

the previous direction. This is, in a nutshell, the clever idea of the CGM: that instead of asking you to precompute the entire set of conjugate directions beforehand, it is saying use the previous conjugate direction p_{k-1} and the current residual, and there is some factor of this, which is a scalar factor, which we will choose very, very cleverly, okay. This is one. Now, this is sort of the first ingredient of the conjugate gradient method.

The second is, well, you can say it is kind of like, how do I know what is... So, what do I need to know to start this algorithm then? I need one p_0 , right? Now, if I want to pick one p_0 , and then, so it seems to me that the recipe is get first p_0 , and then the next one will be p_1 . So, in the absence of anything else, p_0 is chosen in a somewhat obvious way, which is $-r_0$. Why $-r_0$? What is the significance of r_0 or the residual? It is the gradient, right. So, p_0 is in fact what? The steepest descent.

NPTEL

2) $p_0 = -r_0$ → scalar factor
 $(Ax_k - b)$

Conjugacy → $p_i^T A p_j = 0 \quad i \neq j$

LM → $p_k^T A p_k = -p_{k-1}^T A r_k + \beta_k p_{k-1}^T A p_k$

So, in step 1 of conjugate gradient, I do what I already know, which is steepest descent with exact line search, I cannot go wrong, right? I will decrease the function value somewhat. And then the subsequent p 's I do not have to do anything expensive. You see what I need, if I need to figure out some formula for β , r_k is essentially what? $Ax_k - b$, so obviously I know where I am, so I know x_k , so $Ax_k - b$ is not that difficult to realize, so I know r_k , right? Now, how do I...? So, this is basically it, you know, this is the conjugate gradient method and it seems almost like magic because we have not proved right now that the set of directions are conjugate, right, but we will see, we—I mean all of that comes out—but the CGM is just this. So, let us see how do you choose β_k . We want to, in sort of, enforce this conjugate property, right, conjugacy. So, let us recap over here: what did conjugacy, conjugacy, right, what did it tell me? $p_i^T A p_j = 0$, $i \neq j$, right?

I have an expression here in equation, an expression 1, that has p_k and p_{k-1} , right? So, if I wanted to extract out β from here, what would be the logical thing to do? Using this conjugacy

property, left multiply by... okay, left... should I left multiply by $p_k^T A$? If I left multiply by $p_k^T A$, I will get something on the left-hand side, but my β_k term will go to 0, right. So, if I left multiply by $p_{k-1}^T A$, what will I get? I am going to get, ah, okay. So, just write it in one shot.

NPTEL

$p_{k-1}^T A p_k = -p_{k-1}^T A r_k + \beta_k p_{k-1}^T A p_{k-1}$

I want conjugacy.

$\Rightarrow \beta_k = \frac{p_{k-1}^T A r_k}{p_{k-1}^T A p_{k-1}}$

$x_{k+1} = x_k + \alpha_k p_k, \alpha_k = \frac{-r_k^T p_k}{p_k^T A p_k}$

OPTIMIZATION THEORY AND ALGORITHMS

So, left multiplier is going to give you $p_{k-1}^T A p_k = -r_k$, oops, sorry, $p_{k-1}^T A r_k + \beta_k$. Now, I want conjugacy. Remember β is still in my hand, I can choose it the way I want, right. So, I choose, if I want to choose this to be so such that I get conjugacy, so I am going to insist that this guy go to 0. If this guy goes to 0, what am I left with? β_k .

Thus, β_k therefore, has to take on this value: $\beta_k = \frac{p_{k-1}^T A r_k}{p_{k-1}^T A p_{k-1}}$. Right. So, what has this ensured?

That if I choose, for example, β_0 according to this formula, okay, so we have started, our p_0 is $-r_0$, very good, I made the first step using conjugate gradient method. I reached from x_0 to where did I reach? x_1 , because I know the step length also, there is a closed-form expression. Now, I want to calculate p_1 , and I use this formula, I have the expression for β , I will... Am I guaranteed that p_1 and p_0 are conjugate? Yes, right, that is what I—how I chose my β .

So, p_0 and p_1 are conjugate. Similarly, when I go to the next step, p_2 and p_1 will be conjugate, right. So, we are actually showing that immediate vectors are conjugate with respect to each other. We are not showing that p_2 is conjugate with respect to p_1 , and all other p 's below that will come later, but the good news is it turns out to be so, just by this very clever choice of p_k , okay.

So, before we go into detail a little bit of just, you know, what this expression kind of means, the steepest descent was using only $-r_k$, p_k was $-r_k$ that was steepest descent. But here, what am I doing? There is the immediate short-term gain term, which is $-r_k$, and there is a term from the past. So, this is what is somehow giving this method a much more, what should you say, long-term view of the solution approach as compared to your steepest descent, okay. There is an

analog of this in the gradient descent method also. Attributed to one of the gods of optimization called Nesterov, he invented this thing called the accelerated gradient method, which basically keeps track of the current iteration and the previous iteration to give you a faster update, okay. So, we will hopefully, I will give you a short brief on that, yeah.

$x_{k+1} - x_k = r_k (\alpha_{k+1} - \alpha_k) = \alpha_k H p_k.$

Final:

$$\alpha_k = \frac{\gamma_k^T \gamma_k}{p_k^T A p_k}, \quad \beta_k = \frac{\gamma_k^T \gamma_k}{\gamma_{k-1}^T \gamma_{k-1}}$$

To store $p_k = -\gamma_k + \beta_k p_{k-1}$


to get: $\gamma_k, p_{k-1}, \gamma_{k-1}.$

Does β_k change at every step? Have a look at the expression. β_k obviously depends on every iteration; it is not a fixed β , okay. Of course, you know that oops the way to go to the next remains the same and the expression for α_k is also here $r_k^T p_k$. So, notice that in order to progress in this conjugate gradient method, what are the computational things that I need to do? I need to evaluate the α , I need to evaluate the p_k ; these are the two things I need to evaluate. To evaluate α , what do I need to calculate? There is $A p_k$, which is a matrix-vector product. There is one matrix-vector product, then it is a vector-vector product. The numerator is one vector-vector product. How many vector-vector products do I have so far? Two vector-vector products and one matrix-vector product. Similarly, for calculating β , what do I need? I need... yes. So, this is one matrix-vector product, this is one matrix-vector product, right?

So, total two matrix-vector products and then followed by two vector-vector products. This is what I need to do, just keeping track of the cost involved, right. As I mentioned earlier, if you have a way of evaluating a matrix-vector product without storing the matrix A , that works very well for very large problems. So, without storing the matrix A , there are some routines which allow you to just evaluate the product of a matrix into a vector. So, we keep track of the number of these products, okay. Now, this looks like the most basic version of conjugate gradient.

It is actually possible to simplify these expressions a little bit more, okay, which is the more commonly found version of the conjugate gradient method. So, I will just outline some of it and then you will see the even simpler form of the conjugate gradient method, okay. So, $r_{k+1} - r_k$, what is the definition of r ? $r = Ax - b$, right. So, this becomes $Ax_{k+1} - x_k$, right, the b 's will cancel off.

Is $x_{k+1} - x_k$ anything nice? $\alpha_k p_k$, right. So, this is $\alpha_k A p_k$, okay, this is one thing that you have. So, I am just going to write down the final expression that you get from here. The final expression that you get for α and β , α_k turns out to be $\frac{r_k^T r_k}{p_{k-1}^T A p_{k-1}}$, okay. What was the original expression? It is here on the top.



A has eigenvalues: $\lambda \leq \lambda_2 \dots \leq \lambda_n$

Cond no $\kappa = \frac{\lambda_n}{\lambda_1}$

1) If I know the eigenvalues of A

$$\|x_{k+1} - x^*\| \leq \left[\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} + \lambda_1} \right] \|x_0 - x^*\|$$

OPTIMIZATION THEORY AND ALGORITHMS

So, what has changed? The numerator has changed from $r_k^T p_k$ to $r_k^T r_k$. Denominator, um, I think I made a mistake, this is, okay, this is my α_k . Similarly, I am going to write down the expression that you get for β_k . It is a particularly nice expression. So, do you see any advantage of this expression, these expressions, over the previous expressions from a computational point of view? How many matrix-vector products do I need? So, there is only one matrix-vector product, and how many vector-vector products? The numerator of α_k and β_k are the same.

So, that is 1. Then I have in the denominator one more, right. Is there anything else? How many vector-vector products do you see? 1, 2, I claim there are only 2 because I am going to store these values, right. So, $r_{k-1}^T r_{k-1}$ I will get from the previous iteration, right. So, actually this is just one vector-vector product if I am doing it cleverly, okay. So, what should I store? Remember $p_k = -r_k + \beta_k p_{k-1}$, okay.

So, to get this, what all do I need? To get p_k , I need r_k , I need p_{k-1} , and to get β_k , what else do I need? r_{k-1} , right. r_{k-1} . So, if you give me r_k , p_{k-1} , and r_{k-1} , I get my p_k completely. So, I know p_k , what is left for me to go to x_{k+1} ? I need a step length α , right, and the expression for α , do I already have it? Look at the expression for α . Once I get p_k , do I have α_k ? I have it already, right. So, the good news is that as I am going from iteration to iteration, I do not have to store the entire history. What do I need to store? Current iteration, previous iteration.

Everything, you can see the subscripts are either k and $k - 1$. I do not need to store starting from 0, 1, 2 up to all of that. So, what is that telling you? That if my size of the matrix is 1 million by

1 million, I do not need to keep a history of 1 million residuals and the p 's. All I need is current iteration, previous iteration. So, this is why it is also a very, very powerful method.

There is no elaborate history to be kept. So, it is computationally fast. It is also memory-wise very efficient. There is no unnecessary storage required over here. Any questions on the basic flavor over here? It seems almost too easy, right? But it's one of those very elegant, clever results in optimization that came about.

Now, unlike in the steepest descent method where we proved the rate of convergence, this time we are not going to prove the rate of convergence. I am just going to state it. Turns out that the steepest, sorry, the conjugate gradient method also has a linear rate of convergence, okay. So, yes, at most n steps. No, this is telling you at what rate it is happening, right, but remember that when I say it is a little bit misleading to say that both steepest descent and conjugate gradient method have linear rate of convergence, the coefficient of that rate is going to be different from method to method. So, one coefficient can be 1, another coefficient can be 100.

Right. So, you will still see the methods progressing at a different speed, and we will see that later in this lecture when we open up MATLAB, okay. But the rate of convergence in the order notation is still linear, okay. So, this is being stated without proof, okay. There are some more very interesting properties, yes question. Do we have two vector-vector calculations? Which are those? First is $r_k^T r_k$, second is $p_k^T A p_k$, and the second... oh right, yes, he is right, there are actually two vector-vector products. I do not have to calculate $r_{k-1}^T r_{k-1}$ because that was already stored. He is right, there are two vector-vector products.

NPTEL

1) If I know the eig values of A

$$\|x_{k+1} - x^*\| \leq \left[\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} + \lambda_1} \right] \|x_0 - x^*\|$$

$n=10$ $k=1$ $k=5$

$$\|x_2 - x^*\| \quad (\lambda_9 - \lambda_1) / \|x_6 - x^*\| \rightarrow (\lambda_5 - \lambda_1)$$

OPTIMIZATION THEORY AND ALGORITHMS

Now, apart from just saying that it has linear rate of convergence, there are some interesting, you know, rates over here which we will just have a look at the analysis of it, okay. So, let us assume A has, remember we said that it converges in at most n steps. So, that is very tempting, that at most n steps means, can it be faster than n ? So, there is a theorem which tells us when it can be

faster than n , which is again very nice linear algebra. So, let us say that I have eigenvalues of ordered in this form, okay.

When I write it like this obviously, these λ 's may or may not be distinct they can be repeated also. We know that eigenvalues can get repeated, but the eigenvectors will still be orthogonal. Keep that in mind. Now, for a positive definite matrix the condition number how can I write it?

$$\text{Condition number} = \frac{\lambda_{\max}}{\lambda_{\min}}$$

Now, if I know the eigenvalues of A , let us take two cases. Given some big fat matrix A (I mean not fat square matrix, but given a big matrix A) I need not know its eigenvalues. I mean if I know its eigenvalues you have already given me a lot of information about it, but let us say I do not know the eigenvalues that is one case and the other case which is more useful for analysis is let us say I give you the eigenvalues.

NPTEL

$\|x_2 - x^*\|$ ('9 - '1 / \|x_6 - x\| -> ('5 '1)

2) If I don't know eigenvalues of A
 $\rightarrow K$
 $\|x_{k+1} - x^*\| \leq 2 \left(\frac{\sqrt{k}-1}{\sqrt{k}+1} \right)^k \|x_0 - x^*\|$
 In SD $\cdot \sqrt{k} \rightarrow K \quad \checkmark$

So, let us just look at the rate of convergence in both cases, you will see something very surprising. So, what you have is that at the $k + 1$ -th iteration, the distance between where I am and the solution turns out to be.

This turns out to depend on the eigenvalue. Can we try to make sense of this? I mean when do you think this will make a difference? The hint is at most n steps. So, can you see if it is possible for us to make sense of this expression?

Let us say I have repeated eigenvalues. So, let us say the first 5 eigenvalues are distinct and then I have the remaining 5 are repeated and they are the same. So, what is this telling us? Is it telling us in what will happen to the rate of convergence?

Because λ_1 is still the same right? So, if your smallest eigenvalue is repeated, if your smallest eigenvalue is repeated then what is going to happen?

So, let us say the smallest eigenvalue is repeated 5 times. So, at $k = 1$, what will I see over here? On the left-hand side I am going to be looking at $x_2 - x^*$. Over here what will I see?

$$\lambda_9 - \lambda_1$$

$\lambda_9 - \lambda_1$ is not 0, the first 5 eigenvalues are repeated. So, this is fine, it is some number. Now, as the iterations go, supposing I reach where $k = 5$, at $k = 5$ what happens? I am looking at the distance $x_6 - x^*$.

What is the expression I have over here?

$$\lambda_{10} - \lambda_5$$

What is $\lambda_5 - \lambda_1$? It is 0, so boom in the step 5 at $k = 5$, $x_6 - x^* = 0$. So, if I have repeated eigenvalues, that is where this at most n steps result gets invoked that in fact you can have sooner convergence and we can also look at that in the case of MATLAB.

This is of course when I know the eigenvalues of A .

Any questions or doubts on this or is this analysis clear?

Now, if I do not know the eigenvalues of A , but obviously, you need to tell me something and let us say that the something that you are telling me is the condition number. So, I need the same expression that I had, but now in terms of condition number. So, what you get is:

Whether there are repeated eigenvalues or not, I really do not know it. So, does this look familiar from the case of steepest descent? In the case of steepest descent I had almost the same expression with the change that instead of $\sqrt{\kappa}$ I had κ right? So, this is why even though it is a linear rate of convergence, this quantity is going to die down faster.

This is why this is one important aspect of it. So, this is just information, I have not proven it in any way. But both of these things we will visualize when we open up MATLAB in a few minutes. Now, we have not actually proven the entire conjugate vectors to be, I mean the vectors to be conjugate with respect to each other. But we can do some, I am going to show you two interesting properties which are there for the conjugate gradient method which get used a lot.

The first one is this sort of confirms our intuition that every time you make progress in the k^{th} direction you never revisit this is kind of again quantification of that. That the error at the k^{th} step is perpendicular to the error in all previous steps. This is what this statement is saying right. If you replace math by English, r is representing residual, which is representing error. Error at the k^{th} step is perpendicular to error at all previous steps, right?

And this is a surprisingly easy to prove. If you take our expanding subspace theorem, what was the statement of this expanding subspace theorem? Can someone remind me?



$$1) \quad \gamma_k^T \gamma_i = 0 \quad \text{for } i = 0, 1, \dots, k-1$$

$$\text{EST} \rightarrow \quad \gamma_k^T p_i = 0 \quad \text{for } i \in [0, k-1]$$

$$p_i = -\gamma_i + \beta_i p_{i-1}$$

$$\gamma_k^T (\underbrace{-\gamma_i + \beta_i p_{i-1}}_{\leftarrow}) = \underbrace{\gamma_k^T (-\gamma_i)}_{\leftarrow} + \beta_i \underbrace{\gamma_k^T p_{i-1}}_{\leftarrow}$$

$$\Rightarrow \gamma_k^T \gamma_i = 0, \quad i \in [0, k-1]$$



$$r_k^T p_i = 0 \quad \text{for } i = 0 \text{ to } k-1$$

What is the definition of p_i ? By now we should know that p_i is equal to:

So, the way to remember it is p_i is if I were, if I did not know anything I would do gradient descent. If I did that p_i would be equal to $-\gamma_i$. You got the first time, right? Then I said hey we want something a little smarter than gradient descent, which is to remember something about what previous direction. So, there should be a p_{i-1} and there should be a β , right?

So, it is very intuitive to remember this expression. So, if I now left-multiply by r_k^T , what am I going to get?

$$r_k^T p_i = r_k^T (-\gamma_i + \beta_i p_{i-1})$$

This term is zero by the expanding subspace theorem. So, I get:

$$0 = \beta_i r_k^T p_{i-1}$$

Anything else? What about the second term on the right-hand side? Also zero by the same theorem, right? So, it implies that:

$$r_k^T r_i = 0, \quad i = 0 \text{ to } k-1$$

So, there are lots of these very small results you will find in the case of the conjugate gradient method. The ingredients are very few, it is like a khichadi with very few ingredients. What are the ingredients? There is an expression for β . How did we derive that expression for β ? Here, just by taking the initial definition of p_k and demanding conjugacy, I want conjugacy. This gives me an expression for β , I have one expression for β .

I have one expression for α , right? And that is basically it. Everything else just follows from here, some amount of algebra, and I land up with this interesting expression. There are several other such expressions that are used. For example, when I showed you the simplified version here, these simplified versions, they also come just by using these simple algebraic identities.

So, now what we will do is we will have a look at a code implementation of the conjugate gradient method and we will sort of have a trace against steepest descent and let us see what happens. So, before we go to that, is there any difficulty here so far in the expression?