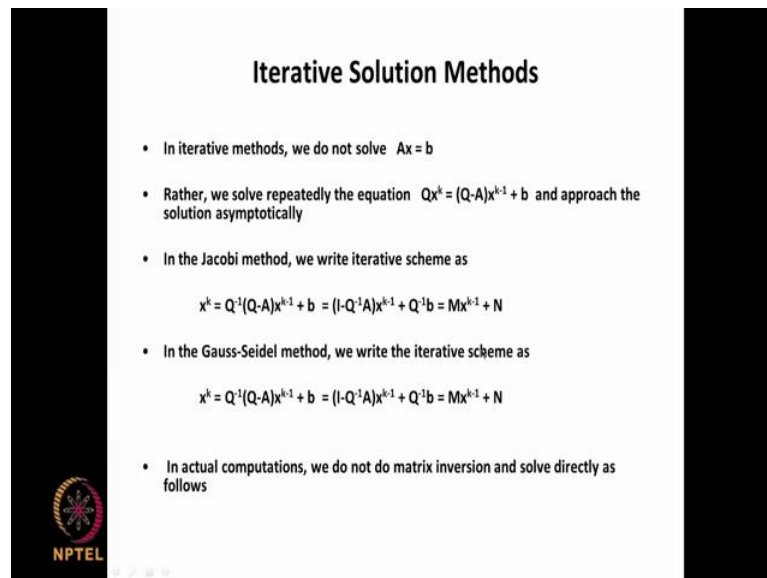


Computational Fluid Dynamics
Prof. Sreenivas Jayanthi
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 48
Recap of Basic Iterative Method


We have been looking at Basic Iterative methods. In iterative methods we do not solve $Ax = b$ directly rather we solve repeatedly the equation $Qx^k = (Q-A)x^{k-1} + b$ and approach the solution asymptotically.

(Refer Slide Time: 00:16)



Iterative Solution Methods

- In iterative methods, we do not solve $Ax = b$
- Rather, we solve repeatedly the equation $Qx^k = (Q-A)x^{k-1} + b$ and approach the solution asymptotically
- In the Jacobi method, we write iterative scheme as
$$x^k = Q^{-1}(Q-A)x^{k-1} + Q^{-1}b = Mx^{k-1} + N$$
- In the Gauss-Seidel method, we write the iterative scheme as
$$x^k = Q^{-1}(Q-A)x^{k-1} + Q^{-1}b = Mx^{k-1} + N$$
- In actual computations, we do not do matrix inversion and solve directly as follows


NPTEL

This is what we have seen, and in one of the basic iterative methods the Jacobi method we write iterative scheme as $x^k = Q^{-1}(Q-A)x^{k-1} + b$ which can also be put in this form finally, in the form of $Mx^{k-1} + N$ and the Gauss-Seidel method we write the iterative scheme as $x^k = Q^{-1}(Q-A)x^{k-1} + b$ equal to in this form; which looks exactly the same except the definition of Q . In the case of Jacobi method Q is the diagonal elements of A and in the case of Gauss-Seidel method it is the diagonal elements and the elements which are lower than the diagonal thus constitute the Q .

So, in both these methods it looks like we have to do inverse of Q and then we have to multiply by Q, it looks like complicated matrix, procedures are there. But it is not really necessary to do it that way, we do not do matrix inversion and actual computations, but solve directly in the following way.


(Refer Slide Time: 01:52)

Basic Iterative Schemes: Jacobi Method

- Consider the set of equations given by

$$\begin{aligned} a_{11}u_1 + a_{12}u_2 + \dots &= c_1 \\ a_{21}u_1 + a_{22}u_2 + \dots &= c_2 \\ &\vdots \\ a_{n1}u_1 + a_{n2}u_2 + \dots &= c_n \end{aligned} \tag{1}$$
- In Jacobi method, we rewrite these as

$$\begin{aligned} a_{11}x_1^{k+1} &= -a_{12}x_2^k - a_{13}x_3^k \dots - a_{1n}x_n^k + b_1 \\ a_{22}x_2^{k+1} &= -a_{21}x_1^k - a_{23}x_3^k \dots - a_{2n}x_n^k + b_2 \\ &\vdots \\ a_{n-1,n}x_{n-1}^{k+1} &= -a_{n-1,1}x_1^k - a_{n-1,2}x_2^k - a_{n-1,3}x_3^k \dots - a_{n-1,n}x_n^k + b_{n-1} \\ a_{nn}x_n^{k+1} &= -a_{n1}x_1^k - a_{n2}x_2^k - a_{n3}x_3^k \dots - a_{n,n}x_n^k + b_n \end{aligned} \tag{2}$$
- Start with initial guess $\{x_1, x_2, \dots, x_n\}^0$, solve the n equations of eqn (2) for $\{x_1, x_2, x_3, \dots, x_n\}^1$
- From $\{x\}^1$, now solve n equations of (2) again for $\{x\}^2$, and from these for $\{x\}^3$ and so on.
- Note that it would take $(n-1)$ multiplications and one division to solve one equation and there are n such equations to be solved to advance from step k to step $(k+1)$.
- But if $[A]$ is sparse and has only 5 (or 7) non-zero coefficients, then it takes only $5n$ (or $7n$) multiplications to advance from k to $k+1$!



So, let us consider the set of equations given by $a_{11}u_1 + a_{12}u_2 + \dots + a_{1n}u_n = c_1$ and so on here, this u should have been x . So, similarly the second equation is given by a new set of coefficients $a_{21}u_1 + a_{22}u_2 + \dots + a_{2n}u_n = c_2$. So, the same variables, but new coefficients similarly for the n th equation new coefficients were the same variables, this is the system of linear algebraic equations.

In the Jacobi method we make use of the first equation and put all these other elements to the right hand side. So, we can put this as $a_{11}x_1^{k+1} = -a_{12}x_2^k - a_{13}x_3^k - \dots - a_{1n}x_n^k + b_1$. So, essentially we have this b is the same as the c here. So, we have rewritten the original equation that we have here with the same b as what we have for the given equation. All the other elements except the first element in the first equation is shifted to the right hand side and. So, from this we can get $x_1^{k+1} = (-a_{12}x_2^k - a_{13}x_3^k - \dots - a_{1n}x_n^k + b_1) / a_{11}$ that

means, all the right hand side is computed and you divide by a_{11} to get x_1^{k+1} . So, there is no inversion as such in this.

Similarly, the second equation here is written in such a way that only this term is kept on the left side all the other terms are taken to the right hand side. So, we have $b_2 - a_{21}x_1^k$ and minus $a_{23}x_3^k - a_{2n}x_n^k - a_{21}x_1^k$ like this. If you consider the last equation here the last equation this term here, this is the n minus x_n minus 1 is the diagonal element. So, this is kept here and all the other terms are taken to the right hand side along with the right hand coefficient here. So, in each case by rewriting it like this we can evaluate x_1^{k+1} by doing all these multiplications with the previous values of x and then subtracting from b_1 here and dividing the result by a_{11} to get x_1 . Similar for x_2 similarly for x_1 , you solve each of these equations for $x_1, x_2, x_3, \dots, x_n$.

So, we start with initial guess x_2, x_3, \dots, x_n , the zero indicating the initial guess and solve n equations of equation 2 for $x_1, x_2, x_3, \dots, x_n$ for the first step. Now you have x_1 for all of these and then again substitute them here and then again solve equation 2 for $x_1, x_2, x_3, \dots, x_n$ for at the end of the second iteration step and once you get x_2 you can again get all of x_3 and so on.

So, in each case you are not exactly solving this equation you are using the first equation to get a new estimate for x_1 knowing the values of all the other variables from the previous iterations. Similarly the second equation is used to get the value of updated value of x_2 making use of all the previous values of this and then we can go on. So, what we see from here is that it would take $n - 1$ multiplication - for example; you have one multiplication, second multiplication, third multiplication up to $n - 1$ multiplication this and this, there are $n - 1$ because this will make it n th. Since it is on this side we have $n - 1$ multiplications to give us this whole thing and then $n - 1$ additions plus this, n additions and the whole result of this $n - 1$ multiplications and n additions is divided by a_{11} to give us x_1^{k+1} .

That means, that you have you need $n - 1$ multiplications and 1 division to solve one equation and in each iterative step you have to solve n such equations because you

have to get an updated value for x_1 and then x_2 updated value for x_2 and all the way to x_n that means, that if you need to go from step k to step $k + 1$ we need to solve we need to use a total of n multiplications or division for one equation and you have n such equations, that means, there are n square number of multiplications or divisions that are needed to go from step k to step $k + 1$, this is in the general case.

And n square does not seem to be such a good thing, but if A is sparse which is what we have in CFD generated compact grid discretizations and has only 5 nonzero coefficients for a 2-D problem or 7 nonzero coefficients in a 3-D problem for example, in central differencing then, what we have is that in each of these equations only there are 7 non-zero coefficients that means, that one of this is kept on this side the other 6 are coming out of this. That means, that we have to multiply only 6 times not n times or $n - 1$ times.

Similarly this is again 6 times. So, each equation is going to have 6 multiplications and 1 division 7. So, that is only 7 n number of multiplications are needed to advance from k to $k + 1$. So, in that sense the Jacobi method takes advantage of the sparsity of matrix A instead of doing it for all of them it does only for those equations which do not have non-zero coefficients and by doing that it goes from n square number of multiplications or divisions to 7 n or 5 n number of multiplications or divisions to go from step k to step $k + 1$.

Now what does that mean if you have 10,000 equations, 10,000 grid points? Then if you do for every one of these in the general case it would take 10,000 square, that is 10 to the power 8 of multiplications to go from k to $k + 1$. But if the same set of equations is such that A is coming from the usually discretized CFD type of equation then we have to do only 70,000 arithmetic operations not 10 to power 8 which is 100 million.

So, that is the kind of advantage that we get when we are dealing with sparse matrix and a sparse matrix advantage is achieved is exploited by Jacobi method. So, in that sense it makes it much better than in the general case. So, in the general case we are not gaining much of an advantage whereas, in this case we are gaining a significant advantage because this n square looks attractive compare to the n^3 for a Gaussian

elimination. But we do not get a solution in one step of k to $k + 1$ you have to do many of those.

So, how many of those depend will tell us the total number of computation and we will see that shortly. But the way that we actually implement Jacobi is such that we take advantage of the non-zero coefficients and the sparsity of matrix a and we go from step k to step $k + 1$ in $7n$ or $5n$ number of or some constant times n where the constant is the order of 10. Number of multiplications and divisions to go from step k to step $k + 1$ and when you look at what is actually required to do these things it is pretty straight forward there is nothing difficult at all in going from step k to step $k + 1$ it is very simple to program.

Now what happens in Gauss-Seidel method for the same set of equations in the Gauss-Seidel method? We have Q becoming $d - e$ and all that kind of thing.


(Refer Slide Time: 11:05)

Basic Iterative Schemes: Gauss-Seidel Method

- For the same set of equations, in the Gauss-Seidel method, we solve as follows:

$$\begin{aligned}
 a_{11}x_1^{k+1} &= -a_{12}x_2^k - a_{13}x_3^k \dots \dots - a_{1n}x_n^k - a_{11}x_1^k + b_1 \\
 a_{22}x_2^{k+1} &= -a_{21}x_1^{k+1} - a_{23}x_3^k \dots \dots - a_{2n}x_n^k - a_{22}x_2^k + b_2 \\
 &\dots \\
 a_{n-1,n-1}x_{n-1}^{k+1} &= -a_{n-1,1}x_1^{k+1} - a_{n-1,2}x_2^{k+1} - a_{n-1,3}x_3^{k+1} \dots \dots - a_{n-1,n}x_n^k + b_{n-1} \\
 a_{nn}x_n^{k+1} &= -a_{n1}x_1^{k+1} - a_{n2}x_2^{k+1} - a_{n3}x_3^{k+1} \dots \dots - a_{n,n-1}x_{n-1}^{k+1} + b_n
 \end{aligned} \tag{3}$$

- Start with initial guesses $\{x\}^0$, solve the n equations of eqn (3) for $\{x\}^1$, from these for $\{x\}^2$, and from these for $\{x\}^3$, and so on.
- Note that if $[A]$ is sparse and has only 5 (or 7) non-zero coefficients, then it takes only $5n$ (or $7n$) multiplications to advance from k to $k+1$.
- Thus, the number of multiplications (or divisions) to go from step k to step $(k+1)$ is the same as that for the Jacobi method.
- The actual number of arithmetic operations depends on how many times we have solve to eqn (2) (or (3) for GS) to get a "converged" solution, i.e., one in which $\{x\}^{k+1}$ is almost equal to $\{x\}^k$.



We can now forget all that and we can rewrite it in this simple form which results in the same form as what we have earlier. So, that is the first equation remains the same as in Jacobi method $a_{11}x_1^{k+1} = b_1 - a_{12}x_2^k - a_{13}x_3^k$ and so on up to $a_{11}x_1^{k+1} = b_1 - a_{11}x_1^k - a_{12}x_2^k - a_{13}x_3^k$. The second equation is similar to the

previous one except that by the time we have come to solving the second equation we have already solved the first. So, we know the k plus 1th iterative value the updated value of x_1 . So, we make use of the updated value here.

We have not solved the third equation. So, we still have only the old value and then all these things are old values and then you have b_2 . By the time you come to the n minus 1th variable x_{n-1} , we have solved for x_1, x_2, x_3 all the way up to x_{n-2} . For all those things we make use of the updated values and only for the yet to be updated value x_n and k is put as old value. So, in that sense the Jacobi method and Gauss-Seidel method are very similar except in the case of Gauss-Seidel method wherever we have an updated value we substitute that into this and otherwise it is the same approach no matrix inversion and we solve again with an initial guess and then from this we solve for x_1 and then x_2, x_3 and so on.

Again, if A is sparse then we do not have to do it for all of these and we have only 5 or 7 n number of non-zero coefficients then it takes only 5 n or 7 n multiplications that means, that we are really taking in the advantage of sparsity here. The Gauss-Seidel method will consume exactly the same number of arithmetic operations as the Jacobi method to advance from step k to step $k+1$. But there is one key difference which sometimes makes the Jacobi method better than the Gauss-Seidel method. One would immediately see the contrast between the two is that we are making use of updated value as soon as it becomes available in the Gauss-Seidel method. So, one would intuitively feel that Gauss-Seidel method should be better than the Jacobi method, which is true in some cases in many cases where we apply this, but there is one advantage of the Jacobi method as opposed to Gauss-Seidel method.

Even though it is slower and the thing is that if you have for example, a million equations and you want to solve these things. Then we can solve each equation independently whereas, if you are doing it using the Gauss-Seidel method if you want to solve this equation you have to wait for this equation to be solved. Similarly if you want to solve this equation you have to wait for all these subsequent equations to be solved. So, that you can the updated value here so that means, that if you are making use of large number of parallel computers and then out of a million equations or 100 million

equations you give to this computer the first million equations and for this you get you give the second million equation, third million equation and so on.

Then this computer cannot operate cannot start the process until this computer has finished all its computations and given to you. But if it is a Jacobi method you can subcontract part of the work to this set of computer and you can subcontract another thing and then you can get back the solution from these sub contract computers and then put together and move on to the next one.

So, that kind of parallel processing will be more readily implementable in the case of Jacobi method because each set of equations can be solved independently from the previous known values. Whereas, in this case you cannot solve this equation until all these things are solved and you need to have these things solved and the information should be supplied to you.

So, there is the information evaluated by solving the equation and then stored and retrieved, sent back stored and retrieved all those kind of operations will become necessary, in the more of them will be necessary in the case of Gauss-Seidel method. So, that is the disadvantage that is there. But precisely because we are making use of updated information we would expect the Gauss-Seidel method to work, but in some special cases that may be a disadvantage.

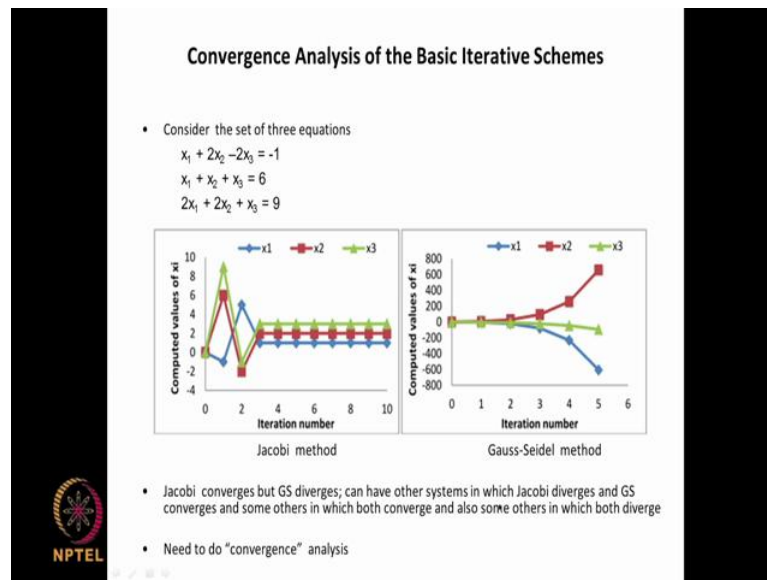
So, let us come back to this thing here. The number of multiplications or divisions to go from step k to $k + 1$ is the same in both Jacobi method and Gauss-Seidel method. The actual number of arithmetic operations depends on how many times we have to solve equation 2 or 3 for the Jacobi method and the Gauss-Seidel method to get a converged solution.

So, this is as pointed out earlier, we need $5n$ or $7n$ number of arithmetic operations multiplications or divisions to go from step k to $k + 1$ that is only one step. How many steps are needed to get a converged solution? Converged solution here we are saying is a solution which x_{k+1} that is updated one is almost equal to x_k that we have updated. If the amount of the updating between the previous and the current one is very

little is very, very small then we can say it is almost converged solution because any further updating is not making bunch of change.

So, if you want to go up to that how many steps you have to take to get down to that position that depends on that tells us because in each step you are using $7n$ number of computations. So, that will tell us the overall amount, and how to get that is something that we will have to see by looking at what is the convergence rate, what are the convergence characters.

(Refer Slide Time: 17:57)



So, here we have a very simple problem a 3 by 3 problem and $x_1 + 2x_2 - 2x_3$ is equal to minus 1 and $x_1 + x_2 + x_3$ is equal to 6 $2x_1 + 2x_2 + x_3$ is equal to 9. We would like to solve this equation using Jacobi method and Gauss-Seidel method and we would like to solve this with all the values to be 0 here. So, what I have plotted is iteration number and the 3 values what are the values that we are predicting and how we are updating here.

Since, we have 3 we can just show a graphically like this and the computed value of x_i is plotted on the y axis and the iteration number is plotted on the x axis - x_1 is given in the blue color, x_2 is in that reddish color and x_3 is in the green color. Each with it is

own symbol diamond, square and triangle respectively. You can see that we start with zero values for all the 3, and at the end of the first iteration, first step x_1 has decreased to something like minus 1.5, x_2 has increased and x_3 has increased.

At the next step x_2 has x_1 has increased from minus value to positive value and the other 2 have come down, and at the next step again x_2 has come down here and x_1 has come down and x_3 and x_2 have come up. And after that there is very little change that means that you can see that there is a big change, a big change, a smaller change and then very small changes and it seems to have converged because here at least graphically we are not seeing significant change between the previous one and this one.

So, for the same initial guess if we use the Gauss-Seidel method for this particular case we were expecting to get much faster solution because of updating and we see that we have a problem here. Because the computed values, the solution finally, we can see is x_2 is 1 x_3 is 2 and x_1 is 1 and x_3 is 3. So, x_1 plus x_2 plus x_3 1 plus 2 plus 3 is 6 again 1 plus 2 x_2 that is 4 minus 6, that is 1 plus 4 minus 6 that is minus 1. So, the true solution is 1 2 and 3 for x_1 , x_2 , x_3 that we have got in very small number of steps here for the Jacobi method.

But we see that with the Gauss-Seidel method at the end of 4 iterations the values are coming as something like in excess of 200 for x_2 , and then in excess of minus 200 for x_1 , and x_3 is also not a small quantity. At the next step here, x_1 has become minus 600, and x_2 has become plus 600, x_3 has become minus 50 or something like that.

So, instead of being converging to values of 1 2 3 they are divergent. So, in this particular case we see that Jacobi converges quickly and Gauss-Seidel method diverges. We can also have other system of equations in which Jacobi method diverges and Gauss-Seidel method converges and some others in which both converge and some others in which both diverge. So, any kind of thing is possible and the difference is the small changes we are making whether to update or not update, it is not something that we can say that updating is universally good. We cannot say that this is very easy and then let us go on do it and we can see that there are methods; there are problems, simple problems in which they do not work.

So, it is precisely for this reason we have to do a proper convergence analysis. Just like in module 3 we took a simple problem and we showed that the wave equation and for the diffusion equation constantly, diffusing equation we have using the simplistic finite difference approximations can give us sometimes to problematic solutions and we see exactly the same kind of thing here. Although the iterative methods, this basic iterative methods are quite easy to implement and program they will not work in all cases, we have to do an a priori convergence analysis to see that the method would actually converge, only then it converges we can attempt this. So, first thing that needs to be done is under what conditions this will a given method converge and then if it converges then we are interested in the rate of convergence.

So, this is what we are going to do in the second part of this module 5. So, what we have learnt in the first part are very basic direct methods and iterative methods for the solution of $Ax = b$ type of things. We have seen focused specifically on the Gaussian elimination method, the lu decomposition method and the tridiagonal method as direct methods. We made the point that Gaussian elimination is the most efficient method for a general purpose $Ax = b$ where A does not exhibit any special properties and it is a full matrix or nearly full matrix. Lu decomposition is a specialized method which is almost as good as Gaussian elimination, but slightly inferior because we have to do 2 substitution process, one forward substitution and one backward substitution whereas, in the case of Gaussian elimination we have to do only one backward substitution.

So, there is a small difference there, but in cases where we have to solve $Ax = b$, several times with the same A then if you do the decomposition of A into L times U once then that can be used for all the other equations in which b is changing. So, in that sense lu decomposition becomes a better method than Gaussian elimination. It has other things also that we will see later on.

The tridiagonal method is a special method which works very well, but it has limitations in its applicability it is much better in terms of computational arithmetic operations than either Gaussian elimination or lu decomposition. But it will work only for matrices, a matrix having only 3 adjacent diagonals including the main diagonal one main diagonal and one above and one below, so only for this case will it work.

And even in this case we require diagonal dominance. So, in such a case we would not have any problem in division by zeros problem will not arise and we can make use of the tridiagonal method. It cannot be used for 2-D problem and it cannot be used for 3-D problem without any modifications.

So, when we come to the iterative methods we have seen today also, in this lecture also we have seen that Jacobi method and the Gauss-Seidel method. We have seen how easy these are to implement and to write programs compared to the direct methods where we have to do quite bit of eliminations and all that, which we have not really gone into. To write a program for Gaussian elimination will take lot more effort than for Gauss-Seidel method or Jacobi method, but for simple 3 by 3 that we have seen in this class here we can see that any method is not converging and we also made the point that this is only a special case and there are methods where Jacobi method will not converge and Gauss-Seidel method will converge and all that.

So, we need to do a convergence analysis and based on this we come up with some criteria for the condition of convergence and then once we are sure of convergence we look at the rate of convergence. We look in the second part of this module at some special methods which improve the rate of convergence over what can be obtained by the Jacobi and Gauss-Seidel methods for conditions in which they converge.

So, that will be part of the second module and we will be looking at advance methods which make use of the iterative approach and they also make use of some special characteristics of the direct methods. We will also finally look at the multi grid approach which is a probably the best general purpose method for sparse matrices that is coming out of CFD type of problems.

So, that is all there in the second part of module 5. So, as of this lecture we can say that the first module is complete.

Thank you.