Lecture 14
MULTIDIMENSIONAL DERIVATIVES

14.1 MULTIDIMENSIONAL FINITE DIFFERENCE FORMULAE

Multidimensional finite difference formulae can be easily written using one dimensional formulae. For example, the formulae for first order derivative using forward difference scheme (FDS) are

$$\left(\frac{\partial f}{\partial x}\right)_{i,j,k} = \frac{f_{i+1,jk} - f_{ijk}}{\Delta x} + O(\Delta x) \tag{14.1}$$

$$\left(\frac{\partial f}{\partial y}\right)_{i,j,k} = \frac{f_{i,j+1,k} - f_{ijk}}{\Delta y} + O(\Delta y) \tag{14.2}$$

$$\left(\frac{\partial f}{\partial z}\right)_{i,j,k} = \frac{f_{i,j,k+1} - f_{ijk}}{\Delta z} + O(\Delta z) \tag{14.3}$$

Similarly, second order central difference formulae for the second order derivatives are:

$$\left(\frac{\partial^2 f}{\partial x^2}\right)_{ijk} = \frac{f_{i+1,jk} + f_{i-1,jk} - 2f_{ijk}}{(\Delta x)^2} + O(\Delta x^2) \tag{14.4}$$

$$\left(\frac{\partial^2 f}{\partial y^2}\right)_{ijk} = \frac{f_{i,j+1,k} + f_{i,j-1,k} - 2f_{ijk}}{(\Delta y)^2} + O(\Delta y^2) \tag{14.5}$$

$$\left(\frac{\partial^2 f}{\partial z^2}\right)_{ijk} = \frac{f_{i,j,k+1} + f_{i,j,k-1} - 2f_{ijk}}{(\Delta z)^2} + O(\Delta z^2) \tag{14.6}$$

14.2 APPROXIMATION OF MIXED DERIVATIVES

In thermo-fluid problems, mixed derivatives are encountered in two situations: (a) heat conduction in anisotropic medium, and (b) the transport equations expressed in non-orthogonal coordinate systems. Mixed derivatives can be treated by combining one dimensional approximation in the same way as described earlier for second order derivative. For example, the mixed derivative $\partial^2 f / \partial x \partial y$ can be re-written as

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) \tag{14.7}$$

Using CDS for approximation of the outer derivative, we obtain

$$\left(\frac{\partial^2 f}{\partial x \partial y}\right)_{i,j} = \frac{\left(\frac{\partial f}{\partial y}\right)_{i+1,j} - \left(\frac{\partial f}{\partial y}\right)_{i-1,j}}{2\Delta x} + O(\Delta x^2) \tag{14.8}$$

We can also use CDS for approximation of $\partial f / \partial y$, i.e.

$$\left(\frac{\partial f}{\partial y}\right)_{i+1,j} = \frac{f_{i+1,j+1} - f_{i+1,j-1}}{2\Delta y} + O\left(\Delta y^2\right), \quad \left(\frac{\partial f}{\partial y}\right)_{i-1,j} = \frac{f_{i-1,j+1} - f_{i-1,j-1}}{2\Delta y} + O\left(\Delta y^2\right) \qquad (14.9)$$

Combining Eq. (14.8) and Eq. (14.9), we get

$$\left(\frac{\partial^2 f}{\partial x \partial y}\right)_{i,j} = \frac{f_{i+1,j+1} - f_{i+1,j-1} - f_{i-1,j+1} + f_{i-1,j-1}}{4\Delta x \Delta y} + O\left(\Delta x^2, \Delta y^2\right) \qquad (14.10)$$

14.3 IMPLEMENTATION OF BOUNDARY CONDITIONS

Numerical solution of a partial differential equation (PDE) using FDM requires finite difference approximation of the PDE at every interior grid point. Further, boundary conditions specified on the domain boundaries must be imposed to obtain a unique solution of the continuum problem. The boundary conditions of the Dirichlet-type (i.e. specified value of the variable) can be incorporated directly. However, the boundary conditions which involve the gradient of a variable would require the use of one sided difference formulae (forward or backward, depending on the location of the boundary node) for approximation of derivatives at the boundary. For example, at the boundary node x_1 the simplest approximation of the derivative is given by the forward difference scheme

$$\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{f_2 - f_1}{x_2 - x_1} \qquad (14.11)$$

which is first order accurate. For more accurate approximation, we can use higher order one-sided difference formulae involving function values at the boundary node and interior grid points. For instance, use of quadratic polynomial fit involving function values at boundary point x_1 and interior nodes x_2 and x_3 leads to the following second order approximation (Ferziger and Peric, 2003):

$$\left(\frac{\partial f}{\partial x}\right)_1 \approx \frac{-(x_2 - x_1)^2 f_3 + (x_3 - x_1)^2 f_2 - \left[(x_3 - x_1)^2 - (x_2 - x_1)^2\right] f_1}{(x_2 - x_1)(x_3 - x_1)(x_3 - x_2)} \qquad (14.12)$$

On a uniform grid, the preceding approximation reduces to

$$\left(\frac{\partial f}{\partial x}\right)_i \approx \frac{-\phi_3 + 4\phi_2 - 3\phi_1}{2\Delta x} \qquad (14.13)$$

One-sided difference formulae (14.11)-(14.13) would also be useful in post processing, e.g. in evaluation of heat flux from computed temperature field.

14.4 FINITE DIFFERENCE DISCRETE ALGEBRAIC SYSTEM

Finite difference approximation of derivatives in a partial differential equation (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, and non-linear for non-linear PDEs. In the latter case, the non-linear equation would require linearization in the process of its numerical solution. 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 (Ferziger and Peric, 2003)

$$A_P\phi_P + \sum_l A_l\phi_l = Q_P \tag{14.14}$$

where index P represents the node at which the PDE is approximated and l denotes the neighbouring node involved in finite difference approximation. Coefficients A_l and A_P are functions of grid size and material properties. The node P and its neighbours form the so-called *computational molecule* (or *stencil*). In the finite difference and the finite volume methods, compass notation is usually employed, i.e. for a node (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. Figure 9.5 depicts the computational molecules in 1D and 2D.

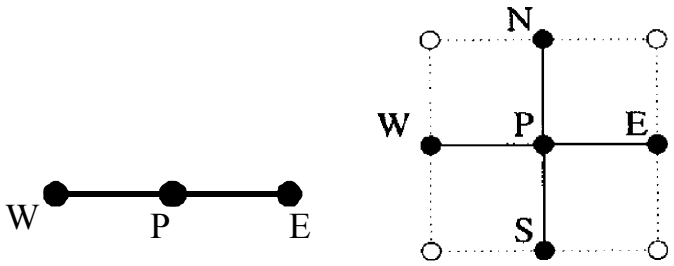

Figure 9.5 Central difference computational molecules (1D and 2D)

Collection of algebraic equation (14.14) at all the computational nodes leads to a system of algebraic equations given by

$$\mathbf{A\Phi} = \mathbf{Q} \tag{14.15}$$

where **A** is the coefficient matrix, Φ is the vector of unknown nodal values of ϕ and **Q** is the vector containing terms on RHS of Eq. (14.14). Matrix **A** is sparse. However, its structure depends on ordering of variables in vector **Φ**.

For structured grids, variables are usually ordered starting from a corner and traversing line after-line in a regular manner. This ordering scheme is called *lexicographic ordering* and results in a poly-diagonal structure for matrix **A**. Note that this ordering is not unique as it depends on the order of line traversal in different directions. For example, if we start ordering the entries in vector **Φ** starting from southwest corner of the domain, proceed eastward along each grid line, and then northward across the domain, we get the ordering of indices as given in Table 14.1.

Table 14.1 Conversion of (i, j, k) grid indices to one dimensional storage locations in vector **Φ.**

Grid location	Compass notation	Storage location
i, j, k	P	$l = (k-1) N_i N_j + (j-1) N_i + 1$
$i-1, j, k$	W	$l-1$
$i+1, j, k$	E	$l+1$
$i, j-1, k$	S	$l - N_i$
$i, j+1, k$	N	$l + N_i$
$i, j, k-1$	B	$l - N_i N_j$
$i, j, k+1$	T	$l + N_i N_j$

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. We can use the directional connection in naming these 1D arrays (calling them as A_P, A_W etc.), and thus write the algebraic equation (14.14) for a two dimensional problem as

$$A_W \phi_W + A_S \phi_S + A_P \phi_P + A_N \phi_N = Q_P \tag{14.16}$$

At present, most of the programming languages (including Fortran 90) support user-defined data types. An alternative storage scheme for matrix **A** can be adopted using records or structure data types wherein the matrix **A** is stored as an array of records (or struct). Each element of this record contains the number of modes connected to a given node, nodal connectivity (i.e. indices of neighbouring nodes) and corresponding matrix entries. This scheme can be used not only for the system matrix arising from finite difference discretization on structured grids, but also for the matrix resulting from finite volume/finite element method on structured as well as unstructured grids. A sample implementation of such a record in C is as follows:

```
struct Matrix
  {
     int  nn;              // number of neighbours of the node
     double AP;            // matrix entry A_P
     double *AL;           // array to hold coefficients A_l
     int *nc;              // array to hold connectivity information
  }
```

On structured grids, we can omit storage of connectivity information by adopting an implicit indexing for neighbouring nodes.

Note that the indexing scheme described above is primarily required if one plans to use a direct solver which require specification of an algebraic system in the standard form **Ax** = **b**. In practical CFD applications, large size of the system matrix normally dictates use of iterative solvers (even for linear systems).

REFERENCE

Ferziger, J. H. And Perić, M. (2003). *Computational Methods for Fluid Dynamics*. Springer.