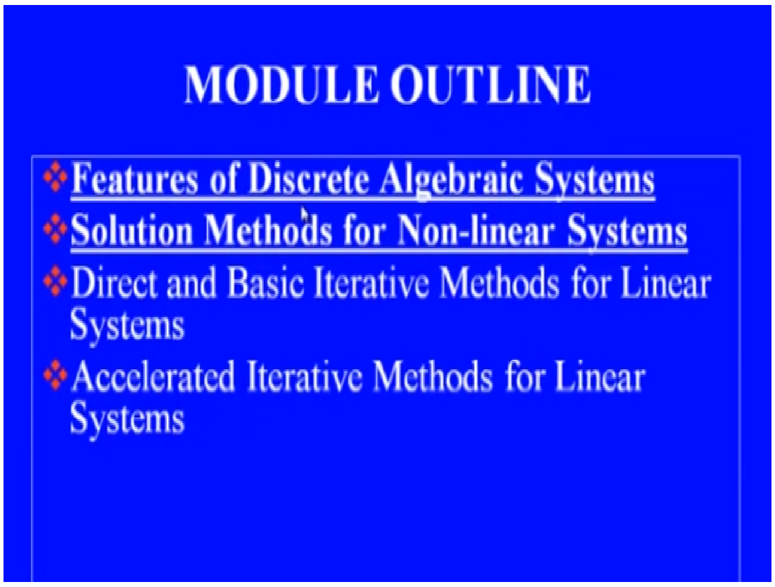
Computational Fluid Dynamics

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

Lecture 22
Solution of Discrete Algebraic Systems

Welcome to module 4 on Solution of Discrete Algebraic Systems. In this module, we would focus on the Solution of Discrete Algebraic Systems equations which we obtained earlier using finite differences. We would get fairly similar systems if we use finite element or finite volume technique for special discretization.

(Refer Slide Time: 00:58)



MODULE OUTLINE

- Features of Discrete Algebraic Systems
- Solution Methods for Non-linear Systems
- Direct and Basic Iterative Methods for Linear Systems
- Accelerated Iterative Methods for Linear Systems

So, we would have a brief look at the features of these Discrete Algebraic Systems and we will discussed in brief which solution makes the best suited for these systems specifically in the context of CFD applications. And we know with our Navier-Stokes equation they represent a system of coupled non-linear equations. So, we will have a brief look at solution methods for non-linear systems.

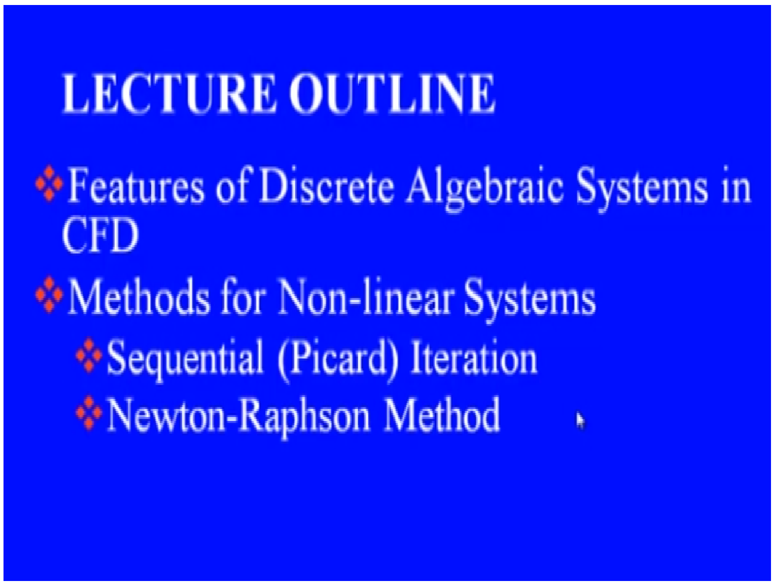
We might also get non-linear systems arising from finite difference or finite element discretization of non-linear heat conduction problems. So, thus we will first have a brief look at the solution methods on non-linear system and then solution of any non-linear system involves solution of a set of linear algebraic equations as the part of the iterative scale. So we will discuss in bit more detail direct as well as iterative methods for solution of linear systems.

We will have a look at what we will call basic iterative methods for linear systems like our Jacobi and Gauss-Seidel iterative schemes and we will see some accelerated schemes based on Krylov acceleration and multi-grid techniques. So in today's lecture we are going to focus on the first 2 topics in this module. While discussing the schemes we will restrict ourselves to the once which are most commonly used in CFD analysis.

Otherwise you can pick up any consolation of linear systems, there are plenty of schemes available. Similarly, they are a number of schemes available for solution of non-linear algebraic equations. So we are not going to discuss each one in detail otherwise each one would require a separate course for itself. So, we would focus very briefly on this schemes which are pertinent to a CFD application.

So in today's lecture it is bit more general, we will have a look at the general features of or generic features of discrete algebraic systems which we obtained in CFD applications.

(Refer Slide Time: 03:07)

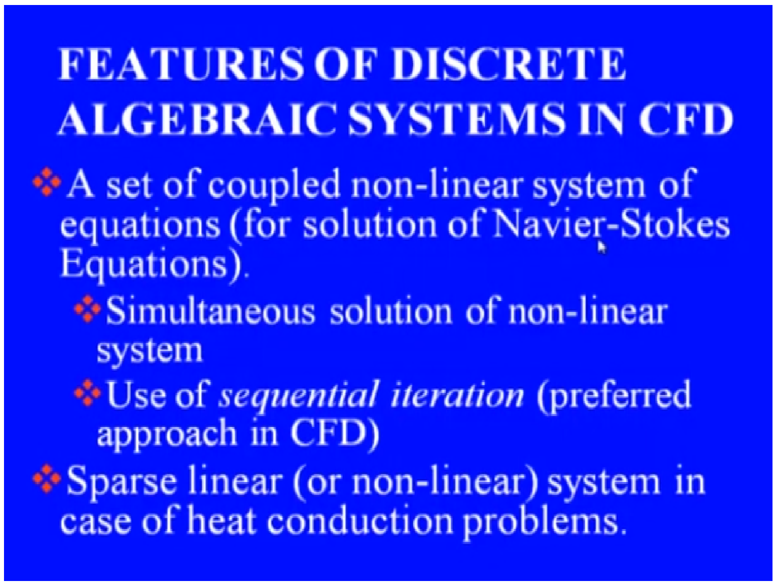


LECTURE OUTLINE

- Features of Discrete Algebraic Systems in CFD
- Methods for Non-linear Systems
  - Sequential (Picard) Iteration
  - Newton-Raphson Method

And there after we would focus on very briefly we will touch upon the methods for non-linear systems. In particular, we will derive or discuss what we call sequential iteration. It is also referred to as Picard iteration and Newton-Raphson method. Okay, now let us come to, let us have a look at our basic features of discrete algebraic systems which we get in CFD analysis.

(Refer Slide Time: 03:34)


FEATURES OF DISCRETE ALGEBRAIC SYSTEMS IN CFD

- A set of coupled non-linear system of equations (for solution of Navier-Stokes Equations).
  - Simultaneous solution of non-linear system
  - Use of *sequential iteration* (preferred approach in CFD)
- Sparse linear (or non-linear) system in case of heat conduction problems.

So, remember our Navier-Stokes equations, there themselves represent a set of free coupled non-linear partial differential equations and they have to be solved in conjunction with the continuity equation and for compressible flows energy equations. So, for compressible flow say for instance you will be dealing with the system of 5 coupled non-linear equations.

And their discretization using finite difference, finite element or finite volume techniques will give us a set of coupled non-linear systems equations. Which must be solved to obtain the numerical solution for velocity filled prism and density filled. Now they are 2 approaches which can be utilized that we can combine all the non-linear systems which we have obtained from discretization of let us say continuity equation, 3 momentum equations and our energy equations.

And there by obtain a big system of non-linear equations and solve this system simultaneously to get our density, velocity filled and temperature filled. But his approach is used rather uncommonly in CFD. What is more common is what is referred to as sequential iteration that is we would start off with that guest value of the velocity pressure and temperature field and solve each equation in turn.

Let us say, x momentum equation, y momentum equation, z momentum equation iteratively and then update the remaining components. So, we will solve each equation into one by one that is why we call it a sequential iteration and this is the preferred approach in CFD. The reasons would become clear when we look today at the solution makes it for non-linear system that favorites we normally preferred the sequential iteration approach in CFD applications.

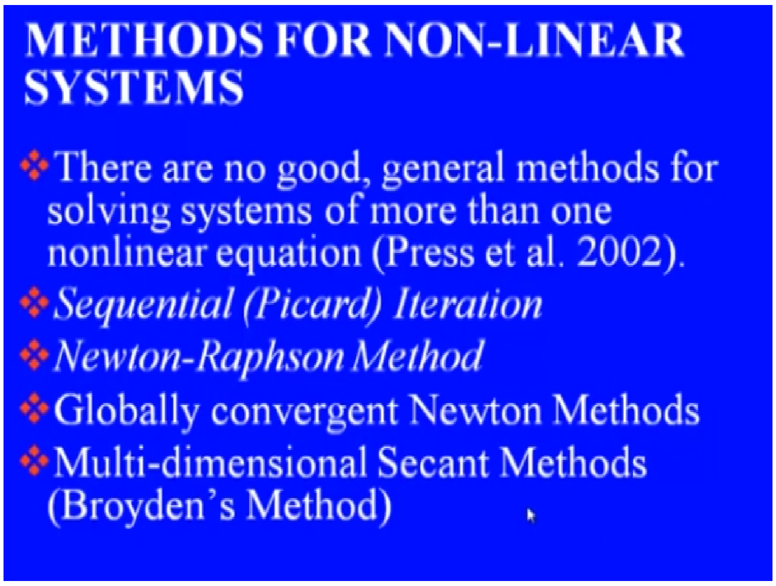
And let us also be very rough that the systems equations which we get they are usually sparse. This sparsity comes in because of limited number of connection which each grid point has whether it is neighboring nodes. So, if we have for instance if dealing with simpler problem of heat conduction, if the problem is linear then we get a sparse linear system. If it is non-linear then again this sparsity pattern is anywhere there.

So, whatever solution techniques which we utilized it suites the port or it should make use of this sparsity feature. Both in the storage of entrance of the matrix and of linear system or linearize systems as well as in the solution process. So, our choice must be based on these the basic nature of discrete algebraic systems which we have just discussed. Now let us have a very brief look at method for non-linear systems.

We are going to begin with a slightly shocking comment from Press et al in their book on Numerical Recipes and the comment is observation size that they are no good general methods for solving system of more than one non-linear equation. If we had a single non-linear equation there are general methods which can guarantee a converse solution or a set of rules which we can definitely obtain for that particular equation.

But the moment we are dealing with more than one equation and that is what we will have an CFD analysis.

(Refer Slide Time: 07:18)



METHODS FOR NON-LINEAR SYSTEMS

- There are no good, general methods for solving systems of more than one nonlinear equation (Press et al. 2002).
- *Sequential (Picard) Iteration*
- *Newton-Raphson Method*
- Globally convergent Newton Methods
- Multi-dimensional Secant Methods (Broyden's Method)

We will have not just one, we will have system of other order of the number of unknowns would be in millions or billions. So, we will have a system of millions or billions of non-linear equations which are to be solved. And this observation size that there is no general method available which can guarantee a converse solution for us. But nevertheless we have got to live with it. That there are certain reddening features of the physics which helps us to solve our Navier-Stokes equations in CFD.

So, we do not have to worry too much about this comment. But at the same time we should not spend too much of time on searching for a general method which is or a black box which we can use for solving all our problems in CFD. So, please be aware of it. Let us not waste our energy on it. Now, let us have a list of few popular techniques for non-linear systems.

We are also used for single non-linear equation as well. We will discuss today what we call sequential or Picard Iteration that is why the first or the most basic iteration scheme suggested for the solution of non-linear problems. So, we will have a look at the form of the equation for a form this method for a single equation and then we are going to generalize it for a systems equations we would say it is few variants as well.

Then Newton-Raphson method celebrated method which is attributed to Newton which is based on Volterra series expansion. So, we will also have a look at this particular method we are going to derive that required formulae and we will write its algorithm and Pseudo-code which you would be asked to translate into an appropriate program using C, C++ or Fortran whichever may the language of your choice be.

Okay, in the next category we have bought what we call Newton type methods. These are popularly refocus globally convergent method. There is a specific reason why we call them as these methods are globally convergent. Sequential iterations also called globally converging method because we can start off from any arbitrary initial guess and theoretically this measure say that we should be able to obtain our convergent solution.

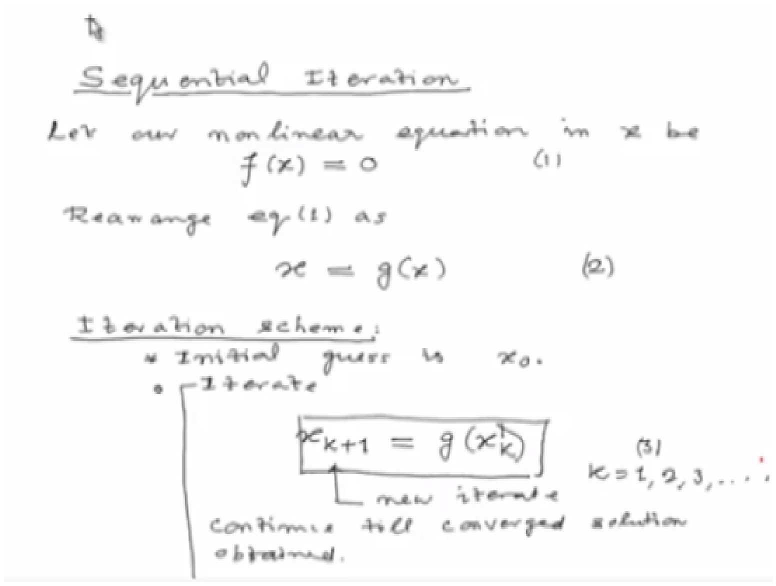
Newton-Raphson methods on the other hand they do not give any such guarantee. Newton-Raphson method converts it very fast but only if our initial guess is close to the final solution. So that is why, at the time it is made to combine the features of both sequential iteration type of globally convergent schemes and Newton-Raphson methods to what we call globally convergent Newton methods.

One popular scheme for solution of single non-linear equations what is called Secant method. This can also be generalized for multi-dimensional problems that is for a system of non-linear equations, okay. So, multi-dimensional Secant method one particular methods what is called Broyden's method. So we are not going to have a detail look at any of these methods in today's lecture.

If you are very keen on looking at these methods you can find all of these methods detailed in the book by Press et al that is Numerical Recipes in C, Fortran or C++, the 3 or 4 such variance of this book. But basic total is Numerical Recipes. You can also find these methods in your numerical analyses, okay. Now, let us first focus on our sequential iteration or peak iteration.

It guarantees converse solution starting from an arbitrary initial guess but the rate of convergent is often very slow. Or let us derive this scheme, how does it work Sequential iteration?

(Refer Slide Time: 11:14)



Sequential Iteration

Let our non linear equation in x be

$$f(x) = 0 \quad (1)$$

Rearrange eq (1) as

$$x = g(x) \quad (2)$$

Iteration scheme:

* Initial guess is x_0.
* Iterate

$$\boxed{x_{k+1} = g(x_k)} \quad (3) \quad k = 1, 2, 3, \ldots$$

new iterate

Continue till converged solution obtained.

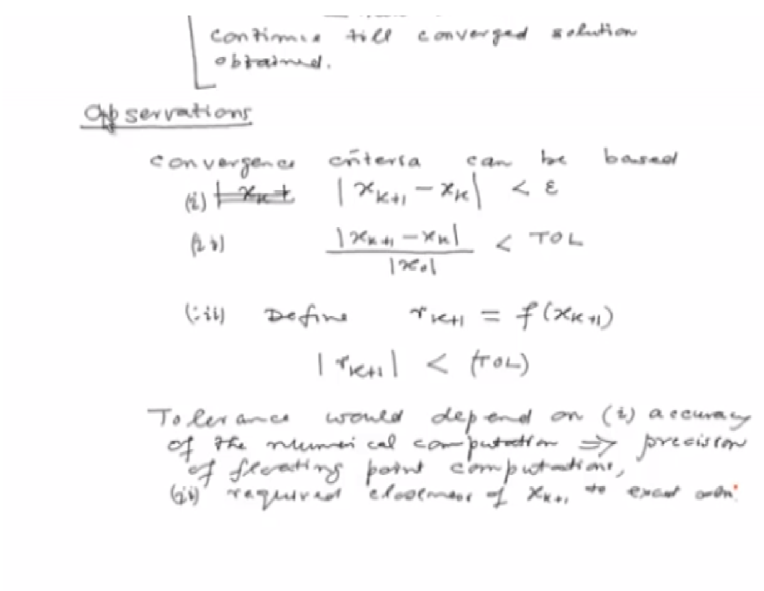
So first have a look at this scheme for a single iteration. So, let have a non-linear equation in x be f(x) = 0. We should try and rearrange it, so rearrange, it is called this equation as 1. So, rearrange equation 1 as x = g(x). Okay, there are various possible ways in which this arrangement could be obtained. If effects is too complicated the simplest thing would be lets add x in both the sides.

So, we get x = x+fx and that x+fx becomes our function g x. Now iterative process or iteration scheme let us say our initial guess is x subscript 0. So, starting from this initial guess we would now iterate that is we are going to now compute a sequence of numbers x of k+1. x of k+1 this would be obtained simply by computing the function g(x) at k. So, xk+1 this we call as our new iterate.

Continue this convergent process till, so continue till converged solution obtained. Normally for this convergence you know this is one big loop if you write a program that is how it would look like k = 1, 2, 3 and so on. We will continue till we achieve the convergence. Normally, we will look at the difference between 2 and we will say convergence has been achieved if the difference between 2 iterates is less than a small specified value.

So, you can choose your convergence criteria it could be based on absolute value or it could be based on a relative magnitude. So, that is your choice.

(Refer Slide Time: 15:26)



continue till converged solution obtained.

<u>Observations</u>

convergence criteria can be based

(i) ~~x_{k+}~~ $|x_{k+1} - x_k| < \varepsilon$

(ii) $\dfrac{|x_{k+1} - x_k|}{|x_0|} < TOL$

(iii) Define $r_{k+1} = f(x_{k+1})$

$|r_{k+1}| < (TOL)$

Tolerance would depend on (i) accuracy of the numerical computation $\Rightarrow$ precision of floating point computations,
(ii) required closeness of x_{k+1} to exact soln.

So, there are some observations. That is query for these ones here. So convergence criteria can be based on that magnitude of x of k+1 – xk being < epsilon. This is one possible criteria and the second possible thing would be that we instead take a relative basis at its xk+1 – xk.

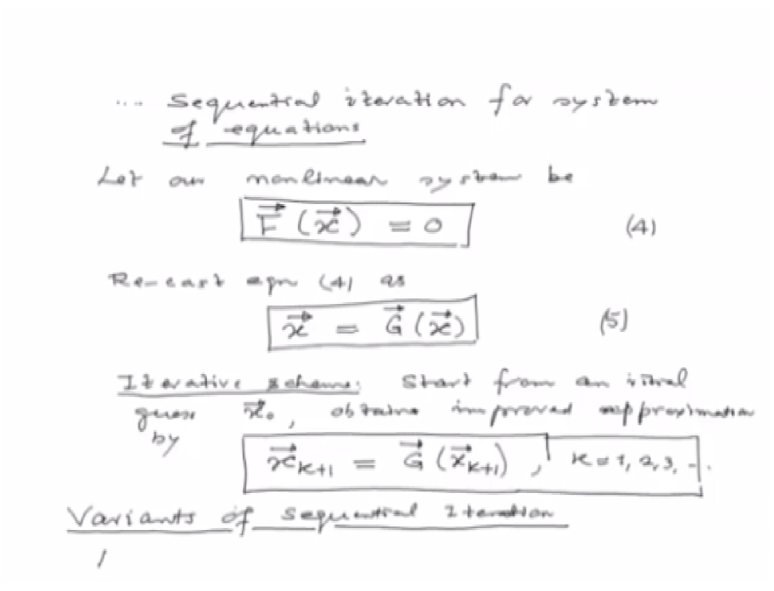
It is absolute value divide by the absolute value of our initial guess. This is less than some has specified tolerance. So, you can use either of these. Some people also take a third criteria which is based on what we call the residual. So, third criteria could be defined that rk+1 = f (x) k+1. So, if xk+1 were a (()) (17:05) solution of system equations 1 then of course this residual would become 0.

So magnitude of this residual rk+1 that also gives us a major of the convergences. So, this is less than similar the tolerance a very small number. We can say that the solution has converged. Now the magnitude of the tolerance that would depend on our application as well as this tolerances would depend on first thing is accuracy of the numerical computations and this is govern by what we call the precision of floating point computations.

And the second one would be more of practical in nature that how close we want of converse solution to be to our exact solution. So, required closeness of xk+1 to exact solution. If there would be certain applications where we may not like to have very accurate solution even in an approximate one will do our job. So, in that case we do not have to waste too much of our time.

So, we can have what we call a broader tolerance or a looser tolerance and if you want a very accurate solution then of course we to pit for what we call a very tight tolerance. Now but remember the tolerance is intrinsically linked to the precision which we are going to use for floating point. If we have used single precision and if you prescribed a tolerance of 10 to power let us say-10 that is meaningless.

So, if you want a tolerance of the order of 10 to power-10 we have to form or computations in double precision arithmetic. And in fact I would recommend that if you are writing your own code whether it is in Fortran or C, C++ please always choose double precision arithmetic for our computations. On this for a single equation we want to generalize it to system equations.
(Refer Slide Time: 20:00)



... Sequential iteration for system of equations

Let our nonlinear system be

$$\boxed{\vec{F}(\vec{x}) = 0} \qquad (4)$$

Re-cast eqn (4) as

$$\boxed{\vec{x} = \vec{G}(\vec{x})} \qquad (5)$$

Iterative scheme: Start from an initial guess $\vec{x}_0$, obtain improved approximation by

$$\boxed{\vec{x}_{k+1} = \vec{G}(\vec{x}_{k+1}), \quad k = 1, 2, 3, \ldots}$$

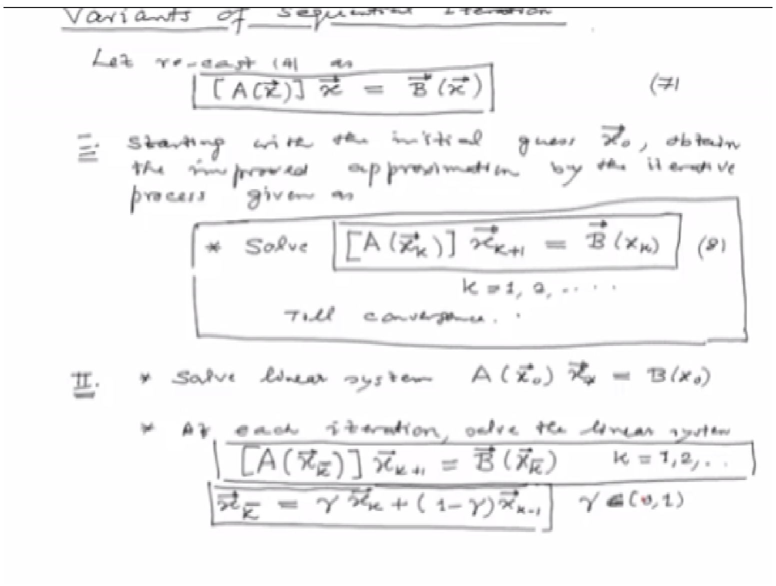
Variants of Sequential Iteration

/

Sequential iteration for system of equations. Now this generalize would again we will follow the same step. So, which ever our systems are so let our non-linear system be let us use capital letter to reinforce the fact that you now dealing with not a single equation we are dealing with the system of equations and you expect to simple. So, x is not a single unknown, it is a number of unknowns combine together. So, it is a vector, so Fx = 0.

This is our non-linear system, let us call this equation 4. Now similar to the way we did it for a single system. Now let us rearrange it or rather what we call recast equation 4 as x = g (x). So, now the algorithm texts fairly simple form, very similar to the way we had written earlier. So, our iterative algorithm start from an initial guess x 0 obtain what we called improved approximations by xk+1 g (xk+1), k = 1, 2, 3 and so on.

(Refer Slide Time: 23:31)



Variants of sequential iteration

Let re-cast (4) as

$$\boxed{[A(\vec{x})]\,\vec{x} = \vec{B}(\vec{x})} \qquad (7)$$

I. Starting with the initial guess $\vec{x}_0$, obtain the improved approximation by the iterative process given as

* Solve $\boxed{[A(\vec{x}_k)]\,\vec{x}_{k+1} = \vec{B}(x_k)}$ (8)

$k = 1, 2, \ldots$

Till convergence.

II. * Solve linear system $A(\vec{x}_0)\,\vec{x}_* = B(x_0)$

* At each iteration, solve the linear system

$$\boxed{[A(\vec{x}_{\bar{k}})]\,\vec{x}_{k+1} = \vec{B}(\vec{x}_{\bar{k}}) \qquad k = 1, 2, \ldots}$$

$$\boxed{\vec{x}_{\bar{k}} = \gamma\,\vec{x}_k + (1-\gamma)\,\vec{x}_{k-1}} \qquad \gamma \in (0,1)$$

So, it is so simple iterative scheme. There are few variants for system of equations. So, variants of sequential iteration what we can do is instead of correcting the system equations such as x = gx. So, let us write or let us recast 4 as A x, we are now A. A would be typically a matrix. So, let me put the [ ] to denote that fact A x into x that is = to B x. Now obtaining this form in fact is a lot easier when we are working with a discretization scheme.

Because thus way we had obtained our generic equation that remember our equation was obtained in terms of these coefficients A p, A w, A e and so on finite discretization. So, this is a natural form. This is a more natural form of non-linear equations which we would get while discretizing our PDE, our Partial Differential Equation using finite difference or finite volume or finite element method.

So, this could be our starting point. And now what we were do is our iteration scheme would be as a starting with the initial guess x 0 obtain the improved approximations by the iteration, iterative process given as solve A (xk) that it forms matrix A using our earlier approximation xk+1 B (xk). So, this is a linear system. So, solve this linear system, 8.

Where K can arrange from let us say 1, 2 so on till convergence and once again for convergence we can check for in 2 ways. One way would be based on the magnitude of the residual vector fx. So, the convergence criteria could be based on the residual of fx vector that is let us say magnitude of that vector or the difference between 2 approximation xk +1 – xk get the magnitude of 9th vector.

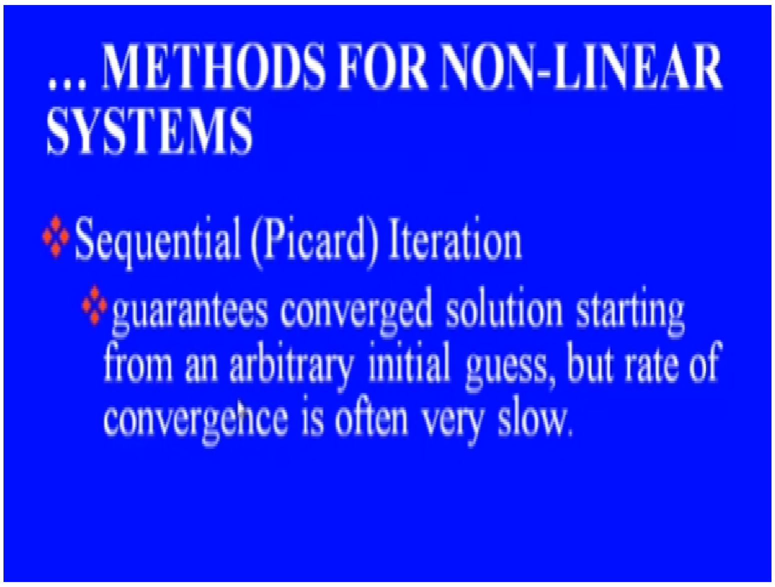
So, this is our simple iterative scheme which is a slight modification of our usual sequential iteration. There is yet another way possible we can make slight modification to this scheme. And that modification is based on the choice of xk+1 in the evaluation of terms of A that is matrix A and the right hand side vector V. So, let us call it as variant 1 or variant 2 would be.

That we would solve linear system A(x0) for a vector less cause it as x star = to B(x0). Okay so the, instead of starting from x 0 now let us obtain a bit more accurate approximation for our initial guess. And then at each iteration solve the linear system A at x k bar now this x k bar is different from a previous approximation and we will see how do we refine this into x of k+1 = B of x k bar.

K = 1, 2 and so on where this x k bar this is defined as gamma times x k+1 – gamma times x of k – 1. I have this parameter gamma which we have introduced to provide us a weighting. Gamma is a number between 0 and 1. Okay, so this x k bar is not = x k bar but in fact it also contains some effect of the iterate previous to that. So, hopefully this gives us a more stable iteration process.

So, this sequential iteration scheme they look, they might look very simple but they work fairly well. Only thing is they require large number of iterations.

(Refer Slide Time: 31:04)

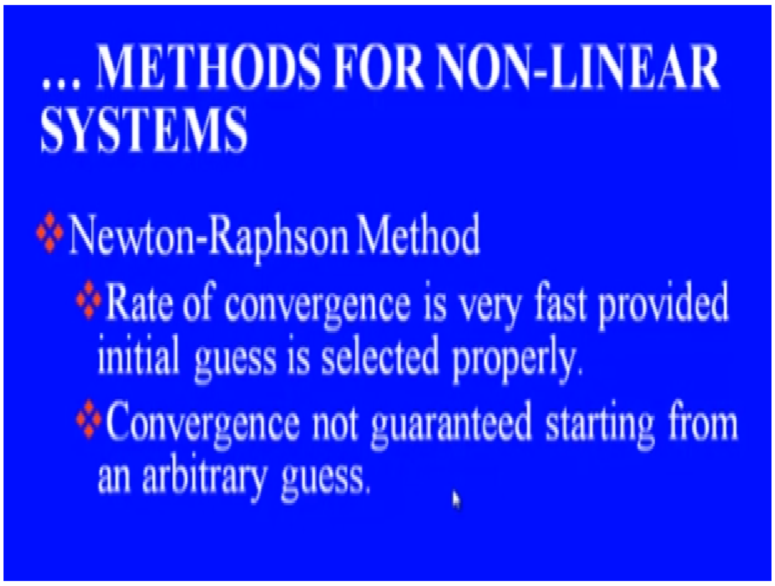


... METHODS FOR NON-LINEAR SYSTEMS

- Sequential (Picard) Iteration
  - guarantees converged solution starting from an arbitrary initial guess, but rate of convergence is often very slow.

Okay and this guaranty like our sequential iteration process guarantees converged solution starting from an arbitrary initial guess. But the rate of convergences is often slow. If you want to get better convergence behavior, we have to go for what we call Newton-Raphson method.

(Refer Slide Time: 32:22)

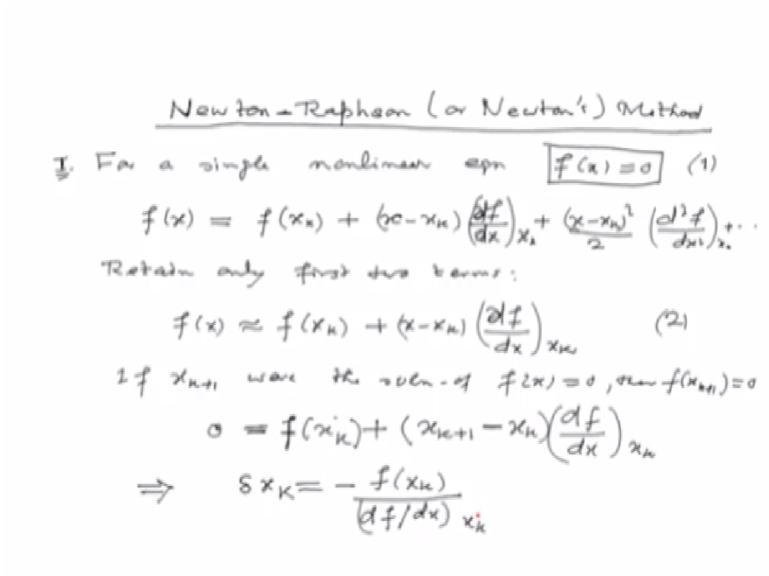


... METHODS FOR NON-LINEAR SYSTEMS

- Newton-Raphson Method
  - Rate of convergence is very fast provided initial guess is selected properly.
  - Convergence not guaranteed starting from an arbitrary guess.

Now rate of convergence is very fast in Newton-Raphson method provided the initial guess is selected properly and please be aware of its next feature that convergence is not guaranteed starting from an arbitrary guess may not get any solution whatsoever with Newton-Raphson method. Now let us have a look at details of what is this Newton-Raphson method or some people simply call it as Newton's method. So, let us see how does this method look like.

(Refer Slide Time: 33:08)



Newton-Raphson (or Newton's) Method

I. For a single nonlinear eqn $\boxed{f(x)=0}$ (1)

$$f(x) = f(x_k) + (x-x_k)\left(\frac{df}{dx}\right)_{x_k} + \frac{(x-x_k)^2}{2}\left(\frac{d^2f}{dx^2}\right)_{x_k} + \cdots$$

Retain only first two terms:

$$f(x) \approx f(x_k) + (x-x_k)\left(\frac{df}{dx}\right)_{x_k} \qquad (2)$$

If x_{k+1} were the soln. of $f(x)=0$, then $f(x_{k+1})=0$

$$0 = f(x_k) + (x_{k+1}-x_k)\left(\frac{df}{dx}\right)_{x_k}$$

$$\Rightarrow \qquad \delta x_k = -\frac{f(x_k)}{(df/dx)_{x_k}}$$

Once again first let us have a look at the form of this method for a single equation and then we would generalize it for a system of equations. So, for a single non-linear equation F x = 0, you will first have a look at how does this scheme look like. Now this F x is a function, so it can be expanded in Taylor series, around suppose we have got iterate called x k. So, around x k let us expand it, so f (x k)+x–x k delta f over delta x.

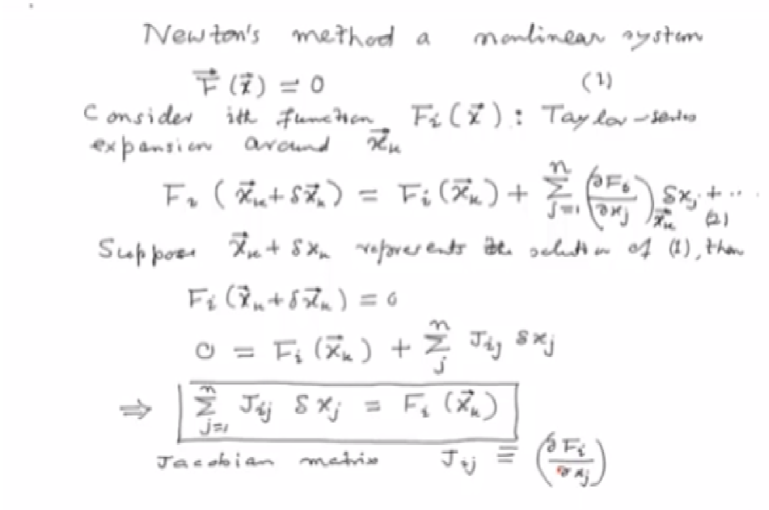
Let us put d f by d x we dealing with a single variable at x k+x–x k whole square divide by 2 d2 f over dx square at x k+so on. So, let us retain only the first term, sorry retain the terms containing the first derivative only ignore the higher order derivatives. So, retain only first 2 terms then what we will get f of x can approximate it as f of x k+x–x k times d f by dx at x k.

Now we are looking for a new iterate or an improved solution. So, let so, if x k+1 were the solution of fx = 0 then of course f of x+1 is also = 0. So, let us substitute that in equation 2. So we get in left turn side you will get 0. f (x k)+x of k+1–xk df over dx at xk. Now let us call this xk+1–xk by x a shorter simple delta xk. So, delta xk = -f xk divided by df by dx at xk.

So, now we have got a way to compute the corrections. If you add delta xk to xk we get an improved solution. So, that is what gives us our iterative process for a single equation. Now this process can also be generalized for a system of equations. The only thing is that for the case of a system equations we will have to work with what we called Jacobian matrix.

We will have set of equations which have to be differentiated with respect to different terms to obtain our delta xk vectors and add this delta xk vector to our starting guess to get the solution. Now let us generalized, let us generalized our Newton's scheme for a system of equations.

(Refer Slide Time: 38:35)


Newton's method a nonlinear system

$$\vec{F}(\vec{x}) = 0 \qquad (1)$$

Consider ith function $F_i(\vec{x})$: Taylor-series expansion around $\vec{x}_k$

$$F_i(\vec{x}_k + \delta\vec{x}_k) = F_i(\vec{x}_k) + \sum_{j=1}^{n} \left(\frac{\partial F_i}{\partial x_j}\right)_{\vec{x}_k} \delta x_j + \cdots \qquad (2)$$

Suppose $\vec{x}_k + \delta x_k$ represents the solution of (1), then

$$F_i(\vec{x}_k + \delta\vec{x}_k) = 0$$

$$0 = F_i(\vec{x}_k) + \sum_{j}^{n} J_{ij}\,\delta x_j$$

$$\Rightarrow \boxed{\sum_{j=1}^{n} J_{ij}\,\delta x_j = F_i(\vec{x}_k)}$$

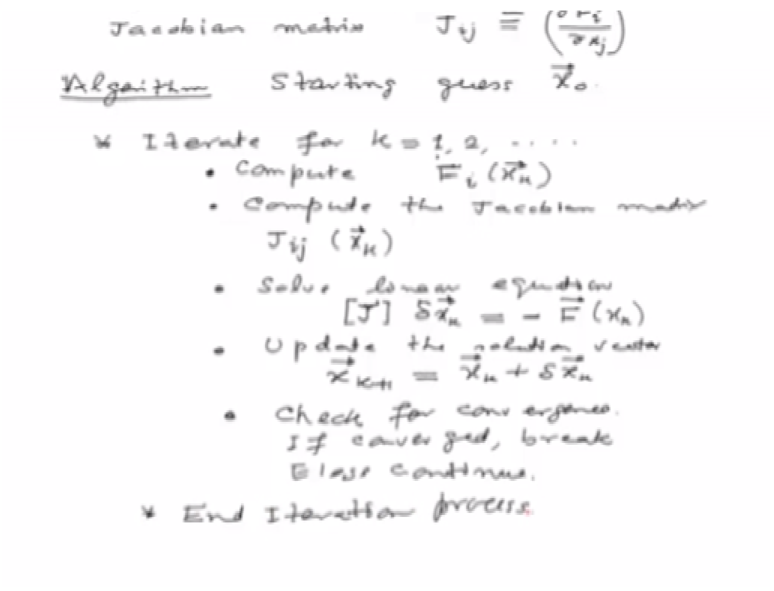
Jacobian matrix $J_{ij} \equiv \left(\frac{\partial F_i}{\partial x_j}\right)$

So it is Newton's method for a non-linear system which we can write as f(x) = 0. Now let us focus on the ith **e**quation, okay and in that case how do we find out the Taylor series expansion. So, consider ith function that is your F i (x). So, now this F i can be thought of as a function of many variables x 1, x 2 and so on. Where x 1, x 2 they are components of vector x. So, for this we have got Taylors series expansion around xk, okay.

Now let us write that, so F i x of k+delta xk now with delta xk represents a small increment in xk vector. So, this can be expressed as the value of this function F i at point x k+sigma j = 1 to n, where n is our number of equations. Delta F i over delta x j times delta x j this derivative is evaluated at point x k+ high order terms. So, we neglect the high order terms and if let us suppose that xk+delta xk represents the solution of let us called our equation as 1.

So, then what we will have F i at xk+delta xk this would be 0. So, left hand side is 0 and the right hand side we have got F i at xk+sigma j = 1 to n. Let us introduce a new term called J ij delta xj, okay. So, we can rewrite this equation or rearrange it as J ij delta xj = F i at xk. Whether this J is popularly, so it is what we called Jacobian matrix. Jacobian matrix J ij this is given by also defined as delta of F i over delta xj.

(Refer Slide Time: 43:49)



Jacobian matrix $J_{ij} \equiv \left(\frac{\partial F_i}{\partial x_j}\right)$

<u>Algorithm</u> Starting guess $\vec{x}_0$

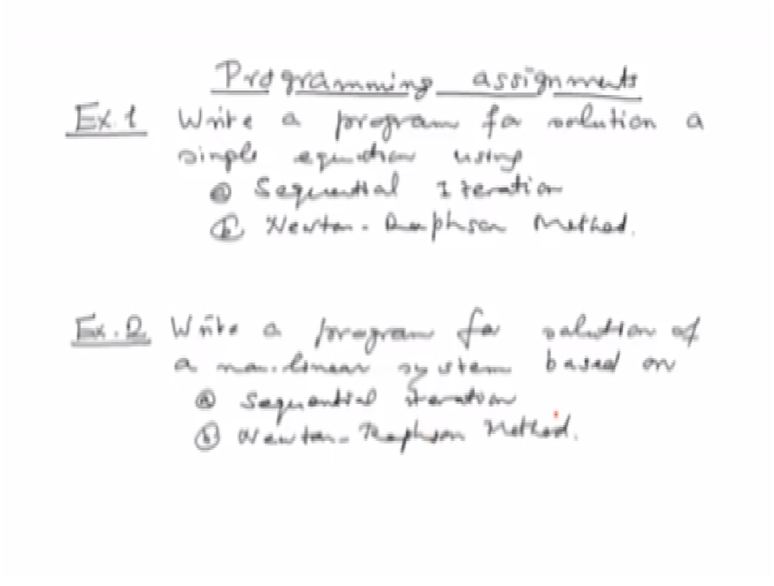
```
* Iterate for $k = 1, 2, \dots$
    • Compute $\vec{F}_i(\vec{x}_k)$
    • Compute the Jacobian matrix
      $J_{ij}(\vec{x}_k)$
    • Solve linear equation
      $[J]\,\delta\vec{x}_k = -\vec{F}(x_k)$
    • Update the solution vector
      $\vec{x}_{k+1} = \vec{x}_k + \delta\vec{x}_k$
    • Check for convergence.
      If converged, break
      Else continue.
* End Iteration process
```

So, now this equation 3 gives us our Newton's algorithm for a solution of non-linear equations. So, we can formularize the algorithm. So, solution algorithm is let see starting guess is xo vector then we would say iterate. And iteration process what we do the first thing is compute for K = 1, 2 and so on till we would achieve the convergence.

So, first we need to compute vector F = F i at xk, compute Jacobian matrix J ij at this point x k and once we have got it solve linear equation has now put our matrix notation form here, J delta xk =-vector F of xk. So, once you solve this linear equation we will get our correction vector Fk. Update this solution vector, so our new or improved iterate x k+1 that is xk+delta xk and check for convergence.

So, if converged break the process, break the iteration process else continue. So this is our body of the iteration process and iteration process. So, in fact what we have just written on screen that is also the Pseudo-code for this Newton-Raphson algorithm. And what I would like you to do is translate it into a code in your favorite programming language and use it to solve sample set of non-linear equations.

(Refer Slide Time: 47:22)



Programming assignments

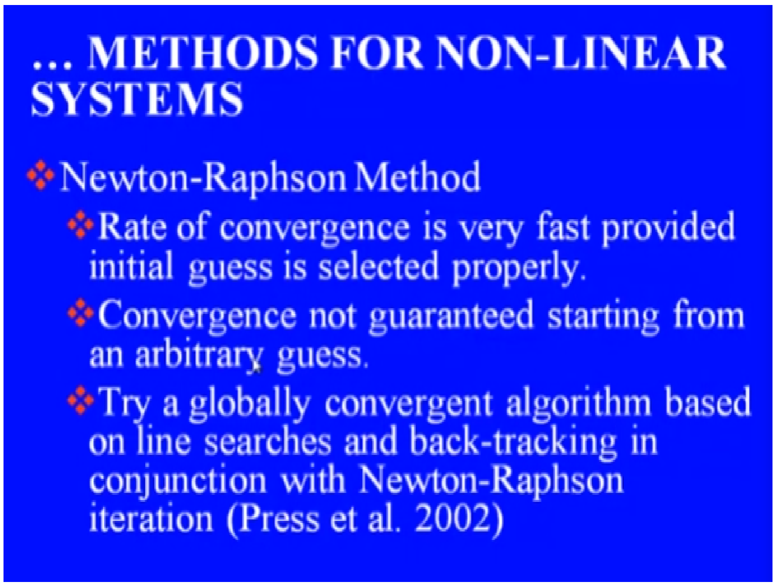
Ex.1 Write a program for solution a single equation using
(a) Sequential Iteration
(b) Newton-Raphson Method.

Ex.2 Write a program for solution of a non-linear system based on
(a) Sequential iteration
(b) Newton-Raphson Method.

So exercise one, they are all coding exercises or programming assignments. Write a program for solution of a single equation using a) sequential iteration and b) Newton-Raphson method. So, once you are comfortable with this program you have tested it for your own example problem then you can expand it and you can use it confidence to write a code for a system of equations.

So, write a program for solution of a non-linear system based on a) sequential iteration the same set basically sequential iteration and b) Newton-Raphson method. For the linear solve which is required as a part of these schemes i would recommend that you search the web get a black box solver, use that as a part of your program.

(Refer Slide Time: 49:34)



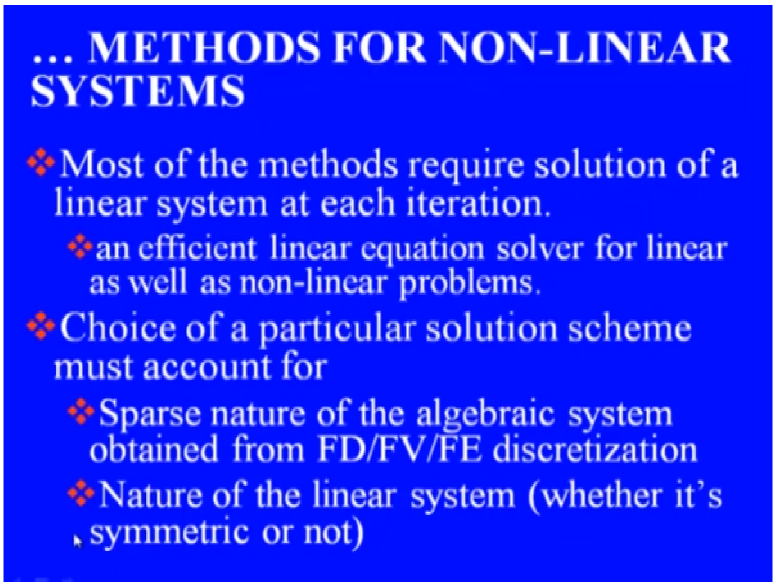
… METHODS FOR NON-LINEAR SYSTEMS

- Newton-Raphson Method
  - Rate of convergence is very fast provided initial guess is selected properly.
  - Convergence not guaranteed starting from an arbitrary guess.
  - Try a globally convergent algorithm based on line searches and back-tracking in conjunction with Newton-Raphson iteration (Press et al. 2002)

So, to conclude our discussions on Newton-Raphson method, we have already noted that rate of convergence of this method is very fast says rate of convergence quadratic. If our initial guess is close to the final solution but remember convergence is not guaranteed is starting from an arbitrary guess. To ameliorate this situation there are various versions, various algorithms has been proposed in literature.

This one popular scheme which we can find in the Numerical Recipes books which is called Globally convergent Newton-Raphson method which is based on line searches and back tracking in conjunction with the Newton-Raphson iterations which you have just seen. So, if you are interested in this algorithm please have a look at the book of Press et al. Now please remember that most of these methods we have already seen in the case of Newton-Raphson method. They require solution of a linear system at each iteration.

(Refer Slide Time: 50:31)



… METHODS FOR NON-LINEAR SYSTEMS

- Most of the methods require solution of a linear system at each iteration.
  - an efficient linear equation solver for linear as well as non-linear problems.
- Choice of a particular solution scheme must account for
  - Sparse nature of the algebraic system obtained from FD/FV/FE discretization
  - Nature of the linear system (whether it's symmetric or not)

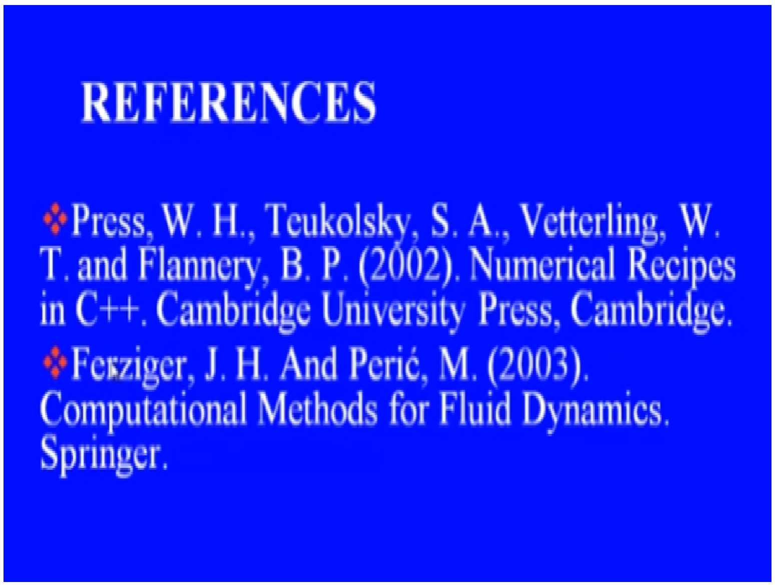
So, we need an efficient linear equation solver irrespective of the nature of a problem whether a problem is linear or non-linear. If it is non-linear we have to use a linear solver in conjunction with Newton-Raphson or sequential iteration scheme. If your problem were linear of course we need a linear solver to solve our problem.

And please remember that our choice of the linear solver must account for the basic nature of the discrete algebraic system that is favors system had a Sparse pattern irrespective of the method which we used. We have already seen the form of algebraic system with finite different discretization with finite volume and finite element. This sparsity pattern is fairly similar; the system is never full it is very very Sparse.

Okay, so this algorithm which we choose for the solution of this system must account for this sparsity**.** It should not require more storage than what is absolutely needed for its Sparse system. There are certain other thing which we can take care of while choosing a particular numerical algorithm for the solution of a linear system, which is based on whether the matrix is general or it is symmetric. If it is symmetric and in addition to that if it is positive definite.

There are set of algorithms available in literature which work much more efficiently than an algorithm for a general linear system. Few of such algorithms we are going to discuss in detail in the next lecture. So, next few lectures we will focus on the linear equations solvers specifically for a Sparse linear systems which arise infinite difference, finite volume or finite elements analysis of CFD problems.

(Refer Slide Time: 52:35)



REFERENCES

- Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2002). Numerical Recipes in C++. Cambridge University Press, Cambridge.
- Ferziger, J. H. And Perić, M. (2003). Computational Methods for Fluid Dynamics. Springer.

So, few references for what we discussed today this Press et all book by press Numerical Recipes it is written in 3-4 versions available. We have got Numerical Recipes in Fortran, numerical recipes in C, numerical recipes in C++ and numerical recipes in Fortran 90. So, depending on your choice of your language you can pick up one of these books or from your library you can get any book.

The basic algorithms are the same, only the coding part would differ at little bit. This published by Cambridge University press. Few algorithms can also you found in this book on Computational Fluid Dynamics by Ferziger and Peric. So, this may be we would stop with this lecture. In the next lecture, we will see few useful algorithms for solution of linear problems.