

Preconditioned Conjugate Gradient - Part 3

Let us have a look at this, let us come back to our choice of L . Now, we had our original system where there was $L^T A L$, right? This was our A . So now again, we are going to pull one rabbit out of the hat. There is a very nice theorem from linear algebra that gives what is called a Cholesky decomposition of a matrix A . You may or may not have heard of it. So, let me write it down, you can look it up. So, the Cholesky decomposition of a symmetric positive definite matrix A is a very nice $C C^T$.

NPTEL

$$\|A - KK^T\|_F \neq 0 \quad \rightarrow \text{sparse lower tri}$$

$$\Rightarrow \text{Choose } L^T = K^{-1}$$

$$LL^T r_k = y_k \rightarrow K^{-1}(K^{-1})^T r_k = y_k$$

OPTIMIZATION THEORY AND ALGORITHMS

So, $C C^T$ where C has some property, what is that property? Does anyone know? The Cholesky decomposition C is lower triangular. Now, this C being lower triangular, let us have a look once again at our $L^T C C^T L$. Let us say that I had my A , right? So, what would be the ideal choice that C should take, or rather, L should take? Let me put it that way.

Supposing I knew the Cholesky decomposition of A , what would you choose as L ? If I can make this into the identity, I am done. So, the best thing would be that $C^T L^T = C^{-1}$, right? Then I would be done. Of course, the Cholesky decomposition or exact Cholesky decomposition is almost as expensive as calculating $Ax = b$, so we are once again back in the same situation. Looks nice on paper, but I cannot use it.

However, here comes the real rabbit out of the hat: there is something called an incomplete Cholesky decomposition, which says that A could be approximately written like this KK^T , where K has a very interesting property. K is very similar to C , okay? In the sense that all the... So, what is K ? K is a sparse lower triangular matrix. So, it does not agree with L or C completely, meaning if I calculate, for example, the Frobenius norm of this, meaning entry-wise take the root mean square, what will I get? I will not get 0. So, it is not exact, and that is why it is called the incomplete Cholesky decomposition.

③ $\hat{\alpha}_k = \frac{\hat{\gamma}_k^T \hat{\gamma}_k}{\hat{P}_k^T \hat{A} \hat{P}_k} = \frac{\gamma_k^T \underbrace{LL^T}_{\text{circled}} \gamma_k}{P_k^T A P_k} = \frac{\gamma_k^T y_k}{P_k^T A P_k} \leftarrow$

④ $\hat{\beta}_{k+1} = \frac{\hat{\gamma}_{k+1}^T \hat{\gamma}_{k+1}}{\hat{\gamma}_k^T \hat{\gamma}_k} = \frac{\gamma_{k+1}^T \underbrace{LL^T}_{\text{circled}} \gamma_{k+1}}{\underbrace{\gamma_k^T LL^T \gamma_k}_{\text{circled}}} = \frac{\gamma_{k+1}^T y_{k+1}}{\gamma_k^T y_k} \leftarrow$

⑤ Introduce $LL^T \gamma_k = y_k$

So, it is cheap to compute, and that is why it is used in this way. Supposing A is, I do this incomplete Cholesky decomposition, I get my KK^T , right? So, how does that help you? So, if this is done... Now comes the practical case. Now, I can choose my L^T to be what exactly like before, right? So, here I have chosen L^T to be C^{-1} , which we said is impractical because it is as good as having solved the problem. Now, I have relaxed my requirements and said fine, so $L^T = K^{-1}$.

So far, it is good because K is inexpensive to compute. Now, the question is, what about K^{-1} ? K^{-1} may not be very nice, right? So, actually, we do not need L^T . What do I need to solve? Let us go back here.

What I need to solve is this: $LL^T r_k = y_k$. So, let us plug that in over. So, $LL^T r_k = y_k$, this is what I need to solve, which will become $K^{-1}K^{-1T} r_k = y_k$. That is what I need to solve, right? So, can I simplify this further? I can take this to the other side, right? So, what will I get? I will get $r_k = K^T K y_k$, right? Or did I get a one transpose wrong?

Let us... So, let us check. Is this expression correct? Sorry, sorry, this is what is wrong. So, L is actually... So, if I write over here, $L = K^{-1T}$, right?

So, this is where I made the mistake. So, this is going to be $L = K^{-1T}$ multiplied by $K^{-1}r_k$, right? So, what do I do next? Inverse and transpose can be swapped, right? And what I am going to get is $r_k = KK^T y_k$, right? So, this has to be solved. Now, when I asked to compute the Cholesky decomposition of A , what was given to me? KK^T , KK^T is appearing right over here.

This has been given by incomplete Cholesky. I do not have to calculate the inverse of a matrix anywhere, okay? So, that is the first... So, that is the first advantage, right? No need to compute this thing. Now, here is, I mean, this is why this whole idea of preconditioning with incomplete Cholesky is so clever. I need y_k , right? Because r_k I can compute, how do I compute r_k ? $Ax_k - b$, if I know I am at x_0 , $Ax_0 - b$, I can compute, I got my r_k , but I need, according to this expression, y_k . How can I cleverly solve this equation without having to spend order n^3 in inverting KK^T ?

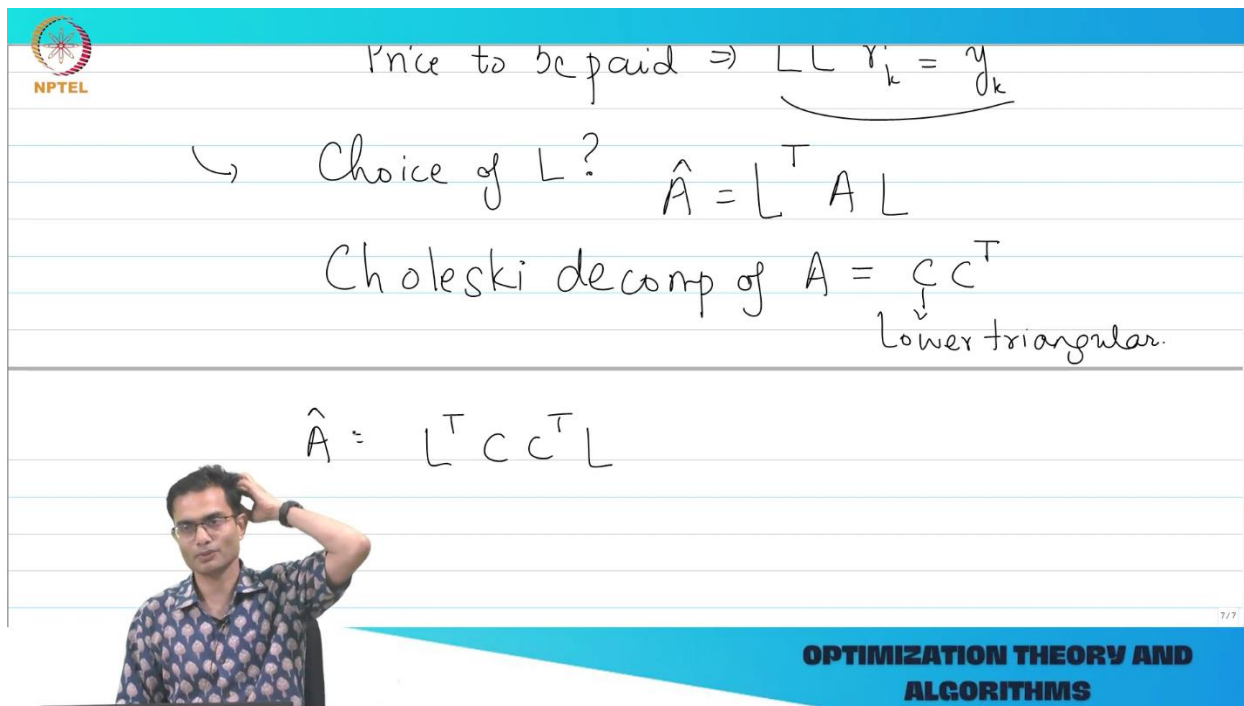
The image shows a whiteboard with handwritten mathematical notes. In the top left corner, there is an NPTEL logo. The notes are numbered 4 through 7. Equation 4: $\hat{\alpha}_k = \frac{\gamma_k^T y_k}{P_k^T A P_k}$ with a checkmark. Equation 5: $x_{k+1} = x_k + \hat{\alpha}_k P_k$ with a checkmark. Equation 6: $\hat{\beta}_{k+1} = \frac{\gamma_{k+1}^T y_{k+1}}{\gamma_k^T y_k}$ with a checkmark. Equation 7: $p_k = -y_k + \hat{\beta}_k P_{k-1}$. To the right of the equations, there is a circled 'FB' and the expression $Ax_0 - b$. In the bottom right corner, there is a video inset of a man in a patterned shirt speaking.

The hint is the structure of K . What is the structure of K ? It is lower triangular, right? So, now let us look at this a little bit: this is $KK^T y_k$. Supposing I combine this and call this z_k , right? So, therefore, this is equal to Kz_k , where K is lower triangular and sparse. It is like this. Do I need to actually invert this, or is there a clever way of solving this to get z_k ? Visualize this matrix, the shaded part is where the numbers are non-zero and the rest is 0.

Identity matrix. Something much simpler you could. So, if you had if you saw this matrix in class 12, you could solve it. What is it? Back substitution, right? The lowest... I mean, for example, z_n , right, is it available immediately? This entry, supposing this is being multiplied by z_1 up to z_n , right, z_n in one... Sorry, not z_n , z_1 . z_1 , I will get z_1 times K_{11} is equal to b_1 , I am going to get z_1 in one shot. z_2 , how will I get? Same thing, back substitution, right? Because z_1 is known, z_2 I will get. There is no inversion happening, this is simple algebra, right? This is called what? Forward substitution, I think, right? I get confused. One is called forward

substitution, the other is called backward substitution, right? Forward probably because you are starting from z_1 and going down. So, let us call this forward substitution. I may be wrong.

So, I get z_k cheaply, and it is a sparse matrix. So, I do not have to actually calculate every single thing. The number of operations may be even lesser than a dense lower triangular matrix. It may not be $O(n^3)$, well, it will be $O(n^2)$ if it were a dense lower triangular matrix, because each one is getting multiple. This is similar, exactly similar to LU decomposition, but the decomposition price was $O(n^3)$; here I am giving you the decomposition at a cheap price. Having gotten the decomposition, the solution is not $O(n^3)$. The decomposition was $O(n^3)$.



Price to be paid \Rightarrow $LL^T y_k = y_k$

Choice of L ? $\hat{A} = L^T A L$

Choleski decomp of $A = C C^T$
Lower triangular.

$\hat{A} = L^T C C^T L$

OPTIMIZATION THEORY AND ALGORITHMS

The decomposition? Yeah, LU decomposition is $O(n^3)$, Cholesky decomposition is $O(n^3)$, incomplete Cholesky is not $O(n^3)$. That is the catch, right? So, I have got z_k . Having gotten z_k , how do I get back y_k ? This will not be order $O(n^2)$ also because it is sparse, right? It may be like $O(n \log n)$ or something. $K^T y_k = z_k$. I know z_k from forward substitution; can I get y_k cheaply? What is the structure of K^T ? Upper triangular, right? So, it is like this, and this is 0. Now, what do I do? From which order do I start? I will start from y_n .

So, I will start with y_n . $y_n \times K_{nn} = z_n$. In one shot, I get my y_n , and I move back. So, I would call this back substitution, and that gives me my y_k , very elegant. And finally, I mean, this does not need to be said, but do I have to calculate a new incomplete Cholesky decomposition at every iteration? No, given A , I get KK^T , and I am with that KK^T for the rest of my iteration. So, all I have to do is one-time cost of computing KK^T , and then forward substitution, back substitution, it is very, very quick. That gives me my y_k 's, right?

Once I get my y_k 's, if I scroll back over here, I got my p_0 in step 7. So, I know the first conjugate direction. It is $-y_0$, right? Once I know my y_0 , then I know my next p is also calculated from step 6. Betas and alphas are available over here. Now, you notice the expressions are very nice in terms of y_k, y_{k+1} . Everything is available with me, right?

So, the update equations are the same. The only places in the code where I have to modify are new expressions for α_k , β_k , and one additional step of solving this forward substitution and back substitution. You should be very careful to put your incomplete Cholesky outside the while loop—a one-time step to calculate KK^T , that is stored and then used every time in each iteration, right? If you put it inside, then you would kill yourself, correct? Right. So, when we look at here r_k , r_k is needed to calculate y_k , right? This final discussion, which we had, we needed r_k to get y_k .

Any other questions? Ruben? Any part that is not clear? Should we walk through it again? We have the starting point. Let us look at our start. Let us try to write this now. Summary: So, what are we given? You gave me A and b , right? And what did I say? I said bad condition number, okay. So, then what did I do? I took A and I did not equal to approximately equal to KK^T , which was what? I am going to use the MATLAB name for it, I chose okay.

This is cheap, fine, yeah. r_k can be simply calculated as $Ax - b$. Which equation? We are never inverting it; we are just calculating it. No, it is wait, here, this expression, that is the expression we are looking at, in terms of... because L and K had an inverse relationship. So, step 1 is this. Step 2: What do I do? I am going to pick a random or your choice of what x_0 . Fine, this is your wish. What would be the next step? I need to now know, I need to move, right?

So, I need to get p_0 , right? p_0 was equal to what? $-y_0$. To get y_0 , what am I doing? Solving this... is that equation over here? r_0 is equal to $KK^T y_k$. r_0 is equal to $KK^T y_0$. How would I get r_0 ? Simply $Ax_0 - b$, right? So, I can do this by, let us give it a nickname, forward substitution, back substitution (FB), FB gave me this. So, I got my y_0 , minus that gave me my p_0 . What is the next step that I need? Alpha, right? I need alpha, α . So, an expression for this, right? The expression for this was what?

NPTEL

$$LL^T r_k = y_k \rightarrow (K^{-1})' K' r_k = y_k$$

$$r_k = KK^T y_k$$

Given by ichol

1) No need for K^{-1} !

$$r_k = KK^T y_k = K z_k$$

OPTIMIZATION THEORY AND ALGORITHMS

$\frac{r_k^T y_k}{p_k^T A p_k}$, right? So, at this point, I need $r_0, y_0, p_0, A p_0$. Do I have all of these done? So, I have this.

The next step would be, I have got my α , I have got my p_0 , where can I go next? I can go to my next x , right? So, I can write $x_{k+1} = x_k + \alpha_k p_k$. Remember, I am not writing this from first principles, I am writing this after that worked-out expression. Otherwise, if I were naively writing this, I would put hats on everything, but I simplified that to simply this. So, this is what I got. Do I know everything here? Yes, I do, right? So, I have reached my new x_{k+1} .

The original A , that is what we worked out. We can do more work, but we do not like to. This is the original A , when we started with the very first step. If you notice this, originally this expression would appear everywhere, $p_k^T A p_k$, but this is actually nothing but $p_k^T A p_j$, if you substitute it, right? So, that way, I avoid calculating some new p_k , because remember, for new p_k , I need this L^{-1} . I do not want to get into any inverses, right? So, α_k is done, it is the original A , x_{k+1} is done. Before I go to the next iteration, I should be careful to calculate what my β_k , I mean...

So that I get my p_{k+1} for the next iteration. So, I will use my β_k expression, which was also in terms of r_k and y_k , right? So, this was $\frac{r_{k+1}^T y_{k+1}}{r_k^T y_k}$. Yeah, that is right, right. Again, can I calculate this? Can I calculate r_{k+1} ? Yes, because I know my new x_{k+1} from the previous step. So, I can calculate my r_{k+1} , and I need y_{k+1} . Can I get y_{k+1} ? Yes, why? Because I know r_k , I know KK^T , and I know therefore, I can get y .

Summary - Given A, b ∇ $\left(\frac{1}{k!} \right)$

- ① $A \approx KK^T$ (ichol) (cheap)
- ② Pick random/gour choice of x_0
- ③ $p_0 = -y_0 \rightarrow$ Solving $r_0 = KK^T y_0 \rightarrow$
 $Ax_0 = b$ FB
- ④ $\hat{\alpha}_k = \frac{r_k^T y_k}{p_k^T A p_k}$ ✓

5/9

So, all of this can also be done and as a result my p_{k+1} update expression also goes through which was

$$p_k = -y_k + \hat{\beta}_k p_{k-1}.$$

Now, I know everything over here, I know my betas, I know my y 's, I know my previous p 's, I can go to the next p , right? So, basically if you look at now this what we have done so far, my earlier CG routine needed minimum modifications. In some places I just had to substitute write this y character in two places. Other than that, this apart from this extra step of forward backward calculation for getting my y 's, it just goes through.

So, this is the entire preconditioned conjugate gradient method. It is a very beautiful piece of linear algebra at a low cost. And this because if you look at I really encourage you to look at the MATLAB documentation of this incomplete Cholesky, they actually give an example of how to speed up CG in that in one of the examples. You take in several arguments you can even increase the threshold to make this more and more accurate. So, remember what I wrote over here this expression.

How different is it from A ? You can make it closer and closer, you can make it coarser or coarser. So, the more effort you expand into this, that is price paid, but your condition number is falling more and more, right? In the ideal case, if this actually became the Cholesky decomposition, your condition numbers become 1, that is as good as it gets. So, obviously, that is not worth it.

So, you may as well do it, right? So, this is if you look at any standard numerical linear algebra routine for solving that implements the conjugate gradient method, no one will have it plain conjugate gradient; everyone will have a preconditioned version of it, and in the market, there are several preconditioners we just spoke about one of them, and all of them are based on some one small or a few small clever tricks of linear algebra, which is why linear algebra is a very strong prerequisite for optimization. I would not say that both are order n^3 , solving $Ax = b$ as a direct method will be n^3 , but we are not doing a direct solve, we are doing an iterative solve, we are going iteration by iteration. At most, it requires n steps. At most, it requires n steps, and why n^2 computations? My K is sparse, lower triangular.

Oh, computing that, yeah, in that sense it is n^2 , but there, so I want to put a little bit of grain of salt over there. If I can multiply two vectors, let us say two n -length vectors, according to you, is that an n^2 operation? It is an n -operation and there are n of them, so n^2 . If it's an n -operation, that means this multiplication of this, this, this, each of them is happening serially.

That is how you're saying n . But if I have vectorized code and a vectorized hardware that does it, it's actually order 1. So that's why this becomes into how properly is your algorithm recognizing the underlying hardware. And with distributed cores, multicores, multithread and all, it becomes very hard to make a definitive statement that this is definitely order n or something. It can be lower. Conjugate direction method, there the trouble was computing p itself because conjugate direction method either you did Gram-Schmidt or you did.

You could parallelize it there, but every step that you could parallelize in conjugate direction method you could parallelize here in the conjugate gradient method. It is not the same because the amount of parallelization required over here is to bring n^2 down to let us say n that would bring n^3 down to maybe $n^2 \log n$ or something. So, that gap will kind of remain there. Right, but at the end of the day, it boils down to actual implementation, how cleverly and how nicely it is done, right, which is why you will be surprised to know that a lot of very, very serious numerical linear algebra routines in the physics community are, for example, written in Fortran to this day,

because it is one of the most efficient languages. Has anyone ever coded in Fortran? It is a beautiful language because it is so simple.

It is like C but it does not have any of the memory allocation nightmare. It is just you have to write everything out explicitly. There is no high-level abstraction there and so it works blazingly fast. So, C and C++ at best they come as close to Fortran.

They never beat it. So, that is kind of the gold standard, right? Very old-fashioned, but you look at the codes written, for example, in your high-energy physics, the Higgs boson discovery and all of that underlying it, you will find Fortran code. LIGO gravitational wave observations with high chance you will find Fortran code. Ok. So, sometimes old is gold. All of the new languages, I mean, the other end of the spectrum is python, right? It will take forever and ever. At most, I mean, if condition number is 1, A is identity, we solve the problem $x = b$.

No, the reason for this is the first graph that I showed in the class.

So, this is the situation. As I am increasing the condition number, say at most n steps was like an ideal result, right, where we did not even mention condition number. In the real world, this is the situation. Which is almost never the case. Often, I mean, in many particularly like in ML applications, I do not want to drive the error down to 10^{-16} , I will work with 10^{-4} or 10^{-5} , and for that, if I can just improve the condition number a little bit, I spend fewer iterations in doing it, right? So, it is all a very, very you have to take a composite view of what are the resources at hand, what do you need to speed up and whatnot, right? You cannot win on every front; somewhere you have to give up, which is the other name for engineering, right?

So, this is it as far as question. How do I know that the condition number of \hat{A} is less than that, right? So, the argument is a little bit roundabout if I did a perfect Cholesky decomposition, the condition number becomes 1. That you agree, we showed that, right? If it is equal to 1. So, now, I back off from perfect Cholesky decomposition to incomplete Cholesky decomposition. So, there the reasoning is that because it is incomplete, this product is not going to be identity, it is going to be something short of it.

So, with a controllable threshold, which is an input argument to the Cholesky algorithm, you can sort of have a knob on the condition number. We are not proving it, but it is numerically found to be the case. So, that is the knob that I have. So, how much resources are you willing to spend in your incomplete Cholesky gives you that much better of a condition number, but if you spend too much on it, then it is not worth it. So, you can play around with this in ICHOL in MATLAB and see that as I increase the change the thresholds it takes more and more time, then at that point it is not worth it.

So, you already have the CG code from the class from last time. So, you can see modifying it for this is a few lines, that is it, there is nothing much to it. So, when you are faced with applying CG to your research problem, right, it is going to be trivial. And then the next step, the next module that we look at, which is the nonlinear CG method. By the end of it, you will have modules ready for everything. For preconditioning, step length using backtracking and Wolfe conditions, everything put together.

You will be able to tackle your first nonlinear problem with all the nuts and bolts already there. Okay, good. Any more questions? Why don't we take any other diagonal matrix? Not diagonal

matrix, orthogonal matrix. Orthogonal matrix, you mean orthogonal means all the columns are perpendicular to each other. You have to prove that that is actually improving the condition number otherwise I mean for example, if it is a rotation matrix.

No use. It may just rotate the Eigen vectors. Eigen values will remain as it is. So, it may not help you, right? So, there are other cheaper tricks also. For example, there is a diagonal preconditioner. So, you just take the, I may be wrong, but I think if you just take the A matrix, take the diagonal elements and invert them and create a new, this thing over there. Correct, there are lots of tricks available depending on if you know something about the structure of A . This is very crucial. If I know something about the structure of A , I can leverage that information into getting a good preconditioner.

So, in the market, you will find general-purpose preconditioners that do not care what A is, and then there are these fine-tuned ones. For example, if A is Toeplitz or this or that, then you have some more tricks that you can do, right? So, these two worlds are very closely intersecting: linear algebra and optimization.