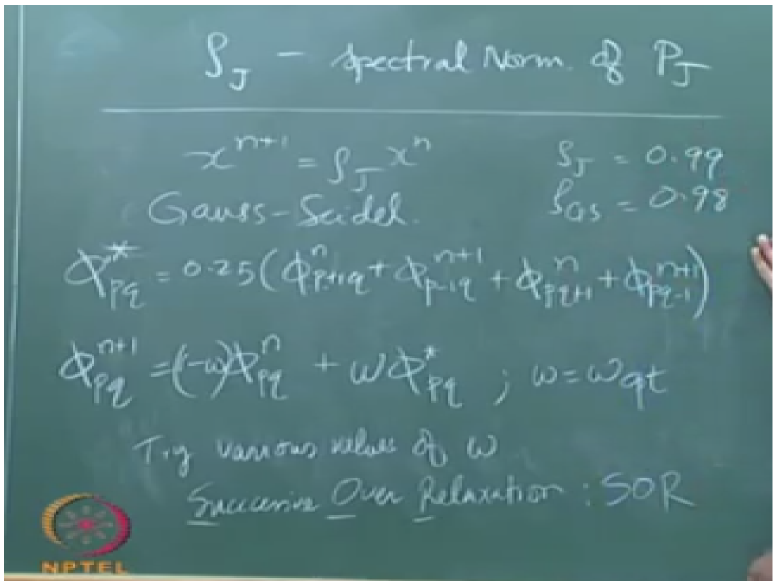
Introduction to Computational Fluid Dynamics
Prof. M. Ramakrishna
Department of Aerospace Engineering
Indian Institute of Technology – Madras

Lecture - 12
Laplace Equation - Convergence Rate (Contd.)

Now we want to explore see, we have seen, we can see that as the number of grid points increases, right, that the number of iterations increases, okay, so for given CPU time for grid point per iteration the number of iteration increases in a very rapid fashion because the spectral norm is so large getting closer and closer to 1, right, it is going to take for ever to essentially can remember we got a geometric series, right.

(Refer Slide Time: 00:43)



ρ_J - Spectral Norm of P_J

$x^{n+1} = \rho_J x^n$ $\rho_J = 0.99$

Gauss-Seidel $\rho_{GS} = 0.98$

$$\phi^*_{pq} = 0.25(\phi^n_{p+1q} + \phi^{n+1}_{p-1q} + \phi^n_{pq+1} + \phi^{n+1}_{pq-1})$$

$$\phi^{n+1}_{pq} = (1-\omega)\phi^n_{pq} + \omega\phi^*_{pq} \; ; \; \omega = \omega_{opt}$$

Try various values of ω

Successive Over Relaxation : SOR

So if you say that xn+1 is rho J times xn or en +1 is rho J times en, right, and you want e to go to 0, the fixed point is 0. If this is very close to 1, if it is 1 of course it is not going to get there, if this is very close to 1, it is going to take forever for the error to go to 0, is that fine, okay, so that is the problem, right.

So we have to think of ways by which are there ways by which we can make the code or the iterations go faster so given that the cost the CPU time for grid point per iteration is the same you cannot change that, you may not mind doing a little more work that is increasing that CPU time per grid point per iteration if we are able to converge to the solution faster okay, now this is only Jacobi.

I am not really going to sit down and work through this for Gauss-Seidel, but we did Gauss-Seidel and it does turn out that Gauss-Seidel is marginally faster than Jacobi, right, but the problem is that if rho Jacobi is 0.99 and rho Gauss-Seidel is 0.98, right, okay, you in fact this is twice as fast as that, right, it is ironical, 0.99 * 0.99 gives me 0.98 essentially right so this is in fact twice as fast as that but in an absolute sense it is no use to right it is only 0.98.

I want something that is much smaller, right, I want something that is much smaller, so we have to come up with way by which we can do this, so what are possible ways by which we can make the program run faster, give me suggestions, or iterations converge faster? What are the things that we have done we have an initial guess, so you could get an improved initial guess, right, clearly see if you manage to guess the solution you get the answer in one iteration.

So clearly an improved initial guess is something that is worthwhile. Higher gird point, so one possibility is that you somehow use, you use the initial, make an initial guess which is a poor guess 0 everywhere on a core script, what we will call it a core script, right and use that as an initial guess for, so you take the 10/10, you take the solution on the 10/10 there is an initial guess for the 20/20 right, in fact you can imagine that you use 10/10 that is an initial guess for 20/20.

Then use 20/20 as an initial guess for 40/40, right, so, you could have a hierarchy of grids we will look at this later in the semester, there is a scheme called multi grid method which is sort of exploits this, but as far as your concern a good initial guess would get us to the answer faster okay, so there seems a possibility there but as I said we will do that a little later, what else? anything else that we can do? Okay.

So one thing that we will try see all we have if you think about it for Gauss-Seidel all we have is phi n+1 pq is 0.25 phi n p+1q you have seen me write this so many times now, phi n p-1q n+1 + phi pq+1 n + phi pq-1 n+1 right, so this is Gauss-Seidel, now this is all we have, we have the latest phi and we have the prior phi and improved we are using know what more can we do, so one of the things that we can try, right.

So this sort of like, just try it out, because this is all we have to play with, is instead of calling this phi and +1, I will call this phi star and I will write phi n +1 at pq in fact is phi n pq + phi star pq and I want to take a liner combination or what is called a convex combination of these 2 in order to form that right, so, I will pick a parameter omega and when omega = 1, I want to get back Gauss-Seidel.

So omega = 1 means that this should be omega, that should be 1 – omega, omega = 1 will get me back the original equation, okay and omega = 0 there is no progress okay, so now we have done a liner combination well what is the big deal, what have we gained by this process, it turns out that there is an optimal value omega for which the convergence is greatly improved.

Okay there is an omega value for which it is greatly improved, there is an omega = omega qt for which the convergence of this iteration scheme, right, is faster than the original Gauss-Seidel is that fine, remember that this was called successive relaxation Gauss-Seidel by itself was called successive relaxation. You can try various values of omega for Laplace equation it turns out that omega believe it or not is >1, optimal value of omega is >1.

Okay and for that reason it is called successive over relaxation or SOR, so instead of successive relaxation which is what Gauss-Seidel was, right, because omega is >1 it is like over relaxation, right, successive over relaxation, is that fine. So we have introduced this omega right now you have to take my word for it that it actually improves, it actually improves the convergence rate.

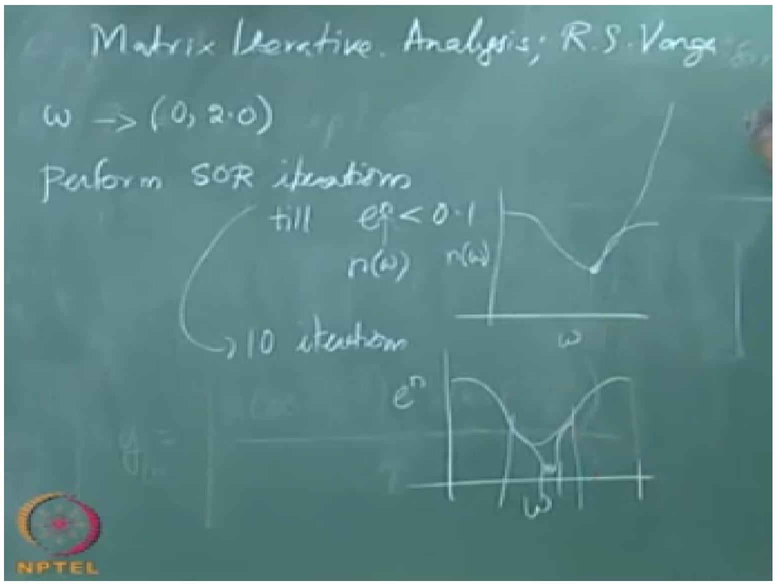
As I said we are not going to go through this whole analysis for it to show that it actually improves convergence rate, we will do it in a more empirical fashion, you can actually write a program and try it out and the question would be, right, if you do not have an expression for omega for the optimal omega as it turns out for Laplace equation, on a unit square with the uniform grid you can actually come up with an expression for an optimal omega.

Okay, right, there are enough books out there that will look at anything on matrix iterative analysis by RS Varga come to mind, there are lots of books out there that will, right, give you expression that they have already derived expressions for an optimal omega fine, but these things work for a unit square uniform distribution, what do we do when we are going to actually solve problems from fluid flow equations or something that is quite complex.

There is no expression, so what you have to do is you have to find the optimal omega by experimentation, you have to hunt for it, is that fine, the advantages once you have found it you are able to do production runs so to speak you can make multiple runs with that optimal omega, right, so how do we find this optimal omega, how do we do this. There are 2 possible ways, 2 possible iterations, one of them may be better than the other we will see that.

There are 2 possible ways that you can do it, so make values of omega and right now without an explanation I will say pick values of omega between 0 and 2.0. We will look at where this 0 and 2.0 comes from.

(Refer Slide Time: 10:20)



Pick values of omega between 0 and 2 right say at equal intervals 0.1, 0.2, 0.3, 0.4 pick values of omega between 0 and 2 and then you can do SOR, perform SOR iterations now comes the critical thing, perform SOR iterations till there are 2 possibilities perform SOR iterations till en < say 0.1 that is 1 possibly some predetermined value, okay and then you can ask the question so this n will be a function of omega that is what we are saying that n will be a function of omega.

You will say allow it to drop by 0.1 and this n will be a function of omega and what we hope is that if I get omega versus n of omega versus omega we hope that we get something like this and we know where there is a minimum, okay, there is a danger here though, I am always vary of the setting value of 0.1, what is the danger, ya what if there is a value of omega for which it does not converge.

You keep on running when it is going and it is going you will never get to 0.1, it never drops away 0.1, see there is always that danger, yes I know for Laplace equation we just showed that the iterations are going to converge right but for SOR we have not really done that, so there is always that danger, if you are solving the Navier-Stokes equations or whatever there is always the danger that you try to get that 0.1 drop and it just does not happen.

That the program runs for ever, right, so in all of this kind of exploratory work, I am citing this example, I am citing this because this is the very important point that I want to make for all exploratory work you always bound the amount of computation that you are willing to perform, you always limit the amount of computation you are willing to perform right, so you do not say till en is < 0.1, that is not the way to do it, right.

What you do is you say we do say 10 iterations or 100 iterations then you have limited the number of iteration, basically going to say I am willing to do this much and no more in which case you can then plot what would you get, what would you plot, the number of iterations is fixed right, so this would be en versus omega. Right and now again we hope that you get something of that sort.

We do not know, we hope that you get something of that sort we do not know, is that fine, okay, so we can run multiple so you can take 10 values for example what I would do is I would take 10 values I would then turn around and say that let me explore for the optimal omega having computed this graph let me explore for the optimal omega in this gap, right and it may turn out that I actually get something like this.

I have deliberately shown it sort of shifting to one side, why I have shown shifting towards one side, we will see whether this actually happens, this is one of those situations that they talk about map classes, but you have not really encountered that often, this is the situation where you have non-uniform convergence, in fact we want non-uniform convergence, we want for different omegas the convergence rate to be different.

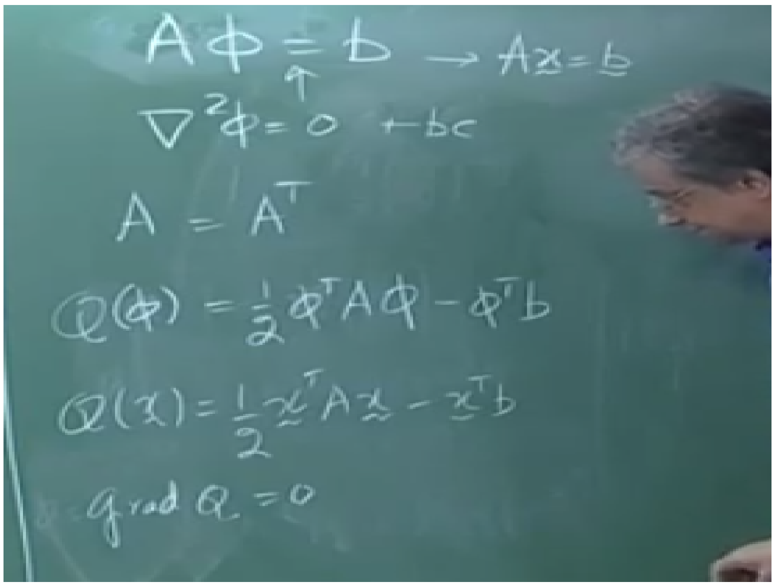
Right, we are looking for a situation where the convergence is non-uniform, specifically we are going out and looking for it. Am I making sense, we have picked a parameter and we are hoping that I really, really, really hope that the convergence rate is non-uniform, if it is uniform then we have not gained anything, right, okay, so we will hunt and we will try to narrow down.

Now it is up to you how much effort you are willing to put in to narrow down that value, am I making sense and if the drop is sufficiently fast you should be able to get an order of magnitude speed up or more, fine, so that we will sort of look at a demo, right, and see what happens, we will try to figure out what happens. Now we have to address one issue, so we have SOR, we have some mechanism in algorithm by which so I do not want you to do this.

Well I want you to do this just for fun, right, if I say I do not want you to do this then you should go and try and do it just to see what is Ramakrishna talking about why should not I do, just try to do it just for fun but once the class, the learning process is over you do not usually do this, what you do is you do this, you bound, you limit the amount of computation that you do in exploratory, right.

You make sure that it cannot become unbounded okay, so right now I will remove that, so we have an algorithm, you can find the optimal omega, why is the hunt limited in this case, in this particular case why is it limited to 0 to 2.0, what is the reason? So let us find out okay. Remember the matrix A, what is this matrix A? this was A phi = b, b had the boundary conditions, this was the discrete version of Laplace equation using central differences.

(Refer Slide Time: 16:30)



$$A\phi \underset{\uparrow}{=} b \rightarrow A\underset{\sim}{x} = \underset{\sim}{b}$$

$$\nabla^2\phi = 0 \; + bc$$

$$A = A^T$$

$$Q(\phi) = \frac{1}{2}\phi^T A \phi - \phi^T b$$

$$Q(\underset{\sim}{x}) = \frac{1}{2}\underset{\sim}{x}^T A \underset{\sim}{x} - \underset{\sim}{x}^T b$$

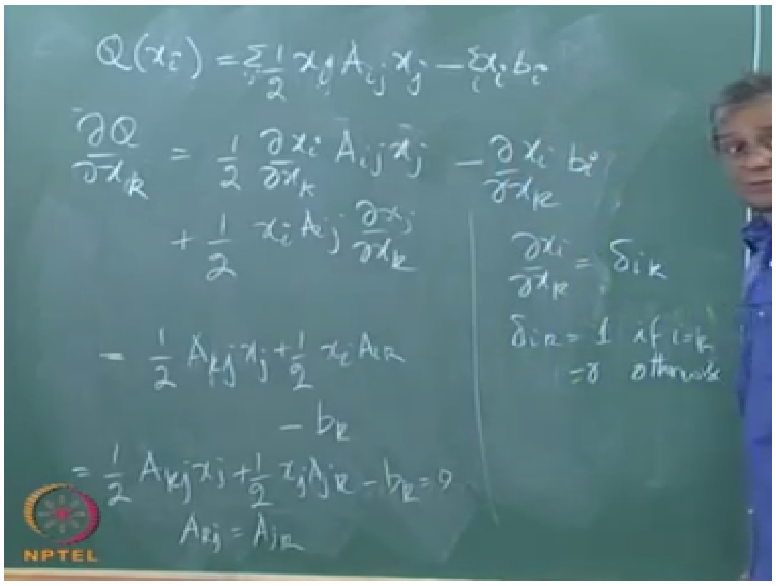
$$\text{grad } Q = 0$$

Lambda square phi = 0 + boundary conditions turned out to be, right, using central differences the matrix of that form and what was the structure of the matrix A, A was symmetric. Remember that A was symmetric, meaning A = A transpose, okay, now consider a function that I just casually introduced well I call it phi, Q of phi is 1/2 phi transpose A phi – phi transpose b.

You write this in the standard form because otherwise I will keep making this mistake of replacing writing x instead of phi, if you write this in the standard form, Ax = b where x and b are vectors, so Q of x is 1/2, put a negative sign in front of it if you want but it does not matter, x transpose Ax – x transpose b, this x has a little tilde in front of it to indicate it is not our coordinate.

So what is Q, given an x or given a phi it gives me a number okay, what does Q do, given an x or given a phi depending on which equation you are looking at it gives me a number is that okay, fine it is mapping this vector into a number into the real line okay, it is quadratic. How do I find it is minimum, how do you find the minimum of a function? Take the gradient and set it = 0, so what we want is grad Q and set it = 0, okay.

(Refer Slide Time: 19:43)



$$Q(x_i) = \sum_{ij} \frac{1}{2} x_i A_{ij} x_j - \sum_i x_i b_i$$

$$\frac{\partial Q}{\partial x_k} = \frac{1}{2}\frac{\partial x_i}{\partial x_k} A_{ij} x_j - \frac{\partial x_i}{\partial x_k} b_i + \frac{1}{2} x_i A_{ij} \frac{\partial x_j}{\partial x_k}$$

$$\frac{\partial x_i}{\partial x_k} = \delta_{ik} \qquad \delta_{ik} = 1 \text{ if } i=k, \; = 0 \text{ otherwise}$$

$$= \frac{1}{2} A_{kj} x_j + \frac{1}{2} x_i A_{ik} - b_k$$

$$= \frac{1}{2} A_{kj} x_j + \frac{1}{2} x_j A_{jk} - b_k = 0$$

$$A_{kj} = A_{jk}$$

You want to take grad Q and set it =0 I think maybe this is better if we use index notation to do the gradient so that you do not get confused okay Q on xi, i is just a subscript is 1/2 xi Aij xj – xi bi this has the summation over i and j and this has a summation over i, so the repeated index indicates that there is a summation, if the subscript is repeated that indicates the summation.

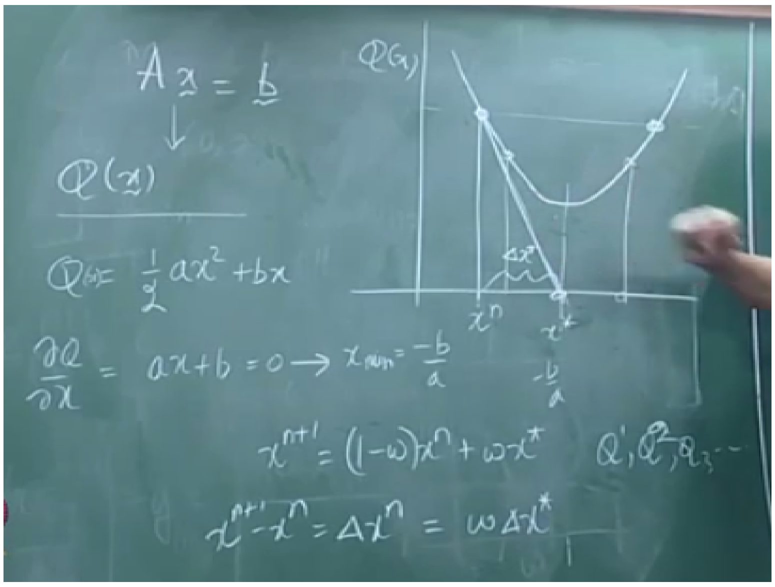
So I am not going to, so what is the gradient, dou Q/dou xk = 1/2 dou xi/dou xk Aij xj + 1/2 xi Aij dou xj/dou xk –dou xi/dou xk bi. I hope all of you are familiar with your index notation. Dou xi/dou xk would be the matrix representation would be unit vector, it would be identity matrix, but in index notation it will be delta ik. Delta ik = 1 if i = k = 0 otherwise, okay, right, so dou x1/dou x1 is 1, dou x1/dou x2 is 0, okay dou x1/dou x3 is 0, right, dou x2/dou x1 is 0, dou x2/dou x2 is 0.

The derivative of the thing with itself is 1, derivative of the thing with something else is 0, okay, that is basically what it says, by itself is 1 with something else is 0, okay and remember that repeated index means summation, so what does that mean, you can check this out, I will let you verify this for yourself, so this gives me 1/2 Akj xj + 1/2 xi Aik – bk. Xi Aik is the same as xj I mean if you are going to sum over the i.

Whether you sum over i or sum over j it does not make a difference so this in fact turns out to be 1/2 Akj xj + 1/2 xj Ajk – bk and you want to set this = 0, but because A is symmetric Akj is the same as Ajk because A is symmetric that is why that symmetry was very important, is that fine. So what is that result in then, those 2 halves add up and you get Ax = b writing it back in vector notation, you get Ax = b.

So solving Ax = b is equivalent to minimizing Q of x, right when A is symmetric solving Ax = b is equivalent to minimizing Q of x is that fine, everyone.

(Refer Slide Time: 24:27)



$A\underset{\sim}{x} = \underset{\sim}{b}$
$Q(\underset{\sim}{x})$
$Q(x) = \frac{1}{2}ax^2 + bx$
$\frac{\partial Q}{\partial x} = ax + b = 0 \rightarrow x_{min} = -\frac{b}{a}$
$Q(x)$
Δx^*
x^n
x^*
$-\frac{b}{a}$
$x^{n+1} = (1-\omega)x^n + \omega x^*$
$Q^1, Q^2, Q_3 \cdots$
$x^{n+1} - x^n = \Delta x^n = \omega \Delta x^*$

So we look at the problem in one dimension, it is easier to understand, we can draw figures, right so we look at the problem in one dimension, so one dimensional equivalent of this, so the tilde will go we will just use x because it is a scalar, so Qx in fact will be 1/2 ax squared + bx fine, this is Q as a function of x and what does dou Q dou x or dQ dx in this case because it is only dependent on one variable ax + b = 0.

Telling us that the minimum is that corresponds to –b/a from your to understanding a quadratic equation you already know that to be a fact. Let us graph this, now in general ax square + bx is not symmetric about the figure is not symmetric about this, but you go back and look at your quadratic equation and you will see that what I am saying is valid in general, right that is if I were to draw quadratic the minimum of here that is –b/a right.

Of course if I have to translate the coordinate system by some c, I would get 2 roots, about that minimum and independent of the orientation of the quadratic those 2 roots will be symmetric about that minimum, is that fine. I want you to go back and check to make sure that what I am saying that you agree with, so if this is your xn because this is a scalar equation life is very easy after all that is what you are doing, you are taking a derivative and setting it = 0 and you will get this point immediately that is what you are doing.

You are solving for that point, this is xn+1, what I am saying is I call it x star okay, and I am going to say that xn+1 is 1-omega times xn + omega times x star that is what I am doing, is that fine and if I subtract out xn from here from this equation I get xn+1 –xn which is delta xn which is the correction that I need to add to my xn to get the new iteration, yes in fact omega times delta x star, okay.

So you just look at this, this is delta x star when omega is 1 this is the value that you get, when omega is 0 delta x star is 0 and what is the value of Q that you get, you get the original value, when omega is 2, you swing over to the other side symmetrically over to the other side and the Q value is the same. For any omega between the value is 0 and 2, you will get Q values to choose omega depending on what omega value that you choose.

We will get a value in between of x and the Q will decrease, so these iterations for omega values between 0 and 2 will generate a sequence of Qs, Q1, Q2, Q3 so on will generate sequence of Qs which are decreasing and they are bounded from below. So you know that there is a limit and you are going to reach that limit, does that make sense.

As long as you pick omega values between 0 and 2, if you pick omega 2.1 it is going to take you there, the Q is increasing, if you pick omega < 0 you are going to go here which is also increasing, if you pick omega 0 they are not going to shift from this point and if you pick omega 2 you are going to oscillate from one point to the other, fine, okay, right so I would suggest that you try this out and see how it goes right, has any questions, fine, thank you.