

Distributed Optimization and Machine Learning

Prof. Mayank Baranwal

Computer Science & Engineering, Electrical Engineering, Mathematics

Indian Institute of Technology Bombay

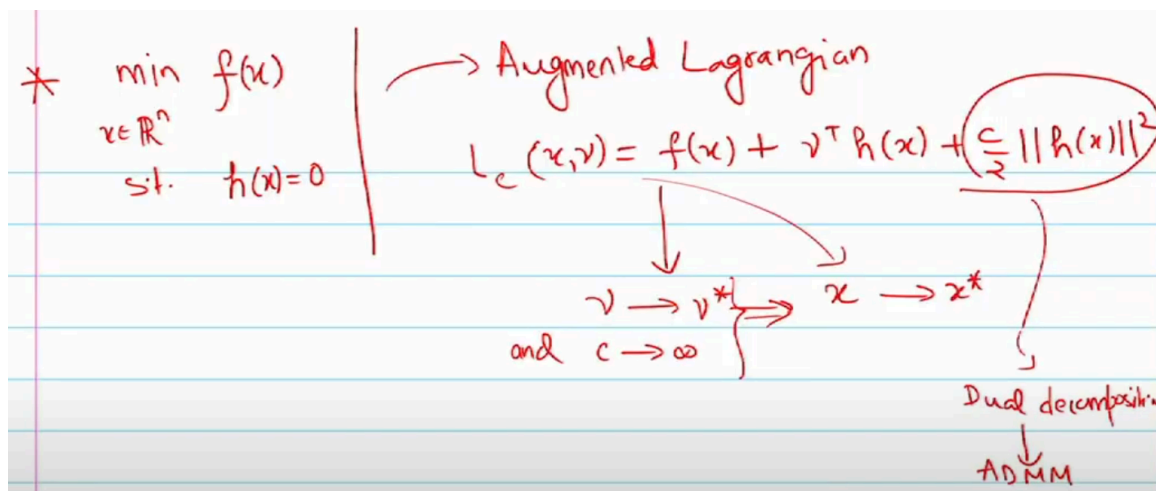
Week-7

Lecture - 23: Method of Multipliers

In the first few lectures we started with the unconstrained optimization and then we in the last lecture we kind of looked at constrained optimization and that too using augmented Lagrangian right. So, just to briefly recap. So we are looking at problems of the form. And what did we construct in order to solve this? So we looked at something called augmented Lagrangian. which was of the form. Let us say you have the current value.

So, essentially you construct a Lagrangian. Let me call it C which looks something like this. we looked at the convergence mechanism this augmented Lagrangian methods or method of multipliers. So, if we if ν goes to ν^* the ν^* is the like if you look consider the original Lagrangian ν^* is the optimal dual variable for the equality constraint.

If ν goes to close is close to ν^* and c goes to infinity right. Then you can show that x that minimizes this augmented Lagrangian would actually go close to x^* right. So, that was the idea and the reason we add this particular term over here is because we want to regularize the landscape, we want to make it much smoother potentially convert non convex problem into convex problem and so on. So, today we are going to be looking at I mean sort of looking at the algorithm I mean last time we sort of discussed the convergence mechanism, but today we are going to be looking at the algorithm. which is the and then that would also lead to something called eventually something called dual decomposition.



And through dual decomposition we would then be introduced to this alternating direction method of multipliers ok alright. So, how does this algorithm work? So, let us say we are at iteration k . we are looking at method of multipliers ok. So, we are at iteration k and at k th iteration you define you find your x_k which is essentially $\arg \min$ ok. So, x_k is essentially the $\arg \min$ of So, so far we know that we can solve unconstrained optimization problem relatively easily using either gradient descent or heavy ball or other things right.

So, we are going to run and we are basically going to solve or basically minimize the unconstrained Lagrangian which is in fact unconstrained augmented Lagrangian right. So, given the value of nu_k the current value of nu_k and the current value of this constant c_k or the coefficient c_k we are going to find our x_k . then we are going to be updating nu_{k+1} using ok . So, this is how we will be updating nu_{k+1} . So the hope is that this eventually converges to new star right that is when your x would also converge x_k would converge to x^* .

So if this converges to new star then this would converge to x^* right that is the idea and in order to do so we also need to ensure that c also becomes very large right. So that means we would also be updating our c like this where β is than 1 and typically β is 1 mean and there is no particular rule, but typically β is chosen to be a number between 5 and 10. So, this is your algorithm ok, this is the this is your algorithm and let us try and see how why this particular choice of nu_{k+1} and so on ok. Are the steps of the algorithm clear? What happens at k th iteration? So, this way you update your new iterates and in the next iteration $k+1$ then you would find x_{k+1} using the c_{k+1} that you have obtained or updated and the new $k+1$. So, there you are going to be running gradients.

So, so far we know that we can solve unconstrained minimization problem right unconstrained optimization. I mean obviously, here we are assuming that we can solve it

exactly, but I mean the results exist where even if you solve it approximately still works it is. Yeah, yeah, but satisfy I mean satisfiability of constraint is important here right. Sir, if you have the x_k . So, you will be at one of the local minima's, but the I mean if you can still I mean.

So, again assuming ν goes to ν^* and c goes to infinity you would be able to do so. So, we will come at an example where you would see that that to like I mean you will see that in action. It will be again like you can only provide local guarantees. So, you it will converge to one of the local as I said like if it is a non convex function with the multiple with multiple local minima in general it is impossible to provide global guarantees right in general right. So, we can only talk about like local guarantees there right, but if it is if it is a different kind of stationary point or maybe a case where like I mean PL inequality kind of non-convexity then you can I mean you can provide global guarantees.

At iteration k: (Method of Multipliers)

$$\begin{aligned} \underline{x}_k &= \arg \min_{\underline{x}} L_c(\underline{x}, \underline{\nu}_k) \rightarrow x^* \\ \underline{\nu}_{k+1} &= \underline{\nu}_k + c_k h(\underline{x}_k) \rightarrow y^* \\ c_{k+1} &= \beta c_k, \beta > 1, \beta \in (5, 10) \end{aligned}$$

$$L_{c_k}(\underline{x}, \underline{\nu}_k) = f(\underline{x}) + \underline{\nu}_k^T h(\underline{x}) + \frac{c_k}{2} \|h(\underline{x}_k)\|^2$$

Sir, you can understand that we want C to be as large as possible. Yes. So, we can see by.

Yeah. Right. But how do we know that the second one. Yeah. So, that is what I was I am going to come to. So, what is the Lagrangian at $C_k \times \nu_k$. So, that is f of x_k .

plus ν_k transpose h of x sorry there is no x_k ok. If you want to minimize this unconstrained Lagrangian. with respect to x . So, you would set the derivative that is the first step that is what you are trying to. So, you would want to ensure that this goes to 0 right for the given value of ν_k and c_k this I mean this is the first step.

No, I mean this is this is what you are trying to mean oh sorry h of x here this is what you want. Now what is the derivative of this? If I set the derivative, this you want to set

to 0 right. May assume this is a element wise of multiplication. So, this you want to set to 0 right. So, this is coming from the augmented Lagrangian part.

Let us say you are not looking at augmented Lagrangian, you are looking at the original Lagrangian. Suppose that is the case, you are looking at the original Lagrangian. and in the original Lagrangian your L would be $L(x, \nu)$ is simply $f(x) + \nu^T h(x)$ and if I take the derivative and set it to 0 that is the stationary condition implies. So, at the optimal point ν^* right. x also needs to be optimal.

So, at the let us call it x^* here we will call it x_k whatever. So, we basically we want this to go to ν^* right and that is why this particular update ν_{k+1} is $\nu_k + c_k h(x_k)$. Because if this goes to ν^* and c goes to infinity then you know that x_k would go to x^* . So, that is how this particular update for the ν_{k+1} or ν_k comes into picture. So, this is the method of multipliers or the basic algorithm again we are we do not assume the value of we do not know the value of ν^* we are just running this, but then it would happen that this converges to ν^* and if c goes to infinity then you are done.

① Step 1: $\nabla_x L_{c_k}(x, \nu_k) \rightarrow 0$

$$\nabla f(x) + \nabla h(x) (\nu_k + c_k h(x)) = 0$$

Suppose you consider the original Lagrangian:

$$L(x, \nu) = f(x) + \nu^T h(x)$$

$$\nabla_x L = 0 \Rightarrow \nabla f(x) + \nu^T \nabla h(x) = 0$$

The other thing that you can look at from here is $h(x_k)$. So, why do you need c to go to infinity? if you look at $h(x_k)$ it is simply $\nu_{k+1} - \nu_k$ divided by c_k and if you want the equality constraints to hold then you want the right hand side to be almost 0 and that happens when c_k is very large right. So, this is another reason why you need c_k to go to be very large. in fact it turns out that for most convex optimization problems you do not even need C_k to be very large and in fact even for non convex problems you again you do not need C_k to be very large as long as it is above certain threshold this would work ok. So, let me write down one particular result we are not going to prove this, but it is a proposition.

So, let f and h are continuous functions and continuous functions and the constraint set is non-convex. So, for k equals 0, 1, so on let x_k be the global minimum of the optimization of the following optimization follow. subject to x_k . So, where ν_k is the sequence ν_k is bounded and $0 < c_k < c$ for all k with c_k going to infinity. So, then every limit point of the sequence is a global minimum of the original problem.

Proposition: Let f and h are continuous functions, and the constraint set $\{h(x)=0\}$ is non-empty.

For $k=0,1,2,\dots$, let x_k be the global min of the following optimization problem,

$$\min_{x \in \mathbb{R}^n} L_{c_k}(x, \nu_k),$$

where $\{\nu_k\}$ is bounded, $0 < c_k < c_{k+1} \forall k$ with $c_k \rightarrow \infty$.

Then every limit point of the seq. $\{x_k\}$ is a global minimum of the original problem.

So, as long as your sequence ν_k is bounded for every k and you have the strict monotonicity in your c_k 's with c_k going to infinity then this basically sequence x_k converges to x^* is what it says ok. So, this is the in fact this is the result that kind of that you can use to essentially prove that this method of multiplier works, but what is one particular caveat with this I mean so far we assume that we can I mean this case x_k happens to be we assume that x_k happens to be the exact minimizer of the augmented Lagrangian right. And even if you are going to use certain algorithm to run this inner minimization, you can maybe only approximately solve this. So, what happens if we only approximately solve this? So, can what can kind of guarantees we can provide? So, assumes so, caveat is it assumes that we can do exact minimization of the augmented So, the question is if we cannot do an exact minimization if you run any numerical algorithm. So, what kind of guarantees you can provide and it turns out that even in that case.

*Caveat: Assumed that we can do exact minimization of the augmented Lagrangian.

So, this algorithm is fairly robust. So, say is let x_k or essentially what you have to say is. So, given your c_k and given your ν_k this gradient of the augmented Lagrangian if that is let us say you try to make it smaller than some ϵ_k ok. So, that means you are only approximately solving the first step for the inner minimization here. So, if you only approximately solve it where where ν_k is bounded and bounded and let the sequence ϵ_k and c_k satisfy $0 < \epsilon_k < \epsilon_{k+1} \forall k$ with $\epsilon_k \rightarrow 0$ Then you can show that an x_k basically converges to x^* which is the solution to the original where you assume that the rank it has a full rank.

So, Grindrow which has rank m and if it has a full rank then you can also show that ν_k plus $c_k h$ of x_k also goes to ν^* . So, again we are not going to prove these results, but I mean what this result essentially states is that even if you solve the inner minimization not exactly as long as you ensure that the sequence ϵ_k converges to 0 and that is

what you eventually want right because as you are closer to the optimal solution you would want to find the optimal solution better, but you can start with larger values of epsilon k and that is fine too. So, you may not exactly solve it in the first few iterations. towards I mean at larger iterations is when you are trying to solve it I mean more sort of exactly or if the even if that is the case you I mean augmented Lagrangian or the method of multiplier it still works. So, in general it is a pretty robust algorithm.

* Proposition: Let x_k for $k=0,1,\dots$ satisfies.

$$\|\nabla_z L_{c_k}(x_k, \nu_k)\| \leq \epsilon_k, \quad u$$

where $\{\nu_k\}$ is bounded, and let $\{\epsilon_k\}$ and $\{c_k\}$ satisfy.

$$0 < c_k < c_{k+1}, \forall k \text{ with } c_k \rightarrow \infty, \quad 0 \leq \epsilon_k \forall k \text{ with } \epsilon_k \rightarrow 0,$$

then $x_k \rightarrow x^*$, where x^* is such that $\nabla h(x^*)$ has rank m ,
and then $\nu_k + c_k h(x_k) \rightarrow \nu^*$.

Algorithm is fairly robust, but there are certain practical issues.

So, algorithm is fairly robust is what we are trying to indicate, but there are certain practical issues. and let us try to look at it through the example that we looked at in the last lecture. Let us consider the same example. You have f of x given by half x_1 square plus x_2 square. So, the problem is minimize f of x subject to x_1 equal to 1.

ok and what are the optimal solutions? What is the optimal solution rather? So, x_1 star x_2 star is 1 comma 0 and what did we find for ν star minus 1 right ok. So, let us look at the augmented Lagrangian for this ν that turns out to be half x_1 square plus c by 2 times x_1 minus 1 whole square plus ν times x_1 minus 1. What is the gradient of this Lagrangian? Because we are running in a minimization, so we need to compute the gradient for sure. So, what is the gradient of this with respect to x ? So, with respect to x_1 it is going to be x_1 plus right and with respect to x_2 is simply going to be x_2 ok. What about the Hessian with this like Hessian let us mean if you want to call it x_6 .

Ex: $f(x) = \frac{1}{2}(x_1^2 + x_2^2)$ $\min_{(x_1, x_2)} f(x)$ $(x_1^*, x_2^*) = (1, 0)$
s.t. $x_1 = 1$ $\lambda^* = -1$

$$L_c(x, \lambda) = \frac{1}{2}(x_1^2 + x_2^2) + \frac{c}{2}(x_1 - 1)^2 + \lambda(x_1 - 1)$$

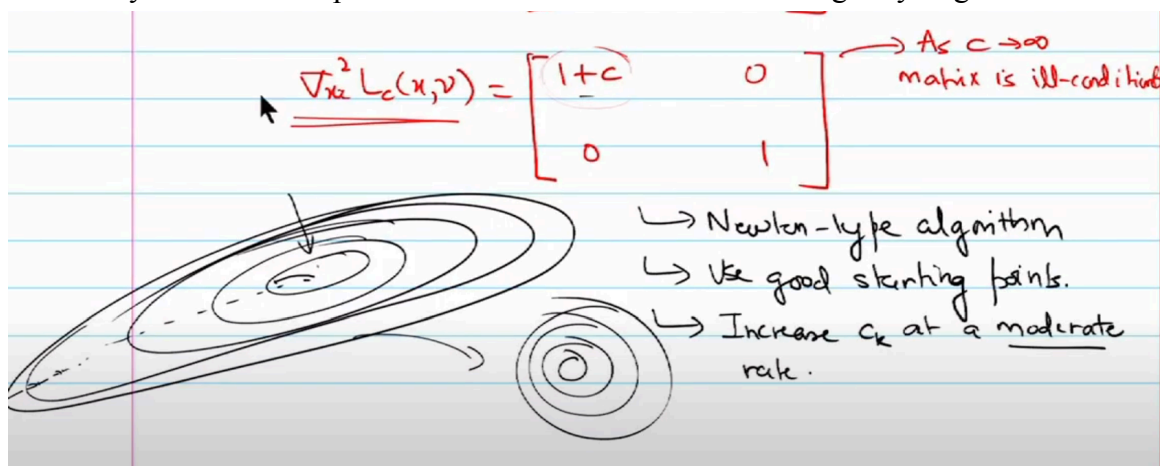
$$\nabla_x L_c(x, \lambda) = \begin{bmatrix} x_1 + c(x_1 - 1) + \lambda \\ x_2 \end{bmatrix}$$

what does the Hessian matrix look like? What is the first term? 1 plus c, 0, 0 and 1. So if you look at this particular Hessian, what happens when c is very large? It is very ill conditioned, right? So when c is very large, one of the eigenvalues is going to be very large, right? So as c goes to infinity, matrix is ill conditioned and that is when you may have practical issues in when you are trying to minimize this particular Lagrangian right. So, what happens when you have ill conditioning your function level curves they look something like this right. So, in one in one particular direction you may be making pretty good update. but your updates in another direction would be very small right and it would take a lot of effort for you to get to the optimal solution a lot of iterations.

And how can one avoid such issues when you are well conditioning? So, one thing is one about starting point good initialization, but how do we make it in like invariant to this or not invariant rather, but like I mean how can what is one way to sort of invert the effect of Use the Hessian inverse. Use the Hessian inverse right, that Newton's method. So, one way out is use Newton type algorithm and what Newton type algorithm does is once you do Hessian inverse basically if you have sort of different curvatures along different directions it basically tries to I mean sort of changes the landscape to maybe concentric circles like these. So, that is one way to. So, instead of running simple gradient descent, you can try Newton's type method on this.

Because as C becomes large, then you would have this issue that the matrix is going to build condition and you would have difficulty in arriving at the optimal solution. I mean it will still converge, but it will converge very slowly right. The other suggestion was use good starting points. If I use a starting point somewhere over here, then it would converge maybe in lesser iterations, but generally that is not known right. use good starting points and the other thing is increase Ck not at a very large rate like you need not use very large values of beta like 5 or 10.

maybe you can use a moderately large rate like let us say beta is 1.5 or something like. So, you do not increase beta c too much. So, increase c k at a moderate rate. So, that you are already closer to the optimal solution before c starts becoming very large ok.



Is this thing clear? So, let us look at one particular step of this for this particular example, let us look at the first step which is minimizing this thing right. So, if I try to minimize this thing what would be the x_2^k ? So, again how did we define x_1 or x_k ? x_k was defined to be arg min of $1 + c_k x_k + \nu_k$ right. So, if you want to find arg min of this essentially we need to find x_1 and x_2 that minimum that basically sets this gradient to 0 right. So, x_2^k is simply 0 right. So, x_2^k is 0 and what about x_1^k it is at it basically you set this to be equal to 0 right for c_k and ν_k .

$$x_k = \arg \min L_{c_k}(x, \nu_k)$$

$$x_{2,k} = 0$$

$$x_{1,k} + \nu_k + c_k(x_{1,k} - 1) = 0$$

$$x_{1,k} = \frac{c_k - \nu_k}{1 + c_k}$$

So, x_1^k would satisfy $x_1^k + \nu_k + c_k(x_1^k - 1) = 0$, right. Yeah, $x_1^k + c_k x_1^k - c_k + \nu_k = 0$, which basically gives you $x_1^k = \frac{c_k - \nu_k}{1 + c_k}$. Now, let us look at the second step of this algorithm which is this particular ν_{k+1} is defined to be $\nu_k + c_k(x_1^k - 1)$. So, ν_{k+1} is defined to be $\nu_k + c_k(x_1^k - 1)$ which is $\nu_k + c_k(x_1^k - 1)$ which is equal to

x_k .

right ok. So, what do you get x_{k+1} is x_k minus c_k into $1 + x_k$ plus c_k . So, if you sort of I think just do a simple algebraic manipulation you can. Yeah you can and using the fact that x_k^* or x^* is basically minus 1, you can show that x_{k+1} if I just subtract x^* from both these equations which is minus 1 here right. It turns out to be $x_k - x^*$ divided by $1 + c_k$. Now if I look at this particular if I look at this particular update.

$$\begin{aligned}x_{k+1} &= x_k + c_k h(x_k) \\&= x_k + c_k (x_{k+1} - 1) \\&= x_k + c_k \left(\frac{c_k - x_k}{1 + c_k} - 1 \right) = x_k + c_k \left(\frac{-1 - x_k}{1 + c_k} \right) \\x_{k+1} &= x_k - c_k \left(\frac{1 + x_k}{1 + c_k} \right) \\x_{k+1} - x^* &= \frac{x_k - x^*}{1 + c_k} \quad c_k > 0\end{aligned}$$

So, what is this update doing? For any value of c_k greater than 0, this is basically defining a contraction on x_k . So, you are getting closer and closer to new star. Your x_{k+1} will be closer to new star than x_k was right for any value of c_k greater than 0. So, if you look at what this algorithm is trying to do, it is essentially defining a contraction like basically you are getting closer and closer to new star with every in fact rather monotonically for any value of c_k greater than 0 right. So, this is how this particular algorithm works, I mean essentially inherently you are trying to make sure that x_k goes to new star and if c_k is very large then this rate of contract like basically quickly tries to go to new star, but at the expense of making possibly making the first part relatively ill condition right.

So, again there is a trade off that way, but this is what this algorithm is doing. So, for any value of c_k greater than 0 this particular works and it may also make sense because this is a convex problem. So, let us look at a problem which is not convex and I think it would be much useful to look at the role of C_k then becomes very important. the same problem pretty much, but then instead of half x_1 square I will just make it minus x_1 square. So, again what are the optimal solution or what is the optimal solution here? x_1^* x_2^*

same as 1 0, but it turns out the nu star in this case is equal to 1 and not minus 1 because nu transpose when you do nu transpose you get minus x 1 from here right.

So, nu star turns out to be 1 and not minus 1. Now, let us see what this like the different steps of the method of multipliers look like. So, when I say x_k is $\arg \min$ of L of $c_k x$ nu k . this basically gives me x_k . So, I am skipping I mean the algebra here, but I think it is clear as to how to derive this right. So, it gives c_k minus ν_k c minus c minus 1 ok, which basically goes to x star 1 comma 0 as c goes to infinity right.

and ν_k I mean even if ν_k does not go to infinity it is still ν_k does not go to ν star it is still c_k goes to here. But the interesting thing happens when we try and write the second step which is ν_{k+1} . So, ν_{k+1} here is essentially you can show that or rather $\nu_{k+1} - \nu^*$ with ν^* equal to 1 now. this turns out to be $\nu_k - \nu^*$ divided by $c_k - 1$. Now unlike the previous case where any value of c_k worked, in fact for a value of c_k bit which is let us say you end up choosing c_k as 1.

EX. non-convex problem.

$$\min_{x_1, x_2} \frac{1}{2} (-x_1^2 + x_2^2) \quad (x_1^*, x_2^*) = (1, 0)$$

$$s.t. \quad x_1 = 1 \quad \nu^* = 1$$

$$x_k = \arg \min_x L_{c_k}(x, \nu_k)$$

$$x_k = \left(\frac{c_k - \nu_k}{c_k - 1}, 0 \right) \xrightarrow{\text{as } c_k \rightarrow \infty} x^* = (1, 0)$$

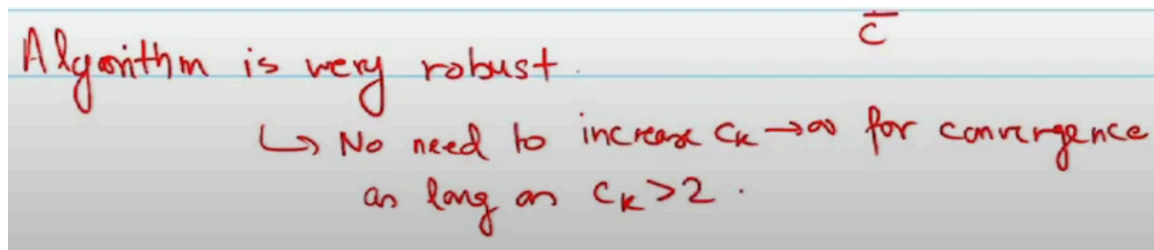
$$\nu_{k+1} - \nu^* = \frac{\nu_k - \nu^*}{c_k - 1} \quad , \text{ we must choose } c_k > 2$$

5. So, this is a diverging sequence now right. In fact ν_k will get farther and farther away from ν^* . to guarantee that this converges you have to choose a value of c_k greater than 2. So, we must choose n . remember in the previous lecture we said as there is there exists some \bar{c} as long as c is more than \bar{c} this would converge. So as long as you choose c_k and even if you do not increase I mean the value of c_k even if you choose c_k to be like you keep it constant but maybe choose it to be 2.

5 or let us say 3 or 5 or something like that right this algorithm would still converge. In practice because we do not know what \bar{c} is we keep increasing the value of c_k so that it eventually hits the \bar{c} but then as long as it has hit the \bar{c} the algorithm would converge and it will give return the correct solution ok. So, that way as I said algorithm in

general is very robust is this clear. So, I hope this example is it suffices right.

So, what is happening when we choose c to be more than 2. So, if I look at the augmented Lagrangian with c more than 2. So, I am adding actually something basically when I add this particular h of x norm square constraint with c more than 2 I am essentially trying to negate. So, there is a non-convexity because of this term and in fact the problem is making it strongly convex not just convex it is making it strongly convex by adding that thing right. So, it makes a landscape not just I mean it converts a non convex problem not just into a convex problem, but also a strongly convex problem right. So, that is why these kind of algorithms have sort of I mean fairly used in in for constraint optimization problem ok.



Algorithm is very robust. \bar{c}
↳ No need to increase $C_k \rightarrow \infty$ for convergence
as long as $C_k > 2$.

So, so no need to increase C_k to infinity for convergence as long as C_k is greater than 2. Any questions on this? Yeah, so we will talk about the convergence rate soon, but let us also look at the, I mean we can, so we know that we can solve the equality constraint optimization. How about inequality constraint optimization problems? So, now we are trying to. So, this is our primal problem minimize f of x subject to let us say h of x equal to 0 and g_j of x So based on what you have seen so far, how can you leverage the same tools to solve this inequality constraint optimization problem? Is it possible to convert inequality constraint optimization problem to equality constraint optimization problem? Then we can leverage the same tools, right? No, right? You still have λ greater than equal to 0.

So you want to work with Yeah let us say I add slack. So, what kind of slack do you want to add? Let us say I add to g_j I add a slack variables $z_{sub j}$, but then we want $z_{sub j}$ to be greater than equal to 0 right. So, we want to completely get rid of the any inequality constraint. The idea is right I mean you are in the right track. But then you want x plus to be greater than x minus. So, adding slack variables is fine, but and in fact that is what we are going to do.

So, how can so the I mean essentially we want to convert this to equality constraint right without having any any additional inequality constraints into picture. So, you are going to add slack of this form. Now, Z_j can be unconstrained ok. So, this way you can convert inequality constraints to equality constraints right.

* Inequality constrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$s.t. \quad h(x) = 0$$

$$g_j(x) \leq 0 \quad \forall j \in \{1, 2, \dots, r\}$$

$$g_j(x) + z_j^2 = 0 \quad \forall j \in \{1, 2, \dots, r\}$$

ok. So, what is the augmented Lagrangian now? So, now the now in this case you essentially have two primal variables x and z ok. There will be one dual variable which well I mean multiple dual variables however you want to call it one corresponding to h and similarly corresponding to g , but these are equality constraints, but then the z is now a primal variable here right. So, if I were to write the augmented Lagrangian. I would write this as a function of x z μ and let us say I am also going to introduce λ for this equality constraint rather and it will be of this form f of x plus μ transpose h of x plus c by 2 norm h of x square plus c by 2 So this is your augmented lagrangian now. So now what do your updates look like? Is there a question? Not a function of x , why should it be a function of x ? Yeah, I mean that that is true for any variable like any even x 1 would be constrained on x 2 and so on.

$$L_c(x, z, \nu, \lambda) = f(x) + \nu^T h(x) + \frac{c}{2} \|h(x)\|^2 + \frac{c}{2} \sum_{j=1}^r |g_j(x) + z_j^2|^2 + \sum_{j=1}^r \lambda_j (g_j(x) + z_j^2) \quad \forall j \in \{1, 2, \dots, r\}$$

But then I mean as it is like it implicitly there are functions of each other, but explicitly it is not. So, your method of multipliers how does it look like now? So, what would be the first step now? So, in the previous case we were getting x_k , now we will be getting x_k and z_k as argument So, that is step 1. Step 2 is updating your ν k right ν k plus 1 was you have a step 3 which is λ update. So, λ update is λ k plus c k times g of x_k or rather λ j if I want to call it.

ok and the fourth step is updating C k plus 1 right. So, this is how you can work with

inequality constraint optimization problem. So, we know that as k goes to infinity this term kind of goes to new star right. What about this particular term? What is lambda star? this does not tell you that it is going to be more than 0 right. So, in fact you can show. So, let us say if I had solved this problem using inequality constraints right.

So, you would have gotten a dual variable corresponding to inequality constraint and let us call them lambda star. So, max of 0 and lambda j k plus c_k d over j x k So, in fact you can show that this goes to lambda j star ok, but I mean that is not super important, but that is I mean just for your reference. So, this particular term actually goes to because you want them to be lambda j star you know they are going to be greater than equal to 0, if these are dual variables corresponding to the inequality constraint. So, max of 0 plus and this particular term that actually goes to lambda j star you can show that this is the case. They are not the same, that is why I am saying that max of 0, this term, this goes to lambda square.

① $(x_k, z_k) \leftarrow \arg \min_{(x, z)} L_{c_k}(x, z, \nu_k, \lambda_k)$

② $\nu_{k+1} = \nu_k + c_k h(x_k) \rightarrow \nu^*$

③ $\lambda_{j,k+1} = \lambda_{j,k} + c_k (g_j(x_k) + z_{j,k}^2)$

④ $c_{k+1} = \beta c_k$

$\max\{0, \lambda_{j,k} + c_k (g_j(x_k))\}$

λ_j^*

No, that is, I mean if one of them is without, let us say if it is satisfied with the, if the inequality constraint is active, right. So, then because of complementary slackness, lambda j is going to be 0. If it is inactive, then it is going to get some certain value. So this is your method of multipliers.