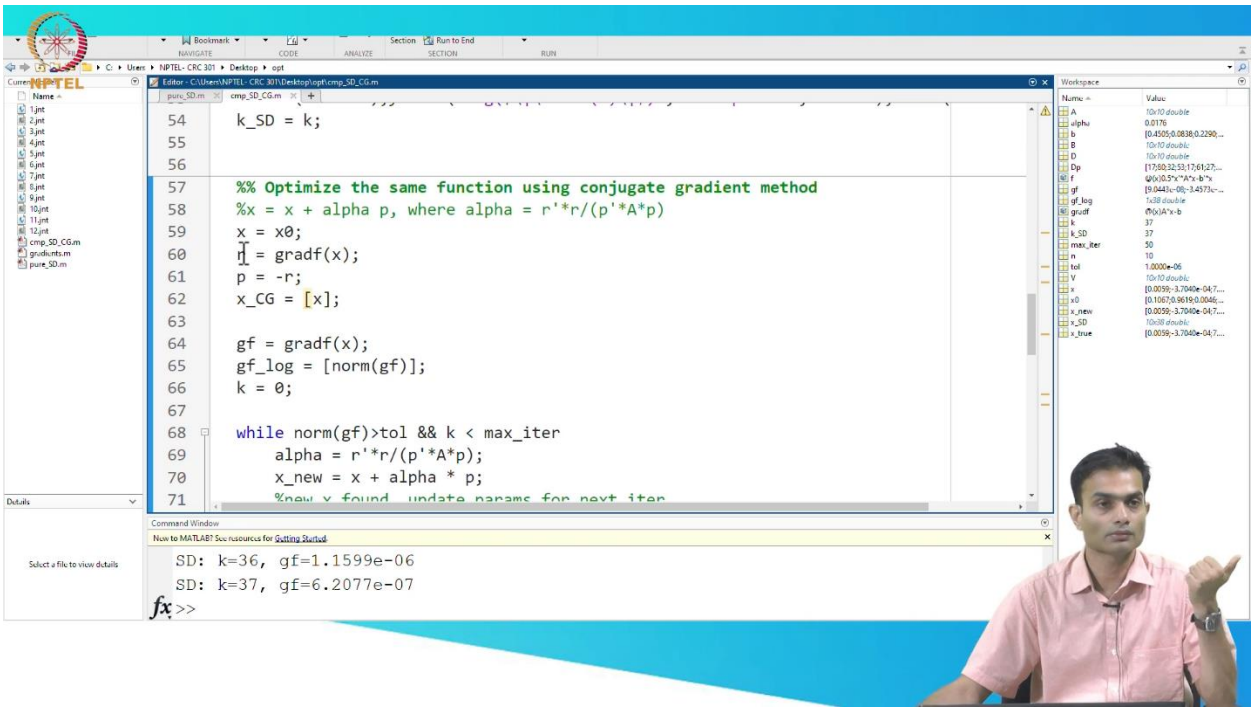
Course Name: Optimization Theory and Algorithms
Professor Name: Dr. Uday K. Khankhoje
Department Name: Electrical Engineering
Institute Name: Indian Institute of Technology Madras
Week - 06
Lecture - 43

MATLAB implementation on CGM

So, the starting part of this code is very similar to what we did last time right to generate a positive definite matrix ok. So, that for those of you who have forgotten it is just that I have created a symmetric matrix by starting with a random matrix this was a random matrix I made it symmetric. Then in line 10 I looked at its eigenvalues and I made them some random integers so that they became positive definite ok and ok and then I reconstructed A matrix as $V\,\mathrm{diag}(\lambda)\,V^T$ ok. Then there is a random b vector which is the right hand side of my problem. So, $A\mathbf{x} = \mathbf{b}$, my A is ready, my b is ready, I am ready for action right. This was the quadratic function as I showed you last time this is how you define a function in terms of **x** right. So,



$$\frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

that is my quadratic function. I have the gradient which I have written as a function ok.

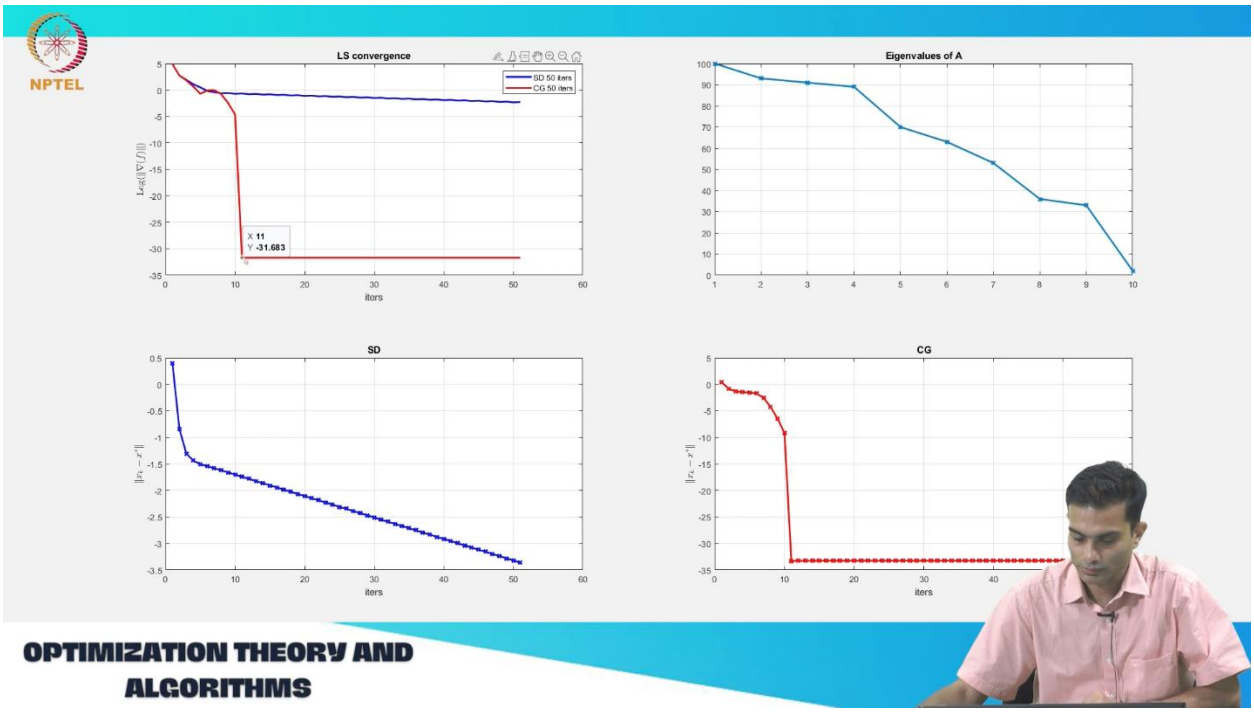
And so, let us actually run this part ok. So, A is done now I am going to prepare the function and this was gradient descent 50 steps. So, we can run this as well. So, this is the convergence of steepest descent.

I do not know why it is called LS, anyway. Now, we are looking at conjugate gradient method. Conjugate gradient method my starting point is some $\mathbf{x_0}$ which I defined earlier on same $\mathbf{x_0}$ right. Once I know my $\mathbf{x_0}$ I can immediately calculate the residual because that is simply $A\mathbf{x} - \mathbf{b}$ right, $A\mathbf{x} - \mathbf{b}$ is the same as ∇f. So, I use $\mathbf{r}$ as $\nabla f(\mathbf{x})$ ok. So, what am I doing in these steps? I am defining all the initial parameters to enter inside the CG loop.

Initial parameters is that the conjugate direction $\mathbf{p_0}$ is $-\mathbf{r}$. So, that is taken care of here. This $\mathbf{x}_{\text{CG}}$ is just for me to keep track. Ideally I should not be storing all of this. This is just so that I can do analysis later ok.

I just read this is GF I am storing it I need not have stored it ok. Now, the logic for the conjugate gradient method loop is the same as before. I am going to say while the norm of the gradient is above some tolerance, tolerance is some low small number ok maybe 10^{-12} or whatever and less than the maximum number of iterations which I decided ok. I have an expression for α. The simplified expression for α right remember that $\mathbf{r}$ is correctly populated at $\mathbf{r_0}$ when I enter this loop.

So, my α_0 is correctly done once I know α I already know $\mathbf{p}$ because I have entered the loop knowing $\mathbf{p}$. I can calculate my $\mathbf{x}_{\text{new}}$, I got my $\mathbf{x}_{\text{new}}$ right once I find out my $\mathbf{x}_{\text{new}}$ I should be careful I should not jump into the next loop without updating what my $\mathbf{p}$ and my β. So, far I have not used my β ok. Now, I write down my $\mathbf{r}_{\text{new}}$ ok $\mathbf{r_k} - \mathbf{r_{k-1}}$ was simply in terms of $A\mathbf{p_k}$ right. So, I get this.



Similarly, now I use my expression for β, I get my β is ready. Once my β is ready I get ready and prepare my $\mathbf{p}$ for the next time I come back inside because you notice the first step of the loop is to calculate my α for which I need the $\mathbf{p}$ to be the correct value. So, I got my $\mathbf{p}$ from here I update my residual once again right. This part is just keeping track of how well the norm of the

gradient dies down etcetera. So, that is why I am calling it bookkeeping ok and it will show me an iteration count every time.

Having done that it is just going to plot all of this and show me right. So, what we saw here was in the case of steepest descent at 37 iterations I got down to $\left| \ |\nabla f| \ \right| \approx 10^{-7}$. Now, let us run this section, you notice what happened? 9 iterations it is down to 10^{-14} right that is that is what machine precision and look at this. Note, so the red curve is the progress of $\left| \ |\nabla f| \ \right|$, I am plotting in log scale so that I can make better sense of it.

So, notice how leisurely steepest descent walks right at the end of 50 not 50, 37 iterations it is reached the log value of 10^{-14}. Whereas, this guy within the 9th or 10th iterations is down to essentially close to machine precision ok. So, this is telling you that conjugate gradient method is much much better than steepest descent over here. And here I am just plotting the eigenvalues are there any repeated eigenvalues? So, 9 and 10 the eigenvalue is the same 17 which is why I finished in 9 steps right one repeated eigenvalue gave it to me and I did not plan this by the way it is the random integer it is so it is so happened that way ok.

So, let us. So, any any questions on this anything that we should try next? So, you want to make tolerance something even smaller or remove it fine, just remove it altogether. So, let us do that. That is what you want. So, what it did was this is the entire trajectory. So, it reached over here basically it reached the solution.

So, this is $\mathbf{x_k} - \mathbf{x}^*$ how far am I from the solution. So, I started making progress in CG reach the final solution now at the 10th step and then what happened I am just wasting time basically from iteration 10 to 50 I am wasting time. So, it had said at most 10 steps, it worked in at most 10 steps. Are there any repeated eigenvalues this time? My random integers turn out to be all distinct. So, it took me 10 steps and I reached the solution in 10 steps after that I just basically wasted time ok.

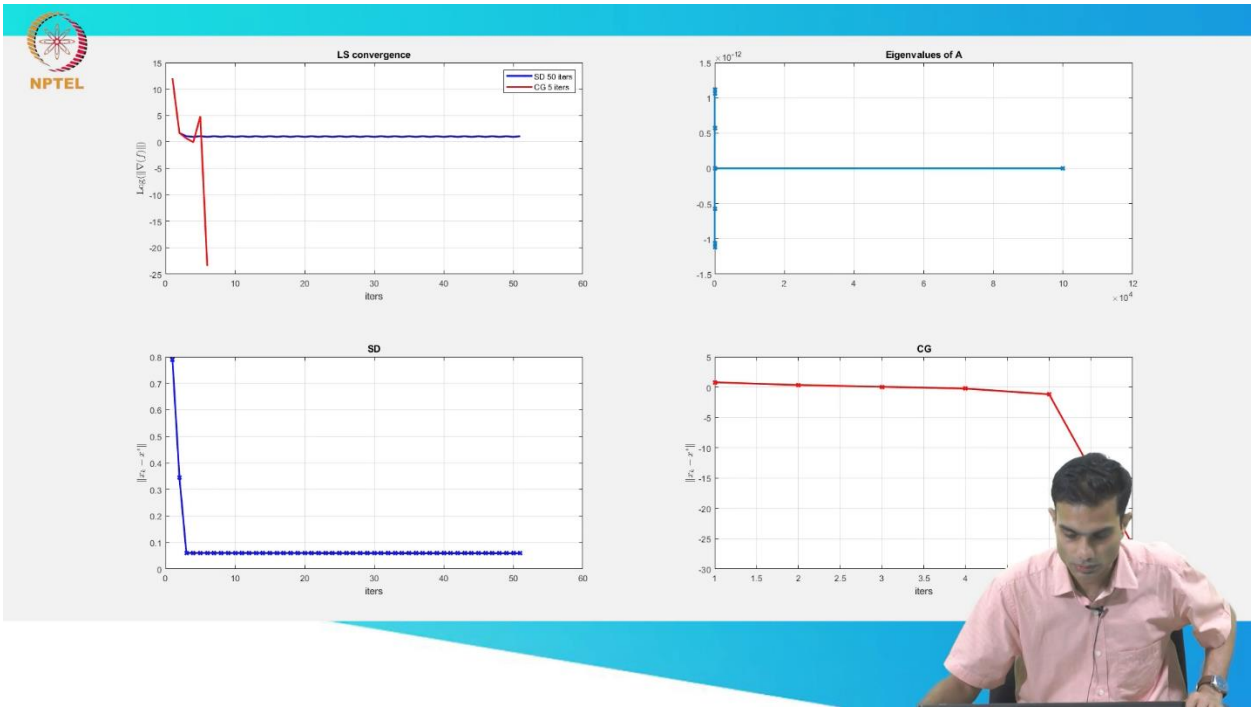
So, if you look at the here the actual values they are all the same. and it is at basically at the 10th step when I hit this and then I am just fluctuating. These are all insignificant fluctuations because it is in 10^{-14} ok. Anything else that you want to try? Why don't we ok. So, let us get this guy back.

we have a few other interesting types of eigenvalues to try. So, let us try that right. So, let us try something like this. So, I have how many distinct eigenvalues? 1,2,3 distinct eigenvalues right 1,5,10 a very artificial problem, but let us see what happens. So, eigenvalues are 10,5, and 1 how many steps did I take? essentially yeah this 3.5.

5 and all is just a plotting artifact it is not actually 3.5 iteration. So, at the end of at the fourth iteration basically I have reached my solution right. So, three repeated eigenvalues took then it just took three steps to reach the solution and you look at the convergence plot it is huge right. So, this is kind of proof that Short term gain ideas based on short term gain do not always pan out right.

When we came up with steepest descent it looked like this is the smartest thing to do just keep going in the negative gradient I am going to do well I cannot go wrong. Well look at the top left plot right steepest descent is in this way going 50 iterations it is down to -10 in the log. This guy is down to -30 in the log within how many iterations 4 iterations right. Okay at best it will

take 10 iterations if the eigenvalues were truly random. So, the more information you know or can get about the problem, the more fine tuned a solution you should choose.



So, supposing you your physics based problem gave you some knowledge that the eigenvalues are going to be clustered around something or they are going to be something something, then you should use a method that can leverage that information. It turns out that the conjugate gradient method is leveraging repeated eigenvalues in a very very nice way. Of course, we have not proven this, this is that same person Luenberger who has another book on optimization who has proved this, but we are not going into it ok. Any other experiment anyone wants to try? All are same eigenvalues ok, let us go back to that. I basically it took me one step to get there.

But what happened to steepest descent? Steepest descent also took one step only. Why? Why is the first step the right step? Because it is a circular thing right. So, wherever you go you are looking directly steepest descent is directly taking me to the center. So, that is what it is. In fact, what we have solved over here is actually a very very simple problem because what will A matrix actually be? It is $V\lambda V^T$.

λ is 7 times identity. Take it out. What am I left with? It is the identity matrix. I basically solved my solution $x = b$. If I had, if this problem was solved by a human instead of a computer, we would beat the computer by one iteration.

I mean i.e. no iteration ok. Any more challenging problem to try over here? A large condition number very good ok. So, let us try this one ok. Highest eigenvalue is 1000 ok. So, what happened to steepest descent? Steepest descent died essentially right. Conjugate gradient method still survived right.

It has reached. So, if I look at $x - x^*$ it has reached it is essentially reached in the this is in the log scale it is reached right. If I look at the residual over here 10^{-13}. and 5 iterations because

there are 5 distinct eigenvalues 1,5,10 and a 1000 right and here I have. So, if I if the eigenvalues were not distinct it would take 10 steps if there were 10 distinct eigenvalues with a bad condition number also it is it is working well we can get even let us let us make it let us get even greedier ok.

At some point CG should also break. Well, at this point it has not, but CG has done something strange over here right. Of course, our steepest descent died much early on. Conjugate gradient method has reached a stationary point which is evident from the fact that $x_k - x^*$ is also gone in the log scale to 10^{-12}. There is something strange that has happened here. So this actually is going to motivate what we will do next.

The way I have implemented CG in this module is you would never implement CG in this way in real life. What I've shown you is a textbook implementation. Why? Because this is not factoring for a bad condition number. So, there is a cousin of the conjugate gradient method called the preconditioned conjugate gradient method where you try to fix this condition number problem.

So, we will come to that next. So, if you take any real life implementation of CG, no one would write it the way I did. They would do a preconditioning and then run it. So, we will we will look at that next. This is purely due to numerical issues. The condition number is what is messing you up.

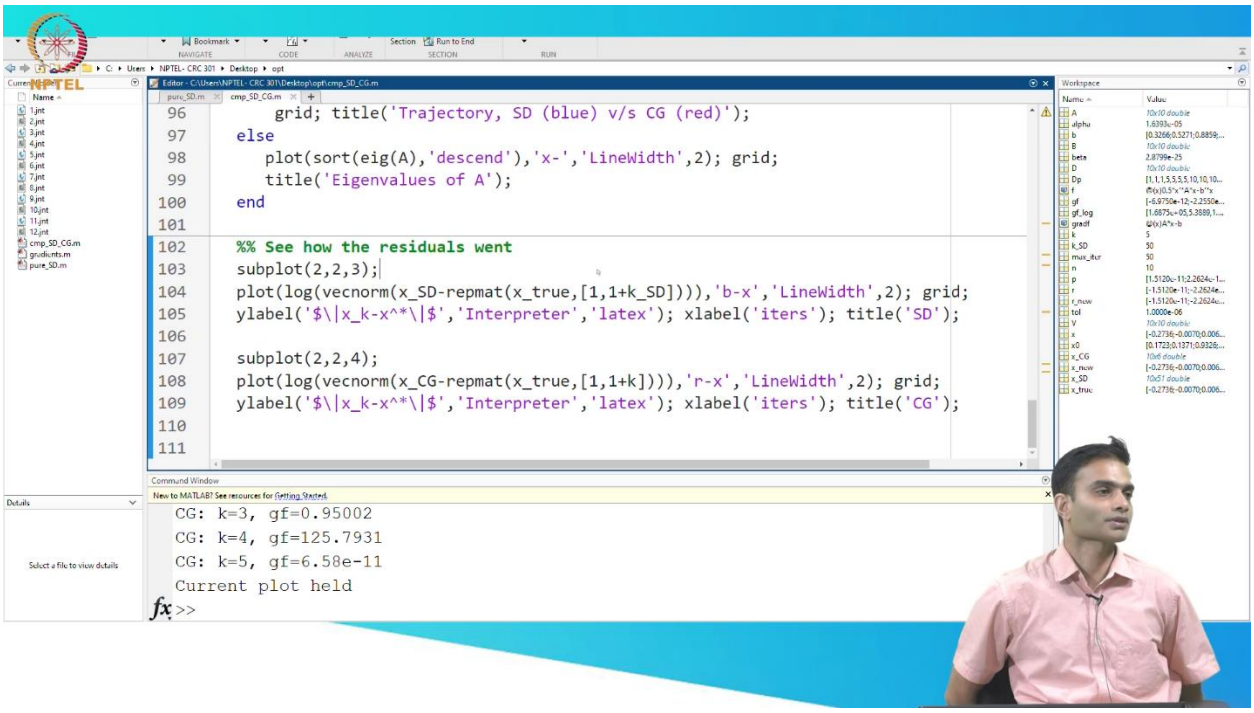
That is that is about it in terms of what I wanted to show you about the conjugate gradient method. So, you have at your disposal now two solid techniques you know steepest descent, you know its linear convergence, you have conjugate gradient which has also linear convergence, but much faster over here with nice properties right. So, you can actually begin applying this in your research problems or whatever you know would be better to wait for the for the preconditioning so that you get even more robust behavior depending on the condition number ok. And after that we will see how we can take this linear CG method and adapt it to a non-linear problem which is very very powerful ok. You can do the same thing with steepest descent also, but CG works better like how it worked earlier ok.

if you will will it work better you mean? How will the algorithm get modified? How will the algorithm get modified if I have a regularization? So, if the regularization is a very nice regularization in the way that I can write it once again as $\frac{1}{2}x^T A'x - b^T x$ then it will basically go the same way. If you cannot write it in that simple way then we will have to probably take the non-linear version of CG ok. The non-linear version we will come to it is also very straight forward. So, there is no problem over there. In the bottom? Bottom I believe it is log you can just confirm it.

yeah log. So, this is log for steepest descent and CG and yep. I will upload this code as before and then you can see. yeah here you go this is actually what. So, log of $x_{SD} - x_2$. So, these are just some bells and whistles if you want to have your legends have latex font in it ok.

When you have too much time you do these things ok, but it gives you good graphs. so as you read this one suggestion to all of you who particularly who are somewhat new to MATLAB go through this code it's a simple hundred line code you should understand every single command that is happening over here you should know it you should internalize it and begin using it in your work right so for example what is repmat does anyone know what is repmat replicate a

matrix ok so small tricks like this so that you avoid writing loops, ok. How does? We have not actually implemented any CDM here. So, it should look similar because it also has the property of at most n steps, but for CDM then you have to tell me how are you picking the p's. So, let us say you chose to do the eigenvalue decomposition, you can do it, I suspect it The only difference between conjugate gradient and conjugate direction method is that you do not invest that cost in evaluating the P's in the case of the conjugate gradient.



You are starting with steepest descent and then building up step by step it is iterative that. So, I do not need p_{100} until I come to step 100 whereas, when I do eigen decomposition I calculate all of them beforehand whether or not I need it. The trajectory will qualitatively be the same because the directions are different that there nowhere did we claim that the conjugate vectors that I got now are eigenvectors. So, different it is going along different directions. about its descent whether or not it is a descent direction.

Well we kind of did in the sense that you remember we said that if when you look at the expression for α_k if it is not a descent direction α_k becomes negative right. There was a $r_k^T p_k$ in the expression for right here right α has ok not in the original expression for α there was a $r_k^T p_k$. If it is a descent direction that became minus already a minus sign this will became positive ok. However, if it is not a descent direction then α will be negative right and that seems like a problem, but the expression for x_{k+1} is $x_{k+1} = x_k + \alpha_k p_k$ and α is turning out to be negative. Instead now I replace p_k by some new vector which is equal to $-q_k$ which is now a descent direction and it is giving me a positive step length.

So, the progress happens in the same way. If it is not a descent direction, if p_k is not a descent direction that means it is an ascent direction, I get a step length which is negative which means effectively I am still descending. So, it is I am in some sense always descending. So, all of them are descending, the only difference is the trajectory in which I descend. Do I go in a zigzag way

or do I go systematically along the axis directions and reach the solution? That is a good question.