convex functions which are much easier to optimize, but at the same time it does not like if I try to look at do augment the method of multipliers directly, there is no way for me to parallelize the method of multipliers because of this particular constraint here. So, all the x i x j kind of terms would start appearing over here and that is when you cannot parallelize it that much. So, the question is can we enjoy the best of both worlds? So, what do we want? we want basically augmented lagrangian in some sense optimization landscape and we also want algorithm to be parallelizable.

which is basically property of dual ascent dual ascent or dual decomposition right. And the answer to this question is yes I mean the fact that we are addressing this here is I mean it should be obvious that I mean it should work and that is the your ADMM algorithm or the alternating direction method of multipliers. So, let us see how ADMM works and then I think it would be clearer. So, basically enjoys the best of both worlds it has this parallelizability as well as you can work with the strongly potentially strongly convex landscape. So, let us consider the problem of the form.



So, we want to minimize. So, x and z are two variables and we this objective function in

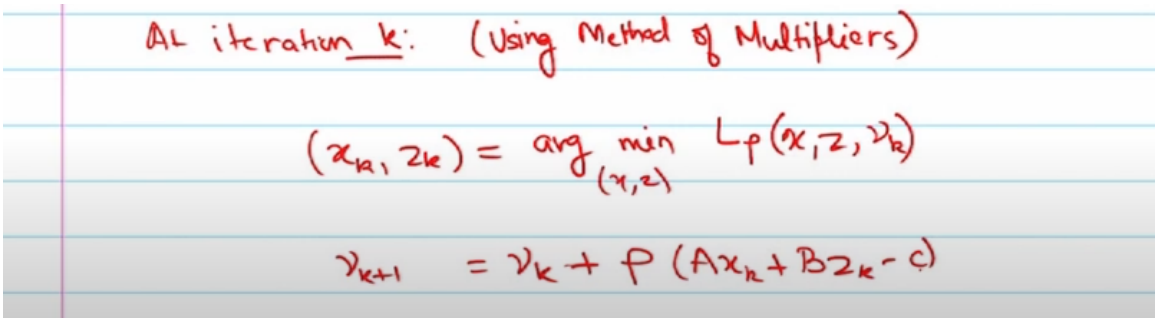
some sense it is decomposable in the primal variables x and z ok. I mean you can view if you want to view it you can view it as f i x i it is it is exactly the same thing, but. So, let us just for now let us just focus on two I mean two different primal block variables. So, subject to a x plus b z or b z is equal to c ok.

So, the usual method of multipliers. So, what would be the iteration look like for the method of multipliers. So, let us say I define let us for now let us fix I mean we are not going to be varying our ck here. or the augmentation coefficient. So, let us say that augmentation coefficient now is going to be rho because I am using c to denote this equality constraint here right.

So, the right hand side of the equality constraint. So, let us denote this augmentation coefficient by rho and what does rho evaluate to? So, first of all it is going to be a function of x z and nu right and this is going to be f of x plus g z. So, this is your augmented Lagrangian. So, if I had approached this problem using method of multipliers, what would have I gotten? So, I would have gotten. So, let me at iteration k, this is using method usual method of multipliers.

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

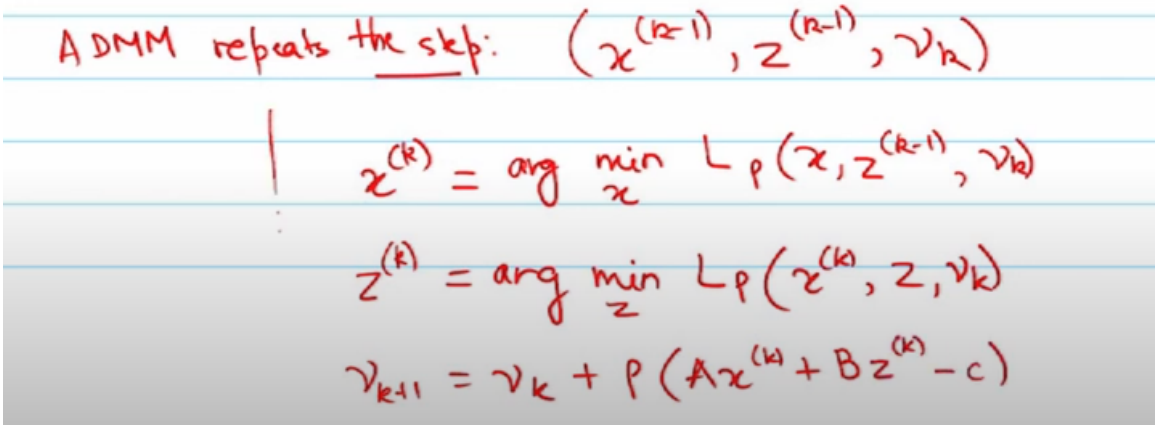
So, I would have gotten xk zk which is going to be arg min of L rho x z and nu k and then you are going to be updating nu k plus 1 as nu k. So, this is for the method of multiplies and here you can see that this particular minimization is not possible here right like if I mean if you want to parallelize it you cannot achieve this particular step because it is a concurrent optimization on x and z. and you cannot parallelize this step and therefore, I mean you cannot decompose it using the usual method of multipliers. So, ADMM is a very simple modification of this algorithm and so ADMM in fact repeats this particular step for any iteration. So, let us say at the end of k minus 1 iterations you have xk minus 1.



z k minus 1 and if you want to call it nu k or nu k minus 1 it is up to you since we have been calling it nu k so far. So, let us call it nu k. So, at iteration k what ADMM does is you basically fix the value of z and you change you basically optimize only over x. So, the reason it is called alternating directional method of multipliers is basically you alternate the minimization step once with respect to x and then with respect to z. So, in that case we can parallelize it right because you have the current values of this every agent.

So, let me first write this. In fact, ADMM works even if A and B are not full rank. It is pretty robust. So, we get current xk and it depends on the previous value of zk minus 1. So, there is no concurrent minimization that way.

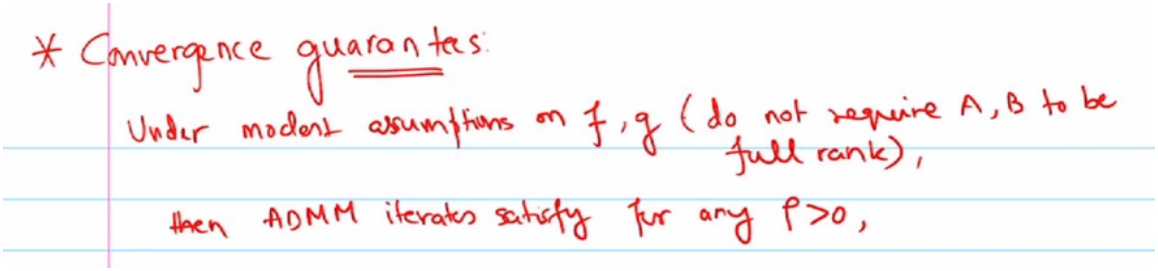
Because if z k is fixed and nu k is fixed, if I look at this particular objective, it is just a function of x and x here and x here. So, there is no minimization on z and therefore, agent like agent i can minimize on its own right without having to know. So, if you fix that let us say, then once you get your x k, then this x k is basically sent over by agent 1 to agent 2 and you can get your z k which is going to be arg min with respect to z L rho x k R z nu k. So, first agent 1 sort of exchanges its z k to let us say agent 2 exchanges its z k minus 1 to agent 1, it computes x k and then it sends over this x k to the next agent ok. And then finally, there is like let us say again just like any dual decomposition if there is a centralized aggregator, they are just going to be updating nu k plus 1 as nu k plus is this clear.



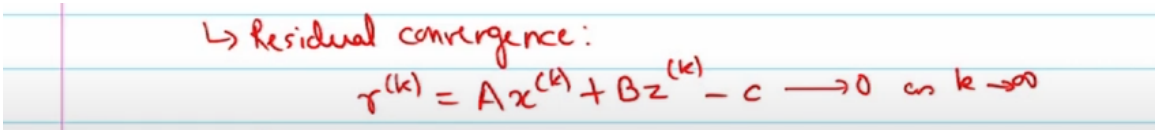
So, it is it is alternating direction method of multipliers because once it you may optimize with respect to x next then you optimize with respect to z. So, agents can basically exchange their solutions to the neighboring agents. let us say in this case and then once those values are I mean once I evaluate my x k and the other agent evaluates their z k, then we basically coordinate with the centralized aggregator and just sends over like we basically send over A x k and B z k and that is how it is. So, in that case you can still parallelize it without having to give away the advantage that we had with the method of multipliers.

I mean the same idea. So, I mean you may define a sequence, let us say that agent 1 is like at the end of k minus 1 iterations, let us say every. Yeah, yeah, yeah. So, yeah, I mean you are going to assume that anyway. But is this clear? So, what kind of convergence guarantees do we have with ADMM? In fact, ADMM is pretty robust.

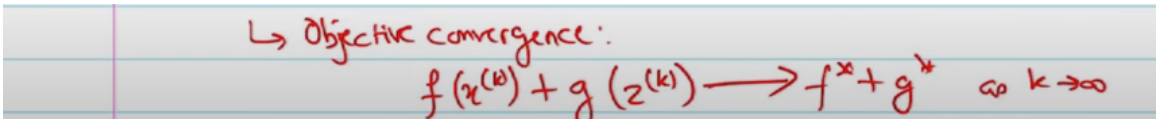
So, under very mild assumptions. So, assumptions on f and g. In fact, as I said you do not require A and B to be full rank as well. then ADMM iterate satisfy for any rho greater than 0.



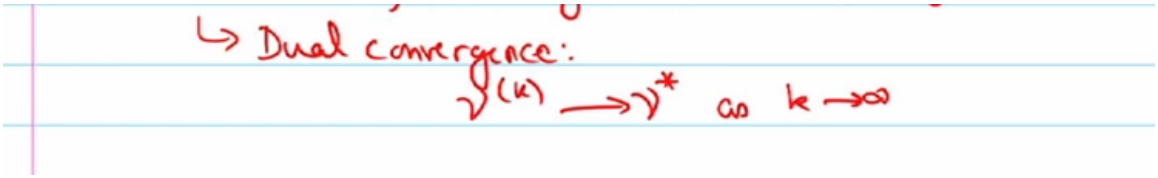
So, the following thing we get residual convergence meaning that this residual rk which is defined as B of x k B of z k minus c. So, this goes to 0 as k goes to infinity.



You have objective convergence meaning f of x k plus g of z k that basically converges to f star plus g star as k goes to infinity.



And then you have the dual convergence as well which is basically convergence on nu and it says that nu k goes to nu star as k goes to infinity. there is a reason we did not use rho k. In fact, it works for any rho greater than 0. You can choose to work with very large value of rho, but that it that comes at a risk of making the problem ill-conditioned.

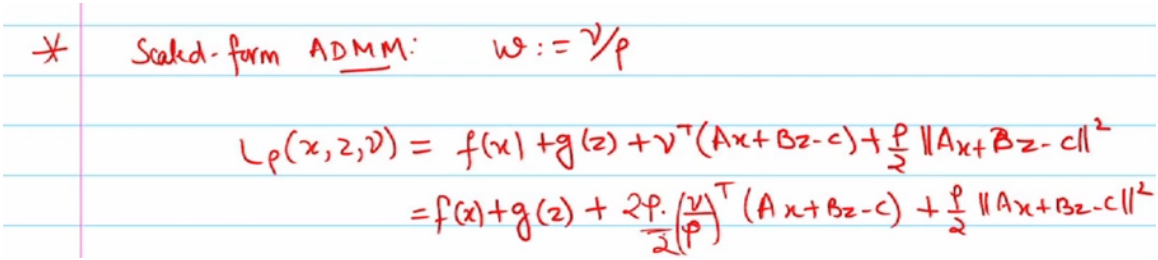


You may potentially get faster convergence with larger rho, but you may also introduce some numerical instability because of the ill-conditioning, but then yeah I mean you do not unlike the previous case you do not really have to keep increasing your rho every time. So, that is that is one advantage of this. You can use rho k plus 1 as like some c times some beta times rho that is fine I mean you would still get the convergence. You can do that as I said like if you use a very large value of rho I mean. you may get faster convergence and that is what because you eventually like in the previous case we had to be above certain threshold so that that's why we were trying to increase ck because we did not know where that threshold was so we were trying to increase it above certain c bar first of all that is not the case here the other thing is if you try to increase your increase the value of rho you may get faster convergence, but it comes at the like the disadvantages that you get ill conditioning of the Hessian.

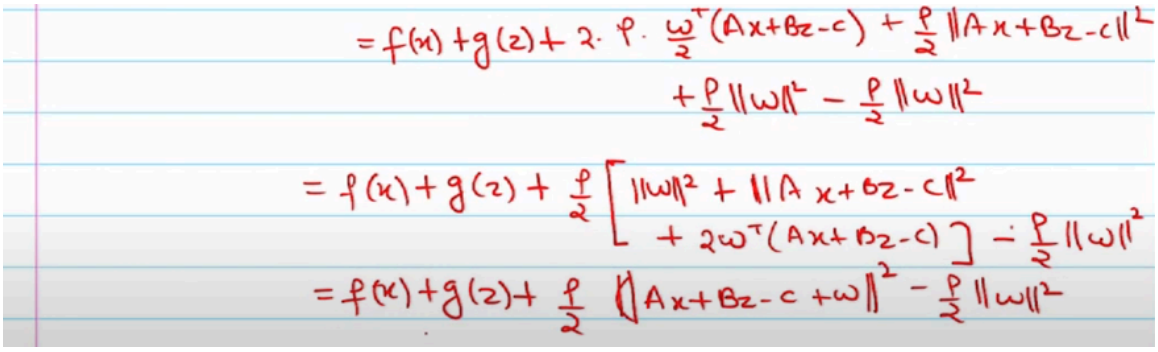
So, your gradient descent or gradient descent rather iterates that may be ill conditioned, but you potentially get you may potentially get faster convergence. A smaller value of rho is more stable, but then the convergence is slower. So, that is that is the tradeoff. So, there is another version of ADMM. So, as you can see that in nu k plus 1 here we kind of I mean there is a this rho sort of appears in before the residual.

So, there is another form of ADMM which is called scaled form ADMM, where instead of working with nu you kind of work with w which is defined to be nu over rho. so that I

mean you directly get these are iterates in terms of the residuals and this directly captures the residuals. So, the idea is. So, if I look at the L rho x z and nu earlier, so that was what f of x plus g of z nu transpose Now if I want to write it in terms of w, this would be f of x plus g of z. So, if I want you can try and basically multiply and divided by 2 right.



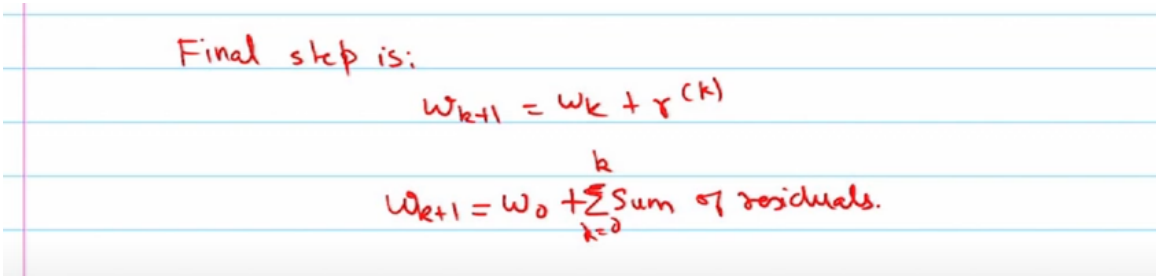
Essentially you want to get a square completion formula. So, this what you get is f of x plus g of c plus 2 times let us say rho into w over 2 transpose ok. and if you want to do a square completion here right. So, you essentially need to add and subtract plus and minus rho by 2 w norm square right and then you can complete the square you can do the square completion here. Because you get if I you can write this as A x plus B z minus c plus w norm whole square right and that is how you can do the square completion as well.



So, what you get is f of x plus g of z plus let us say rho over 2 times. ok and this term over here is nothing but ok. So, this is your I mean writing the same augmented Lagrangian in terms of w. So, if now you try to for a fixed value of w, this is what you want to optimize for a fixed value w and z, you first optimize this with respect to x, then you fix your x, you get a new value of x, and you fix your w and you optimize with respect to z. And then you have a final update, which is this particular update over here.

And if I divide this by row, you get w k plus 1 is equal to w k plus t residual r k, right. The final step is which is directly in terms of the residual. So, if I simply do a telescopic kind of sum here, if I just simply add this up, what do I get? The sum of residuals right. So, you can use this to monitor the algorithm directly by looking at W 2 and that is why

people often prefer scale form ADMM, but that is ok. So, that is what you are going to get, but it is the same algorithm it is just a different way to look at the same algorithm ok.



mean I have already written sum of residuals, so let me not add. Thank you.