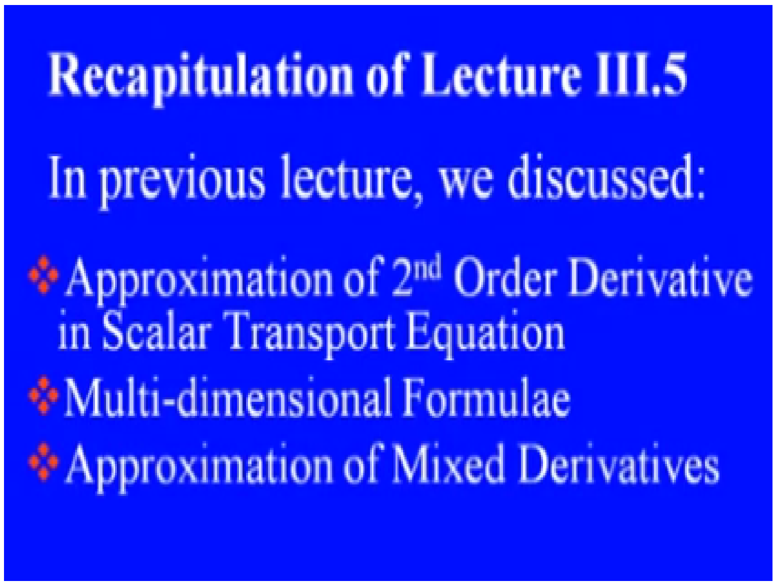
Computational Fluid Dynamics

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

Lecture – 15
Implementation of Boundary Conditions and Finite Difference Algebraic System

(Refer Slide Time: 00:47)



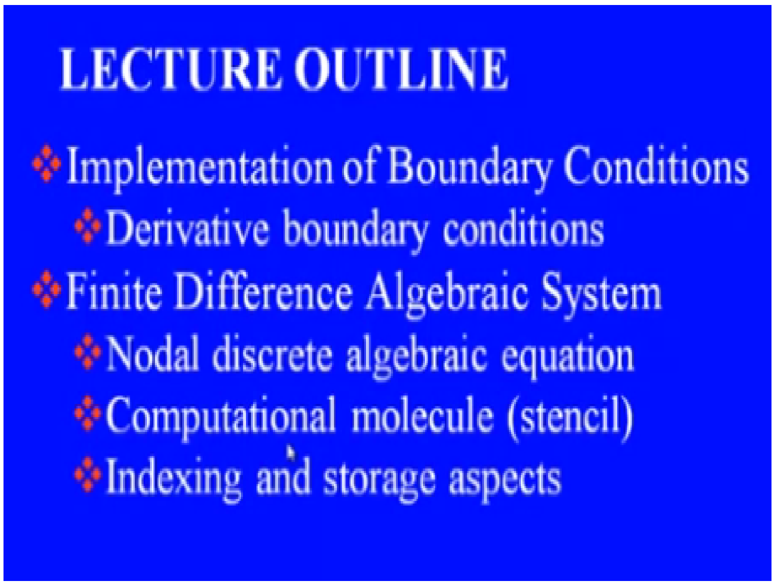
Recapitulation of Lecture III.5

In previous lecture, we discussed:

- Approximation of 2nd Order Derivative in Scalar Transport Equation
- Multi-dimensional Formulae
- Approximation of Mixed Derivatives

Welcome back to the sixth lecture in module 3 on finite difference method. In this lecture, we would focus on implementation of boundary conditions and finite difference algebraic system, so just have a recap of what we did in the previous lecture. We discussed approximation of second order derivative in a scalar transport equation; we discussed multidimensional formulae and approximation of mixed derivatives.

(Refer Slide Time: 01:27)



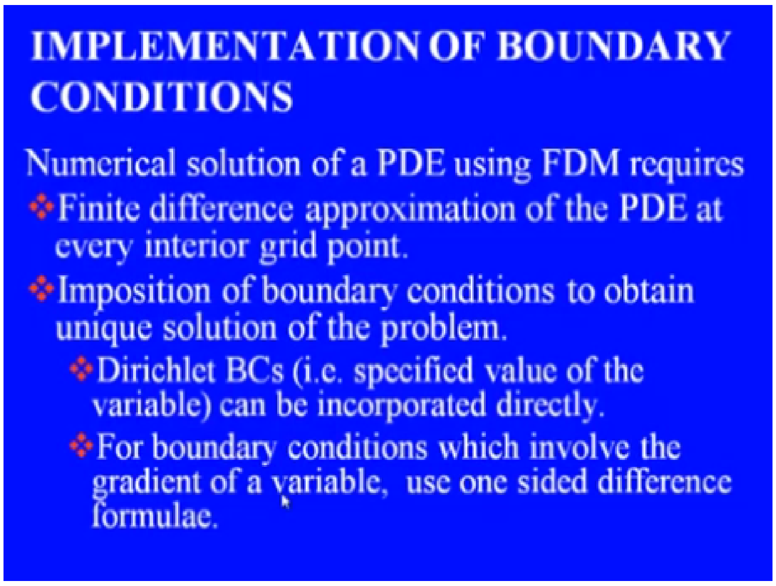
LECTURE OUTLINE

- Implementation of Boundary Conditions
  - Derivative boundary conditions
- Finite Difference Algebraic System
  - Nodal discrete algebraic equation
  - Computational molecule (stencil)
  - Indexing and storage aspects

In this lecture, we will focus on how do we implement boundary conditions? particularly the derivative boundary conditions and we will have a look at the system of algebraic equation which we obtain from finite difference discretization. This is the outline of this lecture. We will first have a look at implementation of boundary conditions; we will derive required higher order formulae for derivative boundary conditions.

And then we will have a look at differential algebraic system or rather find a difference system which we get, we will look at nodal discrete algebraic equations and we will discuss in detail that what we call computational molecule or computational stencil in a finite differences on a structure grid, then we will briefly discuss indexing in storage aspects for discrete algebraic system which we get from finite a difference discretization.

(Refer Slide Time: 02:22)



IMPLEMENTATION OF BOUNDARY CONDITIONS

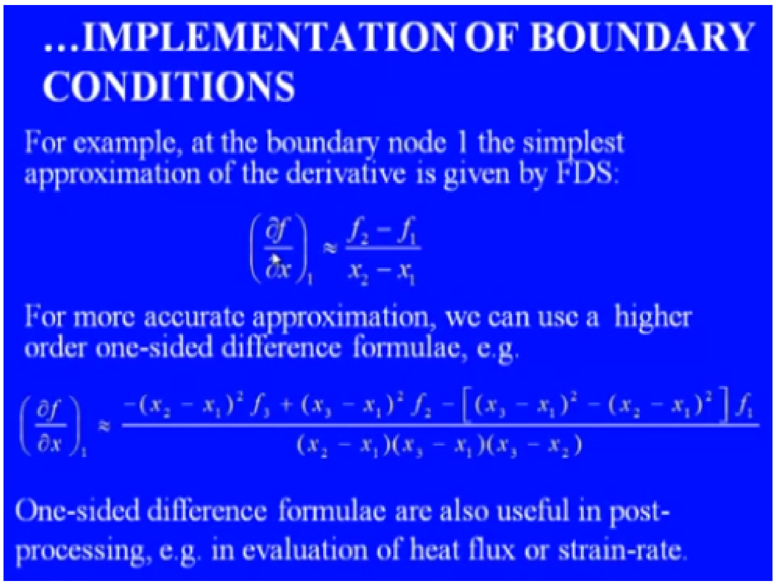
Numerical solution of a PDE using FDM requires
- Finite difference approximation of the PDE at every interior grid point.
- Imposition of boundary conditions to obtain unique solution of the problem.
  - Dirichlet BCs (i.e. specified value of the variable) can be incorporated directly.
  - For boundary conditions which involve the gradient of a variable, use one sided difference formulae.

So, now let us take first topic implementation of boundary conditions. Now, we have our partial differential equation and its numerical space using finite difference requires 2 things; first is finite difference approximation of partial differential equation at every interior grid point, so what we will do? We will replace all the derivatives which appear in our partial differential equation with their corresponding finite difference approximations.

And this would convert our partial differential equation into an algebraic equation at a given interior grid point but this is not sufficient enough we have to impose the boundary conditions which represent the constraints required to obtain a unique solution of a given continuum problem. Now, we can have 2 types of boundary conditions. The first one would call as Dirichlet boundary conditions. Now, Dirichlet BCs, they are basically specified values of the unknown variable.

Now, these can be incorporated directly, we do not have to do any discretization or any difference approximation at those boundary nodes but there are boundary nodes where we will have boundary conditions which involve gradient of a variable, it could be in terms of what we call a Neumann boundary condition where the derivative first order derivative or what we call flux is specified, we could have a mixture of flux plus for derivative plus some multiple of the variable itself what we call robin boundary condition.

(Refer Slide Time: 04:01)



...IMPLEMENTATION OF BOUNDARY CONDITIONS

For example, at the boundary node 1 the simplest approximation of the derivative is given by FDS:

$$\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{f_2 - f_1}{x_2 - x_1}$$

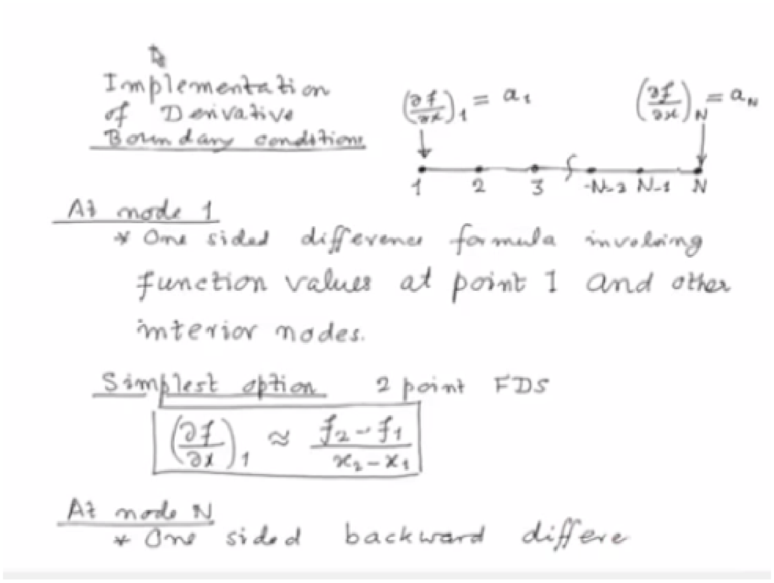
For more accurate approximation, we can use a higher order one-sided difference formulae, e.g.

$$\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{-(x_2 - x_1)^2 f_3 + (x_3 - x_1)^2 f_2 - \left[(x_3 - x_1)^2 - (x_2 - x_1)^2\right] f_1}{(x_2 - x_1)(x_3 - x_1)(x_3 - x_2)}$$

One-sided difference formulae are also useful in post-processing, e.g. in evaluation of heat flux or strain-rate.

So, for such boundary conditions which involved gradient of a variable, we would use one sided a difference formula, to give it as an example, let us have a simple one dimensional problem, where boundary node 1 refers to a left most node and then suppose derivative of the variable or a function f is specified there. Now, if the derivative is specified that has to be discretized or approximated using a difference approximation, so the simplest choice could be the use of forward difference scheme.

(Refer Slide Time: 04:54)



Implementation of Derivative Boundary conditions

$\left(\frac{\partial f}{\partial x}\right)_1 = a_1$ $\left(\frac{\partial f}{\partial x}\right)_N = a_N$

1 2 3 N-2 N-1 N

At node 1

* One sided difference formula involving function values at point 1 and other interior nodes.

Simplest option 2 point FDS

$$\boxed{\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{f_2 - f_1}{x_2 - x_1}}$$

At node N

* One sided backward differe

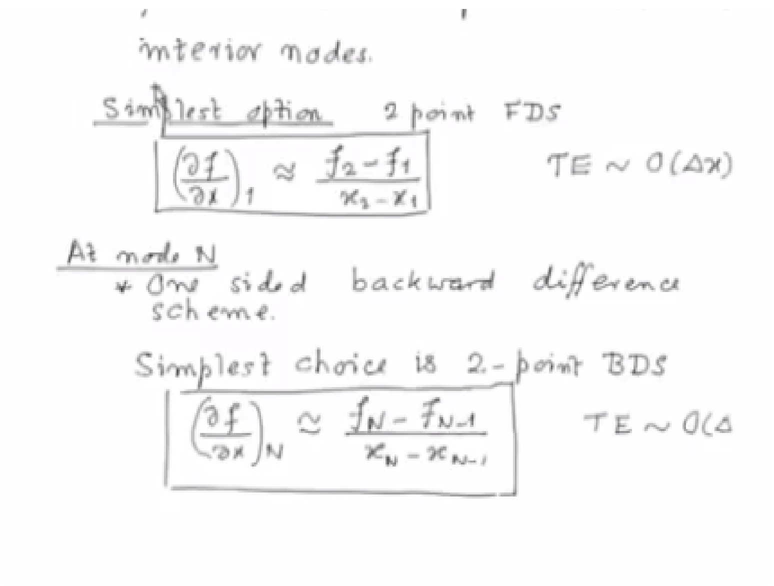
Because we have got now the grid points only to the right of the boundary node 1 okay, so we get a very simple difference approximation del f/del x at node 1 = f2-f1divided by x2 – x1. Let us have a clearer look at it on about, so let us choose, holistic draw our simple one dimensional grid with different node points. Let our nodes be numbered from 1, 2, 3 and so on and the last node is called as capital m.

Now, if the derivative boundary condition is specified here that is del f/del x at point 1 as some known value let us call it a1, at the boundary; similar boundary condition might be specified at node n, suppose we say that del f/del x at node point n that is given by a number, a subscript n. Now, both of these derivatives would require their own finite difference approximation, so implementation of derivative boundary conditions,

So, at node 1, there is no node to the left of 1, so use of central difference approximation is ruled out, so all that we have got we can do we have got to use one sided difference formula; one sided difference formula involving function values; values at point 1 and other interior nodes, okay. The simplest option is of course we use a forward difference scheme; the simplest option is 2 point forward difference scheme which we saw in our slide.

That is to say, this del f over del x at 1, this can be approximated as a difference of f – f1 divided by x2 – x1, okay. Now, if we want to go to the left; so right extreme at node n, forward choices available, we have to look at the node n -1 n -2 and so on we have got to use only values available at these nodes, okay, so here we have got to use this one sided backward difference formula or backward difference scheme.

(Refer Slide Time: 09:24)



interior nodes.

Simplest option 2 point FDS

$$\boxed{\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{f_2 - f_1}{x_2 - x_1}} \qquad TE \sim O(\Delta x)$$

At node N

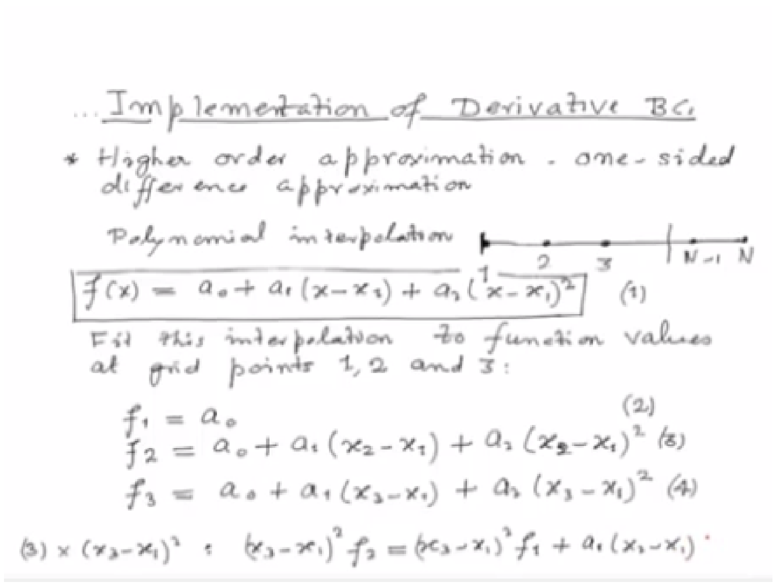
* One sided backward difference scheme.

Simplest choice is 2-point BDS

$$\boxed{\left(\frac{\partial f}{\partial x}\right)_N \simeq \frac{f_N - f_{N-1}}{x_N - x_{N-1}}} \qquad TE \sim O(\Delta$$

So, once again the simplest choice is 2 point BDS, it simply tells us the del f over del x at the rightmost grid point n, this can be approximated in terms of fn-f of n -1 divided by xn-x of n -1. Now, you know most of these formulae which we have just noted down here, this 2 point FDS at the left boundary point 1 or the one at the BDS formula at the rightmost point, in both the cases would truncation error this is of order delta x.

(Refer Slide Time: 10:56)



...Implementation of Derivative BCs

* Higher order approximation - one-sided difference approximation

Polynomial interpolation

$$\boxed{f(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)^2} \quad (1)$$

Fit this interpolation to function values at grid points 1, 2 and 3:

$$f_1 = a_0 \quad (2)$$

$$f_2 = a_0 + a_1(x_2 - x_1) + a_2(x_2 - x_1)^2 \quad (3)$$

$$f_3 = a_0 + a_1(x_3 - x_1) + a_2(x_3 - x_1)^2 \quad (4)$$

$$(3) \times (x_3 - x_1)^2 : \quad (x_3 - x_1)^2 f_2 = (x_3 - x_1)^2 f_1 + a_1(x_2 - x_1)$$

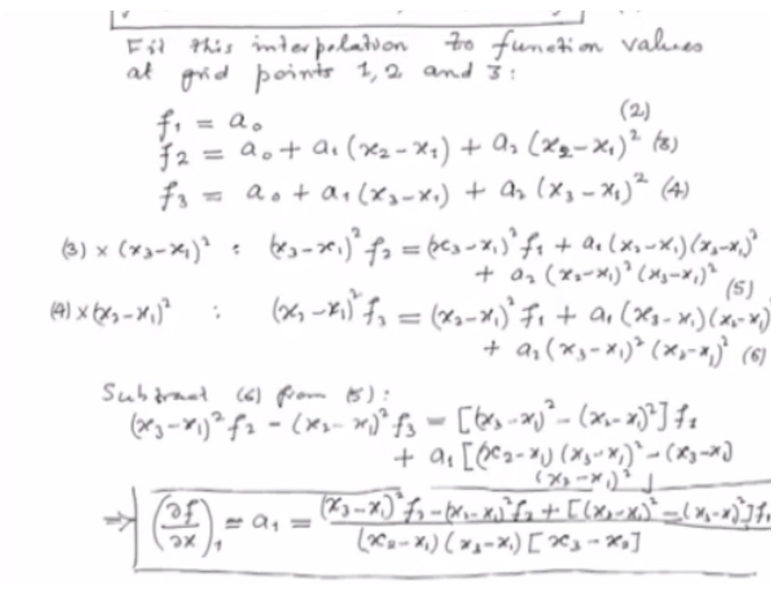
So, that is the accuracy of the approximation is first order, suppose you want to have better accuracy with what we do? We have to involve more number of grid points, so implementation of derivative BCs. Now, let us say we want higher order approximation, once I get one thing this clear that we have got to use a values at only 1 point, so we need one sided difference approximation or one sided formula selects redraw or previous diagram once again.

And let us focus on the leftmost boundary grid point which we have called 1, 1, 2 and 3 and so on. Now, suppose you want to derive or even a higher order formula which would involve values at least one more interior point 1, 2, 3, so now the easiest way to do would be try polynomial fitting; polynomial interpolation for f. So, let us write down fx as a0+a1 times x – x1+a2 times x – x1 squared.

So, we have taken a quadratic polynomial which should hopefully give us a second order difference approximation. Now, fit this interpolation at grid points 1 and 2 interpolation to function values at grid points 1, 2 and 3. So, if you do that, we will get f1 = a0, so we straight we got the value of a0 at point 2 will give us f2 = a0+a1 times x2 – x1+a2 times x3 – x1. So, this number in these equations which have written is called interpolation as equation 1.

A fitted point 1 as equation 2, fitted point 2 as equation 3 and fitted point 3; f3 = a0+a1 times x3 – x1+a2 times x3- x1 square just call this equation 4; a0 is already known from equation 1, a0 = f1. So, now we have to solve for a1 and a2, we are primarily interested in a1 because that is what will give us the derivative value at point 1. So, let us use equation 3 and 4 to eliminate a2, so for this elimination all that we need to do is multiply the coefficients.

(Refer Slide Time: 16:38)



Fit this interpolation to function values at grid points 1, 2 and 3:

$$f_1 = a_0 \quad (2)$$
$$f_2 = a_0 + a_1(x_2 - x_1) + a_2(x_2 - x_1)^2 \quad (3)$$
$$f_3 = a_0 + a_1(x_3 - x_1) + a_2(x_3 - x_1)^2 \quad (4)$$

$(3) \times (x_3 - x_1)^2$: $(x_3 - x_1)^2 f_2 = (x_3 - x_1)^2 f_1 + a_1(x_2 - x_1)(x_3 - x_1)^2 + a_2(x_2 - x_1)^2(x_3 - x_1)^2 \quad (5)$

$(4) \times (x_2 - x_1)^2$: $(x_2 - x_1)^2 f_3 = (x_2 - x_1)^2 f_1 + a_1(x_3 - x_1)(x_2 - x_1)^2 + a_2(x_3 - x_1)^2(x_2 - x_1)^2 \quad (6)$

Subtract (6) from (5):

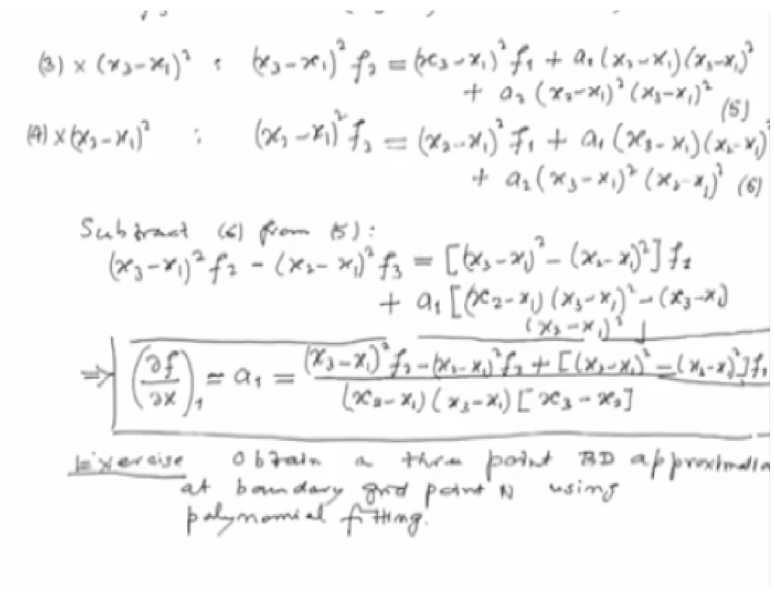
$$(x_3 - x_1)^2 f_2 - (x_2 - x_1)^2 f_3 = [(x_3 - x_1)^2 - (x_2 - x_1)^2] f_1 + a_1[(x_2 - x_1)(x_3 - x_1)^2 - (x_3 - x_1)(x_2 - x_1)^2]$$

$$\Rightarrow \boxed{\left(\frac{\partial f}{\partial x}\right)_1 \simeq a_1 = \frac{(x_3 - x_1)^2 f_2 - (x_2 - x_1)^2 f_3 + [(x_2 - x_1)^2 - (x_3 - x_1)^2] f_1}{(x_2 - x_1)(x_3 - x_1)[x_3 - x_2]}}$$

Cross multiply the coefficient of a2 in these two equations with each other. So, let us try it equation 2 into x3 – x1 whole square, so this will give us x3 – x1 square times f2 = x3 –x1 square times f1, we have substituted for a0 as f1+a1 times x2 – x1 * x3 – x1 square+a2 times x2 – x1 square * x3 – x1 square, let us call this as equation 5. Similarly, let us multiply equation 4 by x2 – x1 square, so this will give us x2 – x1 square times f3 = x2 – x1 square times f1+a1 times x3 – x1 * x2 – x1 square+a2 times x3 – x1 square x2 – x1 square, let us call this equation as 6.

So, now let us subtract 6 from 5, so subtract 6 from 5 and that will eliminate our a2, so we would be left with x3 – x1 square times f2 – x2 – x1 square times f3 = x3 – x1 square – x2 – x1 square times f1+a1 times x2 – x1 * x3 – x1 square – x3 – x1 * x2 – x1 square. Let us rearrange this terms and that gives us the value of the derivative at point 1, del f /del x at point 1 which is equal to a1 and this gives us x3 – x1 square times f2 – x2 – x1 square times f3+x2 – x1 square – x3 – x1 square times f1 divided by x2 – x1, x3 – x1 within brackets x3 – x2.

(Refer Slide Time: 20:55)



$$(3) \times (x_3-x_1)^2 \; : \; (x_3-x_1)^2 f_2 = (x_3-x_1)^2 f_1 + a_1 (x_2-x_1)(x_3-x_1)^2 + a_2 (x_2-x_1)^2 (x_3-x_1)^2 \quad (5)$$

$$(4) \times (x_2-x_1)^2 \; : \; (x_2-x_1)^2 f_3 = (x_2-x_1)^2 f_1 + a_1 (x_3-x_1)(x_2-x_1)^2 + a_2 (x_3-x_1)^2 (x_2-x_1)^2 \quad (6)$$

Subtract (6) from (5):

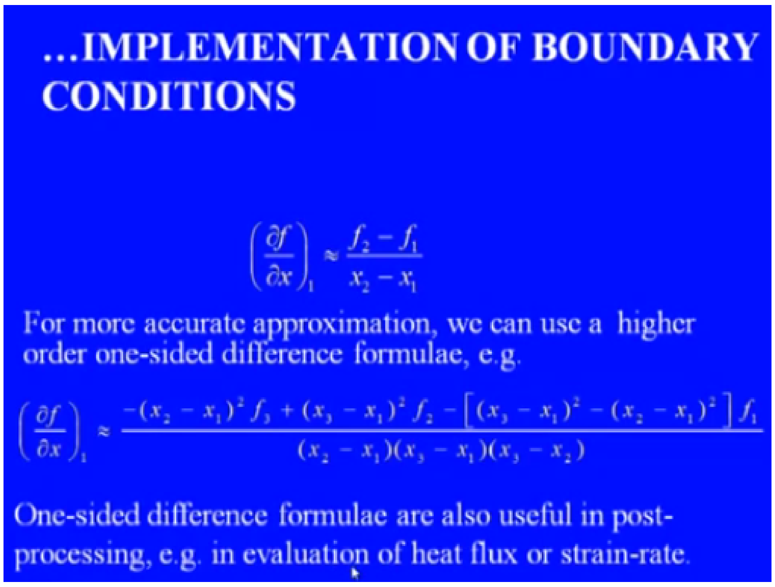
$$(x_3-x_1)^2 f_2 - (x_2-x_1)^2 f_3 = [(x_3-x_1)^2 - (x_2-x_1)^2] f_1 + a_1 [(x_2-x_1)(x_3-x_1)^2 - (x_3-x_1)(x_2-x_1)^2]$$

$$\Rightarrow \boxed{\left(\frac{\partial f}{\partial x}\right)_1 \simeq a_1 = \frac{(x_3-x_1)^2 f_2 - (x_2-x_1)^2 f_3 + [(x_3-x_1)^2 - (x_2-x_1)^2] f_1}{(x_2-x_1)(x_3-x_1)[x_3-x_2]}}$$

Exercise Obtain a three point BD approximation at boundary grid point N using polynomial fitting.

So, now we have got a formula and we have already seen from general results that this should hopefully be more accurate, so this accuracy is very closer to second order. Similar formula you can obtain at point n, so that i would put as an exercise, obtain a 3 point approximation; 3 point, let us call it at point n to be a backward difference, so 3 point backward difference approximation at boundary point or boundary grid point, capital n using all normal fitting.

(Refer Slide Time: 21:53)



...IMPLEMENTATION OF BOUNDARY CONDITIONS

$$\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{f_2 - f_1}{x_2 - x_1}$$

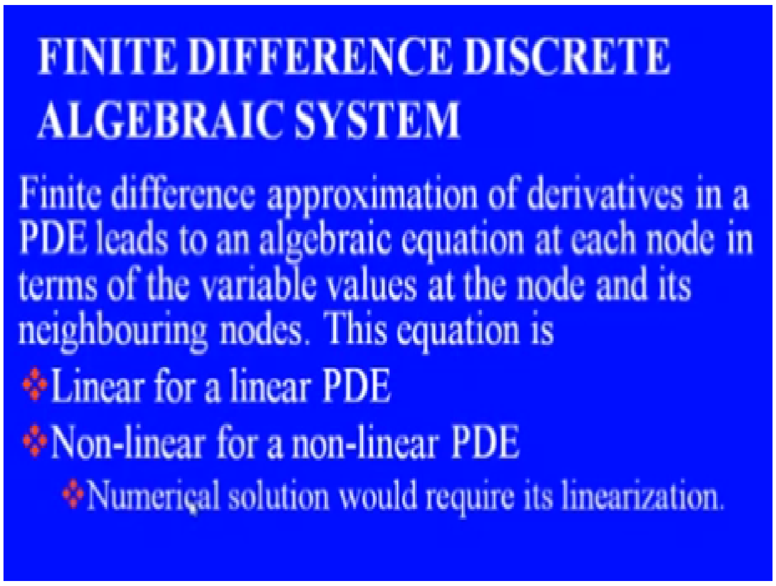
For more accurate approximation, we can use a higher order one-sided difference formulae, e.g.

$$\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{-(x_2-x_1)^2 f_3 + (x_3-x_1)^2 f_2 - \left[(x_3-x_1)^2 - (x_2-x_1)^2\right] f_1}{(x_2-x_1)(x_3-x_1)(x_3-x_2)}$$

One-sided difference formulae are also useful in post-processing, e.g. in evaluation of heat flux or strain-rate.

The equation can be considerably simplified for uniform grid spacing, okay this is just a recap the formula which we have just derived and please note that this one sided difference formula they are also useful apart from the implementation of boundary condition. They can be used in post processing, for instance we want to evaluate the heat flux or let us say stresses at or strain rates at the boundary points.

(Refer Slide Time: 22:28)



FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

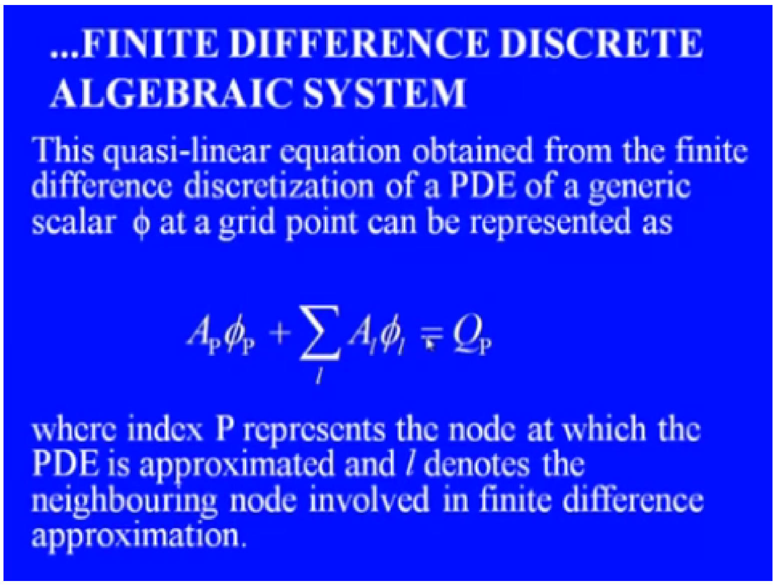
Finite difference approximation of derivatives in a PDE leads to an algebraic equation at each node in terms of the variable values at the node and its neighbouring nodes. This equation is

- Linear for a linear PDE
- Non-linear for a non-linear PDE
  - Numerical solution would require its linearization.

Now suppose, we have done our discretization, we have implemented our boundary conditions, now let us see; let us collect of finite difference system. So, finite difference approximation of derivatives in a partial differential equation leads to an algebraic equation at each node in terms of variable values at the node and its neighbouring values. So, we can write this equation by noting its form.

So, this equation would be linear if a partial differential equation were linear, that is all coefficients of derivatives were either fixed numbers and this equation would be a nonlinear equation, if a partial differential equation were nonlinear. Now, even if the equation were nonlinear, we would write it in a linearized form because that is what we would use in the numerical solution of our discrete is a brake system.

(Refer Slide Time: 23:24)



...FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

This quasi-linear equation obtained from the finite difference discretization of a PDE of a generic scalar ϕ at a grid point can be represented as

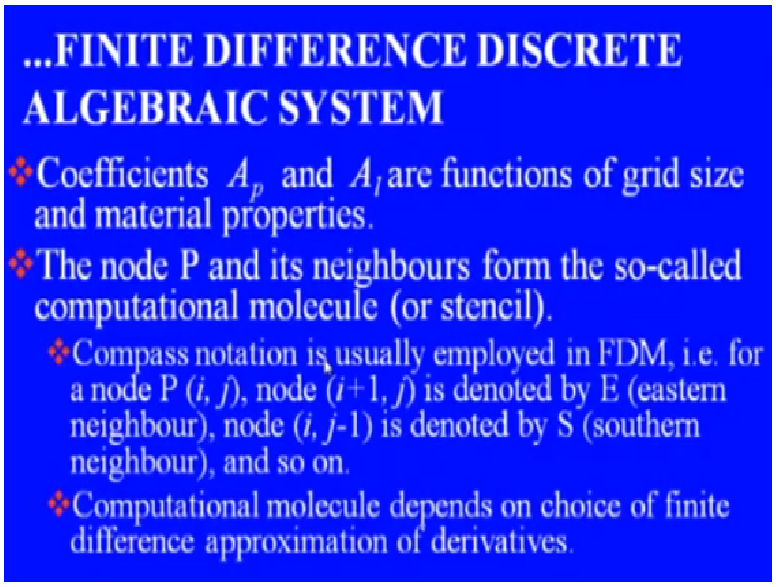
$$A_P\phi_P + \sum_l A_l\phi_l = Q_P$$

where index P represents the node at which the PDE is approximated and l denotes the neighbouring node involved in finite difference approximation.

So, now let us write that linearized form, if our equation were linear, the form would be as such otherwise we will call it as quasi linear equation, so this quasi linear equation obtained from finite difference discretization of a PDE, for let us say a generally scalar or phi could be any velocity component, it could be temperature or if you are solving for pressure approximation equation, it could be pressure.

So, let us call it a generic scalar phi at grid point p or any specific grid point can be represented as Ap * phi p+summation over l Al phi l = Qp. Now, here this p which we have use, p represents a node at which your partial differential equation has been approximated that is; it represents a grid point at which the PDE, all the derivatives in PDE have been replaced by the corresponding finite difference approximations.

(Refer Slide Time: 24:56)



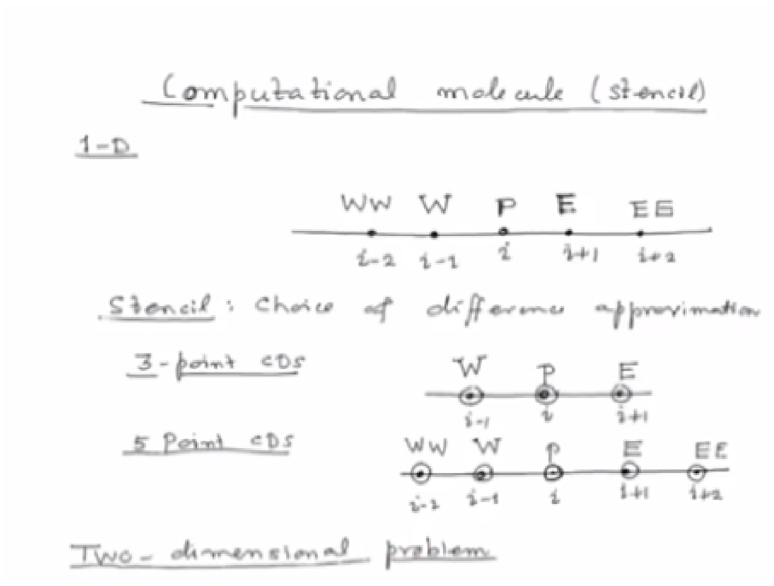
...FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

- Coefficients A_p and A_l are functions of grid size and material properties.
- The node P and its neighbours form the so-called computational molecule (or stencil).
  - Compass notation is usually employed in FDM, i.e. for a node P (i, j), node (i+1, j) is denoted by E (eastern neighbour), node (i, j-1) is denoted by S (southern neighbour), and so on.
  - Computational molecule depends on choice of finite difference approximation of derivatives.

And L denotes the neighbouring nodes which are involved in finite difference approximation of derivative, in QP; we have collected all the terms which are not in terms of phi. These might be result of body forces or source terms and so on, so this is a general linearized system or discrete system which we would obtain for each grid point in our computational domain. Now, note that this coefficient Ap and Al, they are functions of grid sizes as a delta x, delta y and delta z and material properties okay.

Now, in this node p and its neighbouring nodes they form, so called computational molecule or stencil. Now, in finite difference method what we do? We normally imply a structure grid that is to say at each point the grid locally represents an orthogonal grid system, so now in this case we use what we call compass notation okay that is at node pij, node p whose indices are given by ij, its node i+1 at that is a node, which is two words to the right of it.

(Refer Slide Time: 26:02)



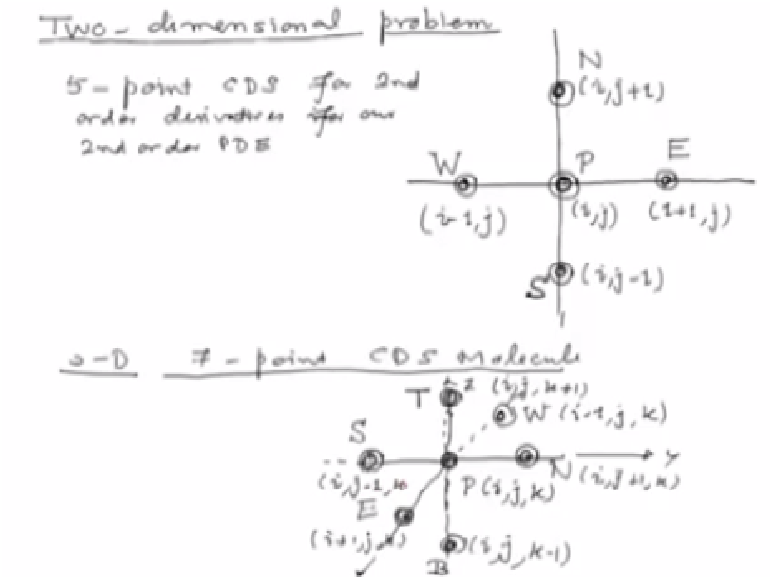
We will denote by eastern node or eastern neighbour node i, j – 1 is denoted by southern neighbour and so on okay and the computational molecule that would depend on the choice of finite difference approximation of the derivative, let us have a bit more look detailed look at this computational molecule. So, let us get back to our board or also we also call it a stencil. So, if we are dealing with one dimensional problem; of course we will have a grid along a line.

And x direction is called i node, i+1, i+2, i-1 and i -2, the central node which is focus of our attention is denoted by capital P that is the present node, P represents the present grid point or node okay the one which is to the right of it, we will say that that corresponds to the eastern node or we will call it as eastern neighbour, i +2, that is to the east of a eastern neighbour, so we will write it as EE i -1, it is to the left.

And we will use the symbol W to denote western neighbour node i -2, that would be called west of west, so WW. So, this is our notation or compressed notation for the grid point. Now, the stencil would depend on the choice of difference approximation, so suppose if you use 3 point CDS, so computational stencil would consist of 3 nodes i, i+1 and i -1, let us use slightly bigger circles to clearly denote the part of a stencil.

So, stencil will consist of node P, E and W. If we use 5 points CDS, then that is the stencil would be broader, this P is i, i +1, i+2, i-1, i-2, this bigger circles to denote distance EE, WW, so choice of stencil will also tell us and we look next at our algebraic system of equations as to how many diagonals; nonzero diagonals are present in our system of equations. So, if you have got a 3 point CDS, it will lead to what we call a tri diagonal matrix.

(Refer Slide Time: 30:29)



If you dealing with 5 points CDS in one dimensions, we will get a penta diagonal matrix. Now, let us take 2 two dimensional case, so for 2 dimensional problems, we will have x as well as y grid lines, so let us draw this x line and this is y line, the intersection point that would be our grid point ij, the grid point to the right of it that is what we will call as eastern neighbour and index i +1, j, the node i -1, j that will be denoted as western neighbour.

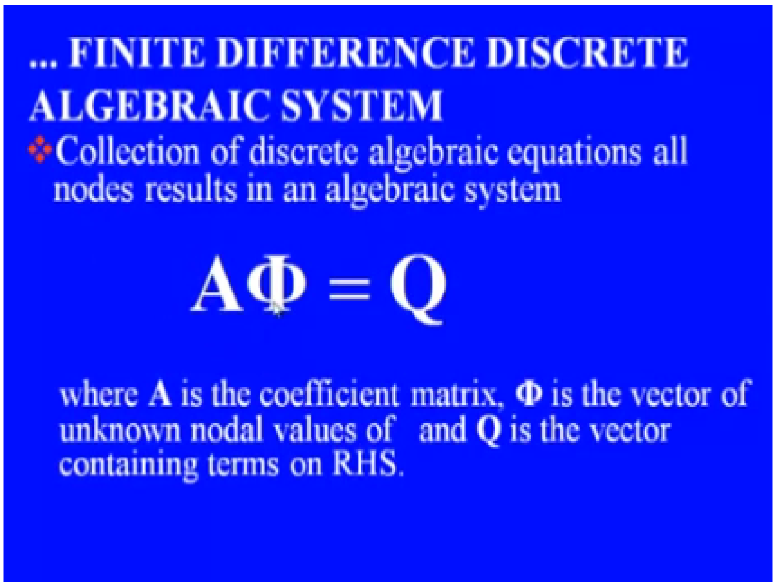
Node towards the upper side following your compass noticing this becomes what we call northern node, so this i j+1, will use a symbol N to denote it. Similarly, this node i j-1 will use symbol S, I will call it as southern node. So, if you have got a 5 point CDS for second order derivatives in our; for our second order PDE, our instance will consist of these points P, E, W, M, S.

In 3 dimensions, we have to add or we would use symbols capital T to denote the top side that is a positive z side and symbol B to denote the negative; neighbour on towards negative z side are what we call the bottom neighbour, so this is a schematic representation in 3D 7 point molecule, so grid point P presented by i, j, k, so for this x direction, y direction and the vertical side we have got our z direction, so to stop that node i j k +1, we represented by symbol capital T.

Similarly, the one towards negative z axis that neighbour is i, j, k – 1; this is denoted by symbol capital B is called bottom neighbour; top neighbour and bottom neighbour. Other two sides, the same thing we have got this term neighbour along positive x axis i+1 j, k western neighbour towards the negative x axis i-1 j, k towards positive y side, we have got what we call the northern neighbour N.

These indices would be i j+1 k and similarly toward negative y side, there is our southern neighbour which will have the index i j-1, so this is the popular compass notation which we use in finite differences. This notation would also be used when we are dealing with structured grids in finite volume, so we will come back to this notation once again when we start off with finite volume method in later modules, okay.

(Refer Slide Time: 35:17)



... FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

- Collection of discrete algebraic equations all nodes results in an algebraic system

$$\mathbf{A\Phi = Q}$$

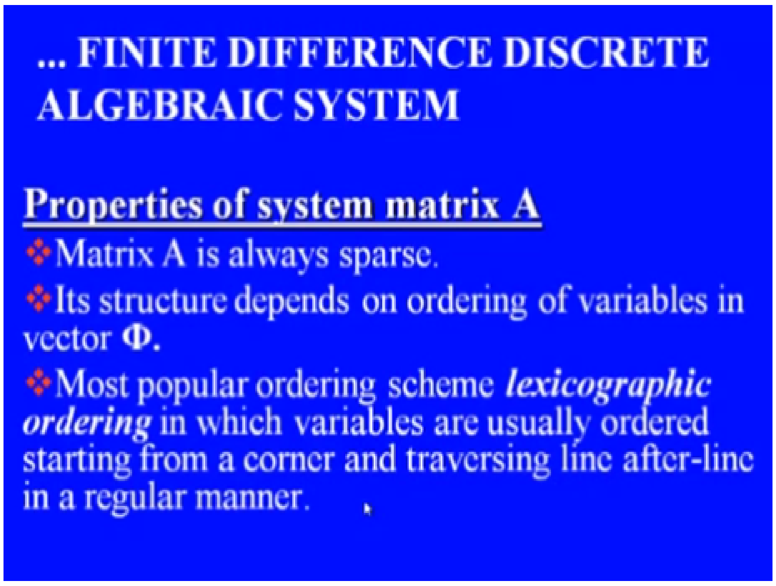
where **A** is the coefficient matrix, **Φ** is the vector of unknown nodal values of and **Q** is the vector containing terms on RHS.

So, now we have seen that look at each grid node, we get assist a simple algebraic equation which we can always write in discretised form, similarly at boundary nodes if there are derivatives present, we can use a finite difference approximation once again we can get an equation algebraic equation for our boundary nodes. For the boundary, nodes where the function itself is specified we get a very simple equation; algebraic equation.

So, if we collect all those discrete algebraic equations at all the nodes that is our internal nodes as well as our external nodes all these else, so we get a system capital A * phi = Q where this A is called the coefficient matrix or system matrix, capital Phi it is the vector of unknown nodal values reach at all the grid points and in this vector capital Q; the bold Q we have collected all the terms on the right hand side that is Q represents the vector of known terms.

Or what we call the right hand side terms, so they are contained in Q. Now, in the case of finite difference and same would be the case in finite volume and finite element techniques. This discrete system matrix A has got some special properties, so what are the property that system matrix A which are obvious, the first obvious properties matrix A is always sparse, there are nonzero entries only on few places in the matrix.

(Refer Slide Time: 37:09)



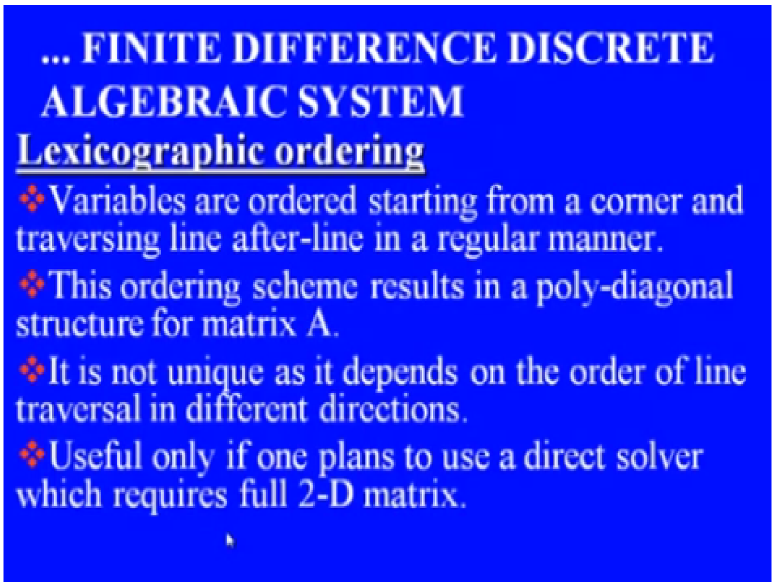
… FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

Properties of system matrix A

- Matrix A is always sparse.
- Its structure depends on ordering of variables in vector **Φ**.
- Most popular ordering scheme ***lexicographic ordering*** in which variables are usually ordered starting from a corner and traversing line after-line in a regular manner.

And as for its the structure of this matrix is concerned, it depends on the ordering of variables in the vector phi, okay and the most popular ordering scheme which were used earlier was what we call lexicographic ordering in which variables are usually ordered starting from a corner and traversing line after line in a regular manner. The only requirement is this start from one corner node then fall one line, the next line parallel to it, next time parallel to it and so on.

(Refer Slide Time: 37:58)



... FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

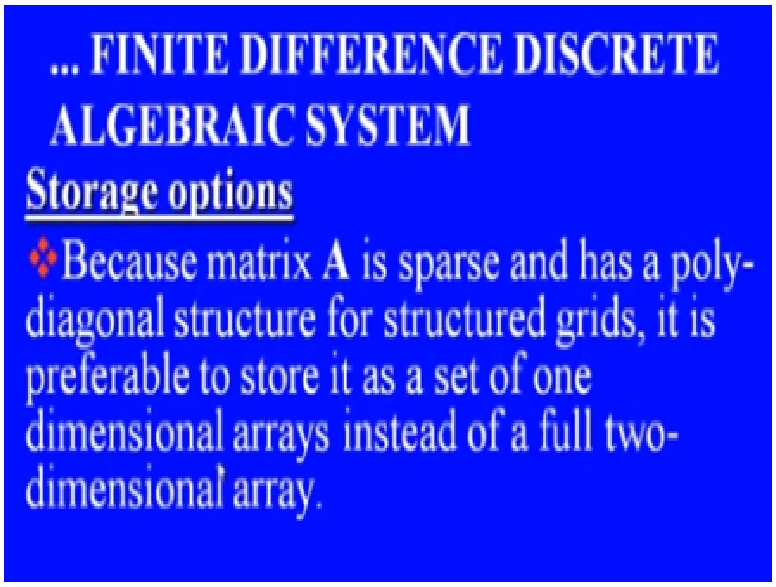
Lexicographic ordering

- Variables are ordered starting from a corner and traversing line after-line in a regular manner.
- This ordering scheme results in a poly-diagonal structure for matrix A.
- It is not unique as it depends on the order of line traversal in different directions.
- Useful only if one plans to use a direct solver which requires full 2-D matrix.

Go to the next plane in a 3D case to generate our indices okay, for instance, if we start the ordering the variable from a corner, this traversal which we call as lexicographic ordering, it results in a poly diagonal structure for matrix A, that is there would be only a set of specific diagonals which are nonzero remaining all the entries of matrix A or 0. Now, we can have different regular orderings.

So, this lexicographic ordering is not unique as it depends on the order of line traversal in different directions x, y and z directions which way we have followed that is up to us, so we can have different types of lexicographic ordering. Now, please note that we need to worry about this lexicographic ordering only if one plans to use a direct solver that is we have assembled a matrix A which is square matrix multiply a vector phi = Q.

(Refer Slide Time: 39:10)



... FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

Storage options

- Because matrix A is sparse and has a poly-diagonal structure for structured grids, it is preferable to store it as a set of one dimensional arrays instead of a full two-dimensional array.

And we want to use a standard a direct linear solver, all in that case, we need to store this full two dimensional matrix. Most of time in fact in CFD, we do not do it. Now or matrix A just pass, that is what we have seen that is the basic property and it has got a poly diagonal structure for structure grids in case of our differences we have; we always use structured grid, so A would have a poly diagonal structure.

So, now you have got 2 options; one option is we store full 2 dimensional grid and thereby we will have many zeros, so we are just storing useless zero numbers which are of no use, we would not, they do not contribute to our solution process in any way. So, what we can instead do is; we can store these diagonals separately, so it is preferable to store this matrix as a set of one dimensional arrays, one one array for each diagonal.

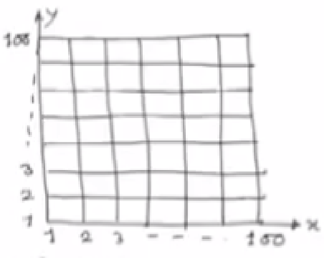
(Refer Slide Time: 39:56)

D Elliptic problem

Total no. of grid point

$N = 100 \times 100$

$= 10^4$



If store a 2-D matrix of order N we require N^2 entries.

Thus storage requirement for full matrix

$\propto N^2$

$\propto 10^8$

With 5 point CDS scheme, [A] will have only non-zero along 5-diagonals. If we store only these 5-diagonal, storage requirement

$\propto 5N$.

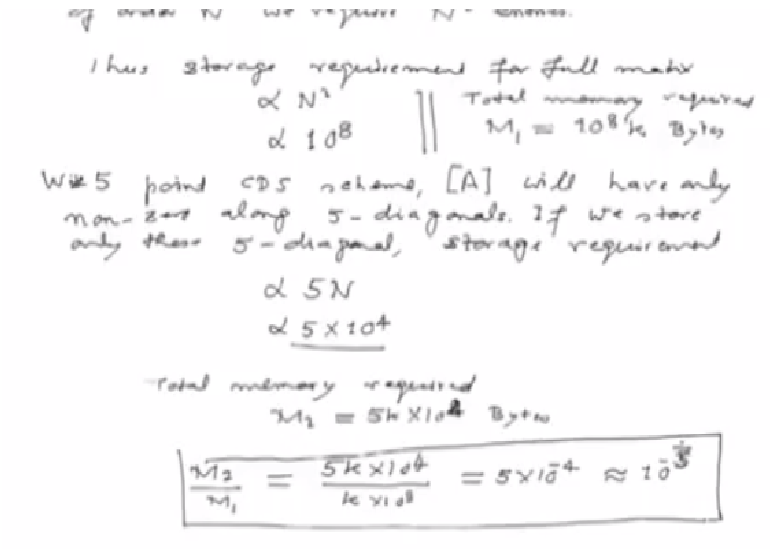
So, that would be a much better choice. Now, to understand this statement let us take a typical example, suppose we are working with 2D problem; 2D elliptic problem and for discretization suppose, we have chosen a rectangular grid is what we would use in finite difference method, will choose a rectangular Cartesian grid.

Just for sake of argument, suppose the number of points; grid points which have used in x direction those are 100, which is fairly common in fact we might be using a much larger number. Similarly, let us say that in y direction, we have chosen the total number of grid lines is again 100, so if we store all the entries, now we will have 100, total number of grid points; total number of grid points that would be 100 cross 100.

That is where we have got 10 to the power 4 grid points in our computational domain. Now, if you store full matrix A; if we store at 2D matrix of order N, how many entries we require? We require N square entries and now here we have put our N is 10 to the power of 4, so if you store all the entries in a finite difference matrix, so the storage requirement; the storage requirement for full matrix is proportional to N square in this particular example.

It is proportional to 10 to the power 8, suppose we have used 5 point CDS, so with 5 point CDS scheme or matrix A will have only nonzero entries along 5 diagonals. So, in this case what will be storage requirement? So, if we store only these 5 diagonals our storage requirement or what this also linked to our memory requirements, so storage requirement this is proportional to 5 times N.

(Refer Slide Time: 44:06)



of order N we require N^2 entries.

Thus storage requirement for full matrix $\propto N^2$ $\propto 10^8$ || Total memory required $M_1 = 10^8 k$ Bytes

With 5 point CDS scheme, [A] will have only non-zero along 5-diagonals. If we store only these 5-diagonal, storage requirement

$\propto 5N$

$\propto \underline{5 \times 10^4}$

Total memory required

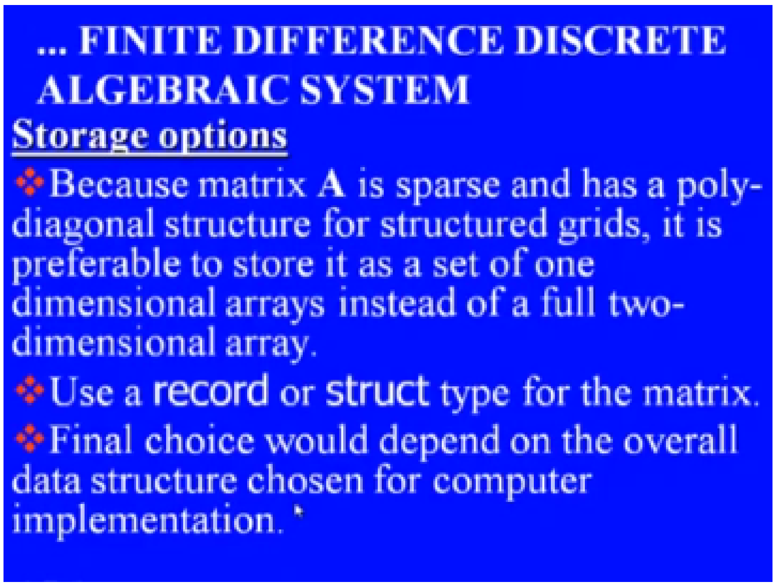
$M_2 = 5k \times 10^4$ Bytes

$$\boxed{\frac{M_2}{M_1} = \frac{5k \times 10^4}{k \times 10^8} = 5 \times 10^{-4} \approx 10^{-3}}$$

That is in our case, N = 10 to a power of 4, so just 5 * 10 to the power 4. Now, you can say if the proportionality constant we take here like that for each real variable, we require let us say k bytes, so the total memory required in the first case, for the full storage this would be 10 to the power 8k bytes and the second case; first case let us call it as capital M1 and the second case total memory required would be M2 that will be 5k * 10 to the power 4 bytes.

So, if you take the ratio of the two; M2/M1 that is 5k * 10 to the power 4 divided by k * 10 to power 8, we get 5 * 10 to the power -4 or which we can say approximately 10 to the power -3, so our storage requirement if we follow or if we make use of this pasty pattern those are 1000 smaller than the full matrix storage and thus what we would in normal programming that is what we do whenever we write of CFD programs, we never ever store full 2 dimensional matrix.

(Refer Slide Time: 46:26)



... FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

Storage options

- Because matrix **A** is sparse and has a poly-diagonal structure for structured grids, it is preferable to store it as a set of one dimensional arrays instead of a full two-dimensional array.
- Use a record or struct type for the matrix.
- Final choice would depend on the overall data structure chosen for computer implementation.

While using finite differences will only store these nonzero entries or nonzero diagonals okay. So, that is why we say it is preferable because we are going to save lots and lots of memory, if we store these diagonals as one dimensional array. (()) (46:37) in modern programming language is to use a record or a struct type for the matrix, now which one you choose? Whether we choose a set of arrays, we can name them as AP, AN and so on.

Or we use his record as struct type, this final choice would depend on the overall data structures chosen for the computer implementation and the language which we have chosen for implementation. For today, we just stop here and we would take up this struct or record type examples in our application part, so in the next lecture we will take up 1 or 2 problems and we would use finite difference approximation to come up with the final discrete system.

We will also try and solve one problem in full and then we will have a look at an outline of a computer program based on C language in the next 1 or 2 lectures.