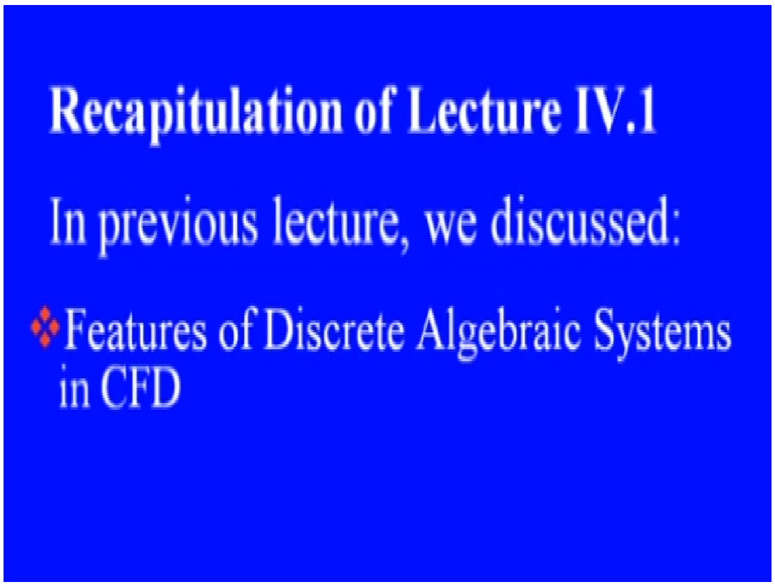
Computational Fluid Dynamics

Dr. Krishna M. Singh
Department of Mechanical and Industrial Engineering
Indian Institute of Technology – Roorkee

Lecture - 23
Direct and Basic Iterative Methods for Linear Systems

Welcome back to the next lecture in module 4 on solution of Discrete Algebraic Systems. We had had a look at these features of discrete algebraic systems which are encountered in CFD in the last lecture and we also had a look at methods for non-linear systems. In this lecture we are going to focus on direct and basic iterative methods for linear systems and accelerate iterative methods for linear systems if time permits today.

(Refer Slide Time: 00:55)



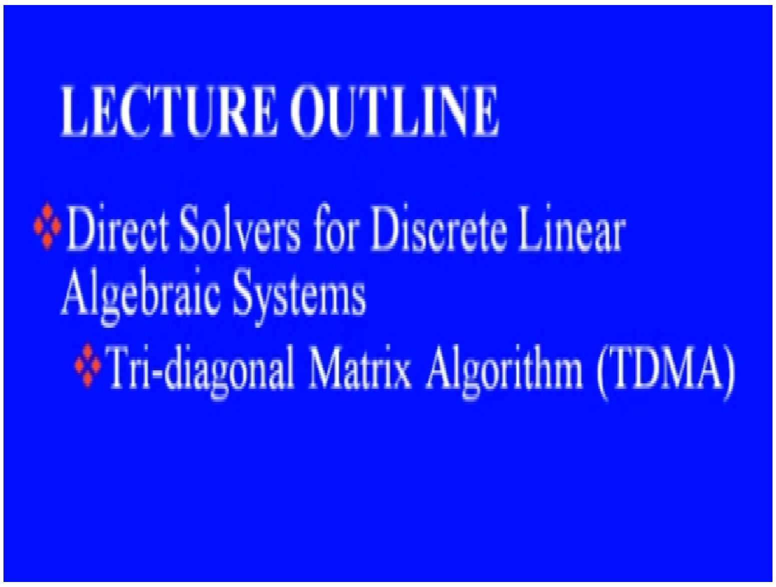
Recapitulation of Lecture IV.1

In previous lecture, we discussed:

- Features of Discrete Algebraic Systems in CFD

So, let us have recap for we did in the previous lecture. We discuss what are the basic features of the discrete algebraic systems encountered in CFD applications are and then we discussed the 2 basic methods for non-linear systems. That is a sequential iteration procedure and Newton-Raphson method.

(Refer Slide Time: 01:16)



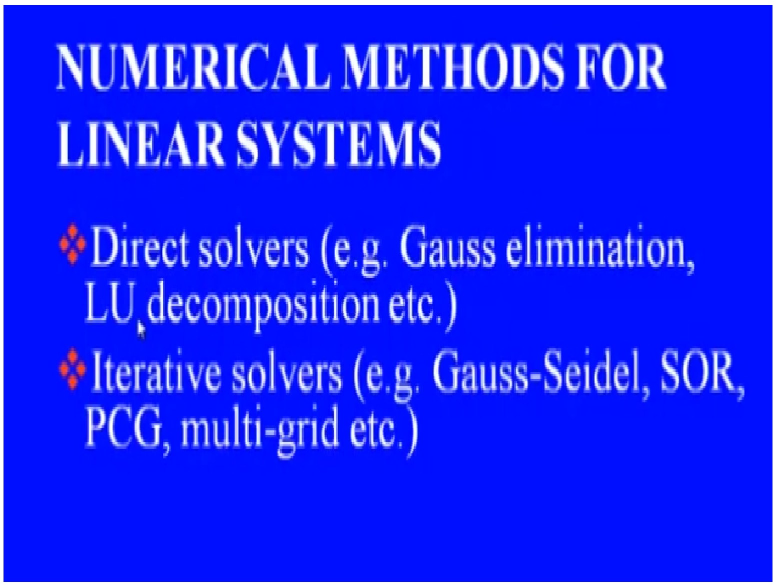
LECTURE OUTLINE

- Direct Solvers for Discrete Linear Algebraic Systems
  - Tri-diagonal Matrix Algorithm (TDMA)

In this lecture we will focus on linear systems. We will first have a brief look at few direct methods and then few basic iterative methods for linear systems. So, we will start off with the Direct Solvers for Discrete Linear Algebraic Systems. In particular, we will derive this tri-diagonal matrix algorithm. So, this is one of a very few Direct Solvers which are extensively used in CFD.

And then we will start off with iterative methods for linear systems, will have a look at few basic iterative methods.

(Refer Slide Time: 01:49)



NUMERICAL METHODS FOR LINEAR SYSTEMS

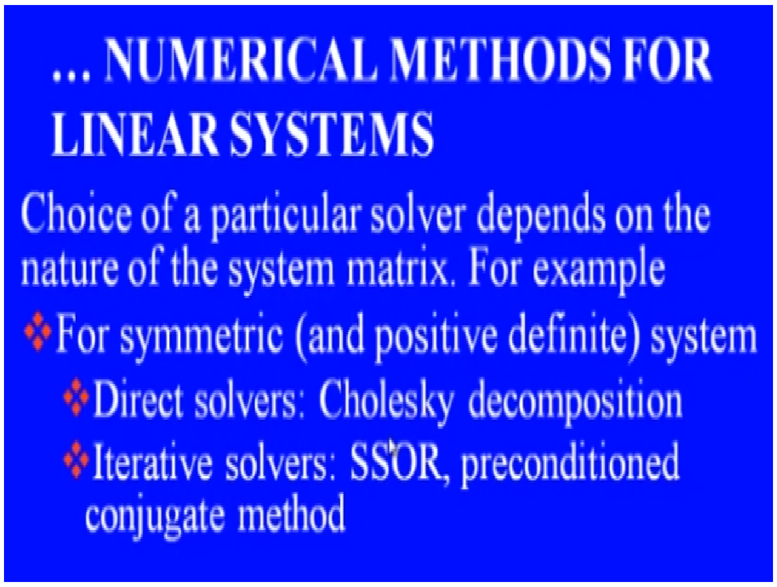
- Direct solvers (e.g. Gauss elimination, LU decomposition etc.)
- Iterative solvers (e.g. Gauss-Seidel, SOR, PCG, multi-grid etc.)

Now let us have a look at basic features of this numerical method for linear systems. We have direct solvers. What do you mean by direct solvers? The once which work directly on a given system of equations and if we had infinite precision in calculations they would lead us to an adjunct solution of the problem without any approximation or any numerical errors. Typical examples or Gauss elimination procedure and factorization base procedures like LU decomposition.

They are many more and we have got literature full of different decomposition procedures for solution of different types linear systems. Then we have also got iterative solvers, typical examples for our Gauss –Seidel process, SOR, which is successive over-relaxation scheme, pre-congregate method, multi-grid method and host of many iterative schemes.

We have got books full of these iterative methods as well as direct solvers available. So, in this lecture we are going to have a look at few simple represent that is which are extensively used in CFD.

(Refer Slide Time: 03:02)



… NUMERICAL METHODS FOR LINEAR SYSTEMS

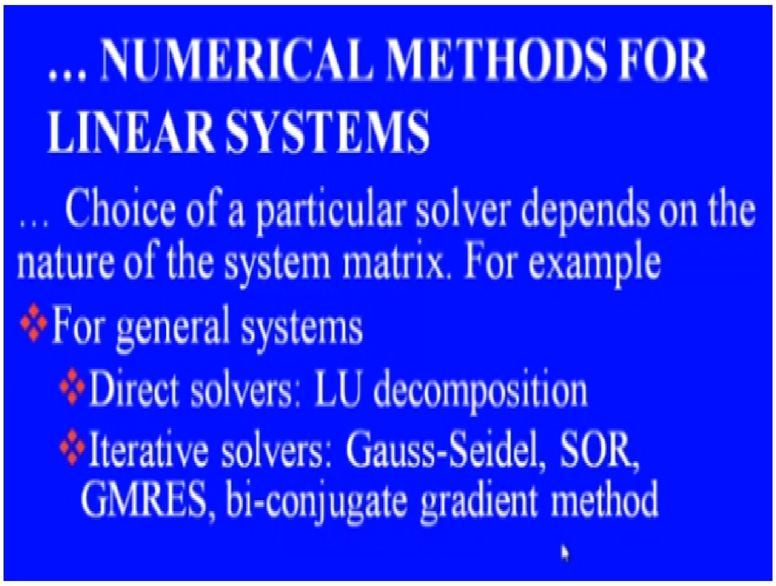
Choice of a particular solver depends on the nature of the system matrix. For example

- For symmetric (and positive definite) system
  - Direct solvers: Cholesky decomposition
  - Iterative solvers: SSOR, preconditioned conjugate method

Now, which particular solver we would use in our numerical code that will depend on the nature of the system matrix. For example, if our system is symmetric and positive definite this would be the case if we had used center difference approximation for a Poisson problem. And similarly if you use (()) (03:24) finite element process for any Poisson equation, both of these describes and processes will lead us to a symmetric and positive definite system.

And in this case we have got simplified versions of decomposition process for instance this Cholesky decomposition which works more efficiently compared to LU decomposition. And similarly we have got specialized versions of iterative solvers such as symmetric SSOR and preconditioned conjugate gradient method.

(Refer Slide Time: 04:04)



... NUMERICAL METHODS FOR LINEAR SYSTEMS

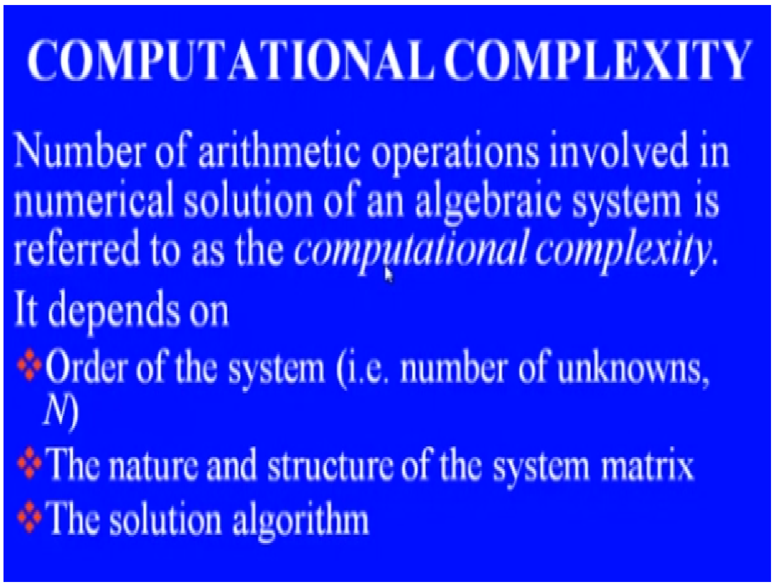
... Choice of a particular solver depends on the nature of the system matrix. For example

- For general systems
  - Direct solvers: LU decomposition
  - Iterative solvers: Gauss-Seidel, SOR, GMRES, bi-conjugate gradient method

But if you are not sure if your system is symmetric and positive definite that is to say your system is what we would say a general system and we know for sure that is going to be invertible a solution exists**.** Then we can use direct solvers such as LU decomposition, Gauss elimination and so on. And iterative solvers some of them are Gauss- Seidel, successive over-relaxation scheme, GMRES and bi-conjugate gradient method.

Now these methods for general system they take definitely more computation time compared to the methods which for symmetric systems will take. Let us introduce a term which we would use very often in presentation of algorithms is called computational complexity.

(Refer Slide Time: 04:56)



COMPUTATIONAL COMPLEXITY

Number of arithmetic operations involved in numerical solution of an algebraic system is referred to as the *computational complexity.*

It depends on

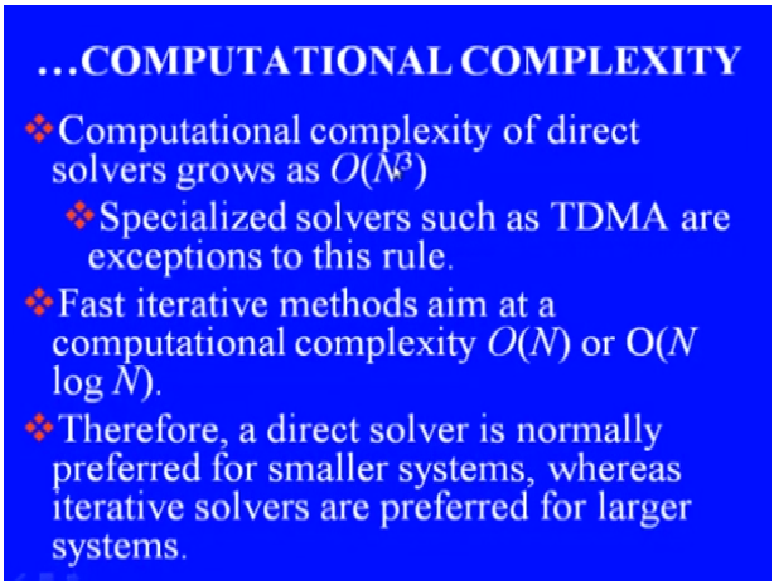
- Order of the system (i.e. number of unknowns, *N*)
- The nature and structure of the system matrix
- The solution algorithm

So, something linked to the number of arithmetic operations. So, number of arithmetic operations involved in numerical solution of an algebraic system is referred to as the computational complexity. So, this term is also extended for any algorithm whether it is being used for algebraic system otherwise. And it depends on in our context the order of system that is the number of unknown's capital N.

It also depends on the nature and its structure of the system matrix. By nature, we mean whether it is symmetric positive definite or it is un-symmetric system. By structure we mean whether it is parts system but multi diagonal structure, a banded matrix or it is a full matrix. So, the computation of complexity will depend on all these features. It also depends on the choice of the solution algorithm.

So, different algorithm they have got different computation complexities. In general, computational complexity of direct solvers,

(Refer Slide Time: 06:00)



...COMPUTATIONAL COMPLEXITY

- Computational complexity of direct solvers grows as $O(N^3)$
  - Specialized solvers such as TDMA are exceptions to this rule.
- Fast iterative methods aim at a computational complexity $O(N)$ or $O(N \log N)$.
- Therefore, a direct solver is normally preferred for smaller systems, whereas iterative solvers are preferred for larger systems.

For example, Gauss elimination or LU decomposition it grows as Order N cube and there is one exception to it. The specialized solvers such as TDMA and similarly cyclical decomposition which are applicable to very special matrixes they are exceptions to this rule. So, we talk about the computational complexity of direct solvers of (()) (06:30) of the order O N cube.

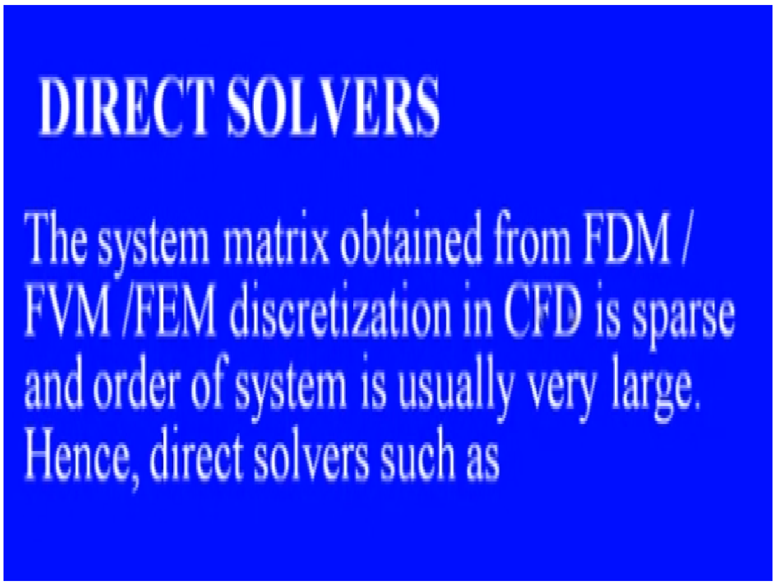
We are talking about our general system not specialized solvers such as TDMA. And we have got another category of algorithm which we called fast methods most, more precisely as fast iterative methods. Thus their aim is to obtain a computational complexity if it grows with this as system size that is computational complexity of the order O N are order O N log N.

Now, this is the ideal case or ideal aim which an algorithm developer aims at. Most critical algorithms will have much higher computational complexity than O N or O N log N. But there are few beautiful iterative algorithms available specifically on its structured grid or geometric multi-grid it can lead us to O N complexity. Now given the computational complexity of these direct solvers being O N cube.

These are normally preferred for smaller systems. Whereas if you got a large system which is, which we typically encounter in CFD analysis. Just remember if you want to have a numerical simulation of a typical industrial flow problem even with Reynolds average Navier Stokes simulation the number of grid points would be in many thousands may be close to a million. So, order of the system algebraic system in such a case would be order of millions.

Similarly, if you want to perform a large area simulation or direct numerical simulation the order of the system would be in billions. So, in such cases we can simply, we cannot afford to user direct solver. We would use iterative solvers for such large scale problems. Now let us have a look at our specific case of FDM, FVM or FEM discretization.

(Refer Slide Time: 08:34)

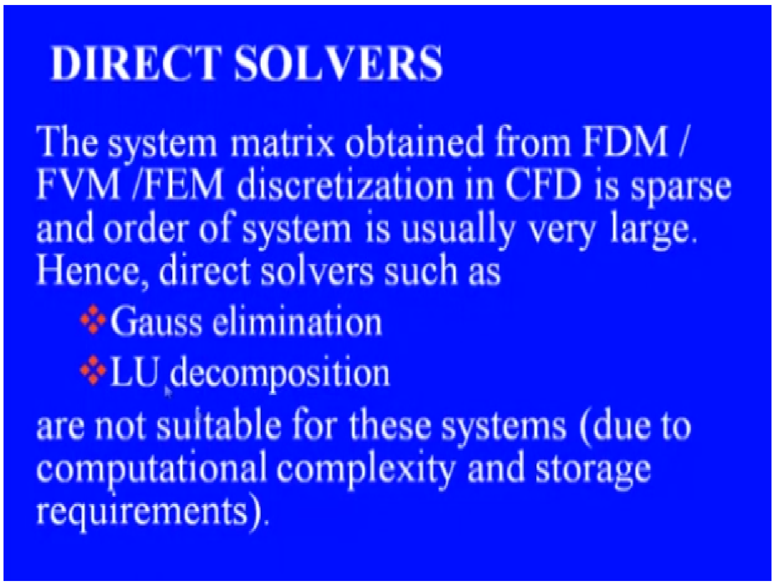


DIRECT SOLVERS

The system matrix obtained from FDM / FVM /FEM discretization in CFD is sparse and order of system is usually very large. Hence, direct solvers such as

When we apply any of these discretization schemes to our governing equations what we get is a sparse system and as I just mentioned few moments ago order of the system is usually large in millions and billions. Hence direct solvers such as Gauss elimination or LU decomposition are rarely used.

(Refer Slide Time: 09:00)



DIRECT SOLVERS

The system matrix obtained from FDM / FVM /FEM discretization in CFD is sparse and order of system is usually very large. Hence, direct solvers such as

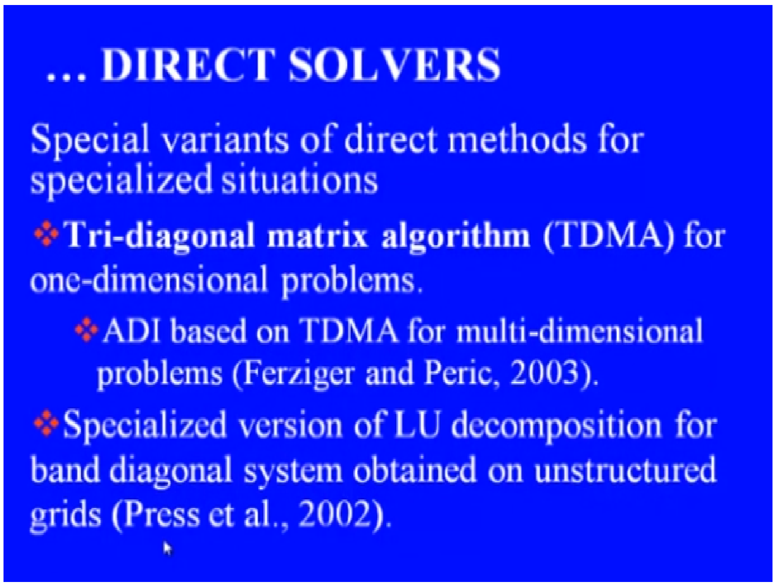
- Gauss elimination
- LU decomposition

are not suitable for these systems (due to computational complexity and storage requirements).

In fact they are not at all suitable for these systems due to their computational complexity which is order O N cube and it's storage requirement which are of order O N square. Even our the system matrix might take only storage of O N and the elimination process or in decomposition process we get the intermediate matrixes which might require much larger amount of storage.

So, hence these methods they are not used per say as solvers for large scale CFD applications. We will see one is specialized over than in Gauss elimination for one dimensional problems which leads us to our TDMA. And similarly this LU decomposition one is, some is specialized more than what we call incomplete LU decompositions. They are used as pre-conditioners for iterative solvers. So, that is in practice that is the only use such of direct solvers in CFD.

(Refer Slide Time: 09:56)



… DIRECT SOLVERS

Special variants of direct methods for specialized situations

- **Tri-diagonal matrix algorithm** (TDMA) for one-dimensional problems.
  - ADI based on TDMA for multi-dimensional problems (Ferziger and Peric, 2003).
- Specialized version of LU decomposition for band diagonal system obtained on unstructured grids (Press et al., 2002).

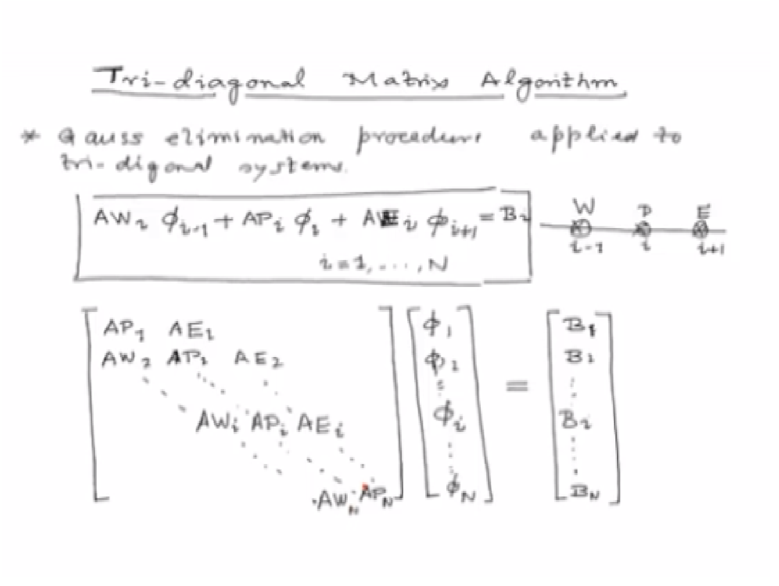
So, special variants for direct methods for a specialized situations we have got like our Tri-diagonal matrix algorithm which we say TDMA in short for one dimensional problems. Because in one dimensional we have used a central difference approximation, but the 3 point computational stand we get a tri-diagonal matrix and this TDMA we will see little later that let us got a computational complexity of O N.

We can also derive for multi-dimensional problem some iterative schemes what we call alternative direction based schemes which are based on TDMA for multi-dimensional problems. We would not take up these schemes in the present lecture. If you are interested, you can have a look at this algorithm in the book of Ferziger and Peric or in the book by Chunk**.**

Or in fact, you pick up any CFD book that will give you some application of this area which is based on TDMA. And similarly specialized versions of LU decomposition for band diagonal systems are also available for band diagonal systems which were obtained on unstructured grid when we apply finite volume or finite element discretization. For details please have a look at the numerical recipe book by Press et al.

Now let us come to those tri-diagonal matrix algorithm, which is applicable for its special case of one dimensional problems. Now let us derive this algorithm, I mean issue.

(Refer Slide Time: 11:39)



Tri-diagonal Matrix Algorithm

* Gauss elimination procedure applied to tri-diagonal systems.

$$\boxed{AW_i \phi_{i-1} + AP_i \phi_i + AE_i \phi_{i+1} = B_i \qquad i=1,\dots,N}$$

$$\begin{bmatrix} AP_1 & AE_1 & & & & \\ AW_2 & AP_2 & AE_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & AW_i & AP_i & AE_i & \\ & & & \ddots & \ddots & \ddots \\ & & & & AW_N & AP_N \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_i \\ \vdots \\ \phi_N \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_i \\ \vdots \\ B_N \end{bmatrix}$$

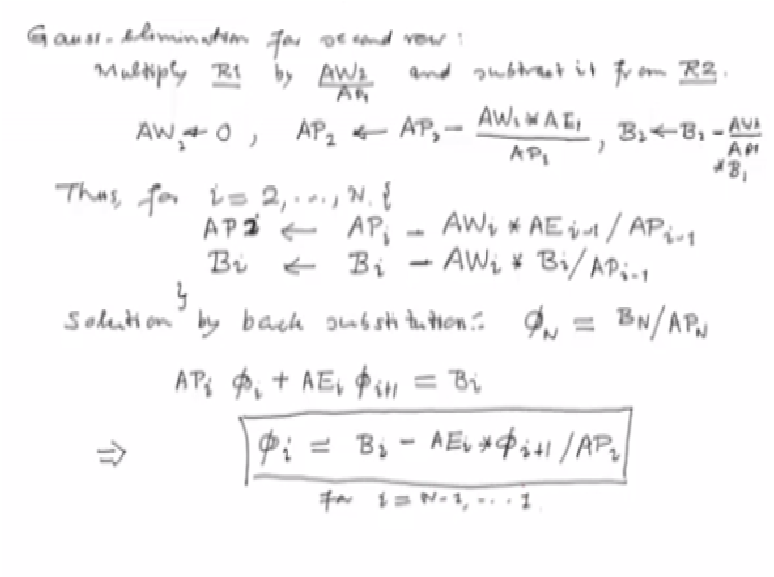
And this particular algorithm is just a special case of Gauss elimination. In fact, this is just the Gauss elimination apply to our current situation. So, it is essentially Gauss elimination procedure applied to tri-diagonal systems. So, our generic equation let us see how do we write our generic equation on 3 point computational stencil PEW. P stands for i, E stands for i+1 and W stands for i – 1.

In short in form we write this equation as AWi for a generic scalar quantity phi i – 1+APi phi of i+AEi phi of i+1 = Bi. Where I, where it is let us say 1 to n. So, this is our equation. If we look at the matrix first say now let us what it in matrix form. AP1, AE1 we will get only these 2 entries in the first row phi 1, phi 2 and so on. These are phi i to phi n. On the right hand side we got this Bi, B1, B 2 so on, Bi to Bn.

Let us complete our tri-diagonal structure the matrix. So in the second row we will have AW2, AP2 and AE2 and so on. So, that is how we will proceed in the i^{th} row we will have AWi, APi, AEi last one we have got our APn and to the best we have got AW of n. Now in Gauss scenario elimination what we want to do we would like to eliminate all he entries in the lower half of the matrix.

So, we would like to eliminate, now here we have got only slow diagonal to be eliminated and for that what we do differences we want eliminate this AW2 entry from the second row.

(Refer Slide Time: 16:20)



Gauss-elimination for second row:

Multiply $\underline{R1}$ by $\frac{AW_2}{AP_1}$ and subtract it from $\underline{R2}$.

$$AW_2 \leftarrow 0, \quad AP_2 \leftarrow AP_2 - \frac{AW_2 * AE_1}{AP_1}, \quad B_2 \leftarrow B_2 - \frac{AW_2}{AP_1} * B_1$$

Thus, for $i = 2, \ldots, N$ {

$$AP_i \leftarrow AP_i - AW_i * AE_{i-1} / AP_{i-1}$$
$$B_i \leftarrow B_i - AW_i * B_i / AP_{i-1}$$

}

Solution by back substitution: $\phi_N = B_N / AP_N$

$$AP_i \, \phi_i + AE_i \, \phi_{i+1} = B_i$$

$$\Rightarrow \quad \boxed{\phi_i = B_i - AE_i * \phi_{i+1} / AP_i}$$

for $i = N-1, \ldots, 1$

So, for that as a process of Gauss elimination for second row, we do not have to do anything for the first row because this entries they form the part of the upper triangular matrix, so, that we will leave us such. We want eliminate AW2 what do we do? We have to multiply the first row R1 by AW2 by AP1 and subtract it from R2. R2 is our row 2. Now you can clearly say that which entries would be affected of the row 2.

When we perform this subtraction process AW2 would be eliminated that will become 0. AP2 would be affected that is all. The AE2 will not be affected because the corresponding entry, the entry in the same column in the first row of 0. So, what we get modified entries would be that AW2 becomes 0 and our AP2 this is now assigned a value for the existing value of AP2 – AW2 into AE1 divided by AP1.

Our right hand side will also get modified in same way because we are performing the same two operation on the load vector as well. So, this B2 would be modified as B2 – AW2 by AP1 into B1. So, we can clearly say that the same process can be applied to any other subsequent rows for instance for eliminating the AWi entry from the ith row. You will multiply the i – nth row by AWi divide by APi – 1, so AWi gets eliminated.

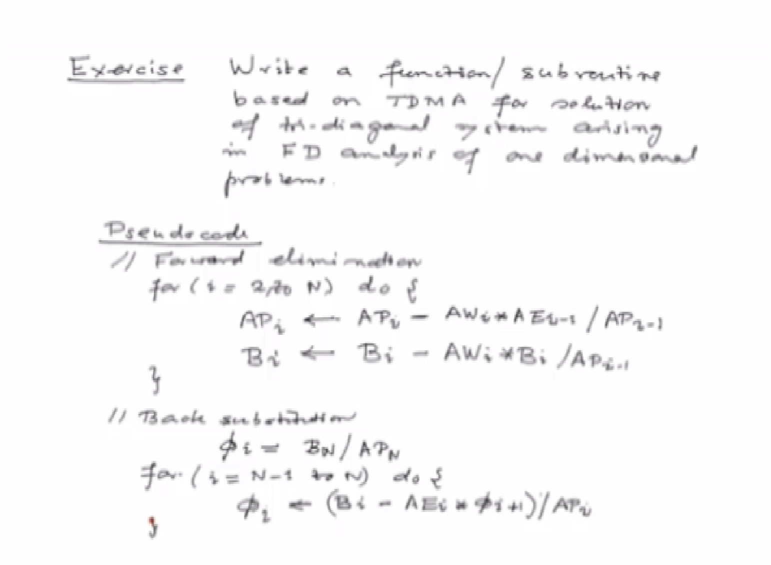
So thus in general thus for i = 2 to N. What we need to do or forward elimination process becomes APi this becomes APi – AWi into AE of i – 1 divided by AP of i – 1. Bi this becomes Bi – AWi into Bi divided by APi – 1. So, this is our forward elimination processes which we need to apply for all the rows starting from second row up to the last row. So, once we have completed this forward elimination how do we obtain the solution? In the last review, we are left with the only 1 entry that is APn phi n = Bn.

So, now our solution is obtained solution by back substitution. We would proceed in backward order, like first we will obtain phi N. So, phi N is now given by BN divided by APN. And how about the remaining entries let us say the ith row we are left with the only entries APi and AEi. So, our ith is equation is now with modified coefficient that is APi phi i+AEi phi i+1 = Bi. So, this leads to equation for phi i = to Bi – AEi into phi of i +1 divided by APi.

Now we need to apply this formula for i = N – 1 to 1 in this order. Okay, so this is just well standard Gauss elimination process applicable to a specialized case and the number of operation which you have formed they are proportional to the number of unknowns in our system. So, if you want to program this method this Pseudo code for this TDMA algorithm becomes very simple.

And I would encourage you to write a subroutine or a function based on TDMA.

(Refer Slide Time: 23:09)



Exercise: Write a function/ subroutine based on TDMA for solution of tri-diagonal system arising in FD analysis of one dimensional problems.

Pseudocode

```
// Forward elimination
for (i = 2, to N) do {
        $AP_i \leftarrow AP_i - AW_i * AE_{i-1} / AP_{i-1}$
        $B_i \leftarrow B_i - AW_i * B_i / AP_{i-1}$
}
// Back substitution
        $\phi_i = B_N / AP_N$
for (i = N-1 to N) do {
        $\phi_i \leftarrow (B_i - AE_i * \phi_{i+1}) / AP_i$
}
```

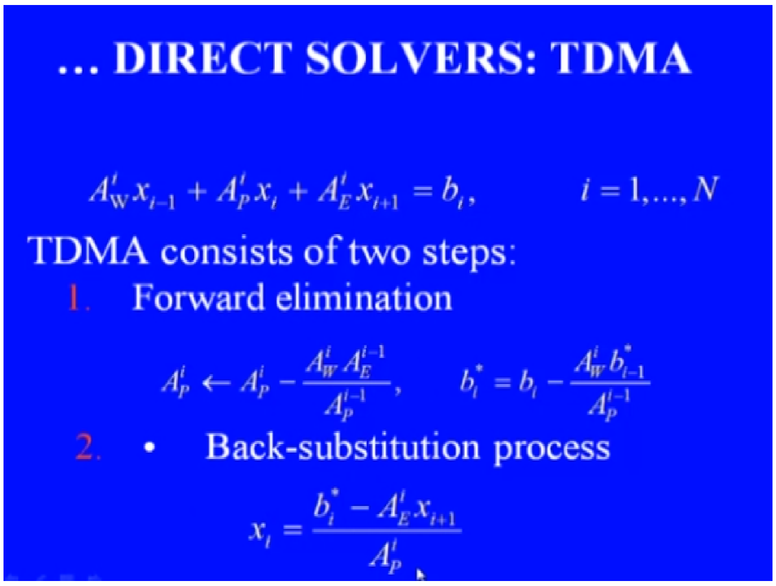
So, thus I would give as an exercise. Write a function or subroutine based on TDMA for solution of tri-diagonal system arising in let us say finite difference analysis of one dimensional problems. For your benefit if you can write this Pseudo code in forward elimination. We just need to apply the elimination part of the formula. So, we can say for i = 2 to N do or modify A.

So, APi that gets modified us APi–AWi into AEi–1 divided by APi–1 and your Bi gets modified us Bi–AWi into Bi divided by AP–1. And then our back substitution phase phi i = BN divided by APN and then for i = N-1 to N do phi i is Bi – AEi into phi of i+1 this whole thing divided by APi. So, you can see this is very simple to code in fact the total number of code lines would be 4+4, 8.

So, in total of 8 code lines you can write this algorithm in a FORTRAN or C language. And there is something's which are commonly being multiplied you can take them out and may be write introduce it additional line of code but reduce the number of multiplications and divisions by doing that. For instance, is AWi divided by APi–1 which we had not forward elimination for each i. It can be calculated only once.

This division process can be done only once and this local variable can be multiplied with respective entries.

(Refer Slide Time: 27:55)



... DIRECT SOLVERS: TDMA

$$A_W^i x_{i-1} + A_P^i x_i + A_E^i x_{i+1} = b_i, \qquad i = 1,\ldots,N$$

TDMA consists of two steps:

1. Forward elimination

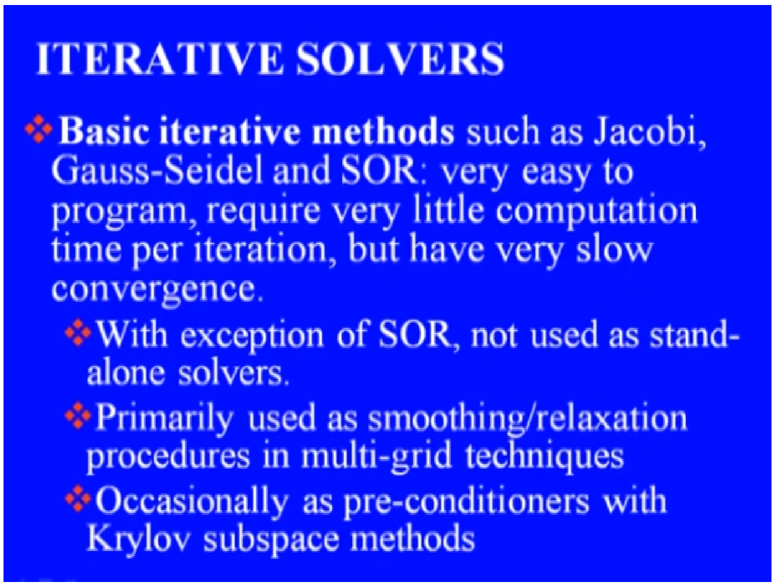
$$A_P^i \leftarrow A_P^i - \frac{A_W^i A_E^{i-1}}{A_P^{i-1}}, \qquad b_i^* = b_i - \frac{A_W^i b_{i-1}^*}{A_P^{i-1}}$$

2. • Back-substitution process

$$x_i = \frac{b_i^* - A_E^i x_{i+1}}{A_P^i}$$

So, forward elimination where by modify this APi the diagonal coefficients and as a consequence our right hand side also gets modified. Once you have completed at this we can easily get our solution by a Back-substitution process. Now, let us move on to the iterative solvers and they are simple classification of these iterative solvers, they are put in 2 categories.

(Refer Slide Time: 28:28)


ITERATIVE SOLVERS

- **Basic iterative methods** such as Jacobi, Gauss-Seidel and SOR: very easy to program, require very little computation time per iteration, but have very slow convergence.
  - With exception of SOR, not used as stand-alone solvers.
  - Primarily used as smoothing/relaxation procedures in multi-grid techniques
  - Occasionally as pre-conditioners with Krylov subspace methods

Basic iterative solvers and accelerated iterative solvers or advanced iterative solvers. So, basic iterative methods solve typical examples are Jacobi's method, Gauss-Seidel method and SOR. These methods are very easy to program. We will see the algorithms in a short while from now. So, they are very easy to program. They require very little computation time per iteration. But they have very slow convergence.

So, for very large systems which we have in CF difference we want to solve the pressure Poisson equation involved in large AD simulation or DNS system is of close to a billion, a few billions this Gauss-Seidel, Jacobi and SOR would require many millions of iterations. And because of slow convergence with exception of SOR Gauss-Seidel and Jacobi these 2 methods are there variance. They are not used as standalone solvers.

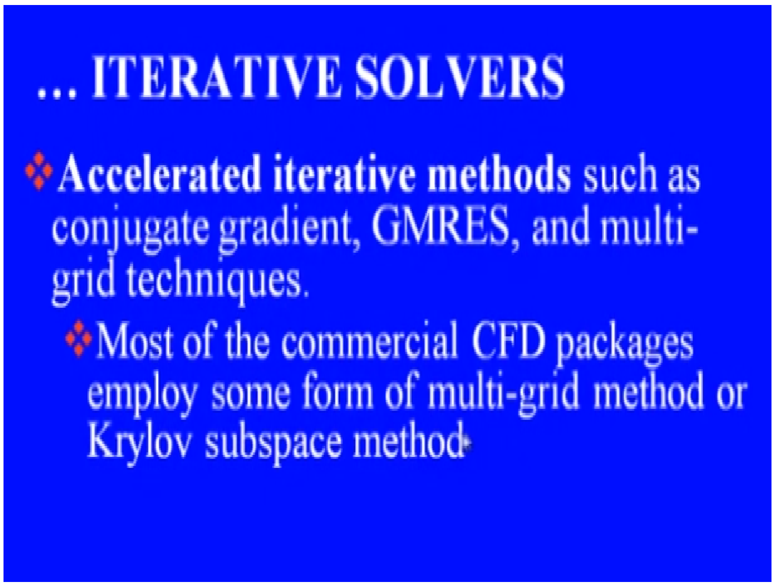
By standalone solver we mean that they are used as an equation solver. So, only SOR with then optimum value of what we called over relaxation parameter. It is used in CFD course**.** Jacobi and Gauss- Seidel we do not use as standalone solver. All these basic iterative method they are primarily used as what we call is smoothing or relaxation procedures in multi-grid techniques. We will take of these multi-grid techniques in the next lecture.

So, both say this category of iterative method they are primarily used as this smoothing or relaxation techniques. And this one more usage which we have got these measures can also be used as pre-conditioners with Krylov subspace methods. We will have a look at 1 or 2 methods of Krylov subspace family namely conjugate gradient methods for symmetric systems and bi-conjugate gradient method for general systems.

And Jacobi's method is used very often as a pre-conditioner to these Krylov subspace methods. One more feature, attractive feature of these basic iterative methods are that they are very easy to paralyze. That is to say if you want to use the massively parallel computer with multiple codes wherein we would like to solve as 1 sub set of problem on different codes.

This Jacobi, Gauss- Seidel or SOR methods can be easily written for such machines. In fact their version would not be very different from the version written for a serial machine. So, that is the advantage which these basic iterative methods for this. Their only down side is their convergence rate is very slow.

(Refer Slide Time: 31:52)



… ITERATIVE SOLVERS

- **Accelerated iterative methods** such as conjugate gradient, GMRES, and multi-grid techniques.
  - Most of the commercial CFD packages employ some form of multi-grid method or Krylov subspace method

Next is our accelerated iterative methods, why we call them as accelerated? We will get to know when we take up each of these methods for instance say this conjugate gradient method. In this conjugate gradient method and GMRES method the corrections show some specific directions to accelerate the convergence of iterations. So that is why verified to these methods as accelerated iterative methods.

Similarly, multi-grid techniques they employ the solutions at multiple grid labels. We will have a sequence of unstructured grids and the corrections or computed at different grid levels which will enhance the convergence of the iteration process. Now the multi-grid methods can be used as standalone methods on structural grid. They have got 2 formats of multi-grid techniques.

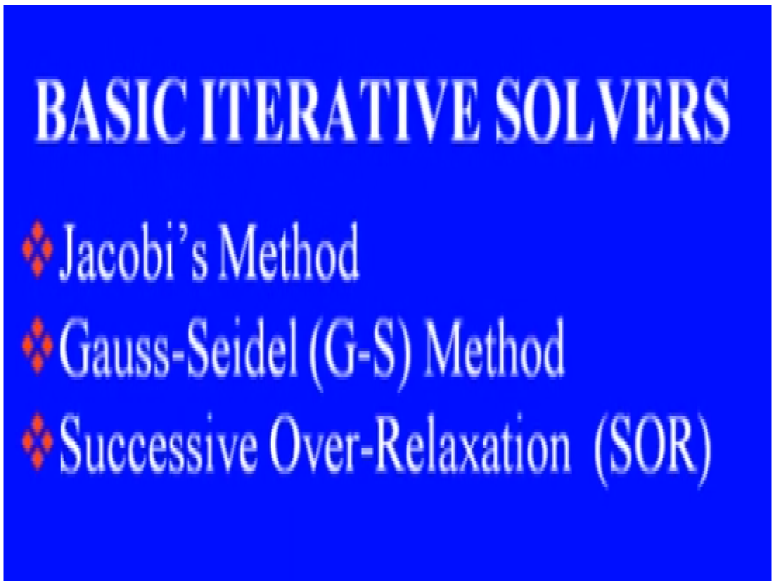
One is what we call geometric multi-grid wherein we use a sequence of unstructured grids. So, if that were possible that if it were possible for us to generate a sequence of hierarchical grid structures. Then we can go for what we call geometric multi-grid. On simple rectangular geometrics we can easily generate a hierarchy of nested grids in the context of structured grid techniques.

It is more difficult we have got unstructured grids for instance we want to use finite volume or finite element method unstructured grid. We can still device multi-grid techniques on that but some of the operators which are required for multi-grid techniques, their derivation becomes a bit more tedious task. In such situations recently a new set techniques have been proposed which are called algebraic multi-grid techniques.

Now these algebraic multi-gird techniques they do not require us to supply a sequence of hierarchal grids. All that we need to supply is our matrix and it creates so called coarsening and prolongation operators just based on this structure of the matrix itself. So, such technique that is why everything is being done algebraically, so these are called algebraic multi-grid techniques.

And today most of the commercial CFD packages employs some form of multi-grid method or Krylov subspace method. In fact, most of them what they were do they use this multi-grid method as a pre-conditioner for Krylov subspace method. Now let us come back to our basic iterative solvers because we might or we would use some of them as pre-conditioners for Krylov subspace methods or multi-grid methods.

(Refer Slide Time: 35:09)

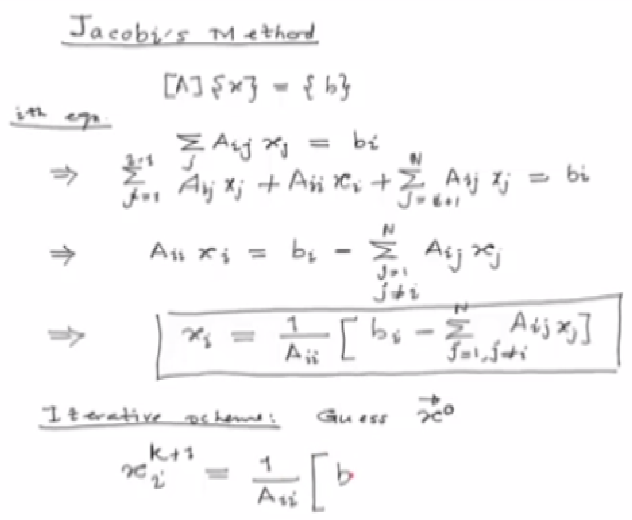


BASIC ITERATIVE SOLVERS

- Jacobi's Method
- Gauss-Seidel (G-S) Method
- Successive Over-Relaxation (SOR)

So, let us see few basic iterative methods like the first one is our Jacobi's method, the second one is Gauss-Seidel method and third one is Successive Over Relaxation method. So, let us have a look at the algorithm for these methods. The first one is Jacobi's method.

(Refer Slide Time: 35:28)



Jacobi's Method

$$[A]\{x\} = \{b\}$$

ith eqn.

$$\sum_j A_{ij} x_j = b_i$$

$$\Rightarrow \sum_{j=1}^{i-1} A_{ij} x_j + A_{ii} x_i + \sum_{j=i+1}^{N} A_{ij} x_j = b_i$$

$$\Rightarrow A_{ii} x_i = b_i - \sum_{\substack{j=1 \\ j \neq i}}^{N} A_{ij} x_j$$

$$\Rightarrow \boxed{x_i = \frac{1}{A_{ii}} \left[b_i - \sum_{j=1, j\neq i}^{N} A_{ij} x_j \right]}$$

Iterative scheme: Guess $\vec{x}^0$

$$x_i^{k+1} = \frac{1}{A_{ii}} \left[b \right.$$

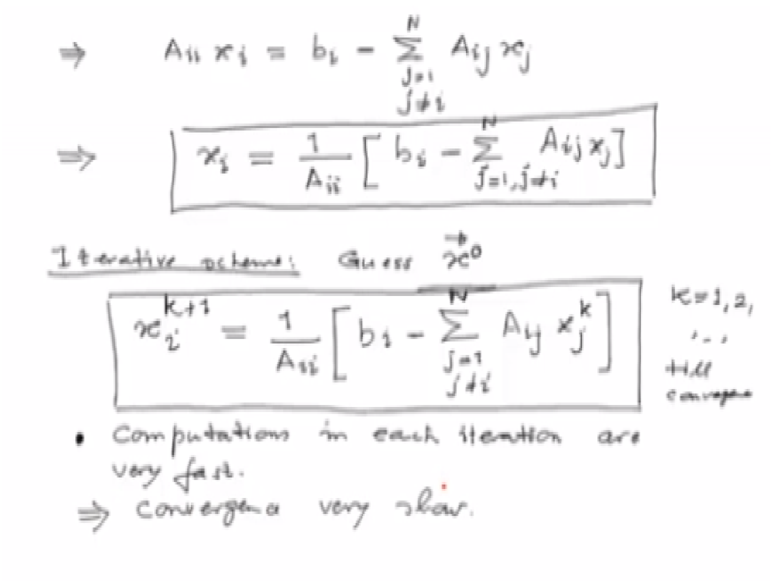
So, our system this write it as Ax =b. So, if you want to write it in index notation we can also write it as the ith row of the system or rather let us call it ith equation that will write as sigma Aij x j = bi. Now let us separate the x item. So, we can rewrite the left hand summation as sigma j =1, 2 i – 1 Aij x j+Aii x i+sigma j = i+1 to N Aij x j = bi. Let us transfer the terms which do not involve xi on the right hand side. So, you get Aii x i = bi – sigma.

These 2 summations we can combine and we can write them in short hand form J = 1 to N with the condition J not = I, Aij xj or xi = 1 by Aii bi – sigma j = 1 and J0 = i to N Aij x j.

Now these basic relations can be utilized as an iterative scheme and this what Jacobi proposed that look if we had the guest value or if we start off with some guest value of for the vector x.

Based on those guestimates we can obtain an improved estimate for the different components of the vector.

(Refer Slide Time: 38:59)



$$\Rightarrow \quad A_{ii} x_i = b_i - \sum_{\substack{j=1 \\ j \neq i}}^{N} A_{ij} x_j$$

$$\Rightarrow \quad \boxed{x_i = \frac{1}{A_{ii}} \left[b_i - \sum_{j=1, j \neq i}^{N} A_{ij} x_j \right]}$$

Iterative scheme: Guess $\vec{x}^0$

$$\boxed{x_i^{k+1} = \frac{1}{A_{ii}} \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{N} A_{ij} x_j^k \right]} \quad k = 1, 2, \ldots \text{ till converge}$$

- Computations in each iteration are very fast.

$\Rightarrow$ Convergence very slow.

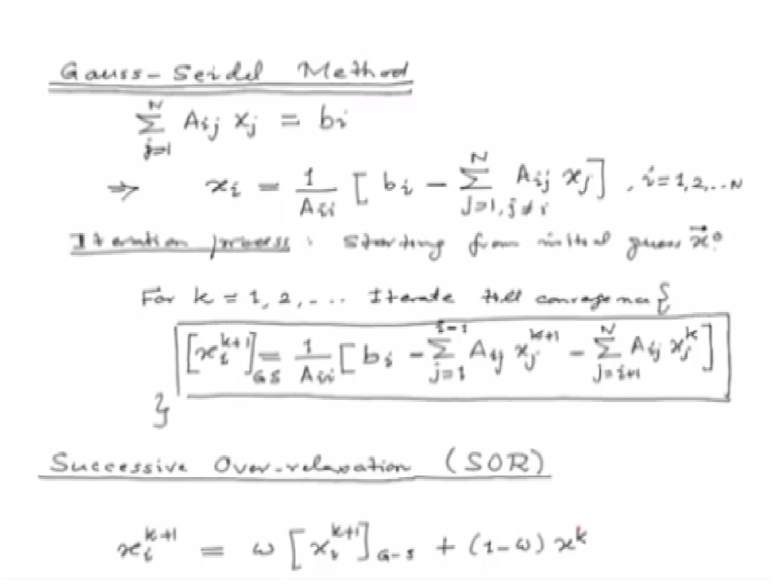
So, iterative scheme becomes Guess x0. Now let us put our iteration counter as a subscript. So, let us get Guess an initial value which we will denote by subscript of 0 and then we will tried we will obtained the next iterate x i K+1. This can be computed in using the values at the previous iterations 1 by Aii bi – sigma j = 1 J0 = i to N Aij x j k.

So, this is a simple algorithm. We cannot think of anything simpler and if you compare with what we discussed for non-linear systems very similar to our Picard iteration or vice versa Picard iteration might have been obtained driving inspiration from Jacobi’s method, okay. And as we have noted this scheme is very simple at one moment. We need only the entries corresponding to the ith row.

That is coefficient Aii, Aij for all j’s and we need the corresponding components. So, that is why computations involved, computations in each iteration are very fast. So, here we can say that K= 1, 2 and so on till convergence. The down side we have already noted convergence is very slow. Is there something which we can do to improve the convergence of this Jacobi iteration?

One of the simplest possible things would be in specifically in computer implementation the moment we have generate it a new estimate x i k+1. We do not put to this value that k+1 of iteration in a separate vector in fact old values or over written, okay. So, should we not use the already available new values and if we can use the new values hopefully the convergence can be improved. So, that was the basic premise of the Gauss-Seidel method.

(Refer Slide Time: 42:50)



Gauss-Seidel Method

$$\sum_{j=1}^{N} A_{ij} x_j = b_i$$

$$\Rightarrow \quad x_i = \frac{1}{A_{ii}}\left[b_i - \sum_{j=1, j\neq i}^{N} A_{ij} x_j \right], \quad i = 1, 2, \ldots N$$

Iteration process : Starting from initial guess $\vec{x}^0$

For $k = 1, 2, \ldots$ Iterate till convergence {

$$\boxed{\left[x_i^{k+1}\right]_{GS} = \frac{1}{A_{ii}}\left[b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{N} A_{ij} x_j^{k} \right]}$$

}

Successive Over-relaxation (SOR)

$$x_i^{k+1} = \omega \left[x_i^{k+1}\right]_{G-S} + (1-\omega)\, x^k$$

Our basic equation remains the same. The starting point that is our, we started off the simple equation the ith equation was j= 1 to n Aij x j = bi and this using this we got a modified form for xi or xi can be written in terms of the remaining components of the vector. So, 1 by Aii bi – sigma j = 1, J0 = i to N Aij x j.

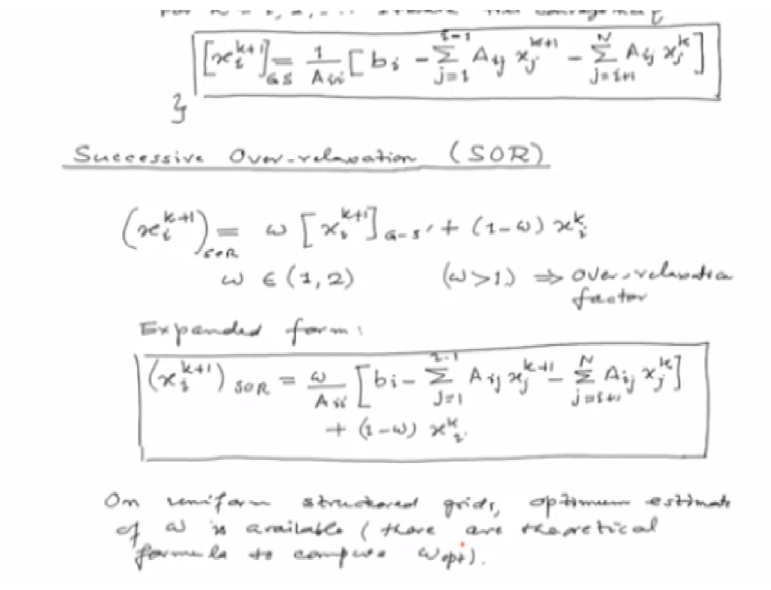
So, when we used the values at previous iteration that is sometimes also referred to in Jacobi iteration that we are trying to annihilate the residual involved in ith equation by using the values at the previous iteration level. What was suggested by Gauss and Seidel was that look use the values which have already been computed at a given xi count. If we will start from i = 1, 2 and so on, so this equation holds good could for all i values.

So, our iteration process what do we do, it is starting from initial guess x of 0 for k = 1, 2 and so on iterate till convergence and what is our iteration process. We should have xi at k+1 = 1 over Aii within bracket bi – now let us break this summation term into 2 parts. One which would use the values of already obtained at this iteration level. That is sigma j = 1 to i–1. For these indices j = 1 to i–1 we would have obtained a new iterate, so use that value.

That is Aij x j k+1. And then the next part-sigma j = i+1 to N Aij x j k. So, this simple modification to Jacobi algorithm leads to rate of convergence which is slightly faster than Jacobi's method and what is our celebrated Gauss-Seidel method. There are versions of Gauss-Seidel method which are specifically designed for parallel computers.

Some of them what we called red-black ordering of the notes rather the planes in multiple dimensions. So, if you are interested you can have look at these versions in some of the standard books which will mention at the end of this lecture. The next method or the last method if you are going to have a look at in this series is what we called successive over relaxation. It is popularly known by acronym SOR.

(Refer Slide Time: 47:30)



$$\boxed{\left[x_i^{k+1}\right]_{GS} = \frac{1}{A_{ii}}\left[b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{N} A_{ij} x_j^{k}\right]}$$

}

Successive Over-relaxation (SOR)

$$\left(x_i^{k+1}\right)_{SOR} = \omega \left[x_i^{k+1}\right]_{G-S} + (1-\omega)\, x_i^k$$

$\omega \in (1,2)$ $(\omega > 1) \Rightarrow$ over-relaxation factor

Expanded form:

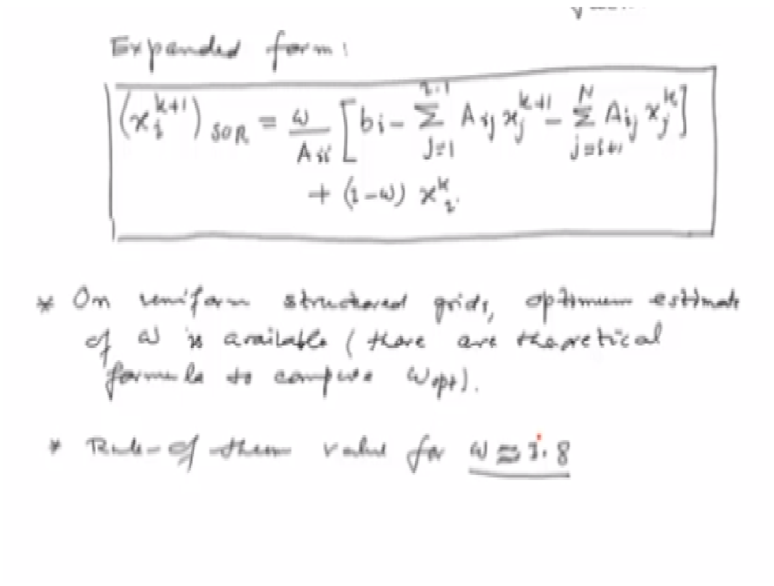
$$\boxed{\left(x_i^{k+1}\right)_{SOR} = \frac{\omega}{A_{ii}}\left[b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{N} A_{ij} x_j^{k}\right] + (1-\omega)\, x_i^k}$$

On uniform structured grids, optimum estimate of ω is available (there are theoretical formula to compute ω_{opt}).

And what is the fun about the successive over relaxation techniques if I look, whatever value or iterate or the new solution which we have obtained let us x i k+1 by Gauss-Seidel iteration do not take that is our final values. In fact, the final at a given iteration level should be of weighted average of the value which we obtained from Gauss-Seidel and the value at the previous iteration.

This simple modification works wonders in the convergence of the scheme. So, if I write this such x i k+1 in our Gauss-Seidel algorithm let us put it under subscript rate. So, in terms of this our successive over relaxation methods say that look now x i k+1 is omega times x i of k+1 Gauss-Seidel+1–omega times x of k where omega as a scalar quantity between 1 and 2. So, omega is always taken greater than 1. So, that is why it is called as over relaxation factor.

(Refer Slide Time: 50:00)



Expanded form:

$$\boxed{\left(x_i^{k+1}\right)_{SOR} = \frac{\omega}{A_{ii}}\left[b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{k+1} - \sum_{j=i+1}^{N} A_{ij}x_j^{k}\right] + (1-\omega)\,x_i^k}$$

* On uniform structured grids, optimum estimate of ω is available (there are theoretical formula to compute ω_{opt}).

* Rule-of-thum value for $\underline{\omega \simeq 1.8}$

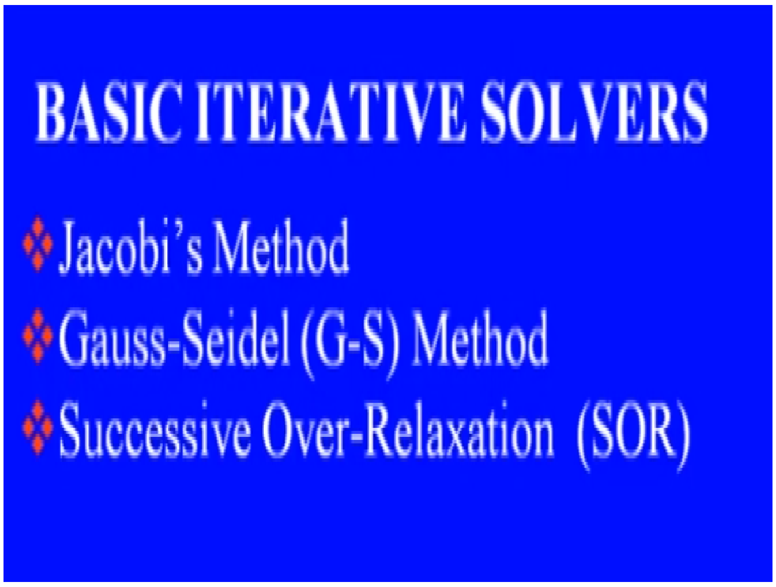
A previous short in form I wrote just to stabilize the close link between the SOR and Gauss-Seidel. For programming purposes we can now write the expanded form that x i k+1 SOR = omega divided by Aii within brackets bi – sigma j = 1 to i – 1Aij x j k+1 – sigma j = i +1 to N Aij x j k+1 – omega times x k i. So, there is certain difference here.

Even this x i k+1 j s prime I would like to write that as because here in this formula this x j k+1 in the right hand side they are the once which have the current iterates obtained by our SOR algorithm. And now on a structured grid on uniform structured grids, optimum estimate of omega is available. In fact, there are formulas, there are theoretical formulas which you can find in any book which gives SOR method theoretical formula to compute omega optimum in this case.

But on general grids if we do not have wherein we cannot find out an estimate of omega opt, rule of thumb value for omega is you take some value close to 1.7 or 1.8. This would give us a reasonable rate of convergence. This was something similar waiting in our case of non-linear problems. The non-linear problems we take omega to be less than 1 so that process is referred to us under relaxation.

But in the case of linear systems we always use over relaxation. That is weighting omega would be chosen to be greater than 1, okay.

(Refer Slide Time: 53:30)

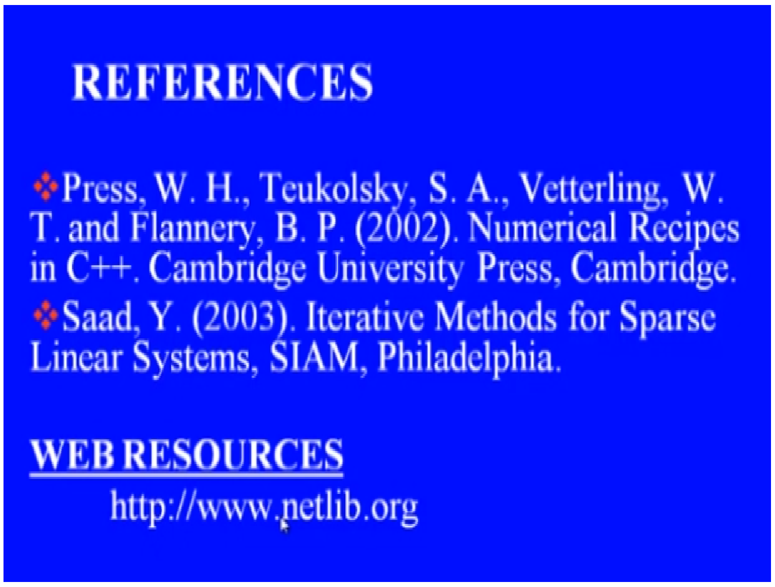


BASIC ITERATIVE SOLVERS

- Jacobi's Method
- Gauss-Seidel (G-S) Method
- Successive Over-Relaxation (SOR)

So, that brings us now to the close of our lecture on the direct solvers and basic iterative solvers. For further details, you can have a look at these 2 books.

(Refer Slide Time: 53:42)



REFERENCES

- Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2002). Numerical Recipes in C++. Cambridge University Press, Cambridge.
- Saad, Y. (2003). Iterative Methods for Sparse Linear Systems, SIAM, Philadelphia.

WEB RESOURCES

http://www.netlib.org

The Press et all's book on Numerical Recipes. This gives the methodologist methods algorithms as well as codes for direct solvers and iterative solvers. For detailed description of iterative methods for Sparse Linear Systems this is what we encounter in CFD. This, it another nice book, book by Saad published by SIAM in 2003.

This is a definitive reference for basic as well as advanced iterative methods with specific applications to this sparse systems obtained from finite difference, finite volume or finite element discretization of partial differential equations. So, for details you can have a look at this book. This pdf is easily available on the net and then this nice collection of routines on the web.

There are many sources but this one very reliable resource that is called Netlib, http://www.netlib.org. So, you can find many routines related to the solution of linear systems on this side as well which can inspire you or help you develop a better understanding of these methods as well as to develop your own computer course. So we stop here and this lecture and the next lecture we will take up accelerated iterative methods for sparse linear systems.