Draft Notes

ME 608

Numerical Methods in Heat, Mass, and Momentum Transfer

Instructor: Jayathi Y. Murthy
School of Mechanical Engineering
Purdue University

Spring 2002

Contents

Chapter 1

Mathematical Modeling

In order to simulate fluid flow, heat transfer, and other related physical phenomena, it is necessary to describe the associated physics in mathematical terms. Nearly all the physical phenomena of interest to us in this book are governed by principles of conservation and are expressed in terms of partial differential equations expressing these principles. For example, the momentum equations express the conservation of linear momentum; the energy equation expresses the conservation of total energy. In this chapter we derive a typical conservation equation and examine its mathematical properties.

1.1 Conservation Equations

Typical governing equations describing the conservation of mass, momentum, energy, or chemical species are written in terms of *specific* quantities - i.e., quantities expressed on a *per unit mass* basis. For example, the momentum equation expresses the principle of conservation of linear momentum in terms of the momentum per unit mass, i.e., velocity. The equation for conservation of chemical species expresses the conservation of the mass of the species in terms of its mass fraction

Let us consider a *specific quantity* ϕ, which may be momentum per unit mass, or the energy per unit mass, or any other such quantity. Consider a control volume of size $\Delta x \times \Delta y \times \Delta z$ shown in Figure 1.1. We want to express the variation of ϕ in the control volume over time. Let us assume that ϕ is governed by a conservation principle that states

$$\begin{aligned}\text{Accumulation of } \phi \text{ in the control volume over time } \Delta t \quad = \quad & \\ & \text{Net influx of } \phi \text{ into control volume} + \\ & \text{Net generation of } \phi \\ & \text{inside control volume}\end{aligned} \tag{1.1}$$

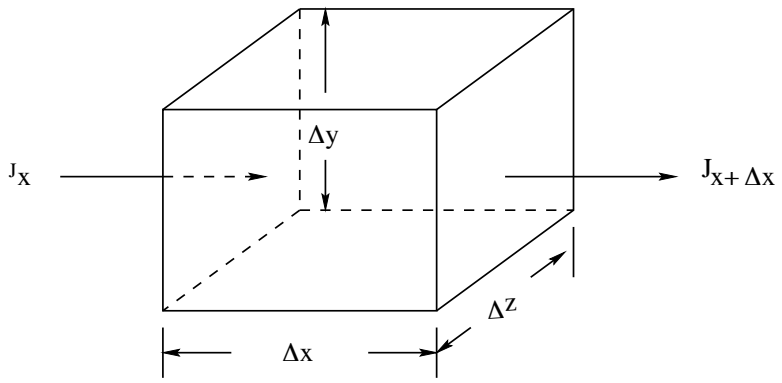


Figure 1.1: Control Volume

The accumulation of ϕ in the control volume over time Δt is given by

$$(\rho\phi\Delta\mathscr{V})_{t+\Delta t} - (\rho\phi\Delta\mathscr{V})_t \tag{1.2}$$

Here, ρ is the density of the fluid, $\Delta\mathscr{V}$ is the volume of the control volume ($\Delta x \times \Delta y \times \Delta z$) and t is time.

The net generation of ϕ inside the control volume over time Δt is given by

$$S\Delta\mathscr{V}\Delta t \tag{1.3}$$

where S is the generation of ϕ per unit volume. S is also sometimes called the source term.

Let us consider the remaining term, the net influx of ϕ into the control volume. Let J_x represent the flux of ϕ coming into the control volume through face x, and $J_{x+\Delta x}$ the flux leaving the face $x+\Delta x$. Similar fluxes exist on the y and z faces respectively. The net influx of ϕ into the control volume over time Δt is

$$\left(J_x - J_{x+\Delta x}\right)\Delta y\Delta z\Delta t + \left(J_y - J_{y+\Delta y}\right)\Delta x\Delta z\Delta t + \left(J_z - J_{z+\Delta z}\right)\Delta x\Delta y\Delta t \tag{1.4}$$

We have not yet said what physical mechanisms cause the influx of ϕ. For physical phenomena of interest to us, ϕ is transported by two primary mechanisms: diffusion due to molecular collision, and convection due to the motion of fluid. In many cases, the diffusion flux may be written as

$$J_{\text{diffusion},x} = -\Gamma\frac{\partial\phi}{\partial x} \tag{1.5}$$

The convective flux may be written as

$$J_{\text{convection},x} = \rho u\phi \tag{1.6}$$

Here, the velocity field is given by the vector $\mathbf{V} = u\mathbf{i} + v\mathbf{j} + w\mathbf{k}$. Thus the net convective and diffusive flux may be written as

$$\begin{aligned} J_x &= \left(\rho u\phi - \Gamma\frac{\partial\phi}{\partial x}\right)_x \\ J_{x+\Delta x} &= \left(\rho u\phi - \Gamma\frac{\partial\phi}{\partial x}\right)_{x+\Delta x} \end{aligned} \tag{1.7}$$

where $(\rho u)_x$ is the mass flux through the control volume face at x. Similar expressions may be written for the y and z directions respectively.

Accumulating terms, and dividing by $\Delta\mathcal{V}\Delta t$ Equation 1.1 may be written as

$$\begin{aligned} \frac{(\rho\phi)_{t+\Delta t} - (\rho\phi)_t}{\Delta t} &= \frac{\left(J_x - J_{x+\Delta x}\right)}{\Delta x} + \frac{\left(J_y - J_{y+\Delta y}\right)}{\Delta y} + \\ &\quad \frac{\left(J_z - J_{z+\Delta z}\right)}{\Delta z} + S \end{aligned} \tag{1.8}$$

Taking the limit $\Delta x, \Delta y, \Delta z, \Delta t \to 0$, we get

$$\frac{\partial(\rho\phi)}{\partial t} = -\frac{\partial J_x}{\partial x} - \frac{\partial J_y}{\partial y} - \frac{\partial J_z}{\partial z} + S \tag{1.9}$$

It is convenient to write Equation 1.9 as

$$\begin{aligned} \frac{\partial}{\partial t}(\rho\phi) + \frac{\partial}{\partial x}(\rho u\phi) + \frac{\partial}{\partial y}(\rho v\phi) + \frac{\partial}{\partial z}(\rho w\phi) = \\ \frac{\partial}{\partial x}\left(\Gamma\frac{\partial\phi}{\partial x}\right) + \frac{\partial}{\partial y}\left(\Gamma\frac{\partial\phi}{\partial y}\right) + \\ \frac{\partial}{\partial z}\left(\Gamma\frac{\partial\phi}{\partial z}\right) + S \end{aligned}$$

or, in vector notation

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla\cdot\rho\mathbf{V}\phi = \nabla\cdot(\Gamma\nabla\phi) + S \tag{1.10}$$

1.1.1 Discussion

It is worth noting the following about the above derivation:

- The differential form is derived by considering balances over a finite control volume.
- Though we have chosen hexahedral control volume on which to do conservation, we can, in principle, choose any shape. We should get the same final governing differential equation regardless of the shape of the volume chosen to do the derivation.

- The conservation equation is written in terms of a *specific* quantity ϕ, which may be energy per unit mass (J/kg), or momentum per unit mass (m/s) or some similar quantity.

- The conservation equation is written on a *per unit volume per unit time* basis. The generation term in Equation 1.10 for example, is the generation of ϕ per unit volume per unit time. If ϕ were energy per unit mass, S would be the generation of energy per unit volume per unit time.

1.1.2 Conservation Form

Equation 1.10 represents the *conservative* or *divergence* form of the conservation equation. This form is characterized by the fact that in steady state, in the absence of generation, the divergence of the flux is zero:

$$\nabla \cdot \mathbf{J} = 0 \tag{1.11}$$

where $\mathbf{J} = J_x\mathbf{i} + J_y\mathbf{j} + J_z\mathbf{k}$. By using the continuity equation, we may write the *non-conservative* form of Equation 1.10

$$\frac{\partial(\rho\phi)}{\partial t} + \rho\mathbf{V}\cdot\nabla\phi = \Gamma\nabla\cdot\nabla\phi + \nabla\Gamma\cdot\nabla\phi + S \tag{1.12}$$

The divergence of $\mathbf{J}$ represents the net efflux per unit volume of $\mathbf{J}$. Thus, the conservative form is a direct statement about the conservation of ϕ in terms of the physical fluxes (convection and diffusion). The non-conservative form does not have a direct interpretation of this sort. Numerical methods that are developed with the divergence form as a starting point can be made to reflect the conservation property *exactly* if care is taken. Those that start from Equation 1.12 can be made to approximate conservation in some limiting sense, but not exactly.

1.2 Governing Equations

The governing equations for fluid flow, heat and mass transfer, as well as other transport equations, may be represented by the conservative form, Equation 1.10. Let us now consider some specific cases of the conservation equation for ϕ.

1.2.1 The Energy Equation

The general form of the energy equation is quite elaborate, though it can also be cast into the general form of Equation 1.10. For simplicity, let us assume low-speed flow and negligible viscous dissipation.

For this case, the energy equation may be written in terms of the specific enthalpy h as

$$\frac{(\partial\rho h)}{\partial t} + \nabla\cdot(\rho\mathbf{V}h) = \nabla\cdot(k\nabla T) + S_h \tag{1.13}$$

where k is the thermal conductivity and T is the temperature. For ideal gases and incompressible substances,

$$dh = C_p dT \tag{1.14}$$

so that Equation 1.13 may be written as

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho \mathbf{V} h) = \nabla \cdot \left(\frac{k}{C_p} \nabla h \right) + S_h \tag{1.15}$$

Comparing Equation 1.15 with Equation 1.10 shows that the energy equation can be cast into the form of the general conservation equation, with $\phi = h$, $\Gamma = k/C_p$ and $S = S_h$.

1.2.2 The Momentum Equation

The momentum equation for a Newtonian fluid in the direction x may be written as

$$\frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho \mathbf{V} u) = \nabla \cdot (\mu \nabla u) - \frac{\partial p}{\partial x} + S_u \tag{1.16}$$

Here, S_u contains those parts of the stress tensor not appearing directly in the diffusion term, and $\partial p/\partial x$ is pressure gradient. We see that Equation 1.16 has the same form as the general conservation equation 1.10, with $\phi = u$, $\Gamma = \mu$ and $S = -\partial p/\partial x + S_u$.

1.2.3 The Species Equation

Consider the transport of a mixture of chemical species. The equation for the conservation of mass for a chemical specie i may be written in terms of its mass fraction, Y_i, where Y_i is defined as the mass of species i per mass of mixture. If Fick's law is assumed valid, the governing conservation equation is

$$\frac{\partial \rho Y_i}{\partial t} + \nabla \cdot (\rho \mathbf{V} Y_i) = \nabla \cdot (\Gamma_i \nabla Y_i) + R_i \tag{1.17}$$

Γ_i is the diffusion coefficient for Y_i in the mixture and R_i is the rate of formation of Y_i through chemical reactions. Again we see that Equation 1.17 has the same form as the general conservation equation 1.10, with $\phi = Y_i$, $\Gamma = \Gamma_i$, and $S = R_i$.

1.3 The General Scalar Transport Equation

We have seen that the equations governing fluid flow, heat and mass transfer can be cast into a single general form which we shall call the general scalar transport equation:

$$\frac{\partial (\rho \phi)}{\partial t} + \nabla \cdot (\rho \mathbf{V} \phi) = \nabla \cdot (\Gamma \nabla \phi) + S \tag{1.18}$$

If numerical methods can be devised to solve this equation, we will have a framework within which to solve the equations for flow, heat, and mass transfer.

1.4 Mathematical Classification of Partial Differential Equations

The general scalar transport equation is a second-order partial differential equation (PDE) governing the spatial and temporal variation of ϕ. If the properties ρ and Γ, or the generation term S_ϕ are functions of ϕ, it is non-linear. Ignoring non-linearities for the moment, we examine the behavior of this equation.

It is instructive to consider a general second-order PDE given by

$$a\phi_{xx} + b\phi_{xy} + c\phi_{yy} + d\phi_x + e\phi_y + f\phi + g = 0 \tag{1.19}$$

The coefficients a,b,c,d,e, f and g are functions of the coordinates (x,y), but not of ϕ itself.

The behavior of Equation 1.19 may be classified according to the sign on the discriminant

$$\mathscr{D} = b^2 - 4ac \tag{1.20}$$

If $\mathscr{D} < 0$ the PDE is called *elliptic*. If $\mathscr{D} = 0$, the PDE is called *parabolic*. If $\mathscr{D} > 0$ the PDE is called *hyperbolic*. Let us consider typical examples of each type of equation.

1.4.1 Elliptic Partial Differential Equations

Let us consider steady heat conduction in a one-dimensional slab, as shown in Figure 1.2. The governing equation and boundary conditions are given by

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) = 0 \tag{1.21}$$

with

$$\begin{aligned} T(0) &= T_0 \\ T(L) &= T_L \end{aligned} \tag{1.22}$$

For constant k, the solution is given by

$$T(x) = T_0 + \frac{(T_L - T_0)}{L}x \tag{1.23}$$

This simple problem illustrates important properties of elliptic PDEs. These are

1. The temperature at any point x in the domain is influenced by the temperatures on *both* boundaries.

2. In the absence of source terms, $T(x)$ is bounded by the temperatures on the boundaries. It cannot be either higher or lower than the boundary temperatures.

It is desirable when devising numerical schemes that these basic properties be reflected in the characteristics of the scheme.

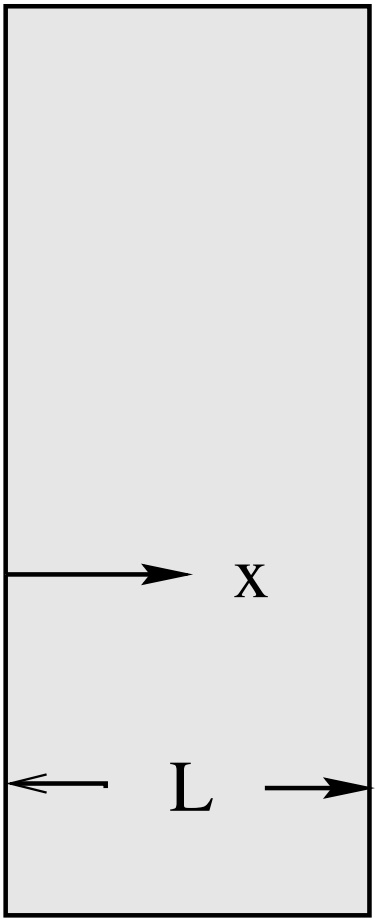


Figure 1.2: Conduction in a One-Dimensional Slab

1.4.2 Parabolic Partial Differential Equations

Consider unsteady conduction in the slab in Figure 1.2. If k, ρ and C_p are constant, Equation 1.13 may be written in terms of the temperature T as

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \tag{1.24}$$

where $\alpha = k/(\rho C_p)$ is the thermal diffusivity. The initial and boundary conditions are given by

$$\begin{aligned} T(x,0) &= T_i(x) \\ T(0,t) &= T_0 \\ T(L,t) &= T_0 \end{aligned} \tag{1.25}$$

Using a separation of variables technique, we may write the solution to this problem as

$$T(x,t) = T_0 + \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L}\right) e^{\frac{-\alpha n^2 \pi^2}{L^2} t} \tag{1.26}$$

where

$$B_n = \frac{2}{L}\int_0^L \left(T_i(x) - T_0\right) \sin\left(\frac{n\pi x}{L}\right) dx \quad n = 1,2,3,... \tag{1.27}$$

We note the following about the solution:

1. The boundary temperature T_0 influences the temperature T(x,t) at every point in the domain, just as with elliptic PDE's.

2. Only *initial* conditions are required (i.e., conditions at $t = 0$). No *final* conditions are required, for example conditions at $t \to \infty$. We do not need to know the future to solve this problem!

3. The initial conditions only affect *future* temperatures, not *past* temperatures.

4. The initial conditions influence the temperature at every point in the domain for all future times. The *amount* of influence decreases with time, and may affect different spatial points to different degrees.

5. A *steady state* is reached for $t \to \infty$. Here, the solution becomes independent of $T_i(x,0)$. It also recovers its elliptic spatial behavior.

6. The temperature is bounded by its initial and boundary conditions in the absence of source terms.

It is clear from this problem that the variable t behaves very differently from the variable x. The variation in t admits only *one-way* influences, whereas the variable x admits *two-way* influences. t is sometimes referred to as the *marching* or *parabolic* direction. Spatial variables may also behave in this way, for example, the axial direction in a pipe flow.

1.4.3 Hyperbolic Partial Differential Equations

Let us consider the one-dimensional flow of a fluid in a channel, as shown in Figure 1.3. The velocity of the fluid, U, is a constant; also $U > 0$. For $t \geq 0$, the fluid upstream of the channel entrance is held at temperature T_0. The properties ρ and C_p are constant and $k = 0$. The governing equations and boundary conditions are given by:

$$\frac{\partial}{\partial t}(\rho C_p T) + \frac{\partial}{\partial x}(\rho C_p U T) = 0 \tag{1.28}$$

with

$$\begin{aligned} T(x,0) &= T_i \\ T(x \leq 0,t) &= T_0 \end{aligned} \tag{1.29}$$

You can convince yourself that Equation 1.28 is hyperbolic by differentiating it once with respect to either t or x and finding the discriminant. The solution to this problem is

$$T(x,t) = T((x - Ut),0) \tag{1.30}$$

or to put it another way

$$\begin{aligned} T(x,t) &= T_i \text{ for } t < \frac{x}{U} \\ &= T_0 \text{ for } t \geq \frac{x}{U} \end{aligned} \tag{1.31}$$

The solution is essentially a step in T traveling in the positive x direction with a velocity U, as shown in Figure 1.4.

We should note the following about the solution:

1. The *upstream* boundary condition ($x = 0$) affects the solution in the domain. Conditions downstream of the domain do not affect the solution in the domain.
2. The inlet boundary condition propagates with a finite speed, U.
3. The inlet boundary condition is not felt at point x until $t = x/U$.

1.4.4 Behavior of the Scalar Transport Equation

The general scalar transport equation we derived earlier (Equation 1.10) has much in common with the partial differential equations we have seen here. The elliptic diffusion equation is recovered if we assume steady state and there is no flow. The same problem solved for unsteady state exhibits parabolic behavior. The convection side of the scalar transport equation exhibits hyperbolic behavior. In most engineering situations, the equation exhibits mixed behavior, with the diffusion terms tending to bring in elliptic influences, and the unsteady and convection terms bringing in parabolic or hyperbolic influences. It is sometimes useful to consider particular coordinates to be elliptic or parabolic. For example, it is useful in parabolic problems to think about time as the parabolic coordinate and to think of space as the elliptic coordinate.

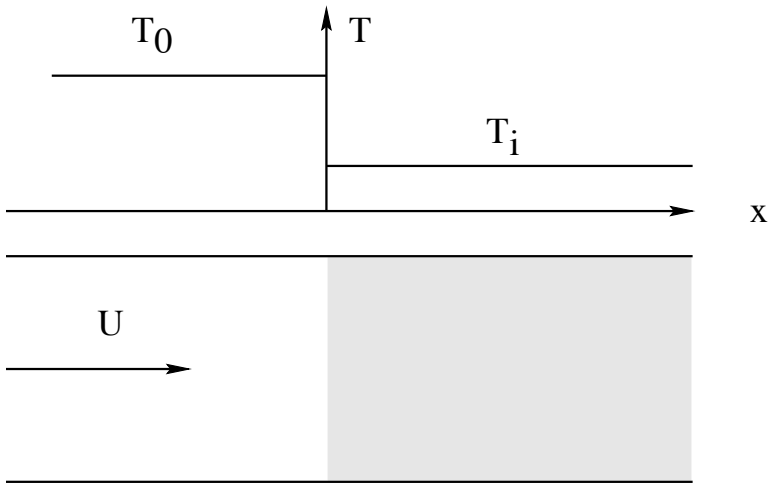


Figure 1.3: Convection of a Step Profile

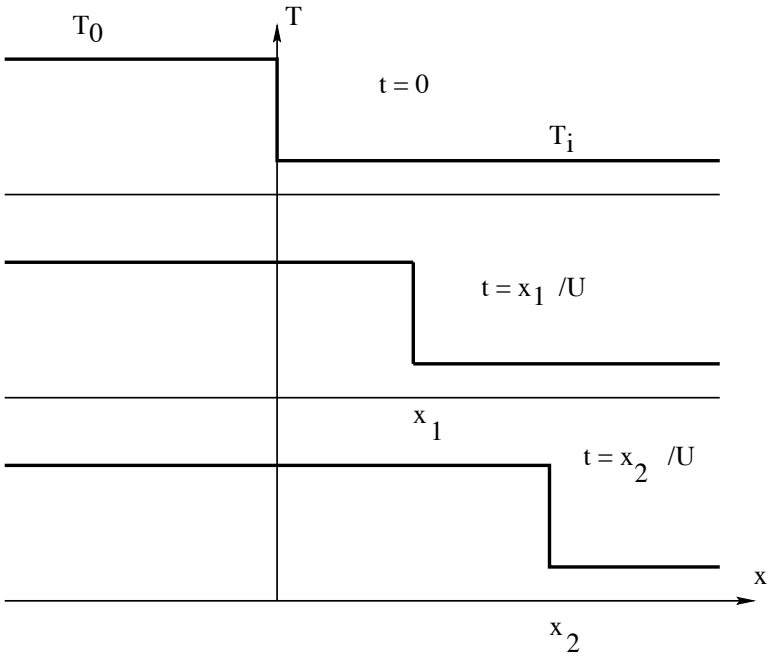


Figure 1.4: Temperature Variation with Time

Though it is possible to devise numerical methods which exploit the particular nature of the general scalar transport equation in certain limiting cases, we will not do that here. We will concentrate on developing numerical methods which are general enough to handle the mixed behavior of the general transport equation.

When we study fluid flow in greater detail, we will have to deal with coupled sets of equations, as opposed to a single scalar transport equation. These sets can also be analyzed in terms similar to the discussion above.

1.5 Closure

In this chapter, we have seen that many physical phenomena of interest to us are governed by conservation equations. These conservation equations are derived by writing balances over finite control volumes. We have seen that the conservation equations governing the transport of momentum, heat and other specific quantities have a common form embodied in the general scalar transport equation. This equation has unsteady, convection, diffusion and source terms. By studying the behavior of canonical elliptic, parabolic and hyperbolic equations, we begin to understand the behavior of these different terms in determining the behavior of the computed solution. The ideal numerical scheme should be able to reproduce these influences correctly.

Chapter 2

Numerical Methods

In the previous chapter, we saw that physical phenomena of interest to us could be described by a general scalar transport equation. In this chapter, we examine numerical methods for solving this type of equation, and identify the main components of the solution method. We also examine ways of characterizing our numerical methods in terms of accuracy, consistency, stability and convergence.

2.1 Overview

Our objective here is to develop a numerical method for solving the general scalar transport equation. Fundamental to the development of a numerical method is the idea of discretization. An analytical solution to a partial differential equation gives us the value of ϕ as a function of the independent variables (x,y,z,t). The numerical solution, on the other hand, aims to provide us with values of ϕ at a *discrete* number of points in the domain. These points are called *grid points*, though we may also see them referred to as *nodes* or *cell centroids*, depending on the method. The process of converting our governing transport equation into a set of equations for the discrete values of ϕ is called the *discretization process* and the specific methods employed to bring about this conversion are called *discretization methods*.

The discrete values of ϕ are typically described by algebraic equations relating the values at grid points to each other. The development of numerical methods focuses on both the derivation of the discrete set of algebraic equations, as well as a method for their solution. In arriving at these discrete equations for ϕ we will be required to assume how ϕ varies between grid points i.e., to make *profile assumptions*. Most widely used methods for discretization require *local* profile assumptions. That is, we prescribe how ϕ varies in the local neighborhood surrounding a grid point, but not over the entire domain.

The conversion of a differential equation into a set of discrete algebraic equations requires the discretization of space. This is accomplished by means of mesh generation. A typical mesh is shown in Figure 2.1. Mesh generation divides the domain of interest into elements or cells, and associates with each element or cell one or more discrete

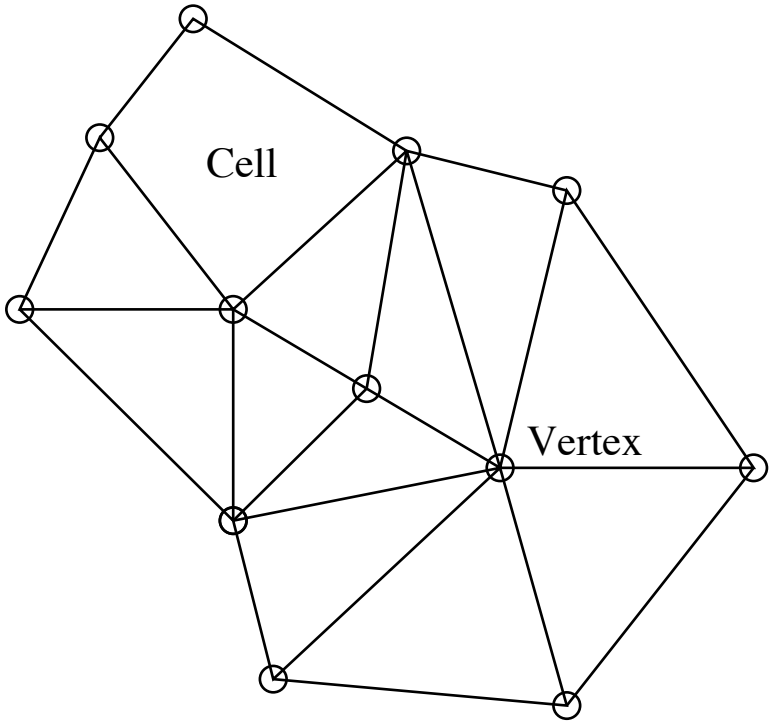


Figure 2.1: An Example of a Mesh

values of ϕ. It is these values of ϕ we wish to compute.

We should also distinguish between the discretized equations and the methods employed to solve them. For our purposes, let us say that the accuracy of the numerical solution, i.e., its closeness to the exact solution, depends only on the discretization process, and not on the methods employed to solve the discrete set (i.e., the path to solution). The path to solution determines whether we are successful in obtaining a solution, and how much time and effort it will cost us. But it does not determine the final answer. (For some non-linear problems, the path to solution can determine which of several possible solutions is obtained. For simplicity, we shall not pursue this line of investigation here.)

Since we wish to get an answer to the original differential equation, it is appropriate to ask whether our algebraic equation set really gives us this. When the number of grid points is small, the departure of the discrete solution from the exact solution is expected to be large. A well-behaved numerical scheme will tend to the exact solution as the number of grid points is increased. The rate at which it tends to the exact solution depends on the type of profile assumptions made in obtaining the discretization. No matter what discretization method is employed, all well-behaved discretization methods should tend to the exact solution when a large enough number of grid points is employed.

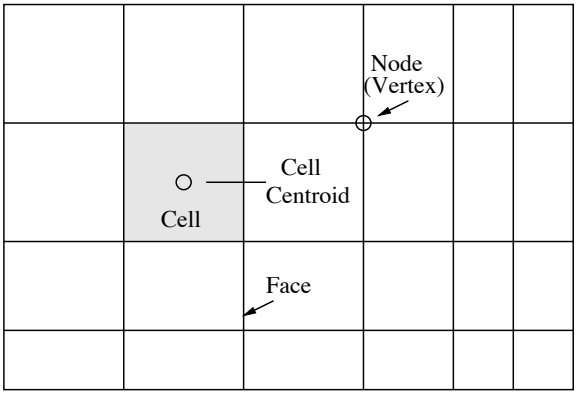


Figure 2.2: Mesh Terminology

2.2 Mesh Terminology and Types

The physical domain is discretized by meshing or gridding it.(We shall use the terms *mesh* and *grid* interchangeably in this book). We shall use the terminology shown in Figure 2.2 in describing our meshes. The fundamental unit of the mesh is the *cell* (sometimes called the element). Associated with each cell is the *cell centroid*. A cell is surrounded by *faces*, which meet at *nodes* or *vertices*. In three dimensions, the face is a surface surrounded by *edges*. In two dimensions, faces and edges are the same.

A variety of mesh types are encountered in practice. These are described below.

2.2.1 Regular and Body-fitted Meshes

In many cases, our interest lies in analyzing domains which are regular in shape: rectangles, cubes, cylinders, spheres. These shapes can be meshed by regular grids, as shown in Figure 2.3(a). The grid lines are orthogonal to each other, and conform to the boundaries of the domain. These meshes are also sometimes called orthogonal meshes.

For many practical problems, however, the domains of interest are irregularly shaped and regular meshes may not suffice. An example is shown in Figure 2.3(b). Here, grid lines are not necessarily orthogonal to each other, and curve to conform to the irregular geometry. If regular grids are used in these geometries, *stair stepping* occurs at domain boundaries, as shown in Figure 2.4. When the physics at the boundary are important in determining the solution, e.g., in flows dominated by wall shear, such an approximation of the boundary may not be acceptable.

2.2.2 Structured, Block Structured, and Unstructured Meshes

The meshes shown in Figure 2.3 are examples of *structured* meshes. Here, every interior vertex in the domain is connected to the same number of neighbor vertices. Figure 2.5 shows a block-structured mesh. Here, the mesh is divided into blocks, and the

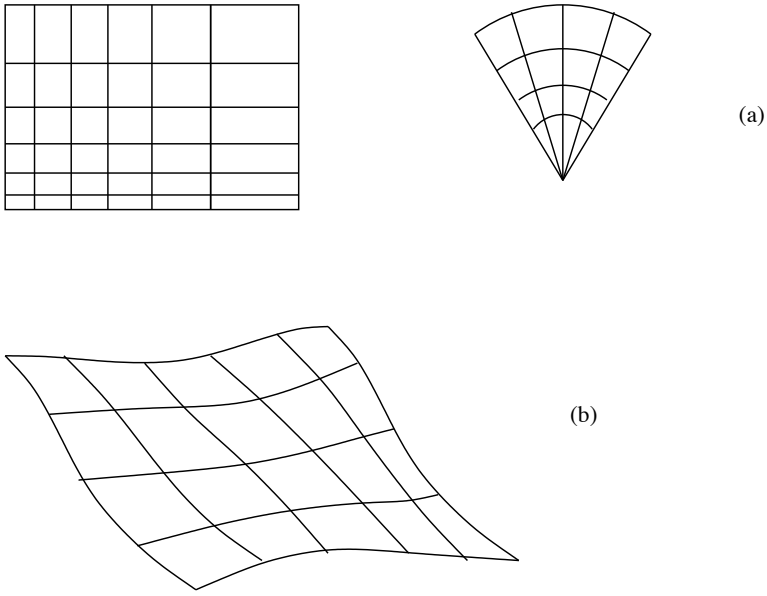


Figure 2.3: Regular and Body-Fitted Meshes

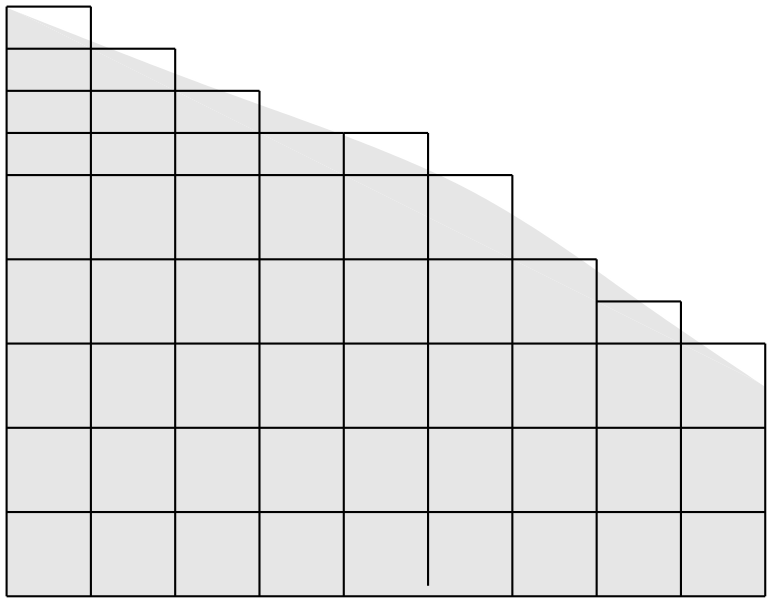


Figure 2.4: Stair-Stepped Mesh

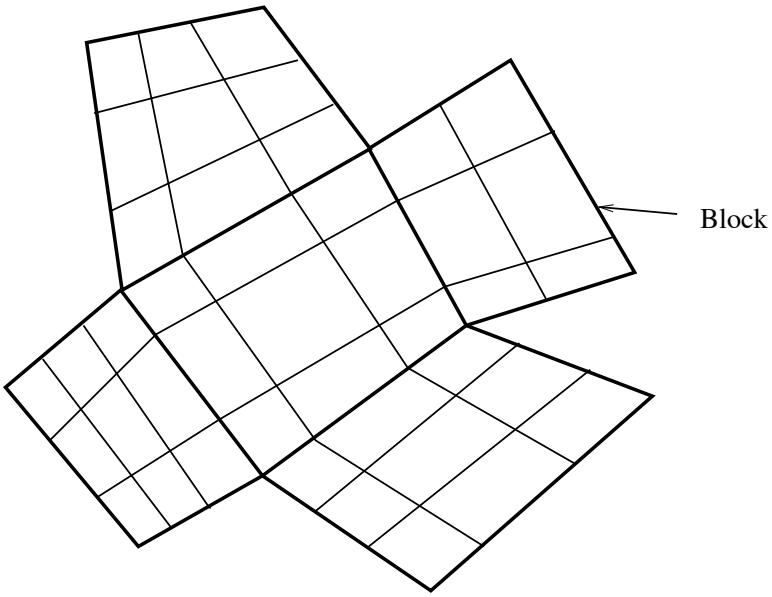


Figure 2.5: Block-Structured Mesh

mesh within each block is structured. However, the arrangement of the blocks themselves is not necessarily structured. Figure 2.6 shows an unstructured mesh. Here, each vertex is connected to an arbitrary number of neighbor vertices. Unstructured meshes impose fewer topological restrictions on the user, and as a result, make it easier to mesh very complex geometries.

2.2.3 Conformal and Non-Conformal Meshes

An example of a non-conformal mesh is shown in Figure 2.7. Here, the vertices of a cell or element may fall on the faces of neighboring cells or elements. In contrast, the meshes in Figures 2.3,2.5 and 2.6 are conformal meshes.

2.2.4 Cell Shapes

Meshes may be constructed using a variety of cell shapes. The most widely used are quadrilaterals and hexahedra. Methods for generating good-quality structured meshes for quadrilaterals and hexahedra have existed for some time now. Though mesh structure imposes restrictions, structured quadrilaterals and hexahedra are well-suited for flows with a dominant direction, such as boundary-layer flows. More recently, as computational fluid dynamics is becoming more widely used for analyzing industrial flows, unstructured meshes are becoming necessary to handle complex geometries. Here, triangles and tetrahedra are increasingly being used, and mesh generation techniques for their generation are rapidly reaching maturity. As of this writing, there are no general

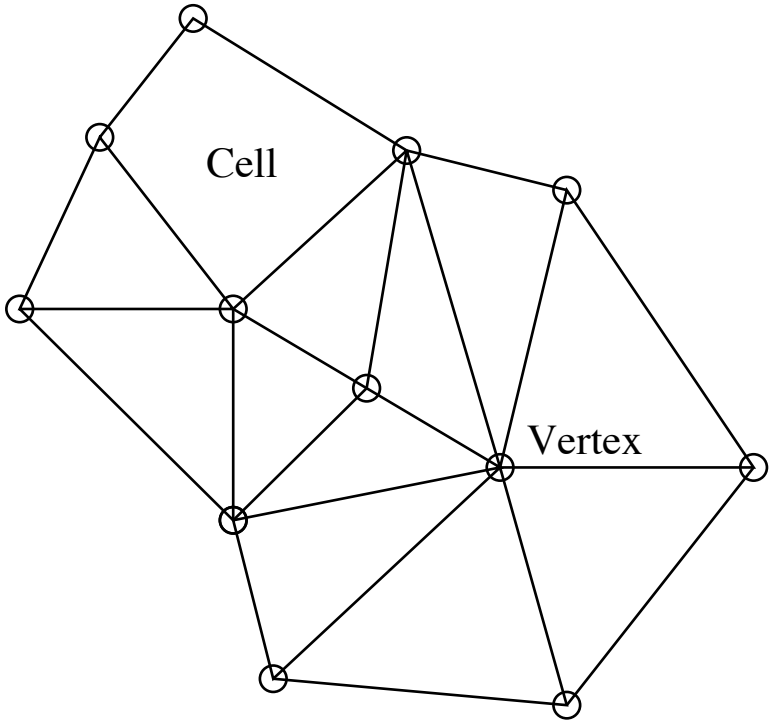


Figure 2.6: Unstructured Mesh

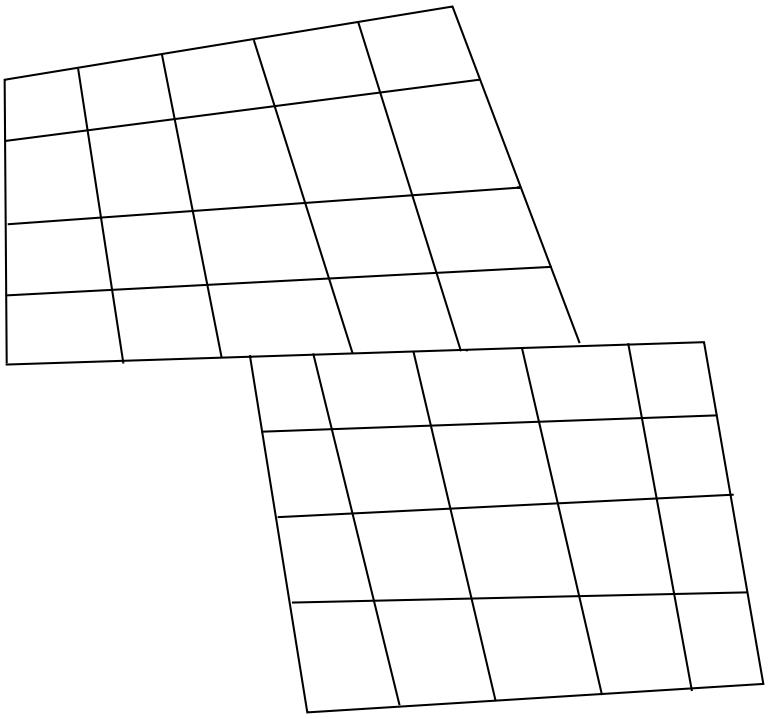


Figure 2.7: Non-Conformal Mesh

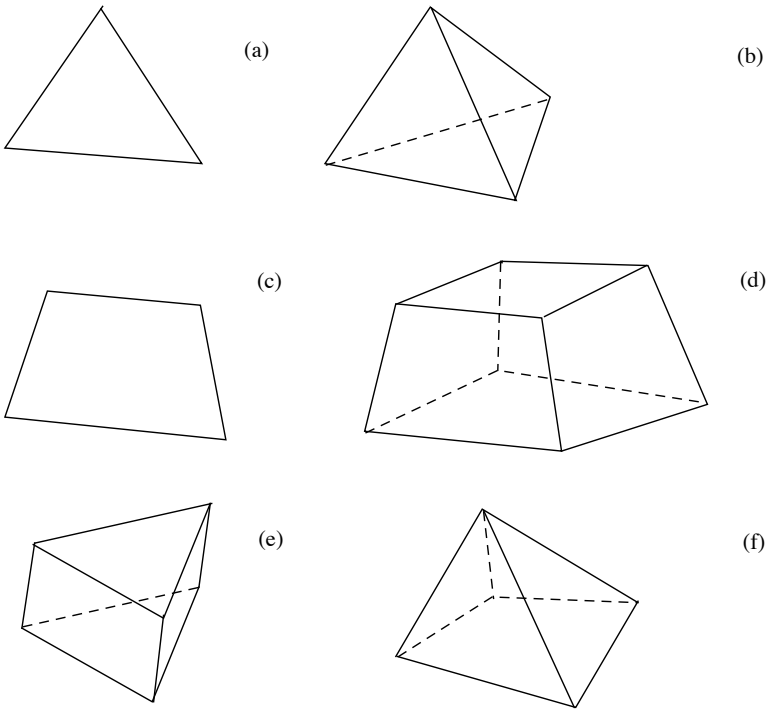


Figure 2.8: Cell Shapes: (a) Triangle, (b) Tetrahedron, (c) Quadrilateral, (d) Hexahedron, (e) Prism, and (f) Pyramid

purpose techniques for generating unstructured hexahedra. Another recent trend is the use of hybrid meshes. For example, prisms are used in boundary layers, transitioning to tetrahedra in the free-stream. In this book, we will develop numerical methods capable of using all these cell shapes.

2.2.5 Node-Based and Cell-Based Schemes

Numerical methods which store their primary unknowns at the node or vertex locations are called *node-based* or *vertex-based* schemes. Those which store them at the cell centroid, or associate them with the cell, are called *cell-based* schemes. Finite element methods are typically node-based schemes, and many finite volume methods are cell-based. For structured and block-structured meshes composed of quadrilaterals or hexahedra, the number of cells is approximately equal to the number of nodes, and the spatial resolution of both storage schemes is similar for the same mesh. For other cell shapes, there may be quite a big difference in the number of nodes and cells in the mesh. For triangles, for example, there are twice as many cells as nodes, on average. This fact must be taken into account in deciding whether a given mesh provides adequate resolution for a given problem. From the point of view of developing numerical methods, both schemes have advantages and disadvantages, and the choice will depend

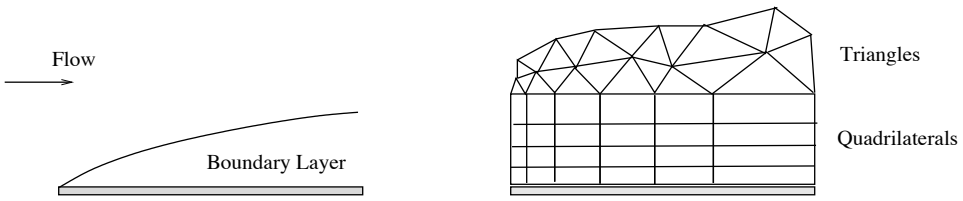


Figure 2.9: Hybrid Mesh in Boundary Layer

on what we wish to achieve.

2.3 Discretization Methods

So far, we have alluded to the discretization method, but have not said specifically what method we will use to convert our general transport equation to a set of discrete algebraic equations. A number of popular methods are available for doing this.

2.3.1 Finite Difference Methods

Finite difference methods approximate the derivatives in the governing differential equation using truncated Taylor series expansions. Consider a one-dimensional scalar transport equation with a constant diffusion coefficient and no unsteady or convective terms:

$$\Gamma \frac{d^2\phi}{dx^2} + S = 0 \tag{2.1}$$

We wish to discretize the diffusion term. Referring to the one-dimensional mesh shown in Figure 2.10, we write

$$\phi_1 = \phi_2 - \Delta x \left(\frac{d\phi}{dx}\right)_2 + \frac{(\Delta x)^2}{2}\left(\frac{d^2\phi}{dx^2}\right)_2 + O\left((\Delta x)^3\right) \tag{2.2}$$

and

$$\phi_3 = \phi_2 + \Delta x \left(\frac{d\phi}{dx}\right)_2 + \frac{(\Delta x)^2}{2}\left(\frac{d^2\phi}{dx^2}\right)_2 + O\left((\Delta x)^3\right) \tag{2.3}$$

The term $O((\Delta x)^3)$ indicates that the terms that follow have a dependence on $(\Delta x)^n$ where $n > 3$. Subtracting Equations 2.2 from Equation 2.3 gives

$$\left(\frac{d\phi}{dx}\right)_2 = \frac{\phi_3 - \phi_1}{2\Delta x} + O((\Delta x)^2) \tag{2.4}$$

By adding the two equations together, we can write

$$\left(\frac{d^2\phi}{dx^2}\right)_2 = \frac{\phi_1 + \phi_3 - 2\phi_2}{\Delta x^2} + O((\Delta x)^2) \tag{2.5}$$

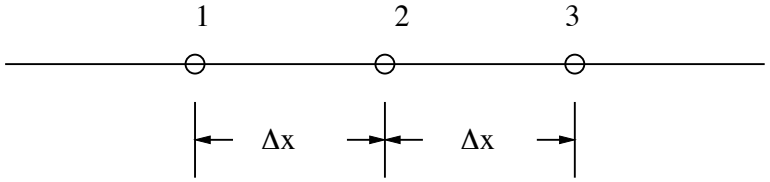


Figure 2.10: One-Dimensional Mesh

By including the diffusion coefficient and dropping terms of $O((\Delta x)^2)$ or smaller, we can write

$$\Gamma\left(\frac{d^2\phi}{dx^2}\right)_2 = \Gamma\frac{\phi_1+\phi_3-2\phi_2}{\Delta x^2} \tag{2.6}$$

The source term S is evaluated at the point 2 using

$$S_2 = S\left(\phi_2\right) \tag{2.7}$$

Substituting Equations 2.6 and 2.7 into Equation 2.1 gives the equation

$$\frac{2\Gamma}{(\Delta x)^2}\phi_2 = \frac{\Gamma}{(\Delta x)^2}\phi_1 + \frac{\Gamma}{(\Delta x)^2}\phi_3 + S_2 \tag{2.8}$$

This is the discrete form of Equation 2.1. By obtaining an equation like this for every point in the mesh, we obtain an set of algebraic equations in the discrete values of ϕ. This equation set may be solved by a variety of methods which we will discuss later in the book.

Finite difference methods do not explicitly exploit the conservation principle in deriving discrete equations. Though they yield discrete equations that look similar to other methods for simple cases, they are not guaranteed to do so in more complicated cases, for example on unstructured meshes.

2.3.2 Finite Element Methods

We consider again the one-dimensional diffusion equation, Equation 2.1. There are different kinds of finite element methods. Let us look at a popular variant, the Galerkin finite element method. Let $\overline{\phi}$ be an approximation to ϕ. Since $\overline{\phi}$ is only an approximation, it does not satisfy Equation 2.1 exactly, so that there is a residual R:

$$\frac{d^2\overline{\phi}}{dx^2} + S = R \tag{2.9}$$

We wish to find a $\overline{\phi}$ such that

$$\int_{\text{domain}} WR\,dx = 0 \tag{2.10}$$

W is a weight function, and Equation 2.10 requires that the residual R become zero in a weighted sense. In order to generate a set of discrete equations we use a family of

weight functions W_i, $i = 1,2,...N$, where N is the number of grid points, rather than a single weight function. Thus, we require

$$\int_{\text{domain}} W_i R dx = 0 \quad i = 1,2,...N \tag{2.11}$$

The weight functions W_i are typically local in that they are non-zero over element i, but are zero everywhere else in the domain. Further, we assume a *shape function* for $\overline{\phi}$, i.e., assume how $\overline{\phi}$ varies between nodes. Typically this variation is also local. For example we may assume that ϕ assumes a piece-wise linear profile between points 1 and 2 and between points 2 and 3 in Figure 2.10. The Galerkin finite element method requires that the weight and shape functions be the same. Performing the integration in Equation 2.11 results in a set of algebraic equations in the nodal values of ϕ which may be solved by a variety of methods.

We should note here that because the Galerkin finite element method only requires the residual to be zero in some weighted sense, it does not enforce the conservation principle in its original form. We now turn to a method which employs conservation as a tool for developing discrete equations.

2.3.3 Finite Volume Method

The finite volume method (sometimes called the control volume method) divides the domain in to a finite number of non-overlapping cells or control volumes over which conservation of ϕ is enforced in a discrete sense. It is possible to start the discretization process with a direct statement of conservation on the control volume, as in Equation 1.9 in the previous chapter. Alternatively we may start with the differential equation and integrate it over the control volume. Let us examine the discretization process by looking at one-dimensional diffusion with a source term:

$$\frac{d}{dx}\left(\Gamma \frac{d\phi}{dx}\right) + S = 0 \tag{2.12}$$

Consider a one-dimensional mesh, with cells as shown in Figure 2.11. Let us store discrete values of ϕ at cell centroids, denoted by W, P and E. The cell faces are denoted by w and e. Let us assume the face areas to be unity.

We focus on the cell associated with P. We start by integrating Equation 2.12 over the cell P. This yields

$$\int_w^e \frac{d}{dx}\left(\Gamma \frac{d\phi}{dx}\right) dx + \int_w^e S dx = 0 \tag{2.13}$$

so that

$$\left(\Gamma \frac{d\phi}{dx}\right)_e - \left(\Gamma \frac{d\phi}{dx}\right)_w + \int_w^e S dx = 0 \tag{2.14}$$

We note that this equation can also be obtained by writing a heat balance over the cell P from first principles. Thus far, we have made no approximation.

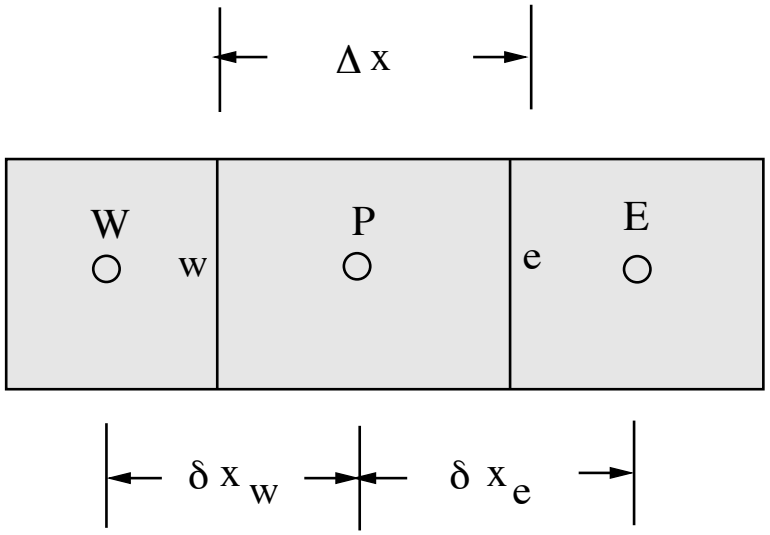


Figure 2.11: Arrangement of Control Volumes

We now make a profile assumption, i.e., we make an assumption about how ϕ varies between cell centroids. If we assume that ϕ varies linearly between cell centroids, we may write

$$\frac{\Gamma_e(\phi_E-\phi_P)}{(\delta x_e)}-\frac{\Gamma_w(\phi_P-\phi_W)}{(\delta x_w)}+\overline{S}\Delta x=0 \tag{2.15}$$

Here $\overline{S}$ is the average value of S in the control volume. We note that the above equation is no longer exact because of the approximation in assuming that ϕ varies in a piecewise linear fashion between grid points.

Collecting terms, we obtain

$$a_P\phi_P=a_E\phi_E+a_W\phi_W+b \tag{2.16}$$

where

$$\begin{aligned} a_E &= \Gamma_e/(\delta x_e) \\ a_W &= \Gamma_w/(\delta x_w) \\ a_P &= a_E+a_W \\ b &= \overline{S}\Delta x \end{aligned} \tag{2.17}$$

Equations similar to Equation 2.16 may be derived for all cells in the domain, yielding a set of algebraic equations, as before; these may be solved using a variety of direct or iterative methods.

We note the following about the discretization process.

1. The process starts with the statement of conservation over the cell. We then find cell values of ϕ which satisfy this conservation statement. Thus conservation is guaranteed for each cell, regardless of mesh size.

2. Conservation does not guarantee accuracy, however. The solution for ϕ may be inaccurate, but conservative.

3. The quantity $-(\Gamma d\phi/dx)_e$ is diffusion flux on the e face. The cell balance is written in terms of *face* fluxes. The gradient of ϕ must therefore be evaluated at the *faces* of the cell.

4. The profile assumptions for ϕ and S need not be the same.

We will examine additional properties of this discretization in the next chapter.

2.4 Solution of Discretization Equations

All the discretization methods described here result in a set of discrete algebraic equations which must be solved to obtain the discrete values of ϕ. These equations may be linear (i.e. the coefficients are independent of ϕ) or they may be non-linear (i.e. the coefficients are functions of ϕ). The solution techniques are independent of the discretization method, and represent the *path to solution*. For the linear algebraic sets we will encounter in this book, we are guaranteed that there is only one solution, and if our solution method gives us a solution, it is the solution we want. All solution methods (i.e. all paths to solution) which arrive at a solution will give us the same solution for the same set of discrete equations. For non-linear problems, we do not have this guarantee, and the answer we get may depend on factors like the initial guess, and the actual path to solution. Though this is an important issue in computing fluid flows, we will not address it here.

Solution methods may be broadly classified as direct or iterative. We consider each briefly below.

2.4.1 Direct Methods

Using one of the discretization methods described previously, we may write the resulting system of algebraic equations as

$$\mathbf{A}\phi = \mathbf{B} \tag{2.18}$$

where $\mathbf{A}$ is the coefficient matrix, $\phi = [\phi_1, \phi_2, ...]^T$ is a vector consisting of the discrete values of ϕ, and $\mathbf{B}$ is the vector resulting from the source terms.

Direct methods solve the equation set 2.18 using the methods of linear algebra. The simplest direct method is inversion, whereby ϕ is computed from

$$\phi = \mathbf{A}^{-1}\mathbf{B} \tag{2.19}$$

A solution for ϕ is guaranteed if $\mathbf{A}^{-1}$ can be found. However, the operation count for the inversion of an $N \times N$ matrix is $O(N^2)$. Consequently, inversion is almost never employed in practical problems. More efficient methods for linear systems are available. For the discretization methods of interest here, $\mathbf{A}$ is sparse, and for structured meshes it is banded. For certain types of equations, for example, for pure diffusion, the

matrix is symmetric. Matrix manipulation can take into account the special structure of **A** in devising efficient solution techniques for Equation 2.18. We will study one such method, the tri-diagonal matrix algorithm (TDMA), in a later chapter.

Direct methods are not widely used in computational fluid dynamics because of large computational and storage requirements. Most industrial CFD problems today involve hundreds of thousands of cells, with 5-10 unknowns per cell even for simple problems. Thus the matrix **A** is usually very large, and most direct methods become impractical for these large problems. Furthermore, the matrix **A** is usually non-linear, so that the direct method must be embedded within an iterative loop to update non-linearities in **A**. Thus, the direct method is applied over and over again, making it all the more time-consuming.

2.4.2 Iterative Methods

Iterative methods are the most widely used solution methods in computational fluid dynamics. These methods employ a guess-and-correct philosophy which progressively improves the guessed solution by repeated application of the discrete equations. Let us consider an extremely simple iterative method, the Gauss-Seidel method. The overall solution loop for the Gauss-Seidel method may be written as follows:

1. Guess the discrete values of ϕ at all grid points in the domain.

2. Visit each grid point in turn. Update ϕ using

$$\phi_P = \frac{(a_E \phi_E + a_W \phi_W + b)}{a_P} \tag{2.20}$$

   The neighbor values, ϕ_E and ϕ_W are required for the update of ϕ_P. These are assumed known at prevailing values. Thus, points which have already been visited will have recently updated values of ϕ and those that have not will have old values.

3. Sweep the domain until all grid points are covered. This completes one iteration.

4. Check if an appropriate convergence criterion is met. We may, for example, require that the maximum change in the grid-point values of ϕ be less than 0.1 %. If the criterion is met, stop. Else, go to step 2.

The iteration procedure described here is not guaranteed to converge to a solution for arbitrary combinations of a_P, a_E and a_W. Convergence of the process is guaranteed for linear problems if the *Scarborough criterion* is satisfied. The Scarborough criterion requires that

$$\begin{aligned} \frac{|a_E| + |a_W|}{|a_P|} &\leq 1 \quad \text{for all grid points} \\ &< 1 \quad \text{for at least one grid point} \end{aligned} \tag{2.21}$$

Matrices which satisfy the Scarborough criterion have *diagonal dominance*. We note that direct methods do not require the Scarborough criterion to be satisfied to obtain a

solution; we can always obtain a solution to our linear set of equations as long as our coefficient matrix is not singular.

The Gauss-Seidel scheme can be implemented with very little storage. All that is required is storage for the discrete values of ϕ at the grid points. The coefficients a_P, a_E, a_W and b can be computed on the fly if desired, since the entire coefficient matrix for the domain is not required when updating the value of ϕ at any grid point. Also, the iterative nature of the scheme makes it particularly suitable for non-linear problems. If the coefficients depend on ϕ, they may be updated using prevailing values of ϕ as the iterations proceed.

Nevertheless, the Gauss-Seidel scheme is rarely used in practice for solving the systems encountered in CFD. The rate of convergence of the scheme decreases to unacceptably low levels if the system of equations is large. In a later chapter, we will use a *multigrid method* to accelerate the rate of convergence of this scheme and make it usable as a practical tool.

2.5 Accuracy, Consistency, Stability and Convergence

In this section, we turn to certain important properties of numerical methods.

2.5.1 Accuracy

Accuracy refers to the correctness of a numerical solution when compared to an exact solution. In most cases, we do not know the exact solution. It is therefore more useful to talk of the *truncation error* of a discretization method. The truncation error associated with the diffusion term using the finite difference method is $O((\Delta x)^2)$, as shown by Equation 2.5.This simply says that if $d^2\phi/dx^2$ is represented by the first term in Equation 2.5, the terms that are neglected are of $O((\Delta x)^2)$. Thus, if we refine the mesh, we expect the truncation error to decrease as $(\Delta x)^2$. If we double the x-direction mesh, we expect the truncation error to decrease by a factor of four. The truncation error of a discretization scheme is the largest truncation error of each of the individual terms in the equation being discretized. The *order* of a discretization method is n if its truncation error is $O((\Delta x)^n)$. It is important to understand that the truncation error tells us how fast the error will decrease with mesh refinement, but is not an indicator of how high the error is on the current mesh. Thus, even methods of very high order may yield inaccurate results on a given mesh. However, we are guaranteed that the error will decrease more rapidly with mesh refinement than with a discretization method of lower order.

2.5.2 Consistency

A *consistent* numerical method is one for which the truncation error tends to vanish as the mesh becomes finer and finer. (For unsteady problems, both spatial and temporal truncation errors must be considered). We are guaranteed this if the truncation error is some power of the mesh spacing Δx (or Δt). Sometimes we may come across schemes

where the truncation error of the method is $O(\Delta x/\Delta t)$. Here, consistency is not guaranteed unless Δx is decreased faster than Δt. Consistency is a very important property. Without it, we have no guarantee that mesh refinement will improve our solution.

2.5.3 Stability

The previous two properties refer to the behavior of the discretization method. Stability is a property of the *path to solution*. For steady state problems, for example, we obtain a discretized set of algebraic equations which must be solved. We may choose to solve this set using an iterative method. Depending on the properties of the method, solution errors may be amplified or damped. An iterative solution method is unstable or divergent if it fails to yield a solution to the discrete set.

It is also possible to speak of the stability of time-marching schemes. When solving unsteady problems, we will use numerical methods which compute the solution at discrete instants of time, using the solution at one or more previous time steps as initial conditions. Stability analysis allow us to determine whether errors in the solution remain bounded as time marching proceeds. An unstable time-marching scheme would not be able to reach steady state in an unsteady heat conduction problem, for example (assuming that a steady state exists).

It is possible to analyze iterative and time marching methods using *stability analysis*. However, this is most convenient for linear problems, and is usually too difficult for most realistic problems. Here, non-linearities in the governing equations, boundary conditions, and properties, as well as coupling between multiple governing equations, make a formal analysis difficult. In reality the practitioner of CFD must rely on experience and intuition in devising stable solution methods.

2.5.4 Convergence

We distinguish between two popular usages of the term convergence. We may say that an iterative method has converged to a solution, or that we have obtained convergence using a particular method. By this we mean that our iterative method has successfully obtained a solution to our discrete algebraic equation set. We may also speak of convergence to mesh independence. By this, we mean the process of mesh refinement, and its use in obtaining solutions that are essentially invariant with further refinement. We shall use the term in both senses in this book.

2.6 Closure

In this chapter, we have presented a broad overview of discretization and introduced terminology associated with numerical methods. We have learned that there are a number of different philosophies for discretizing the scalar transport equation. Of these, only the finite volume method enforces conservation on each cell, and thus ensures that both local and global conservation are guaranteed no matter how coarse the mesh. In the next chapter, we consider the finite volume method in more detail, and study the properties of the discretizations it produces when applied to diffusion problems.

Chapter 3

The Diffusion Equation: A First Look

In this chapter we turn our attention to an important physical process, namely diffusion. Diffusion operators are common in heat, mass and momentum transfer and can also be used to model electrostatics, radiation, and other physics. We consider the discretization and solution of the scalar transport equation for both steady and unsteady diffusion problems. We will attempt to relate the properties of our discrete equations with the behavior of the canonical partial differential equations we studied previously. The methodology we develop in this chapter will allow us to examine more complicated mesh types and physics in later chapters.

3.1 Two-Dimensional Diffusion in Rectangular Domain

Let us consider the steady two-dimensional diffusion of a scalar ϕ in a rectangular domain. From Equation 1.10, the governing scalar transport equation may be written as

$$\nabla \cdot \mathbf{J} = S \tag{3.1}$$

where $\mathbf{J} = J_x\mathbf{i} + J_y\mathbf{j}$ is the diffusion flux vector and is given by

$$\mathbf{J} = -\Gamma \nabla \phi \tag{3.2}$$

In Cartesian geometries, the gradient operator is given by

$$\nabla = \frac{\partial}{\partial x}\mathbf{i} + \frac{\partial}{\partial y}\mathbf{j} \tag{3.3}$$

We note that Equation 3.1 is written in conservative or divergence form. When Γ is constant and S is zero, the equation defaults to the familiar Laplace equation. When Γ is constant and S is non-zero, the Poisson equation is obtained.

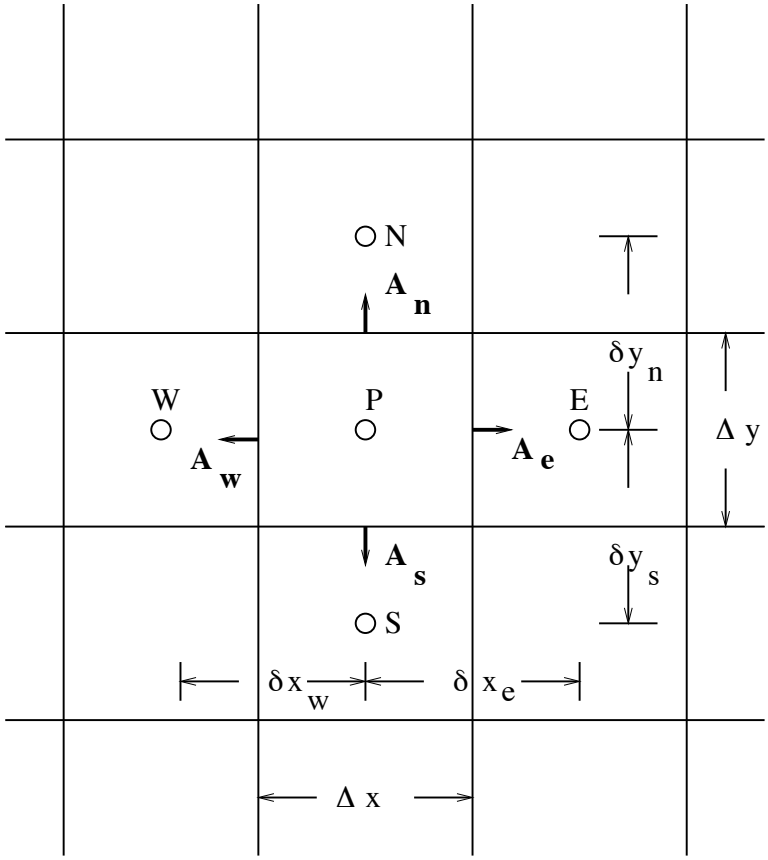


Figure 3.1: Two-Dimensional Control Volume

3.1.1 Discretization

The arrangement of cells under consideration is shown in Figure 3.1. As in the previous chapter, we focus on cell P and its neighbors, the cells E, W, N and S. Discrete values of ϕ are stored at cell centroids. We also store the diffusion coefficient Γ at cell centroids. The faces e,w, n and s are associated with area vectors $\mathbf{A}_e$, $\mathbf{A}_w$, $\mathbf{A}_n$ and $\mathbf{A}_s$. The vectors are positive pointing outwards from the cell P. The volume of the cell P is $\Delta\mathscr{V} = \Delta x \times \Delta y$.

We begin the process of discretization by integrating Equation 3.1 over the cell P:

$$\int_{\Delta\mathscr{V}} \nabla \cdot \mathbf{J} d\mathscr{V} = \int_{\Delta\mathscr{V}} S d\mathscr{V} \tag{3.4}$$

Next, we apply the divergence theorem to yield

$$\int_A \mathbf{J} \cdot d\mathbf{A} = \int_{\Delta\mathscr{V}} S d\mathscr{V} \tag{3.5}$$

The first integral represents the integral over the control surface A of the cell. We have made no approximations thus far.

We now make a profile assumption about the flux vector $\mathbf{J}$. We assume that $\mathbf{J}$ varies linearly over each face of the cell P, so that it may be represented by its value at the face centroid. We also assume that the mean value of the source term S over the control volume is $\overline{S}$. Thus,

$$(\mathbf{J}\cdot\mathbf{A})_e + (\mathbf{J}\cdot\mathbf{A})_w + (\mathbf{J}\cdot\mathbf{A})_n + (\mathbf{J}\cdot\mathbf{A})_s = \overline{S}\Delta\mathscr{V} \tag{3.6}$$

or, more compactly

$$\sum_{\text{f=e,w,n,s}} \mathbf{J}_f \cdot \mathbf{A}_f = \overline{S}\Delta\mathscr{V} \tag{3.7}$$

The face areas $\mathbf{A}_e$ and $\mathbf{A}_w$ are given by

$$\begin{aligned} \mathbf{A}_e &= \Delta y\, \mathbf{i} \\ \mathbf{A}_w &= -\Delta y\, \mathbf{i} \end{aligned} \tag{3.8}$$

The other area vectors may be written analogously. Further

$$\begin{aligned} \mathbf{J}_e \cdot \mathbf{A}_e &= -\Gamma_e \Delta y \left(\frac{\partial \phi}{\partial x}\right)_e \\ \mathbf{J}_w \cdot \mathbf{A}_w &= \Gamma_w \Delta y \left(\frac{\partial \phi}{\partial x}\right)_w \end{aligned} \tag{3.9}$$

The transport in the other directions may be written analogously.

In order to complete the discretization process, we make one more round of profile assumptions. We assume that ϕ varies linearly between cell centroids. Thus, Equation 3.9 may be written as

$$\begin{aligned} \mathbf{J}_e \cdot \mathbf{A}_e &= -\Gamma_e \Delta y \frac{\phi_E - \phi_P}{(\delta x)_e} \\ \mathbf{J}_w \cdot \mathbf{A}_w &= \Gamma_w \Delta y \frac{\phi_P - \phi_W}{(\delta x)_w} \end{aligned} \tag{3.10}$$

Similar expressions may be written for the other fluxes.

Let us assume that the source term S has the form

$$S = S_C + S_P \phi \tag{3.11}$$

with $S_P \leq 0$. We say that S has been *linearized*. We will see later how general forms of S can be written in this way. We write the volume-averaged source term $\overline{S}$ in the cell P as

$$\overline{S} = S_C + S_P \phi_P \tag{3.12}$$

Substituting Equations 3.10, 3.8 and 3.12 into Equation 3.6 yields a discrete equation for ϕ_P:

$$a_P \phi_P = a_E \phi_E + a_W \phi_W + a_N \phi_N + a_S \phi_S + b \tag{3.13}$$

where

$$\begin{aligned} a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\ a_W &= \frac{\Gamma_w \Delta y}{(\delta x)_w} \\ a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\ a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\ a_P &= a_E + a_W + a_N + a_S - S_P \Delta x \Delta y \\ b &= S_C \Delta x \Delta y \end{aligned} \tag{3.14}$$

Equation 3.13 may be written in a more compact form as

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b \tag{3.15}$$

Here, the subscript *nb* denotes the cell neighbors E, W, N, and S.

3.1.2 Discussion

We make the following important points about the discretization we have done so far:

1. The discrete equation expresses a balance of discrete flux (the **J**'s) and the source term. Thus conservation over individual control volumes is guaranteed. However, overall conservation in the calculation domain is not guaranteed unless the diffusion transfer from one cell enters the next cell. For example, in writing the balance for cell E, we must ensure that the flux used on the face e is $\mathbf{J}_e$, and that it is discretized *exactly* as in Equation 3.10.

2. The coefficients a_P and a_{nb} are all of the same sign. In our case they are all positive. This has physical meaning. If the temperature at E is increased, we would expect the temperature at P to increase, not decrease. (The solution to our elliptic partial differential equation also has this property). Many higher order schemes do not have this property. This does not mean these schemes are wrong – it means they do not have a property we would like to have if at all possible.

3. We require that S_P in Equation 3.11 be negative. This also has physical meaning. If for example S is a temperature source, we do not want a situation where as T increases, S increases indefinitely. We control this behavior in the numerical scheme by insisting that S_P be kept negative.

4. When $S_P = 0$, we have

$$a_P = \sum_{\text{nb}} a_{\text{nb}} \tag{3.16}$$

Equation 3.13 may then be written as

$$\phi_P = \sum_{\text{nb}} \left(\frac{a_{\text{nb}}}{a_P} \phi_{\text{nb}} \right) \tag{3.17}$$

where $\sum_{\text{nb}} (a_{\text{nb}}/a_P) = 1$. Since ϕ_P is the weighted sum of its neighbor values, it is always bounded by them. By extension, ϕ_P is always bounded by the boundary values of ϕ. We notice that this property is also shared by our canonical elliptic equation.

When $S \neq 0$, ϕ_P need not be bounded in this manner, and can overshoot or undershoot its boundary values, but this is perfectly physical. The amount of overshoot is determined by the magnitude of S_C and S_P with respect to the a_{nb}'s.

5. If $S_P = 0$ and $a_P = \sum_{\text{nb}} a_{\text{nb}}$, we notice that ϕ and $\phi + C$ are solutions to Equation 3.13. This is also true of the original differential equation, Equation 3.1. The solution can be made unique by specifying boundary conditions on ϕ which fix the value of ϕ at some point on the boundary.

3.2 Boundary Conditions

A typical boundary control volume is shown in Figure 3.2. A boundary control volume is one which has one or more faces on the boundary. Discrete values of ϕ are stored at cell centroids, as before. In addition, we store discrete values of ϕ at the centroids of boundary faces.

Let us consider the discretization process for a near-boundary control volume centered about the cell centroid P with a face on the boundary. The boundary face centroid is denoted by b. The face area vector of the boundary face is $\mathbf{A}_b$, and points outward from the cell P as shown.

Integrating the governing transport equation over the cell P as before yields

$$(\mathbf{J} \cdot \mathbf{A})_b + (\mathbf{J} \cdot \mathbf{A})_e + (\mathbf{J} \cdot \mathbf{A})_n + (\mathbf{J} \cdot \mathbf{A})_s = \overline{S} \Delta \mathscr{V} \tag{3.18}$$

The fluxes on the interior faces are discretized as before. The boundary area vector $\mathbf{A}_b$ is given by

$$\mathbf{A}_b = -\Delta y \, \mathbf{i} \tag{3.19}$$

Let us assume that the boundary flux $\mathbf{J}_b$ is given by the boundary face centroid value. Thus

$$\mathbf{J}_b = -\Gamma_b \nabla \phi_b \tag{3.20}$$

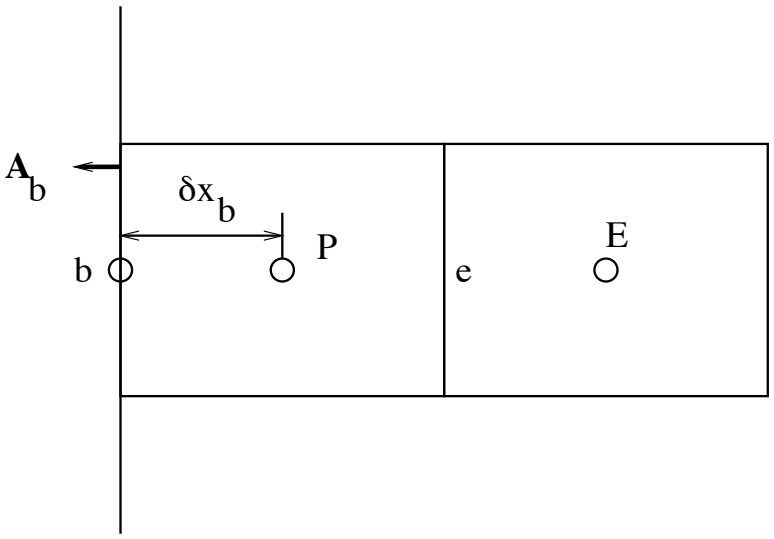


Figure 3.2: Boundary Control Volume

so that

$$\mathbf{J}_b \cdot \mathbf{A}_b = \Delta y \Gamma_b \nabla \phi_b \tag{3.21}$$

Assuming that ϕ varies linearly between b and P, we write

$$\mathbf{J}_b \cdot \mathbf{A}_b = \Delta y \Gamma_b \frac{(\phi_P - \phi_b)}{(\delta x)_b} \tag{3.22}$$

The specification of boundary conditions involves either specifying the unknown boundary value ϕ_b, or alternatively, the boundary flux $\mathbf{J}_b$. Let us consider some common boundary conditions next.

3.2.1 Dirichlet Boundary Condition

The boundary condition is given by

$$\phi_b = \phi_{b,\text{given}} \tag{3.23}$$

Using $\phi_{b,\text{given}}$ in Equation 3.22, and including $\mathbf{J_b} \cdot \mathbf{A}_b$ in Equation 3.18 yields the following discrete equation for boundary cell P:

$$a_P \phi_P = a_E \phi_E + a_N \phi_N + a_S \phi_S + b \tag{3.24}$$

where

$$\begin{aligned} a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\ a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\ a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\ a_b &= \frac{\Gamma_b \Delta y}{(\delta x)_b} \\ a_P &= a_E + a_N + a_S + a_b - S_P \Delta x \Delta y \\ b &= a_b \phi_b + S_C \Delta x \Delta y \end{aligned} \tag{3.25}$$

We note the following important points about the above discretization:

1. At Dirichlet boundaries, $a_P > (a_E + a_N + a_S)$. This property ensures that the Scarborough criterion is satisfied for problems with Dirichlet boundary conditions.

2. ϕ_P is guaranteed to be bounded by the values of ϕ_E, ϕ_N, ϕ_S and ϕ_b if S_C and S_P are zero. This is in keeping with the behavior of the canonical elliptic partial differential equation we encountered earlier.

3.2.2 Neumann Boundary Condition

Here, we are given the normal gradient of ϕ at the boundary:

$$-(\Gamma \nabla \phi)_b \cdot \mathbf{i} = q_{b,\text{given}} \tag{3.26}$$

We are in effect given the flux $\mathbf{J}_b$ at Neumann boundaries:

$$\mathbf{J}_b \cdot \mathbf{A}_b = -q_{b,\text{given}} \Delta y \tag{3.27}$$

We may thus include $(-q_{b,\text{given}} \Delta y)$ directly in Equation 3.18 to yield the following discrete equation for the boundary cell P:

$$a_P \phi_P = a_E \phi_E + a_N \phi_N + a_S \phi_S + b \tag{3.28}$$

where

$$\begin{aligned} a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\ a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\ a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\ a_P &= a_E + a_N + a_S - S_P \Delta x \Delta y \\ b &= q_{b,\text{given}} \Delta y + S_C \Delta x \Delta y \end{aligned} \tag{3.29}$$

We note the following about the discretization at the boundary cell P:

1. $a_P = (a_E + a_N + a_S)$ at Neumann boundaries if $S = 0$.

2. If both $q_{b,\text{given}}$ and S are zero, ϕ_P is bounded by its neighbors. Otherwise, ϕ_P can exceed (or fall below) the neighbor values of ϕ. This is admissible. If heat were being added at the boundary, for example, we would expect the temperature in the region close to the boundary to be higher than that in the interior.

3. Once ϕ_P is computed, the boundary value, ϕ_b may be computed using Equation 3.22:

$$\phi_b = \frac{q_{b,\text{given}} + (\Gamma_b/\delta x_b)\phi_P}{(\Gamma_b/\delta x_b)} \tag{3.30}$$

3.2.3 Mixed Boundary Condition

The mixed boundary condition is given by

$$-(\Gamma\nabla\phi)_b \cdot \mathbf{i} = h_b(\phi_\infty - \phi) \tag{3.31}$$

Since $\mathbf{A}_b = \Delta y\mathbf{i}$, we are given that

$$\mathbf{J}_b \cdot \mathbf{A}_b = -h_b(\phi_\infty - \phi)\Delta y \tag{3.32}$$

Using Equation 3.22 we may write

$$\Gamma_b \frac{(\phi_P - \phi_b)}{\delta x_b} = -h_b(\phi_\infty - \phi_b) \tag{3.33}$$

We may thus write ϕ_b as

$$\phi_b = \frac{h_b\phi_\infty + (\Gamma_b/\delta x_b)\phi_P}{h_b + (\Gamma_b/\delta x_b)} \tag{3.34}$$

Using Equation 3.34 to eliminate ϕ_b from Equation 3.33 we may write

$$\mathbf{J}_b \cdot \mathbf{A}_b = -R_{eq}(\phi_\infty - \phi_P)\Delta y \tag{3.35}$$

where

$$R_{eq} = \frac{h_b(\Gamma_b/\delta x_b)}{h_b + (\Gamma_b/\delta x_b)} \tag{3.36}$$

We are now ready to write the discretization equation for the boundary control volume P. Substituting the boundary flux from Equation 3.35 into Equation 3.18 given the following discrete equation for ϕ_P:

$$a_P\phi_P = a_E\phi_E + a_N\phi_N + a_S\phi_S + b \tag{3.37}$$

where

$$\begin{aligned} a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\ a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\ a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\ a_b &= R_{eq} \Delta y \\ a_P &= a_E + a_N + a_S + a_b - S_P \Delta x \Delta y \\ b &= R_{eq} \Delta y \phi_\infty + S_C \Delta x \Delta y \end{aligned} \tag{3.38}$$

We note the following about the above discretization:

1. $a_P > (a_E + a_N + a_S)$ for the boundary cell P if $S = 0$. Thus, mixed boundaries are like Dirichlet boundaries in that the boundary condition helps ensure that the Scarborough criterion is met.

2. The cell value ϕ_P is bounded by its neighbor values ϕ_E, ϕ_N and ϕ_S, and the external value, ϕ_∞.

3. The boundary value, ϕ_b, may be computed from Equation 3.34 once a solution has been obtained. It is bounded by ϕ_P and ϕ_∞, as shown by Equation 3.34.

3.3 Unsteady Conduction

Let us now consider the unsteady counterpart of Equation 3.1:

$$\frac{\partial}{\partial t}(\rho\phi) + \nabla \cdot \mathbf{J} = S \tag{3.39}$$

We are given initial conditions $\phi(x, y, 0)$. As we saw in a previous chapter, time is a "marching" coordinate. By knowing the initial condition, and taking discrete time steps Δt, we wish to obtain the solution for ϕ at a each discrete time instant.

In order to discretize Equation 3.39, we integrate it over the control volume as usual. We also integrate it over the time step Δt, i.e., from t to $t + \Delta t$.

$$\int_{\Delta t}\int_{\Delta \mathcal{V}} \frac{\partial}{\partial t}(\rho\phi)\, d\mathcal{V}\, dt + \int_{\Delta t}\int_{\Delta \mathcal{V}} \nabla \cdot \mathbf{J}\, d\mathcal{V}\, dt = \int_{\Delta t}\int_{\Delta \mathcal{V}} S\, d\mathcal{V}\, dt \tag{3.40}$$

Applying the divergence theorem as before, we obtain

$$\int_{\Delta \mathcal{V}} \left((\rho\phi)^1 - (\rho\phi)^0\right) d\mathcal{V} + \int_{\Delta t}\int_A \mathbf{J} \cdot d\mathbf{A}\, dt = \int_{\Delta t}\int_{\Delta \mathcal{V}} S\, d\mathcal{V}\, dt \tag{3.41}$$

The superscripts 1 and 0 in the first integral denote the values at the times $t + \Delta t$ and t respectively. Let us consider each term in turn. If we assume that

$$\int_{\Delta \mathcal{V}} \rho\phi\, d\mathcal{V} = (\rho\phi)_P \Delta \mathcal{V} \tag{3.42}$$

we may write the unsteady term as

$$\Delta\mathcal{V}\left((\rho\phi)_P^1-(\rho\phi)_P^0\right) \tag{3.43}$$

We now turn to the flux term. If we assume as before that the flux on a face is represented by its centroid value, we may write the term as

$$\int_{\Delta t}\sum_{\mathrm{f=e,w,n,s}}\mathbf{J}_f\cdot\mathbf{A}_f dt \tag{3.44}$$

We are now required to make a profile assumption about how the flux $\mathbf{J}$ varies with time. Let us assume that it can be interpolated between time instants $t+\Delta t$ and t using a factor f between zero and one:

$$\int_{\Delta t}\mathbf{J}\cdot\mathbf{A}dt=\left(f\mathbf{J}^1\cdot\mathbf{A}+(1-f)\mathbf{J}^0\cdot\mathbf{A}\right)\Delta t \tag{3.45}$$

Proceeding as before, making linear profile assumptions for ϕ between grid points, we may write

$$\begin{aligned}\mathbf{J}_e^1\cdot\mathbf{A}_e &= -\Gamma_e\Delta y\frac{\phi_E^1-\phi_P^1}{(\delta x)_e}\\ \mathbf{J}_w^1\cdot\mathbf{A}_w &= \Gamma_w\Delta y\frac{\phi_P^1-\phi_W^1}{(\delta x)_w}\end{aligned} \tag{3.46}$$

and

$$\begin{aligned}\mathbf{J}_e^0\cdot\mathbf{A}_e &= -\Gamma_e\Delta y\frac{\phi_E^0-\phi_P^0}{(\delta x)_e}\\ \mathbf{J}_w^0\cdot\mathbf{A}_w &= \Gamma_w\Delta y\frac{\phi_P^0-\phi_W^0}{(\delta x)_w}\end{aligned} \tag{3.47}$$

Let us now examine the source term. Linearizing S as $S_C+S_P\phi$ and further assuming that

$$\int_{\Delta\mathcal{V}}\left(S_C+S_P\phi\right)d\mathcal{V}=\left(S_C+S_P\phi_P\right)\Delta\mathcal{V} \tag{3.48}$$

we have

$$\int_{\Delta t}\int_{\Delta\mathcal{V}}Sd\mathcal{V}dt=\int_{\Delta t}\left(S_C+S_P\phi_P\right)\Delta\mathcal{V}dt \tag{3.49}$$

Again, interpolating S between $t+\Delta t$ and t using a weighting factor f between zero and one:

$$\int_{\Delta t}\left(S_C+S_P\phi_P\right)\Delta\mathcal{V}dt=f\left(S_C+S_P\phi_P\right)^1\Delta\mathcal{V}\Delta t+(1-f)\left(S_C+S_P\phi_P\right)^0\Delta\mathcal{V}\Delta t \tag{3.50}$$

For simplicity, let us drop the superscript 1 and let the un-superscripted value represent the value at time $t+\Delta t$. The values at time t are represented as before with the superscript 0. Collecting terms and dividing through by Δt, we obtain the following discrete equation for ϕ:

$$a_P\phi_P=\sum_{\mathrm{nb}}a_{\mathrm{nb}}\left(f\phi_{\mathrm{nb}}+(1-f)\phi_{\mathrm{nb}}^0\right)+b+\left(a_P^0-(1-f)\sum_{\mathrm{nb}}a_{\mathrm{nb}}\right)\phi_P^0 \tag{3.51}$$

where *nb* denotes E,W, N and S. Further

$$\begin{aligned}
a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\
a_W &= \frac{\Gamma_w \Delta y}{(\delta x)_w} \\
a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\
a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\
a_P^0 &= \frac{\rho \Delta \mathscr{V}}{\Delta t} \\
a_P &= f \sum_{\text{nb}} a_{\text{nb}} - f S_P \Delta \mathscr{V} + a_P^0 \\
b &= \left(f S_C + (1-f) S_C^0 + (1-f) S_P^0 \phi_P^0 \right) \Delta \mathscr{V}
\end{aligned} \tag{3.52}$$

It is useful to examine the behavior of Equation 3.51 for a few limiting cases.

3.3.1 The Explicit Scheme

If we set $f = 0$, we obtain the explicit scheme. This means that the flux and source terms are evaluated using values exclusively from the previous time step. In this limit, we obtain the following discrete equations:

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}}^0 + b + \left(a_P^0 - \sum_{\text{nb}} a_{\text{nb}} \right) \phi_P^0 \tag{3.53}$$

and

$$\begin{aligned}
a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\
a_W &= \frac{\Gamma_w \Delta y}{(\delta x)_w} \\
a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\
a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\
a_P^0 &= \frac{\rho \Delta \mathscr{V}}{\Delta t} \\
a_P &= a_P^0 \\
b &= \left(S_C^0 + S_P^0 \phi_P^0 \right) \Delta \mathscr{V}
\end{aligned} \tag{3.54}$$

We notice the following about the discretization:

1. The right hand side of Equation 3.53 contains values exclusively from the previous time t. Thus, given the condition at time t, it is possible for us to evaluate the right hand side completely, and find the value of ϕ_P at time $t + \Delta t$.

2. We do not need to solve a set of linear algebraic equations to find ϕ_P.

3. When $\Delta t \to \infty$, we see that the discrete equation for steady state is recovered. This is also true when steady state is reached through time marching, i.e., when $\phi_P = \phi_P^0$. Thus, we are assured that our solution upon reaching steady state is the same as that we would have obtained if we had solved a steady problem in the first place.

4. The explicit scheme corresponds to the assumption that ϕ_P^0 prevails over the entire time step.

5. We will see later in the chapter that the explicit scheme has a truncation error of $O(\Delta t)$. Thus, the error reduces only linearly with time-step refinement.

This type of scheme is very simple and convenient, and is frequently used in CFD practice. However, it suffers from a serious drawback. We see that the term multiplying ϕ_P^0 can become negative when

$$a_P^0 < \sum_{\text{nb}} a_{\text{nb}} \tag{3.55}$$

When $a_P^0 < \sum_{\text{nb}} a_{\text{nb}}$, we see that an increase in ϕ at the previous time instant can cause a decrease in ϕ at the current instant. This type of behavior is not possible with a parabolic partial differential equation. We can avoid this by requiring $a_P^0 \geq \sum_{\text{nb}} a_{\text{nb}}$. For a uniform mesh, and constant properties, this restriction can be shown , for one-, two- and three-dimensional cases to be, respectively

$$\Delta t \leq \frac{\rho(\Delta x)^2}{2\Gamma} \tag{3.56}$$

$$\Delta t \leq \frac{\rho(\Delta x)^2}{4\Gamma} \tag{3.57}$$

and

$$\Delta t \leq \frac{\rho(\Delta x)^2}{6\Gamma} \tag{3.58}$$

This condition is sometimes called the *von Neumann* stability criterion in the literature. It requires that the forward time step be limited by the *square* of the mesh size. This dependence on $(\Delta x)^2$ is very restrictive in practice. It requires us to take smaller and smaller time steps as our mesh is refined, leading to very long computational times.

3.3.2 The Fully-Implicit Scheme

The fully-implicit scheme is obtained by setting $f = 1$ in Equation 3.51. In this limit, we obtain the following discrete equation for ϕ_P.

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b + a_P^0 \phi_P^0 \tag{3.59}$$

with

$$\begin{aligned} a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\ a_W &= \frac{\Gamma_w \Delta y}{(\delta x)_w} \\ a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\ a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\ a_P^0 &= \frac{\rho \Delta \mathcal{V}}{\Delta t} \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} - S_P \Delta \mathcal{V} + a_P^0 \\ b &= S_C \Delta \mathcal{V} \end{aligned} \tag{3.60}$$

We note the following important points about the implicit scheme:

1. In the absence of source terms, $a_P = \sum_{\text{nb}} a_{\text{nb}} + a_P^0$. Because of this property, we are guaranteed that ϕ_P is bounded by the values of its spatial neighbors at $t + \Delta t$ and by the value at point P at the previous time. This is in keeping with the behavior of canonical parabolic partial differential equation. We may consider ϕ_P^0 to be the *time neighbor* of ϕ_P. Also, the Scarborough criterion is satisfied.

2. The solution at time $t + \Delta t$ requires the solution of a set of algebraic equations.

3. As with the explicit scheme, as $\Delta t \to \infty$, we recover the discrete equations governing steady state diffusion. Also, if we reach steady state by time marching, i.e., $\phi_P^0 = \phi_P$, we recover the discrete algebraic set governing steady diffusion.

4. The fully-implicit scheme corresponds to assuming that ϕ_P prevails over the entire time step.

5. There is no time step restriction on the fully-implicit scheme. We can take as big a time step as we wish without getting an unrealistic ϕ_P. However, physical plausibility does not imply accuracy – it is possible to get plausible but inaccurate answers if our time steps are too big.

6. We will see later in this chapter that the truncation error of the fully-implicit scheme is $O(\Delta t)$, i.e., it is a first-order accurate scheme. Though the coefficients it produces have useful properties, the rate of error reduction with time step is rather slow.

3.3.3 The Crank-Nicholson Scheme

The Crank-Nicholson Scheme is obtained by setting $f = 0.5$. With this value of f, our discrete equation becomes

$$a_P\phi_P = \sum_{\text{nb}} a_{\text{nb}} \left(0.5\phi_{\text{nb}} + 0.5\phi^0_{\text{nb}}\right) + b + \left(a^0_P - 0.5\sum_{\text{nb}} a_{\text{nb}}\right)\phi^0_P \tag{3.61}$$

with

$$\begin{aligned}
a_E &= \frac{\Gamma_e \Delta y}{(\delta x)_e} \\
a_W &= \frac{\Gamma_w \Delta y}{(\delta x)_w} \\
a_N &= \frac{\Gamma_n \Delta x}{(\delta y)_n} \\
a_S &= \frac{\Gamma_s \Delta x}{(\delta y)_s} \\
a^0_P &= \frac{\rho \Delta \mathscr{V}}{\Delta t} \\
a_P &= 0.5\sum_{\text{nb}} a_{\text{nb}} - 0.5 S_P \Delta \mathscr{V} + a^0_P \\
b &= 0.5\left((S_C + S^0_C) + S^0_P \phi^0_P\right)\Delta \mathscr{V}
\end{aligned} \tag{3.62}$$

We note the following about the Crank-Nicholson scheme:

1. For $a^0_P < 0.5\sum_{\text{nb}} a_{\text{nb}}$ the term multiplying ϕ^0_P becomes negative, leading to the possibility of unphysical solutions. Indeed, any value of f different from one will have this property.

2. The Crank-Nicholson scheme essentially makes a linear assumption about the variation of ϕ_P with time between t and $t + \Delta t$. We will see later in this chapter that though this leads to a possibility of negative coefficients for large time steps, the scheme has a truncation error of $O((\Delta t)^2)$. Consequently, if used with care, the error in the our solutions can be reduced more rapidly with time-step refinement than the other schemes we have encountered thus far.

3.4 Diffusion in Polar Geometries

Since Equation 3.1 is written in vector form, it may be used to describe diffusive transport in other coordinate systems as well. Indeed much of our derivation thus far can be applied with little change to other systems. Let us consider two-dimensional polar geometries next. A typical control volume is shown in Figure 3.3. The control volume is located in the $r - \theta$ plane and is bounded by surfaces of constant r and θ. The grid point P is located at the cell centroid. The volume of the control volume is $\Delta \mathscr{V} = r_P \Delta\theta \Delta r$.

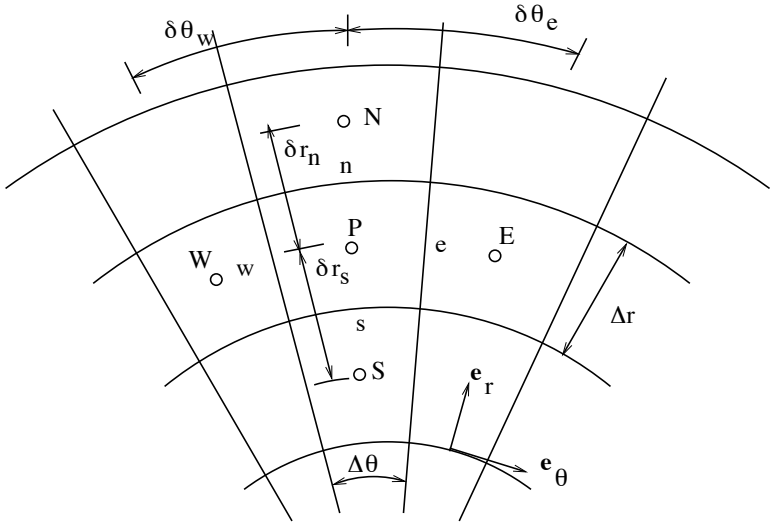


Figure 3.3: Control Volume in Polar Geometry

We assume $\partial\phi/\partial x = 0$, so that all transport is confined to the $r-\theta$ plane. We also assume steady diffusion, though the unsteady counterpart is easily derived.

Integrating Equation 3.1 over the control volume as before, and applying the divergence theorem yields Equation 3.6:

$$(\mathbf{J}\cdot\mathbf{A})_e + (\mathbf{J}\cdot\mathbf{A})_w + (\mathbf{J}\cdot\mathbf{A})_n + (\mathbf{J}\cdot\mathbf{A})_s = \overline{S}\Delta\mathcal{V} \tag{3.63}$$

The face area vectors are given by

$$\begin{aligned}
\mathbf{A}_e &= \Delta r\ \mathbf{e}_{\theta,e} \\
\mathbf{A}_w &= -\Delta r\ \mathbf{e}_{\theta,w} \\
\mathbf{A}_n &= r_n\Delta\theta\ \mathbf{e}_r \\
\mathbf{A}_s &= -r_s\Delta\theta\ \mathbf{e}_r
\end{aligned} \tag{3.64}$$

We recall that the diffusive flux $\mathbf{J}$ is given by

$$\mathbf{J} = -\Gamma\nabla\phi \tag{3.65}$$

For polar geometries the gradient operator is given by

$$\nabla = \frac{\partial}{\partial r}\mathbf{e}_r + \frac{1}{r}\frac{\partial}{\partial\theta}\mathbf{e}_\theta \tag{3.66}$$

Thus the fluxes on the faces are given by

$$\begin{aligned}
\mathbf{J}_e \cdot \mathbf{A}_e &= -\Gamma_e \Delta r \frac{1}{r_e} \left(\frac{\partial \phi}{\partial \theta} \right)_e \\
\mathbf{J}_w \cdot \mathbf{A}_w &= \Gamma_w \Delta r \frac{1}{r_w} \left(\frac{\partial \phi}{\partial \theta} \right)_w \\
\mathbf{J}_n \cdot \mathbf{A}_n &= -\Gamma_n r_n \Delta \theta \left(\frac{\partial \phi}{\partial r} \right)_n \\
\mathbf{J}_s \cdot \mathbf{A}_s &= \Gamma_s r_s \Delta \theta \left(\frac{\partial \phi}{\partial r} \right)_s
\end{aligned} \tag{3.67}$$

Assuming that ϕ varies linearly between grid points yields

$$\begin{aligned}
\mathbf{J}_e \cdot \mathbf{A}_e &= -\Gamma_e \Delta r \frac{\phi_E - \phi_P}{r_e (\delta \theta)_e} \\
\mathbf{J}_w \cdot \mathbf{A}_w &= \Gamma_w \Delta r \frac{\phi_P - \phi_W}{r_w (\delta \theta)_w} \\
\mathbf{J}_n \cdot \mathbf{A}_n &= -\Gamma_n r_n \Delta \theta \frac{\phi_N - \phi_P}{(\delta r)_n} \\
\mathbf{J}_s \cdot \mathbf{A}_s &= \Gamma_s r_s \Delta \theta \frac{\phi_P - \phi_S}{(\delta r)_s}
\end{aligned} \tag{3.68}$$

The source term may be written as

$$(S_C + S_P \phi_P) \Delta \mathscr{V} \tag{3.69}$$

Collecting terms, we may write the discrete equation for the cell P as

$$a_P \phi_P = a_E \phi_E + a_W \phi_W + a_N \phi_N + a_S \phi_S + b \tag{3.70}$$

where

$$\begin{aligned}
a_E &= \frac{\Gamma_e \Delta r}{r_e (\delta \theta)_e} \\
a_W &= \frac{\Gamma_w \Delta r}{r_w (\delta \theta)_w} \\
a_N &= \frac{\Gamma_n r_n \Delta \theta}{(\delta r)_n} \\
a_S &= \frac{\Gamma_s r_s \Delta \theta}{(\delta r)_s} \\
a_P &= a_E + a_W + a_N + a_S - S_P \Delta \mathscr{V} \\
b &= S_C \Delta \mathscr{V}
\end{aligned} \tag{3.71}$$

We note the following about the above discretization:

1. Regardless of the shape of the control volume, the basic process is the same. The integration of the conservation equation over the control volume and the application of the divergence theorem results in a control volume balance equation, regardless of the shape of the control volume.

2. The only differences are in the form of the face area vectors and the gradient operator for the coordinate system. The latter manifests itself in the expressions for the distances between grid points.

3. The polar coordinate system is *orthogonal*, i.e., $\mathbf{e}_r$ and $\mathbf{e}_\theta$ are always perpendicular to each other. Because the control volume faces are aligned with the coordinate directions, the line joining the cell centroids (P and E, for example) is perpendicular the face (e, for example). As a result, the flux normal to the face can be written purely in terms of the cell centroid ϕ values for the two cells sharing the face. We will see in the next chapter that additional terms appear when the mesh is non-orthogonal, i.e., when the line joining the cell centroids is not perpendicular to the face.

3.5 Diffusion in Axisymmetric Geometries

A similar procedure can be used to derive the discrete equation for axisymmetric geometries. We assume steady conduction. Since the problem is axisymmetric, $\partial\phi/\partial\theta = 0$. A typical control volume is shown in Figure 3.4, and is located in the $r-x$ plane. The grid point P is located at the cell centroid. The volume of the control volume is $\Delta\mathscr{V} = r_P \Delta r \Delta x$.

The face area vectors are given by

$$\begin{aligned} \mathbf{A}_e &= r_e \Delta r\, \mathbf{i} \\ \mathbf{A}_w &= -r_w \Delta r\, \mathbf{i} \\ \mathbf{A}_n &= r_n \Delta x\, \mathbf{e}_r \\ \mathbf{A}_s &= -r_s \Delta x\, \mathbf{e}_r \end{aligned} \tag{3.72}$$

For axisymmetric geometries the gradient operator is given by

$$\nabla = \frac{\partial}{\partial r}\mathbf{e}_r + \frac{\partial}{\partial x}\mathbf{i} \tag{3.73}$$

Thus the fluxes on the faces are given by

$$\begin{aligned} \mathbf{J}_e \cdot \mathbf{A}_e &= -\Gamma_e r_e \Delta r \left(\frac{\partial\phi}{\partial x}\right)_e \\ \mathbf{J}_w \cdot \mathbf{A}_w &= \Gamma_w r_w \Delta r \left(\frac{\partial\phi}{\partial x}\right)_w \\ \mathbf{J}_n \cdot \mathbf{A}_n &= -\Gamma_n r_n \Delta x \left(\frac{\partial\phi}{\partial r}\right)_n \\ \mathbf{J}_s \cdot \mathbf{A}_s &= \Gamma_s r_s \Delta x \left(\frac{\partial\phi}{\partial r}\right)_s \end{aligned} \tag{3.74}$$

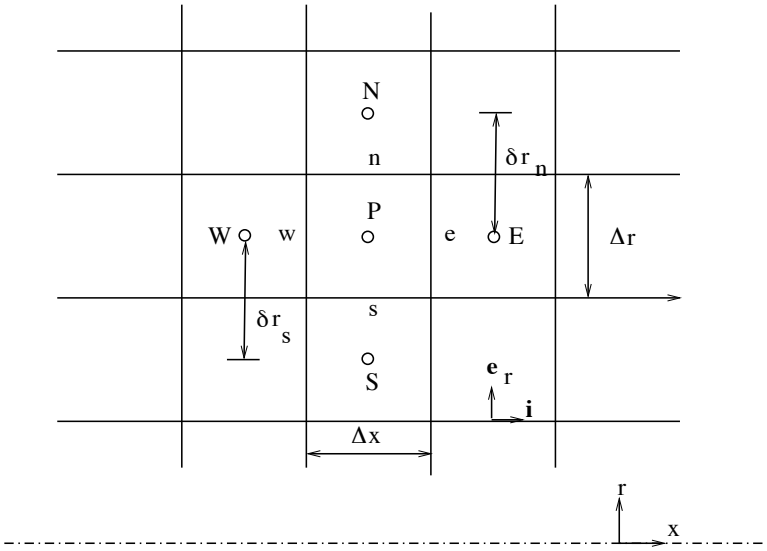


Figure 3.4: Control Volume in Axisymmetric Geometry

Assuming that ϕ varies linearly between grid points yields

$$\begin{aligned}
\mathbf{J}_e \cdot \mathbf{A}_e &= -\Gamma_e r_e \Delta r \frac{\phi_E - \phi_P}{(\delta x)_e} \\
\mathbf{J}_w \cdot \mathbf{A}_w &= \Gamma_w r_w \Delta r \frac{\phi_P - \phi_W}{(\delta x)_w} \\
\mathbf{J}_n \cdot \mathbf{A}_n &= -\Gamma_n r_n \Delta x \frac{\phi_N - \phi_P}{(\delta r)_n} \\
\mathbf{J}_s \cdot \mathbf{A}_s &= \Gamma_s r_s \Delta x \frac{\phi_P - \phi_S}{(\delta r)_s}
\end{aligned} \tag{3.75}$$

The source term may be written as

$$(S_C + S_P \phi_P) \Delta \mathscr{V} \tag{3.76}$$

Collecting terms, we may write the discrete equation for the cell P as

$$a_P \phi_P = a_E \phi_E + a_W \phi_W + a_N \phi_N + a_S \phi_S + b \tag{3.77}$$

where

$$\begin{aligned} a_E &= \frac{\Gamma_e r_e \Delta r}{(\delta x)_e} \\ a_W &= \frac{\Gamma_w r_w \Delta r}{(\delta x)_w} \\ a_N &= \frac{\Gamma_n r_n \Delta x}{(\delta r)_n} \\ a_S &= \frac{\Gamma_s r_s \Delta x}{(\delta r)_s} \\ a_P &= a_E + a_W + a_N + a_S - S_P \Delta \mathscr{V} \\ b &= S_C \Delta \mathscr{V} \end{aligned} \tag{3.78}$$

3.6 Finishing Touches

A few issues remain before we have truly finished discretizing our diffusion equation. We deal with these below.

3.6.1 Interpolation of Γ

We notice in our discretization that the diffusion flux is evaluated at the *face* of the control volume. As a result, we must specify the face value of the diffusion coefficient Γ. Since we store Γ at cell centroids, we must find a way to interpolate Γ to the face.

Referring to the notation in Figure 3.5, it is possible to simple interpolate Γ linearly as:

$$\Gamma_e = f_e \Gamma_P + (1 - f_e)\Gamma_E \tag{3.79}$$

where

$$f_e = \frac{0.5\Delta x_E}{(\delta x)_e} \tag{3.80}$$

As long as Γ_e is smoothly varying, this is a perfectly adequate interpolation. When ϕ is used to represent energy or temperature, step jumps in Γ may be encountered at conjugate boundaries. It is useful to devise an interpolation procedure which accounts for these jumps.

Our desire is to represent the interface flux correctly. Let J_e be the magnitude of the flux vector $\mathbf{J}_e$. Let us assume locally one-dimensional diffusion. In this limit, we may write

$$J_e = -\frac{(\phi_E - \phi_P)}{(0.5\Delta x_P)/\Gamma_P + (0.5\Delta x_E)/\Gamma_E} \tag{3.81}$$

Thus, an equivalent interface diffusion coefficient may be defined as

$$\frac{\delta x_e}{\Gamma_e} = \frac{0.5\Delta x_P}{\Gamma_P} + \frac{0.5\Delta x_E}{\Gamma_E} \tag{3.82}$$

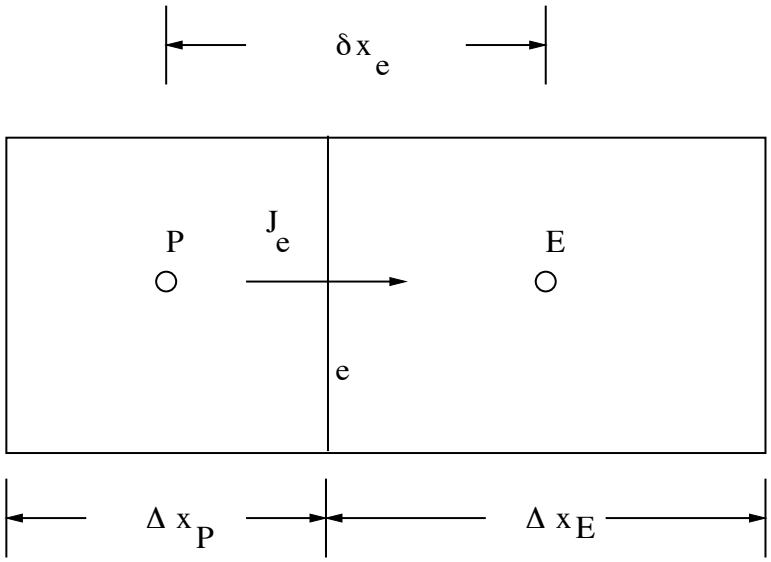


Figure 3.5: Diffusion Transport at Conjugate Boundaries

The term $\delta x_e/\Gamma_e$ may be seen as a *resistance* to the diffusion transfer between P and E. We may write Γ_e as

$$\Gamma_e = \left(\frac{1 - f_e}{\Gamma_P} + \frac{f_e}{\Gamma_E} \right)^{-1} \tag{3.83}$$

Equation 3.83 represents a *harmonic mean* interpolation for Γ. The properties of this interpolation may be better understood if we consider the case $f_e = 0.5$, i.e., the face e lies midway between P and E. In this case,

$$\Gamma_e = \frac{2\Gamma_P \Gamma_E}{\Gamma_P + \Gamma_E} \tag{3.84}$$

In the limit $\Gamma_P \gg \Gamma_E$, we get $\Gamma_e = 2\Gamma_E$. This is as expected since the high-diffusion coefficient region does not offer any resistance, and the effective resistance is that offered by the cell E, corresponding to a distance of $0.5\Delta x_E$.

It is important to realize that the use of harmonic mean interpolation for discontinuous diffusion coefficients is exact only for one-dimensional diffusion. Nevertheless, its use for multi-dimensional situations has an important advantage. With this type of interpolation, nothing special need be done to treat conjugate interfaces. We simply treat solid and fluid cells as a part of the same domain, with different diffusion coefficients stored at cell centroids. With harmonic-mean interpolation, the discontinuity in temperature gradient at the conjugate interface is correctly captured.

3.6.2 Source Linearization and Treatment of Non-Linearity

Our goal is to reduce our differential equation to a set of algebraic equations in the discrete values of ϕ. When the scalar transport is non-linear, the resulting algebraic set is also non-linear. Non-linearity can arise from a number of different sources. For example, in diffusion problems, the diffusion coefficient may be a function of ϕ, such as in the case of temperature-dependent thermal conductivity. The source term S may also be a function of ϕ. In radiative heat transfer in participating media, for example, the source term in the energy equation contains fourth powers of the temperature. There are many ways to treat non-linearities. Here, we will treat non-linearities through *Picard* iteration. In this method, the coefficients a_P, a_{nb}, S_C and S_P are evaluated using prevailing values of ϕ. They are updated as ϕ is updated by iteration.

We said previously that the source term S could be written in the form

$$S = S_C + S_P \phi \tag{3.85}$$

We now examine how this can be done when S is a non-linear function of ϕ.

Let the prevailing value of ϕ be called ϕ^*. This is the value that exists at the current iteration. We write a Taylor series expansion for S about its prevailing value $S^* = S(\phi^*)$:

$$S = S^* + \left(\frac{\partial S}{\partial \phi}\right)^* (\phi - \phi^*) \tag{3.86}$$

so that

$$\begin{aligned} S_C &= S^* - \left(\frac{\partial S}{\partial \phi}\right)^* \phi^* \\ S_P &= \left(\frac{\partial S}{\partial \phi}\right)^* \end{aligned} \tag{3.87}$$

Here, $(\partial S/\partial \phi)^*$ is the gradient evaluated at the prevailing value ϕ^*. For most problems of interest to us, $\partial S/\partial \phi$ is negative, resulting in a negative S_P. This ensures that the source tends to decrease as ϕ increases, providing a stabilizing effect. However, this type of dependence is not always guaranteed. In an explosion or a fire, for example, the application of a high temperature (the lighting of a match) causes energy to be released, and increases the temperature further. (The counter-measure is provided by the fact that the fuel is eventually consumed and the fire burns out). From a numerical viewpoint, a negative S_P makes $a_P > \sum_{\text{nb}} a_{\text{nb}}$ in our discretization, and allows us to satisfy the Scarborough criterion. It aids in the convergence of iterative solution techniques.

3.6.3 Under-Relaxation

When using iterative methods for obtaining solutions or when iterating to resolve non-linearities, it is frequently necessary to control the rate at which variables are changing during iterations. When we have a strong non-linearity in a temperature source term, for example, and our initial guess is far from the solution, we may get large oscillations in the temperature computed during the first few iterations, making it difficult for the iteration to proceed. In such cases, we often employ *under-relaxation*.

Let the current iterate of ϕ be ϕ_P^*. We know that ϕ_P satisfies

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b \tag{3.88}$$

so that after the system has been solved for the current iteration, we expect to compute a value of ϕ_P of

$$\phi_P = \frac{\sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b}{a_P} \tag{3.89}$$

We do not, however, want ϕ_P to change as much as Equation 3.89 implies. The change in ϕ_P from one iteration to the next is given by

$$\frac{\sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b}{a_P} - \phi_P^* \tag{3.90}$$

We wish to make ϕ_P change only by a fraction α of this change. Thus

$$\phi_P = \phi_P^* + \alpha \left(\frac{\sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b}{a_P} - \phi_P^* \right) \tag{3.91}$$

Collecting terms, we may write

$$\frac{a_P}{\alpha} \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b + \frac{1-\alpha}{\alpha} a_P \phi_P^* \tag{3.92}$$

We note the following about Equation 3.92:

1. When the iterations converge to a solution, i.e., when $\phi_P = \phi_P^*$, the original discrete equation is recovered. So we are assured that under-relaxation is only a change in the path to solution, and not in the discretization itself. Thus, both under-relaxed and un-underrelaxed equations yield the same final solution.

2. Though over-relaxation ($\alpha > 1$) is a possibility, we will for the most part be using $\alpha \leq 1$. With $\alpha \leq 1$, we are assured that $a_P/\alpha > \sum_{\text{nb}} a_{\text{nb}}$. This allows us to satisfy the Scarborough criterion.

3. The optimum value of α depends strongly on the nature of the system of equations we are solving, on how strong the non-linearities are, on grid size and so on. A value close to unity allows the solution to move quickly towards convergence, but may be more prone to divergence. A low value keeps the solution close to the initial guess, but keeps the solution from diverging. We use intuition and experience in choosing an appropriate value.

4. We note the similarity of under-relaxation to time-stepping. The initial guess acts as the initial condition. The terms a_P/α and $((1-\alpha)/\alpha)a_p\phi_P^*$ represent the effect of the unsteady terms.

3.7 Discussion

At this point, our discretized equation set is ready for solution. We have seen that our diffusion equation can be discretized to yield coefficients that guarantee physical plausibility for the orthogonal meshes we have considered here. This property is very useful in a numerical scheme. However, we will see in the next chapter that it cannot usually be obtained when meshes are non-orthogonal or unstructured.

We have thus far not addressed the issue of solving the discretization equations. For the moment, we shall use the simple Gauss-Seidel scheme to solve our equation set. This is admittedly slow for large meshes, and far better iteration schemes exist. These will be covered in a later chapter.

3.8 Truncation Error

We now examine the various profile assumptions we have made in the course of discretizing our diffusion equation to quantify the truncation error of our finite volume scheme.

3.8.1 Spatial Truncation Error

In coming up with our spatial approximations we made the following assumptions:

1. The face flux ($\mathbf{J}_e$, for example) was represented by the face centroid value. That is, the mean flux through the face e was represented by the face centroid value.

2. The source term $\overline{S}\Delta\mathscr{V}$ was written as $(S_C + S_P\phi_P)\Delta\mathscr{V}$,i.e., the cell centroid value ϕ_P was used to represent the mean ϕ value in the cell.

3. The gradient at the face was computed by assuming that ϕ varies linearly between cell centroids. Thus $(d\phi/dx)_e$ was written as $(\phi_E - \phi_P)/(\delta x)_e$.

Items 1 and 2 essentially involve the same approximation: that of representing the mean value of a variable by its centroid value. Item 3 involves an approximation to the face gradient. Let us examine each of these approximations in turn. We will consider a one-dimensional control volume and a uniform grid.

Mean Value Approximation

Consider the control volume in Figure 3.6 and the function $\phi(x)$ in the control volume. We expand $\phi(x)$ as

$$\phi(x) = \phi_P + (x - x_P)\left(\frac{d\phi}{dx}\right)_P + \frac{(x-x_P)^2}{2!}\left(\frac{d^2\phi}{dx^2}\right)_P + \frac{(x-x_P)^3}{3!}\left(\frac{d^3\phi}{dx^3}\right)_P + O\left((\Delta x)^4\right) \tag{3.93}$$

Integrating Equation 3.93 over the control volume, we have

$$\begin{aligned}\int_{x_w}^{x_e}\phi(x)\,dx &= \phi_P\int_{x_w}^{x_e}dx+\left(\frac{d\phi}{dx}\right)_P\int_{x_w}^{x_e}(x-x_P)dx \\ &\quad+\left(\frac{d^2\phi}{dx^2}\right)_P\int_{x_w}^{x_e}\frac{(x-x_P)^2}{2!}dx+O\left((\Delta x)^4\right)\end{aligned} \tag{3.94}$$

so that

$$\begin{aligned}\int_{x_w}^{x_e}\phi(x)\,dx &= (\Delta x)\phi_P+\frac{1}{2}\left((x_e-x_P)^2-(x_w-x_P)^2\right)\left(\frac{d\phi}{dx}\right)_P \\ &\quad+\frac{1}{6}\left((x_e-x_P)^3-(x_w-x_P)^3\right)\left(\frac{d^2\phi}{dx^2}\right)_P \\ &\quad+O\left((\Delta x)^4\right)\end{aligned} \tag{3.95}$$

For a uniform mesh, Equation 3.95 may be written as

$$\int_{x_w}^{x_e}\phi(x)\,dx=(\Delta x)\phi_P+\frac{1}{24}\left((\Delta x)^3\right)\left(\frac{d^2\phi}{dx^2}\right)_P+O\left((\Delta x)^4\right) \tag{3.96}$$

Dividing through by Δx we get

$$\overline{\phi}=\frac{1}{\Delta x}\int_{x_w}^{x_e}\phi(x)\,dx=\phi_P+O\left((\Delta x)^2\right) \tag{3.97}$$

Thus, we see that the centroid value ϕ_P represents the mean value with a truncation error of $O((\Delta x)^2)$. For a constant ϕ, all derivatives are zero. If $\phi(x)$ is linear, all derivatives of order higher than $d\phi/dx$ are zero. For these two cases, $\overline{\phi}=\phi_P$ is true *exactly*. The same analysis can be applied to the face flux $\mathbf{J}_e$, or indeed to any variable being represented by its centroid value.

Gradient Approximation

Let us now examine the truncation error in representing the gradient $(d\phi/dx)_e$ as $(\phi_E-\phi_P)/(\delta x)_e$. Referring to Figure 3.6, we write:

$$\begin{aligned}\phi_E &= \phi_e+\frac{\Delta x}{2}\left(\frac{d\phi}{dx}\right)_e+\frac{(\Delta x)^2}{8}\left(\frac{d^2\phi}{dx^2}\right)_e+\frac{(\Delta x)^3}{48}\left(\frac{d^3\phi}{dx^3}\right)_e \\ &\quad+O\left((\Delta x)^4\right) \\ \phi_P &= \phi_e-\frac{\Delta x}{2}\left(\frac{d\phi}{dx}\right)_e+\frac{(\Delta x)^2}{8}\left(\frac{d^2\phi}{dx^2}\right)_e-\frac{(\Delta x)^3}{48}\left(\frac{d^3\phi}{dx^3}\right)_e \\ &\quad+O\left((\Delta x)^4\right)\end{aligned} \tag{3.98}$$

Subtracting the second equation from the first and dividing by Δx yields

$$\left(\frac{d\phi}{dx}\right)_e=\frac{\phi_E-\phi_P}{\Delta x}+O\left((\Delta x)^2\right) \tag{3.99}$$

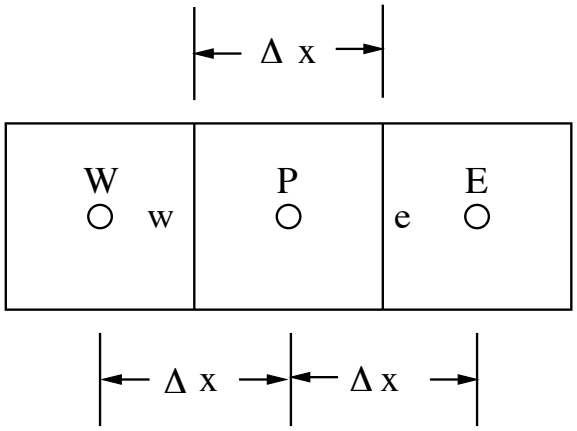


Figure 3.6: Uniform Arrangement of Control Volumes

Thus the assumption that ϕ varies linearly between grid points leads to a truncation error of $O((\Delta x)^2)$ in $d\phi/dx$. We see that all the approximations in a steady diffusion equation are $O((\Delta x)^2)$. So the discretization scheme has a truncation error of $O((\Delta x)^2)$.

The mean-value approximation is $O((\Delta x)^2$ even for a non-uniform mesh. The gradient approximations is $O((\Delta x))^2$ only for uniform meshes. For non-uniform meshes, the $O((\Delta x)^2)$ terms in the Taylor series expansion do not cancel upon subtraction, leaving a formal truncation error of $O(\Delta x)$ on $d\phi/dx$.

3.8.2 Temporal Truncation Error

Let us consider the fully-implicit scheme. The profile assumptions made in discretizing the unsteady diffusion equation using this scheme are

1. The cell centroid values $(\rho\phi)_P^1$ and $(\rho\phi)_P^0$ in the unsteady term are assumed to represent the average value for the cell.

2. The spatial assumptions are as described above.

3. The flux and source terms from time $t + \Delta t$ are assumed to prevail over the time step Δt.

The first assumption is equivalent to the mean value assumption analyzed above and engenders a spatial error of $O((\Delta x)^2$. We have already seen that the other spatial assumptions result in a truncation error of $O((\Delta x)^2)$. Let us examine the truncation error implicit in item 3. In effect, we wish to evaluate a term of the type

$$\int_{t_0}^{t_1} S(t)dt \tag{3.100}$$

Expanding S(t) about S^1, its value at t_1, we have

$$S(t) = S^1 + \left(\frac{dS}{dt}\right)^1 \left(t - t_1\right) + \left(\frac{d^2S}{dt^2}\right)^1 \frac{\left(t - t_1\right)^2}{2} + O\left((\Delta t)^3\right) \tag{3.101}$$

Integrating over the time step Δt, i.e. from t_0 to t_1, we get

$$\int_{t_0}^{t_1} S(t)dt = S^1\Delta t + \left(\frac{dS}{dt}\right)^1 \int_{t_0}^{t_1} \left(t - t_1\right) dt + O\left((\Delta t)^3\right) \tag{3.102}$$

Integrating and dividing by Δt we get

$$\frac{1}{\Delta t}\int_{t_0}^{t_1} S(t)dt = \overline{S} = S^1 - \left(\frac{dS}{dt}\right)^1 \frac{\Delta t}{2} + O\left((\Delta t)^2\right) \tag{3.103}$$

Thus the temporal truncation error of the fully implicit scheme is $O(\Delta t)$.

We state without proof that the truncation error of the explicit scheme is also $O(\Delta t)$ and that of the Crank-Nicholson scheme is $O((\Delta t)^2)$.

3.9 Stability Analysis

In this section, we perform a *von Neumann* stability analysis. Stability can be understood in two ways. For steady state problems, we wish to determine whether the path to solution is stable, i.e., we wish to analyze a particular iterative method for stability. For unsteady problems, we ask whether a particular marching scheme is stable. For example, for an unsteady heat conduction problem, we may want to determine whether taking successive time steps cause the errors in the solution to grow. This similarity between iteration and time-stepping is not surprising. We may consider any time-stepping scheme to be an iterative scheme, i.e., a way of obtaining a steady state solution, if one exists.

Let us consider the stability of the explicit scheme. For simplicity, let us consider a one-dimensional case with constant properties and no source term. The discretized diffusion equation may be written as:

$$a_P\phi_P = a_E\phi_E^0 + a_W\phi_W^0 + \left(a_P^0 - a_E - a_W\right)\phi_P^0 \tag{3.104}$$

and

$$\begin{aligned} a_E &= \frac{\Gamma_e}{(\delta x)_e} \\ a_W &= \frac{\Gamma_w}{(\delta x)_w} \\ a_P^0 &= \frac{\rho\Delta x}{\Delta t} \\ a_P &= a_P^0 \end{aligned} \tag{3.105}$$

Let Φ represent the *exact* solution to Equation 3.104. By *exact* we mean that it is the solution to the discrete equation, obtained using a computer with infinite precision. However, all real computers available to us have finite precision, and therefore, we must contend with *round-off* error. Let this finite-precision solution be given by ϕ. The error ε is given by

$$\varepsilon = \phi - \Phi \tag{3.106}$$

We ask the following question: Will the explicit scheme cause our error ε to grow, or will the process of time-stepping keep the error within bounds?

Substituting Equation 3.106 into Equation 3.104 yields

$$a_P\left(\Phi_P + \varepsilon_P\right) = a_E\left(\Phi_E^0 + \varepsilon_E^0\right) + a_W\left(\Phi_W^0 + \varepsilon_W^0\right) + \left(a_P^0 - a_E - a_W\right)\left(\Phi_P^0 + \varepsilon_P^0\right) \tag{3.107}$$

As before, the Φ and ε terms superscripted 0 represent the values at the time step t, and the un-superscripted values represent the values at time $t + \Delta t$. Since Φ_P is the exact solution, it satisfies Equation 3.104. Therefore, the Φ terms in Equation 3.107 cancel, leaving

$$a_P\varepsilon_P = a_E\varepsilon_E^0 + a_W\varepsilon_W^0 + \left(a_P^0 - a_E - a_W\right)\varepsilon_P^0 \tag{3.108}$$

Let us expand the error ε in an infinite series

$$\varepsilon\left(x,t\right) = \sum_m e^{\sigma_m t} e^{i\lambda_m x} \quad m = 0,1,2,...M \tag{3.109}$$

where σ_m may be either real or complex, and λ_m is given by

$$\lambda_m = \frac{m\pi}{L}, \quad m = 0,1,2,...M \tag{3.110}$$

L is the width of the domain. Equation 3.109 essentially presents the spatial dependence of the error by the sum of periodic functions $e^{i\lambda_m x}$, and the time dependence by $e^{\sigma_m t}$. If σ_m is real and greater than zero , the error grows with time, and if σ_m is real and less than zero, the error is damped. If σ_m is complex, the solution is oscillatory in time. We wish to find the *amplification factor*

$$\frac{\varepsilon(x,t+\Delta t)}{\varepsilon(x,t)} \tag{3.111}$$

If the amplification factor is greater than one, our error grows with time step, and our scheme is unstable. If it is less than one, our error is damped in the process of time marching, and our scheme is stable.

Since the diffusion equation for constant Γ and zero S is linear, we may exploit the principle of superposition. Thus, we can examine the stability consequences of a single error term

$$\varepsilon = e^{\sigma_m t} e^{i\lambda_m x} \tag{3.112}$$

rather than the summation in Equation 3.109. Substituting Equation 3.112 into Equation 3.108, we get

$$\begin{aligned} a_P e^{\sigma_m(t+\Delta t)} e^{i\lambda_m x} &= e^{\sigma_m t}\left(a_E e^{i\lambda_m(x+\Delta x)} + a_W e^{i\lambda_m(x-\Delta x)}\right) \\ &\quad + \left(a_P^0 - a_E - a_W\right) e^{\sigma_m t} e^{i\lambda_m x} \end{aligned} \tag{3.113}$$

Dividing through by $a_P e^{\sigma_m t} e^{i\lambda_m x}$ we get

$$\begin{aligned} e^{\sigma_m \Delta t} &= \frac{a_E}{a_P} e^{i\lambda_m \Delta x} + \frac{a_W}{a_P} e^{-i\lambda_m \Delta x} \\ &\quad + \frac{a_P^0 - a_E - a_W}{a_P} \end{aligned} \tag{3.114}$$

For a uniform mesh $a_E = a_W = \Gamma/\Delta x$ and $a_P^0 = \rho \Delta x/\Delta t$. Thus

$$e^{\sigma_m \Delta t} = \frac{\Gamma \Delta t}{\rho (\Delta x)^2} \left(e^{i\lambda_m \Delta x} + e^{-i\lambda_m \Delta x} \right) + \left(1 - \frac{2\Gamma \Delta t}{\rho (\Delta x)^2} \right) \tag{3.115}$$

Using $\beta = \lambda_m \Delta x$ and the identities

$$e^{i\lambda_m \Delta x} + e^{-i\lambda_m \Delta x} = 2\cos\beta = 2 - 4\sin^2 \frac{\beta}{2} \tag{3.116}$$

we get

$$e^{\sigma_m \Delta t} = 1 - \frac{4\Gamma \Delta t}{\rho (\Delta x)^2} \sin^2 \frac{\beta}{2} \tag{3.117}$$

We recognize that the amplification factor is

$$\frac{\varepsilon(x, t + \Delta t)}{\varepsilon(x,t)} = e^{\sigma_m \Delta t} = 1 - \frac{4\Gamma \Delta t}{\rho (\Delta x)^2} \tag{3.118}$$

We require that

$$\left| \frac{\varepsilon(x, t + \Delta t)}{\varepsilon(x,t)} \right| \leq 1 \tag{3.119}$$

or

$$\left| 1 - \frac{4\Gamma \Delta t}{\rho (\Delta x)^2} \right| \leq 1 \tag{3.120}$$

If $(1 - \frac{4\Gamma \Delta t}{\rho (\Delta x)^2}) > 0$, we require

$$\frac{4\Gamma \Delta t}{\rho (\Delta x)^2} \geq 0 \tag{3.121}$$

This is always guaranteed since all terms are positive. If, on the other hand, $(1 - \frac{4\Gamma \Delta t}{\rho (\Delta x)^2}) \leq 0$, we require

$$\frac{4\Gamma \Delta t}{\rho (\Delta x)^2} \leq 2 \tag{3.122}$$

or

$$\Delta t \leq \frac{\rho (\Delta x)^2}{2\Gamma} \tag{3.123}$$

We recognize this criterion to be the same as Equation 3.56. Using an error expansion of the type

$$\varepsilon(x, y, t) = e^{\sigma_m t} e^{i\lambda_m x} e^{i\lambda_n y} \tag{3.124}$$

we may derive an equivalent criterion for two-dimensional uniform meshes ($\Delta x = \Delta y$) as

$$\Delta t \leq \frac{\rho(\Delta x)^2}{4\Gamma} \tag{3.125}$$

A similar analysis may also be done to obtain the criterion for three-dimensional situations. It is also possible to show, using a similar analysis, that the fully-implicit and Crank-Nicholson schemes are unconditionally stable.

Though the von Neumann stability analysis and our heuristic requirement of positive coefficients have yielded the same criterion in the case of the explicit scheme, we should not conclude that the two will always yield identical results. The von Neumann analysis yields a time-step limitation required to keep round-off errors bounded. It is possible to get bounded but unphysical solutions. This happens in the case of the Crank-Nicholson scheme, which the von Neumann stability analysis classifies as unconditionally stable. However, we know that for $\rho\Delta x/\Delta t < 0.5 \sum_{\text{nb}} a_{\text{nb}}$, it is possible to get unphysical results. In this case, the von Neumann stability analysis tells us that the oscillations in our solution will remain bounded, but it cannot guarantee that they will be physically plausible.

Von Neumann stability analysis is a classic tool for analyzing the stability of linear problems. However, we see right away that it would be substantially more complicated to do the above analysis if Γ were a function of ϕ, or if we had non-linear source terms. It becomes very difficult to use when we solve coupled non-linear equations such as the Navier-Stokes equations. In practice, we use von Neumann stability analysis to give us guidance on the baseline behavior of idealized systems, realizing that the coupled non-linear equations we really want to solve will probably have much stringent restrictions. For these, we must rely on intuition and experience for guidance.

3.10 Closure

In this chapter, we have completed the discretization of the diffusion equation on regular orthogonal meshes for Cartesian, polar and axisymmetric geometries. We have seen that our discretization guarantees physically plausible solutions for both steady and unsteady problems. We have also seen how to handle non-linearities. For uniform meshes, we have shown that our spatial discretization is formally second-order accurate, and that if we use the fully-implicit scheme, our temporal discretization is first-order accurate. We have also see how to conduct a stability analysis for the explicit scheme as applied to the diffusion equation. In the next chapter, we will address the issue of mesh non-orthogonality and understand the resulting complications.

Chapter 4

The Diffusion Equation: A Closer Look

In the last chapter, we saw how to discretize the diffusion equation for regular (orthogonal) meshes composed of quadrilaterals. In this chapter, we address non-orthogonal meshes, both structured and unstructured. We will see that non-orthogonality leads to extra terms in our discrete equations which destroy some of the properties we had particularly prized in our discretization scheme. We will also address the special issues associated with the computation of gradients on unstructured meshes.

4.1 Diffusion on Orthogonal Meshes

Let us consider a mesh consisting of convex polyhedra, and particularly, equilateral triangles, as shown in Figure 4.1. Regardless of the shape of the cells, any face f in the mesh is shared by only two neighbor cells. We shall consider the mesh to be orthogonal if the line joining the centroids of adjacent cells is orthogonal to the shared face f. This is guaranteed for the equilateral triangular mesh in the figure. A detail of the cells $C0$ and $C1$ is shown in Figure 4.2.

Consider the steady diffusion equation:

$$\nabla \cdot \mathbf{J} = S \tag{4.1}$$

where

$$\mathbf{J} = -\Gamma \nabla \phi \tag{4.2}$$

We focus on the cell $C0$. To discretize Equation 4.1, we integrate it over the control volume $C0$ as before:

$$\int_{\Delta\mathcal{V}_0} \nabla \cdot \mathbf{J} d\mathcal{V} = \int_{\Delta\mathcal{V}_0} S d\mathcal{V} \tag{4.3}$$

Applying the divergence theorem, we get

$$\int_A \mathbf{J} \cdot d\mathbf{A} = \int_{\Delta\mathcal{V}_0} S d\mathcal{V} \tag{4.4}$$

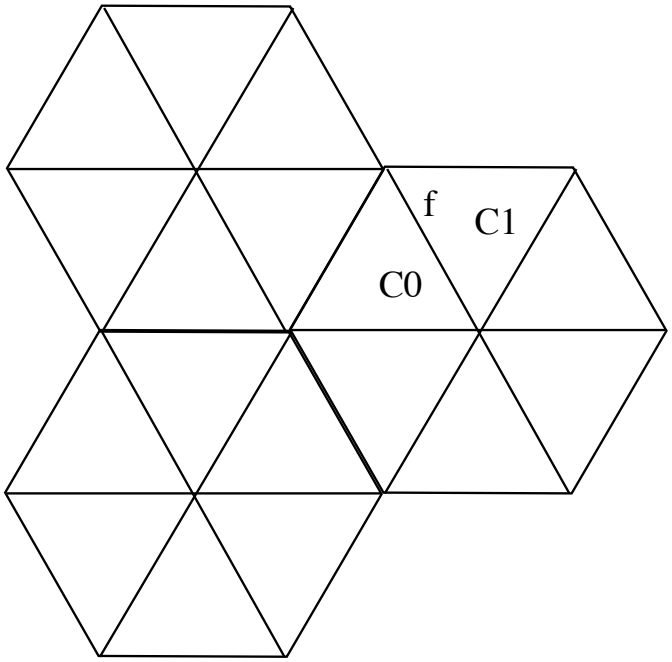


Figure 4.1: Orthogonal Mesh

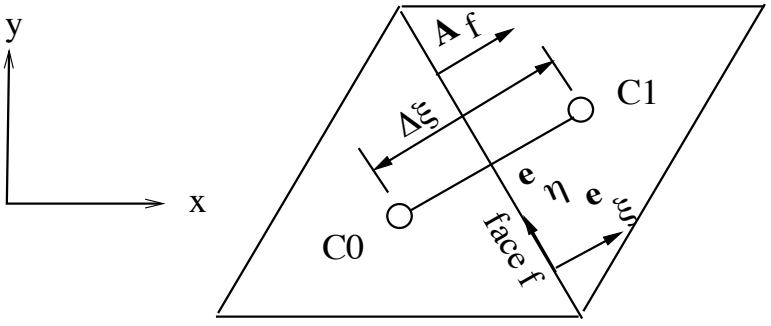


Figure 4.2: Arrangement of Cells in Orthogonal Mesh

We now make a profile assumption. We assume that $\mathbf{J}$ can be approximated by its value at the centroid of the face f. Also, let us assume that $S = S_C + S_P\phi_0$ as before. Thus

$$\sum_f \mathbf{J}_f \cdot \mathbf{A}_f = \left(S_C + S_P\phi_0\right)\Delta\mathcal{V}_0 \tag{4.5}$$

The summation in the first term is over all the faces of the control volume $C0$, and $\mathbf{A}_f$ is the area vector on the face f pointing outwards from cell $C0$. Further,

$$\begin{aligned}\mathbf{J}_f &= -\Gamma_f(\nabla\phi)_f \\ &= -\Gamma_f\left(\frac{\partial\phi}{\partial x}\mathbf{i} + \frac{\partial\phi}{\partial y}\mathbf{j}\right)_f\end{aligned} \tag{4.6}$$

In order to evaluate $\partial\phi/\partial x$ and $\partial\phi/\partial y$ at the face f, we need cell centroid values of ϕ which are suitably placed along the x and y axes. This is easy to arrange for a rectangular mesh, as we saw in the previous chapter. We see from Figure 4.2 that such values are not available for the mesh under consideration. We must take an alternative approach.

Consider the coordinate directions ξ and η in Figure 4.2. The unit vector $\mathbf{e}_\xi$ is aligned with the line joining the centroids. The unit vector $\mathbf{e}_\eta$ is tangential to the face f. Because the mesh is orthogonal, the unit vectors are perpendicular to each other. We may write the face flux vector $\mathbf{J}_f$ in the coordinate system $\xi - \eta$ as

$$\mathbf{J}_f = -\Gamma_f\left(\frac{\partial\phi}{\partial\xi}\mathbf{e}_\xi + \frac{\partial\phi}{\partial\eta}\mathbf{e}_\eta\right) \tag{4.7}$$

The area vector $\mathbf{A}_f$ may be written as

$$\mathbf{A}_f = A_f\mathbf{e}_\xi \tag{4.8}$$

Therefore

$$\mathbf{J}_f \cdot \mathbf{A}_f = -\Gamma_f A_f\left(\frac{\partial\phi}{\partial\xi}\right)_f \tag{4.9}$$

We see that if the line joining the centroids is perpendicular to the face, the normal diffusion transport $\mathbf{J_f}\cdot\mathbf{A}_f$ only depends on the gradient of ϕ in the coordinate direction ξ and not on η. We realize that since cell centroid values of ϕ are available along the $\mathbf{e}_\xi$ direction, it is easy for us to write $\partial\phi/\partial\xi$.

As before, let us make a linear profile assumption for the variation of $\phi(\xi)$ between the centroids of cells $C0$ and $C1$. Thus

$$\mathbf{J}_f \cdot \mathbf{A}_f = -\Gamma_f A_f\frac{\left(\phi_1 - \phi_0\right)}{\Delta\xi} \tag{4.10}$$

where $\Delta\xi$ is the distance between the cell centroids, as shown in Figure 4.2, and ϕ_0 and ϕ_1 are the discrete values of ϕ at the cell centroids. We see that if the mesh is orthogonal, the diffusion transport normal to the face f can be written purely in terms of values of ϕ at the centroids of the cells sharing the face.

So far we have looked at the transport through a single face f. Substituting Equation 4.10 into Equation 4.5 and summing over all the faces of the cell $C0$ yields an equation of the form

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b \tag{4.11}$$

where

$$\begin{aligned} a_{\text{nb}} &= \left(\frac{\Gamma_f A_f}{\Delta \xi} \right)_{\text{nb}}, \quad nb = 1,2,...M \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} - S_P \Delta \mathcal{V}_0 \\ b &= S_C \Delta \mathcal{V}_0 \end{aligned} \tag{4.12}$$

In the above equations, *nb* denotes the *face* neighbors of the cell under consideration, P. That is, the "neighbor" cells are those which share faces with the cell under consideration. The cell P shares vertices with other cells, but these do not appear in the discretization. M is the number of face neighbors. If the cell is a triangle, there are three face neighbors, and $M = 3$ in the above equation. We note the following:

1. In the development above, we have not used the fact that cell under consideration is a triangle. All we have required is that the cell be a polyhedron and that line joining the cell centroids be orthogonal to the face.

2. It is necessary for the mesh to consist of convex polyhedra. If the cells are not convex, the cell centroid may not fall inside the cell.

3. We need not make the assumption of two-dimensionality. The above development holds for three dimensional situations.

4. We have not made any assumptions about mesh structure, though it is usually difficult to generate orthogonal meshes without some type of structure.

5. The scheme is conservative because we use the conservation principle to derive the discrete equations. The face flux $\mathbf{J}_f$ leaves one cell and enters the adjacent cell. Thus overall conservation is guaranteed. It is worth noting here that $\mathbf{J}_f$ should be thought of as belonging to the face f rather that to either cell $C0$ or $C1$. Thus, whatever profile assumptions are used to evaluate $\mathbf{J}_f$ are used consistently for both cells $C0$ and $C1$.

6. As with our orthogonal rectangular mesh in the previous chapter, we get a well-behaved set of discrete equations. In the absence of source terms, $a_P = \sum_{\text{nb}} a_{\text{nb}}$. We are thus guaranteed that ϕ_P is bounded by its face neighbors when $S = 0$.

7. All the development from the previous chapter regarding unsteady flow, interpolation of Γ_f and linearization of source terms may be carried over unchanged.

8. It is possible to solve the discrete equation set using the Gauss-Seidel iterative scheme, which does not place any restrictions on the number of neighbors.

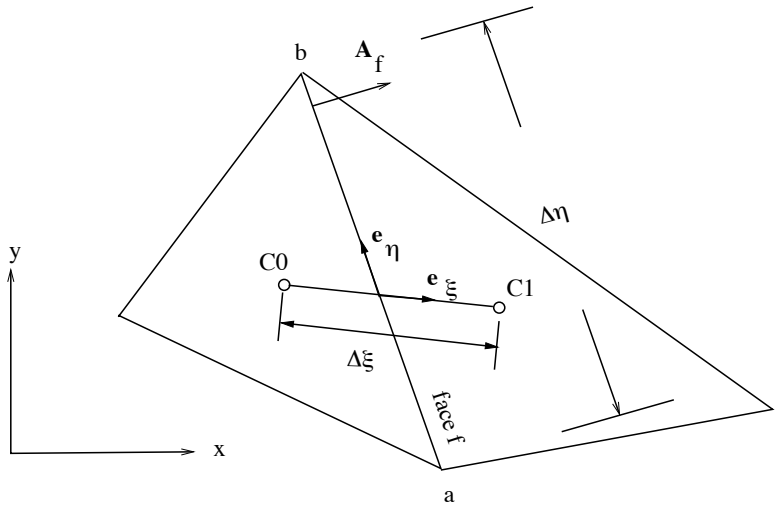


Figure 4.3: Non-Orthogonal Mesh

4.2 Non-Orthogonal Meshes

In practice, it is rarely possible to use orthogonal meshes in industrial geometries because of their complexity. Our interest therefore lies in developing methods which can deal with non-orthogonal meshes, and preferably, with unstructured meshes. We will start by looking at general non-orthogonal meshes and derive expressions for structured meshes as a special case.

Consider the cells $C0$ and $C1$ shown in Figure 4.3. We consider this mesh to be *non-orthogonal* because the line joining the cell centroids $C0$ and $C1$ is not perpendicular to the face f.

As before, we consider the steady diffusion equation

$$\nabla \cdot \mathbf{J} = S \tag{4.13}$$

and focus on the cell $C0$. We integrate the equation over the control volume $C0$ and apply the divergence theorem as before. Assuming that $\mathbf{J}_f$, the flux at the face centroid, prevails over the face f, we obtain:

$$\sum_f \mathbf{J}_f \cdot \mathbf{A}_f = \left(S_C + S_P \phi_0\right) \Delta\mathcal{V}_0 \tag{4.14}$$

Thus far, the procedure is the same as that for an orthogonal mesh. The area vector $\mathbf{A}_f$ is given by

$$\mathbf{A}_f = A_x\,\mathbf{i} + A_y\,\mathbf{j} \tag{4.15}$$

As before, writing $\mathbf{J}_f$ in terms of $\partial\phi/\partial x$ and $\partial\phi/\partial y$ is not useful since we do not have cell centroid values of ϕ along these directions. Let us consider instead the coordinate system $\xi - \eta$ on the face f. The unit vector $\mathbf{e}_\xi$ is parallel to the line joining the centroids. The unit vector $\mathbf{e}_\eta$ is tangential to the face. However, since the mesh is not

orthogonal, $\mathbf{e}_\xi$ and $\mathbf{e}_\eta$ are not orthogonal to each other. Writing $\mathbf{J}_f$ in terms of the $\xi - \eta$ coordinate system seems promising since we have cell centroid values aligned along the ξ direction.

As usual, we may write $\mathbf{J}_f \cdot \mathbf{A}_f$ as

$$\mathbf{J}_f \cdot \mathbf{A}_f = -\Gamma_f \left(\frac{\partial \phi}{\partial x} A_x + \frac{\partial \phi}{\partial y} A_y \right)_f \tag{4.16}$$

In order to express $\partial\phi/\partial x$ and $\partial\phi/\partial y$ in terms of ξ and η, we start be writing

$$\begin{aligned} \phi_\xi &= \phi_x x_\xi + \phi_y y_\xi \\ \phi_\eta &= \phi_x x_\eta + \phi_y y_\eta \end{aligned} \tag{4.17}$$

where ϕ_ξ denotes $\partial\phi/\partial\xi$, x_ξ denote $\partial x/\partial\xi$ and so on. Solving for ϕ_x and ϕ_y, we get

$$\begin{aligned} \phi_x &= \frac{\phi_\xi y_\eta - \phi_\eta y_\xi}{\mathscr{J}} \\ \phi_y &= \frac{-\phi_\xi x_\eta + \phi_\eta x_\xi}{\mathscr{J}} \end{aligned} \tag{4.18}$$

where

$$\mathscr{J} = x_\xi y_\eta - x_\eta y_\xi \tag{4.19}$$

Therefore

$$\begin{aligned} \mathbf{J}_f \cdot \mathbf{A}_f &= -\Gamma_f \left(\frac{A_x y_\eta - A_y x_\eta}{\mathscr{J}} \right) \left(\phi_\xi \right)_f \\ &\quad -\Gamma_f \left(\frac{-A_x y_\xi + A_y x_\xi}{\mathscr{J}} \right) \left(\phi_\eta \right)_f \end{aligned} \tag{4.20}$$

Furthermore, we may write

$$\begin{aligned} x_\xi &= \frac{x_1 - x_0}{\Delta\xi} \\ y_\xi &= \frac{y_1 - y_0}{\Delta\xi} \\ x_\eta &= \frac{x_b - x_a}{\Delta\eta} \\ y_\eta &= \frac{y_b - y_a}{\Delta\eta} \\ A_x &= (y_b - y_a) \\ A_y &= -(x_b - x_a) \end{aligned} \tag{4.21}$$

The unit vectors $\mathbf{e}_\xi$ and $\mathbf{e}_\eta$ may be written as

$$\begin{aligned} \mathbf{e}_\xi &= \frac{(x_1 - x_0)\mathbf{i} + (y_1 - y_0)\mathbf{j}}{\Delta\xi} \\ \mathbf{e}_\eta &= \frac{(x_b - x_a)\mathbf{i} + (y_b - y_a)\mathbf{j}}{\Delta\eta} \end{aligned} \tag{4.22}$$

where $\Delta\xi$ is the distance between the centroids and $\Delta\eta$ is the distance between the vertices a and b. We recognize that

$$\Delta\eta = A_f \tag{4.23}$$

Now, let us examine the Jacobian $\mathscr{J}$. Using the equations 4.21, we may write

$$\begin{aligned}
\mathscr{J} &= x_\xi y_\eta - x_\eta y_\xi \\
&= \frac{x_1 - x_0}{\Delta\xi}\frac{y_b - y_a}{\Delta\eta} - \frac{y_1 - y_0}{\Delta\xi}\frac{x_b - x_a}{\Delta\eta} \\
&= \frac{\mathbf{A}_f \cdot \mathbf{e}_\xi}{\Delta\eta}
\end{aligned} \tag{4.24}$$

The ϕ_ξ term in Equation 4.20 may be written as

$$\begin{aligned}
\phi_\xi \left(\frac{A_x y_\eta - A_y x_\eta}{\mathscr{J}} \right) &= \phi_\xi \frac{\left[A_x(y_b - y_a) - A_x(x_b - x_a) \right] / \Delta\eta}{\mathbf{A}_f \cdot \mathbf{e}_\xi / \Delta\eta} \\
&= \phi_\xi \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A} \cdot \mathbf{e}_\xi}
\end{aligned} \tag{4.25}$$

The ϕ_η term in Equation 4.20 may be written as

$$\begin{aligned}
\phi_\eta \left(\frac{-A_x y_\xi + A_y x_\xi}{\mathscr{J}} \right) &= -\phi_\eta \frac{\left[(y_1 - y_0)(y_b - y_a) + (x_1 - x_0)(x_b - x_a) \right] / \Delta\xi}{\mathbf{A}_f \cdot \mathbf{e}_\xi / \Delta\eta} \\
&= -\phi_\eta (\Delta\eta)^2 \frac{\mathbf{e}_\xi \cdot \mathbf{e}_\eta}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \\
&= -\phi_\eta \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \mathbf{e}_\xi \cdot \mathbf{e}_\eta
\end{aligned} \tag{4.26}$$

Collecting terms, we may write

$$\mathbf{J}_f \cdot \mathbf{A}_f = -\Gamma_f \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \left(\phi_\xi \right)_f + \Gamma_f \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \mathbf{e}_\xi \cdot \mathbf{e}_\eta \left(\phi_\eta \right)_f \tag{4.27}$$

We now need profile assumptions for ϕ. For the moment, we shall consider only the ϕ_ξ term. Assuming that ϕ varies linearly between cell centroids, we may write

$$\left(\phi_\xi \right)_f = \frac{\phi_1 - \phi_0}{\Delta\xi} \tag{4.28}$$

Furthermore, we define

$$\mathscr{S}_f = \Gamma_f \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \mathbf{e}_\xi \cdot \mathbf{e}_\eta \left(\phi_\eta \right)_f \tag{4.29}$$

Therefore

$$\mathbf{J}_f \cdot \mathbf{A}_f = -\frac{\Gamma_f}{\Delta\xi} \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \left(\phi_1 - \phi_0 \right) + \mathscr{S}_f \tag{4.30}$$

4.2.1 Discussion

We have thus far derived an expression for the quantity $\mathbf{J}_f \cdot \mathbf{A}_f$ for the face f. We see that, unlike for orthogonal meshes, the flux cannot be written in terms of $\partial\phi/\partial\xi$ alone – an additional gradient, $\partial\phi/\partial\eta$ is involved. We call the term involving $\partial\phi/\partial\xi$ the *primary gradient* or the *primary diffusion* term. The term $\mathscr{S}_f$ is called the *secondary gradient* or the *secondary diffusion* term. For orthogonal meshes, $\mathbf{e}_\xi \cdot \mathbf{e}_\eta$ is zero since the line joining the centroids is perpendicular to the face. Therefore $\mathscr{S}_f$ is zero for orthogonal meshes. Furthermore, from Equation 4.8, the term $(\mathbf{A}_f \cdot \mathbf{A}_f)/(\mathbf{A}_f \cdot \mathbf{e}_\xi)$ reduces to A_f for an orthogonal mesh. Thus, we are assured that our formulation defaults to the right expression when the mesh is orthogonal.

The secondary gradient term is a little problematic to compute since we do not have any cell centroid values of ϕ available in the η direction. We must devise a method by which $\partial\phi/\partial\eta$ can be computed on the face f. Precisely how this is done will depend on whether our mesh is structured or unstructured.

4.2.2 Secondary Gradient Calculation

For structured meshes, the computation of the gradient $\partial\phi/\partial\eta$ poses no particular problem. In two dimensions, the problem may be boiled down to either finding the ϕ_a and ϕ_b by interpolation and thus finding $\partial\phi/\partial\eta$, or alternatively, finding $\partial\phi/\partial\eta$ at the cells C0 and C1 and interpolating these values to the face f.

For three-dimensional situations on structured meshes, again, there is no particular difficulty. The structured mesh shown in the x-y plane in Figure 4.4(a) consists of mesh lines of $\xi = c$ and $\eta = c$ which form the faces of the cell. In three dimensions, a cell is bounded by faces of constant ξ, η and ζ. The gradient $\nabla\phi$ may be decomposed in these three non-orthogonal directions, resulting in secondary gradient terms involving $\partial\phi/\partial\eta$ and $\partial\phi/\partial\zeta$. These *tangential* gradients may be written in terms of values of ϕ at the points a, b, c and d shown in Figure 4.4(b). These values in turn may be interpolated from neighboring cell-centroid values.

For unstructured meshes, it is possible to write $\partial\phi/\partial\eta$ in terms of ϕ_a and ϕ_b in two dimensions since the coordinate direction η can be uniquely defined. In three dimensions, however, it not possible to define the η direction uniquely. The face f is in general an n-sided polyhedron, with no unique mesh directions, and the two tangential directions η and ζ would have to be chosen arbitrarily. We should note however that the ξ direction is uniquely defined as the centroid-to-centroid direction.

For unstructured meshes, we write

$$\mathbf{J}_f \cdot \mathbf{A}_f = -\frac{\Gamma_f}{\Delta\xi}\frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi}\left(\phi_1 - \phi_0\right) - \Gamma_f (\nabla\phi)_f \cdot \mathbf{A}_f + \frac{\Gamma_f}{\Delta\xi}\frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi}(\nabla\phi)_f \cdot \mathbf{e}_\xi \Delta\xi \quad (4.31)$$

so that the secondary gradient term is given by

$$\mathscr{S}_f = -\Gamma_f (\nabla\phi)_f \cdot \mathbf{A}_f + \frac{\Gamma_f}{\Delta\xi}\frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi}(\nabla\phi)_f \cdot \mathbf{e}_\xi \Delta\xi \quad (4.32)$$

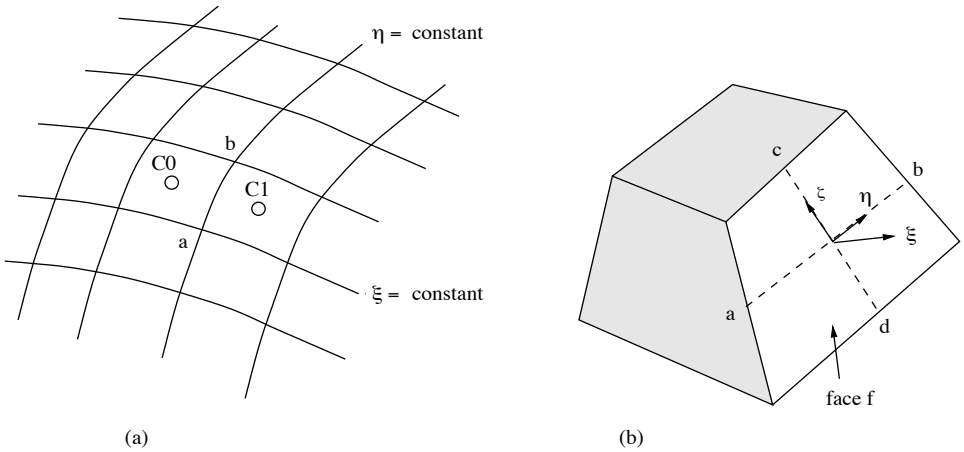


Figure 4.4: Definition of Face Tangential Directions in (a) Two Dimensions and (b) Three Dimensions

Equation 4.32 writes the secondary gradient term as the *difference* of the total transport through the face f and the transport in the direction $\mathbf{e}_\xi$. Both terms require the face gradient $(\nabla\phi)_f$; its computation is addressed in a later section.

Thus the problem of computing the secondary gradient is reduced to the problem of computing the face gradient of ϕ. It is possible to compute $(\nabla\phi)_f$ directly at the face using the methods we will present in a later section. It is often more convenient to store $\nabla\phi$ at the cells $C0$ and $C1$. Assuming that the gradient of ϕ in each cell is a constant, we may find an average as

$$(\nabla\phi)_f = \frac{\nabla\phi_0 + \nabla\phi_1}{2} \tag{4.33}$$

The unstructured mesh formulation can of course be applied to structured meshes. However, it is usually simpler and less expensive to exploit mesh structure when it exists.

4.2.3 Discrete Equation for Non-Orthogonal Meshes

The discretization procedure at the face f can be repeated for each of the faces of the cell $C0$. The resulting discrete equation may be written in the form

$$a_P\phi_P = \sum_{\text{nb}} a_{\text{nb}}\phi_{\text{nb}} + b \tag{4.34}$$

where

$$\begin{aligned} a_{\text{nb}} &= \left(\frac{\Gamma_f}{\Delta\xi}\frac{\mathbf{A}_f\cdot\mathbf{A}_f}{\mathbf{A}_f\cdot\mathbf{e}_\xi}\right)_{\text{nb}} \qquad \text{nb}=1,2,...\text{M} \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} - S_P\cdot\mathscr{V}_0 \\ b &= S_C\Delta\mathscr{V}_0 - \sum_{\text{nb}}\left(\mathscr{S}_f\right)_{\text{nb}} \end{aligned} \tag{4.35}$$

As before, *nb* denotes the cell neighbors of the cell under consideration, P. The quantities Γ_f, $\mathbf{e}_\xi$, $\Delta\xi$, $\mathbf{A}_f$ and $\mathscr{S}_f$ correspond to the face f shared by the cell P and the neighbor cell *nb*.

We see that the primary terms result in a coefficient matrix which has the same structure as for orthogonal meshes. The above discretization ensures that $a_P = \sum_{\text{nb}} a_{\text{nb}}$ if $S = 0$. However, we no longer have the guarantee that ϕ_P is bounded by its cell neighbors. This is because the secondary gradient term, $\mathscr{S}_f$, involves gradients of ϕ which must be evaluated from the cell centroid values by interpolation. As we will see later, this term can result in the possibility of spatial oscillations in the computed values of ϕ.

Though the formulation has been done for steady diffusion, the methods for unsteady diffusion outlined in the previous chapter are readily applicable here. Similarly the methods for source term linearization and under-relaxation are also readily applicable. The treatment of interfaces with step jumps in Γ requires a modest change, which we leave as an exercise to the reader.

4.3 Boundary Conditions

Consider the cell $C0$ with the face b on the boundary, as shown in Figure 4.5. The cell value ϕ_0 is stored at the centroid of the cell $C0$. As with regular meshes, we store a discrete value ϕ_b at the centroid of the boundary face b. The boundary area vector $\mathbf{A}_b$ points outwards from the cell $C0$ as shown. The cell balance for cell $C0$ is given as before by

$$\sum_f \mathbf{J}_f\cdot\mathbf{A}_f + \mathbf{J}_b\cdot\mathbf{A}_b = \left(S_C + S_P\phi_0\right)\Delta\mathscr{V}_0 \tag{4.36}$$

Here the summation over f in the first term on the left hand side is over all the *interior* faces of cell $C0$, and the second term represents the transport of ϕ through the boundary face. We have seen how the interior fluxes are discretized. Let us now consider the boundary term $\mathbf{J}_b\cdot\mathbf{A}_b$.

The boundary transport term is written as

$$\mathbf{J}_b\cdot\mathbf{A}_b = -\Gamma_b\left(\nabla\phi\right)_b\cdot\mathbf{A}_b \tag{4.37}$$

We define the direction ξ to be the direction connecting the cell centroid to the face centroid, as shown in Figure 4.5, and the direction η to be tangential to the boundary

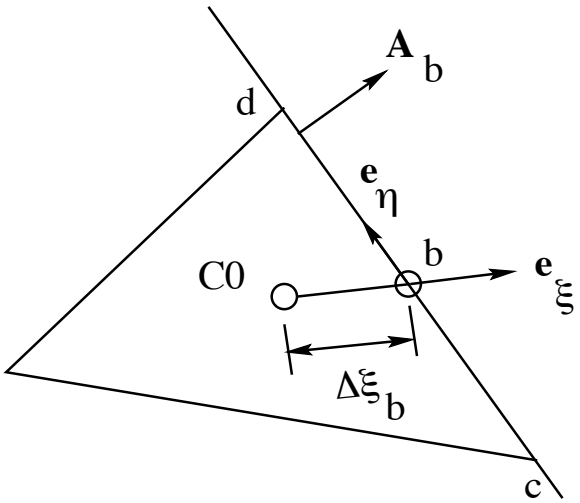


Figure 4.5: Boundary Control Volume for Non-Orthogonal Mesh

face. Decomposing the flux $\mathbf{J}_b$ in the ξ and η directions as before, we obtain

$$\mathbf{J}_b \cdot \mathbf{A}_b = -\Gamma_b \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} \left(\phi_\xi\right)_b + \Gamma_f \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} \mathbf{e}_\xi \cdot \mathbf{e}_\eta \left(\phi_\eta\right)_b \tag{4.38}$$

As before, we define the secondary diffusion term

$$\mathscr{S}_b = \Gamma_f \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} \mathbf{e}_\xi \cdot \mathbf{e}_\eta \left(\phi_\eta\right)_b \tag{4.39}$$

We note that if $\mathbf{e}_\xi$ and $\mathbf{e}_\eta$ are perpendicular to each other, $\mathscr{S}_b$ is zero.

Thus, the total transport across the boundary face b is given by

$$\mathbf{J}_b \cdot \mathbf{A}_b = -\Gamma_b \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} \left(\phi_\xi\right)_b + \mathscr{S}_b \tag{4.40}$$

As before we make a linear profile assumption for the variation of ϕ between points $C0$ and b. This yields

$$\mathbf{J}_b \cdot \mathbf{A}_b = -\frac{\Gamma_b}{(\Delta\xi)_b} \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} \left(\phi_b - \phi_0\right) + \mathscr{S}_b \tag{4.41}$$

As with interior faces, we are faced with the question of how to compute ϕ_η at the face b. For structured meshes, and two-dimensional unstructured meshes, we may use interpolation to obtain the vertex values ϕ_c and ϕ_d in Figure 4.5. For three-dimensional unstructured meshes, however, the boundary face tangential directions are not uniquely

defined. Consequently, we write the boundary secondary gradient term as the difference of the total and primary terms:

$$\mathscr{S}_b = -\Gamma_b (\nabla\phi)_b \cdot \mathbf{A}_b + \frac{\Gamma_b}{(\Delta\xi)_b} \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} (\nabla\phi)_b \cdot \mathbf{e}_\xi (\Delta\xi)_b \tag{4.42}$$

Further assuming that

$$(\nabla\phi)_b = (\nabla\phi)_0 \tag{4.43}$$

we may write the secondary gradient term at the boundary as

$$\mathscr{S}_b = -\Gamma_b (\nabla\phi)_0 \cdot \mathbf{A}_b + \frac{\Gamma_b}{(\Delta\xi)_b} \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} (\nabla\phi)_0 \cdot \mathbf{e}_\xi (\Delta\xi)_b \tag{4.44}$$

We will see in a later section how the cell gradient $(\nabla\phi)_0$ is computed.

Having seen how to discretize the boundary flux, we now turn to the application of boundary conditions.

Dirichlet Boundary Condition

At Dirichlet boundaries, we are given the value of ϕ_b

$$\phi_b = \phi_{b,\text{given}} \tag{4.45}$$

Using $\phi_{b,\text{given}}$ in Equation 4.41 and including $\mathbf{J}_b \cdot \mathbf{A}_b$ in the boundary cell balance yields a discrete equation of the following form:

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b \tag{4.46}$$

where

$$\begin{aligned}
a_{\text{nb}} &= \left(\frac{\Gamma_f}{\Delta\xi} \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \right)_{\text{nb}} \qquad \text{nb} = 1,2,...\text{M} \\
a_b &= \frac{\Gamma_b}{(\Delta\xi)_b} \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} \\
a_P &= \sum_{\text{nb}} a_{\text{nb}} + a_b - S_P \cdot \mathscr{V}_0 \\
b &= S_C \Delta\mathscr{V}_0 - \sum_{\text{nb}} \left(\mathscr{S}_f \right)_{\text{nb}} + a_b \phi_{b,\text{given}} - \mathscr{S}_b
\end{aligned} \tag{4.47}$$

Here, *nb* denotes the interior cell neighbors of the cell under consideration, P. The quantities $\Gamma_f, \mathbf{e}_\xi, \Delta\xi, \mathbf{A}_f$ and $\mathscr{S}_f$ correspond to the face f shared by the cell P and the neighbor cell *nb*. In the a_b term, $\mathbf{e}_\xi$ corresponds to the direction shown in Figure 4.5.

As with interior cells, we see that the primary terms result in a coefficient matrix which has the same structure as for orthogonal meshes. The above discretization ensures that $a_P > \sum_{\text{nb}} a_{\text{nb}}$ if $S = 0$. However, we no longer have the guarantee that ϕ_P

is bounded by its cell neighbors. This is because the secondary gradient terms, $\mathscr{S}_f$ and $\mathscr{S}_b$, involve gradients of ϕ which must be evaluated from the cell centroid values by interpolation. As we will see later, this term can cause spatial oscillations in the computed values of ϕ.

Equivalent discrete equations for Neumann and mixed boundary conditions may be derived starting with Equation 4.41 and following the procedures in the previous chapter. We leave this as an exercise for the reader.

4.4 Gradient Calculation

As we saw in the previous section, for non-orthogonal grids we need to determine gradients of ϕ at the cell face centroids to compute the secondary diffusion term. Gradients are also required in many other cases. For example, velocity derivatives are required to compute the production term in turbulence models or to compute the strain rate for non-Newtonian viscosity models. In this section, we will learn about techniques for evaluating gradients for different types of mesh topologies.

4.4.1 Structured Meshes

For a one-dimensional problem, a linear profile assumption for the variation of ϕ between cell centers results in the following expression for the derivative at the cell face

$$\left.\frac{d\phi}{dx}\right|_f = \frac{\phi_E - \phi_P}{\Delta x} \tag{4.48}$$

In the same manner we can write the derivative at a cell center using the values at the two adjacent cells.

$$\left.\frac{d\phi}{dx}\right|_P = \frac{\phi_E - \phi_W}{2\Delta x} \tag{4.49}$$

This expression is usually referred to as the "central difference" approximation for the first derivative.

For Cartesian grids in multiple dimensions, we can compute the derivatives by applying the same principle along the respective coordinate directions. Consider, for example the grid shown in Fig. 4.6. For simplicity, we restrict the following development to equispaced grids. Using the central difference approximation introduced earlier we can write

$$\left.\frac{\partial\phi}{\partial x}\right|_P = \frac{\phi_E - \phi_W}{2\Delta x} \tag{4.50}$$

$$\left.\frac{\partial\phi}{\partial y}\right|_P = \frac{\phi_N - \phi_S}{2\Delta y} \tag{4.51}$$

The procedure is similar in case of general non-orthogonal structured grids, where we use central difference approximations to write the derivatives in the transformed

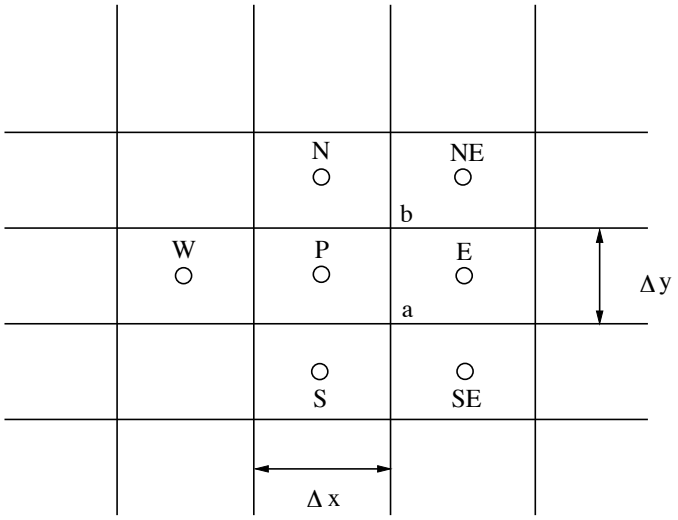


Figure 4.6: Arrangement of Cells in Cartesian Mesh

coordinates. Figure 4.7 shows a non-orthogonal mesh in the $x-y$ plane and the corresponding transformed mesh in the $\xi-\eta$ plane. We write

$$\left.\frac{\partial\phi}{\partial\xi}\right|_P = \frac{\phi_E-\phi_W}{2\Delta\xi} \tag{4.52}$$

$$\left.\frac{\partial\phi}{\partial\eta}\right|_P = \frac{\phi_N-\phi_S}{2\Delta\eta} \tag{4.53}$$

Knowing the cell gradients, we can compute the face value required in the secondary diffusion term (Equation 4.29) by averaging. For example, for a uniform mesh

$$\left.\frac{\partial\phi}{\partial\eta}\right|_f = \frac{\left.\frac{\partial\phi}{\partial\eta}\right|_P + \left.\frac{\partial\phi}{\partial\eta}\right|_E}{2} \tag{4.54}$$

An alternative approach would be to write the face derivative in terms of the values at the nodes a and b

$$\left.\frac{\partial\phi}{\partial\eta}\right|_f = \frac{\phi_b-\phi_a}{\Delta\eta} \tag{4.55}$$

For Cartesian grids, it is easy to show that the two approximations are equivalent if the nodal values are obtained by linear interpolation. For an equispaced Cartesian grid this means that the nodal value ϕ_a is

$$\phi_a = \frac{\phi_P+\phi_E+\phi_{SE}+\phi_S}{4} \tag{4.56}$$

The second interpretation (Equation 4.55) is useful because it suggests one way of obtaining derivatives at faces for general unstructured grid where we no longer have

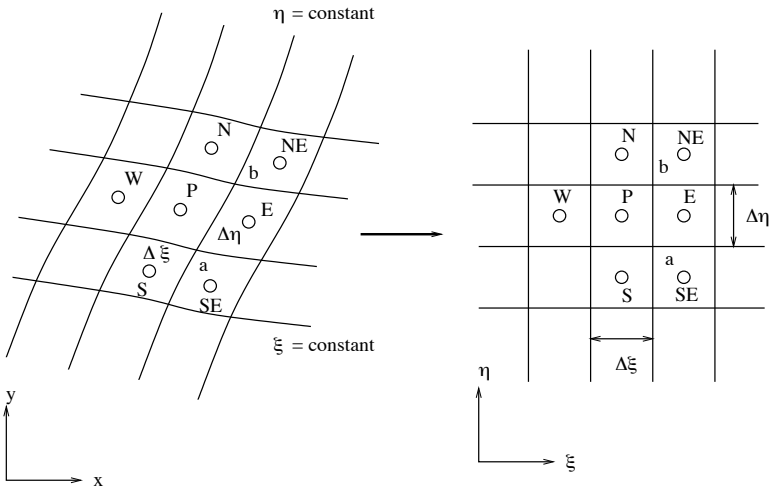


Figure 4.7: Structured Mesh in (a) Physical Space and (b) Mapped Space

orthogonal directions to apply the finite-difference approximation. Thus, we can use Equation 4.55 for the face f shown in Figure 4.3. The nodal values in this case are obtained by some averaging procedure over the surrounding cells.

4.4.2 Unstructured Meshes

The method outlined above is applicable for arbitrary unstructured grids in two dimensions. However, as we commented earlier, extension to three dimensions is not straightforward. This is because in 3D we no longer have unique directions in the plane of the face to use a finite-difference approximation. Also, in many instances we need to know all three components of the derivative at cell centers, not just derivatives in the plane of the cell face. Therefore we need to seek other ways of calculating derivatives that are applicable for arbitrary grids.

Gradient Theorem Approach

One approach is suggested by the gradient theorem which states that for any closed volume $\Delta\mathscr{V}_0$ surrounded by surface A

$$\int_{\Delta\mathscr{V}_0} \nabla\phi d\mathscr{V} = \int_A \phi d\mathbf{A} \tag{4.57}$$

where d$\mathbf{A}$ is the outward-pointing incremental area vector. To obtain a discrete version of Equation 4.57, we make our usual round of profile assumptions. First, we assume that the gradient in the cell is constant. This yields

$$\nabla\phi = \frac{1}{\Delta\mathscr{V}_0} \sum_f \int_A \phi d\mathbf{A}_f \tag{4.58}$$

Next, we approximate the integral over a cell face by the face centroid value times the face area. Thus we can write

$$\nabla\phi = \frac{1}{\Delta\mathcal{V}_0}\sum_f \phi_f \mathbf{A}_f \tag{4.59}$$

We still need to define the face value, ϕ_f before we can use this formula. The simple approximation is to use the average of the two cells values sharing the face:

$$\phi_f = \frac{\phi_0 + \phi_1}{2} \tag{4.60}$$

The advantage of this approach is that it is applicable for arbitrary cell shapes, including non-conformal grids. All the operations involved in this procedure are face-based just like the operations involved in the discretization of the transport equations and do not require any additional grid connectivity. Also, this procedure is easily extended to three-dimensional cases.

Once we have obtained the derivative by using Equations 4.59 and 4.60, we can improve on our initial approximation of the face average by *reconstructing* from the cell. Thus, from Figure 4.8 we can write

$$\phi_f = \frac{(\phi_0 + \nabla\phi_0 \cdot \Delta\mathbf{r}_0) + (\phi_1 + \nabla\phi_1 \cdot \Delta\mathbf{r}_1)}{2} \tag{4.61}$$

This suggests an iterative approach for computing successively better approximations to the gradients. During each iteration, we can compute the face average value using the gradients computed from the previous iteration and use these face values to compute new values of the gradients. However, this increases the effective stencil with increasing iterations and can lead to oscillatory results. In practice, therefore only one or two iterations are typically used. In addition, as we will see in the next chapter, the gradients used to reconstruct face values are also *limited* to the bounds dictated by suitable neighbor values, so as to avoid undershoots and overshoots in the solution.

Note that in applying the gradient theorem we used the cell around the point *C*0 as the integration volume. While this practice involves the smallest possible stencil, it is not mandatory that we use the same control volume for computing the gradient that we use for applying the discrete conservation laws. Other integration volumes are often used, specially in node-based discretization algorithms.

The gradient resulting from the use of the cell as the integration volume involves values of ϕ only at the face neighbors and is not always the most optimal solution. In the next section we learn about the use of another approach that can use a bigger stencil.

Least Squares Approach

The idea here is to compute the gradient at a cell such that it reconstructs the solution in the neighborhood of the cell. For example, consider cell *C*0. We would like the value of ϕ computed at the centroid of cell C1 in Figure 4.9 to be equal to ϕ_1. By assuming a locally linear variation of ϕ, we write

$$\phi_0 + \nabla\phi_0 \cdot \Delta\mathbf{r}_1 = \phi_1 \tag{4.62}$$

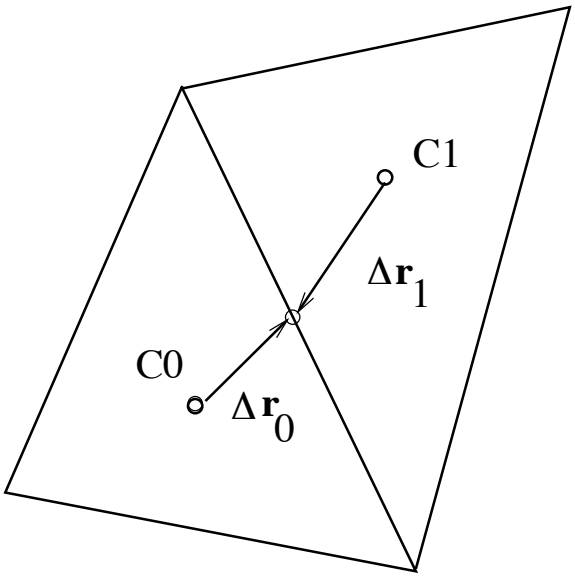


Figure 4.8: Arrangement of Cells in Unstructured Mesh

Here $\Delta\mathbf{r}_1$ s the vector from the centroid of cell 0 to the centroid of cell 1

$$\Delta\mathbf{r}_1 = \Delta x_1\mathbf{i} + \Delta y_1\mathbf{j} \tag{4.63}$$

We rewrite Equation 4.62 as

$$\Delta x_1 \left.\frac{\partial\phi}{\partial x}\right|_0 + \Delta y_1 \left.\frac{\partial\phi}{\partial y}\right|_0 = \phi_1 - \phi_0 \tag{4.64}$$

We require that the same be true at all other cells surrounding cell $C0$. For a cell Cj the equation reads

$$\Delta x_j \left.\frac{\partial\phi}{\partial x}\right|_0 + \Delta y_j \left.\frac{\partial\phi}{\partial y}\right|_0 = \phi_j - \phi_0 \tag{4.65}$$

It is convenient to assemble all the equations in a matrix form as follows

$$\mathbf{Md} = \blacksquare\phi \tag{4.66}$$

Here $\mathbf{M}$ is the $J \times 2$ matrix

$$\mathbf{M} = \begin{bmatrix} \Delta x_1 & \Delta y_1 \\ \Delta x_2 & \Delta y_2 \\ \vdots & \vdots \\ \Delta x_J & \Delta y_J \end{bmatrix} \tag{4.67}$$

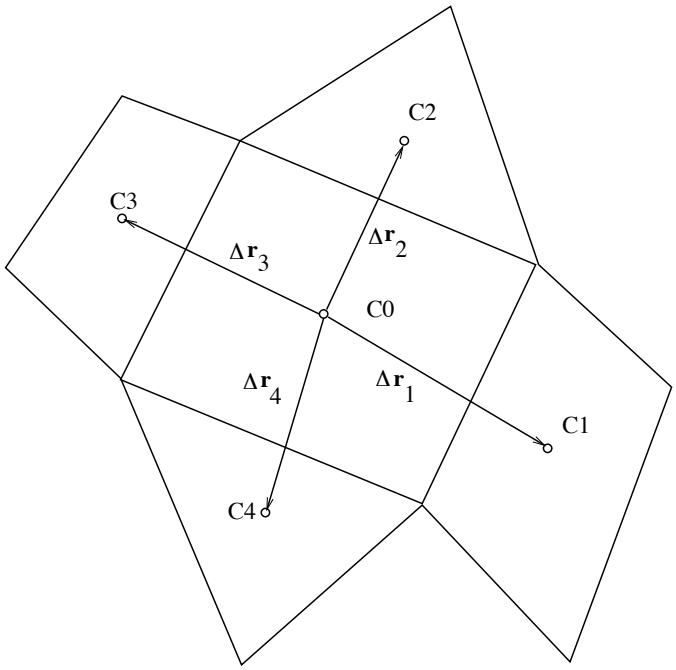


Figure 4.9: Least Squares Calculation of Cell Gradient

and $\mathbf{d}$ is the vector of the components of gradients of ϕ at cell $C0$

$$\mathbf{d} = \begin{bmatrix} \left.\frac{\partial \phi}{\partial x}\right|_0 \\ \\ \left.\frac{\partial \phi}{\partial y}\right|_0 \end{bmatrix} \tag{4.68}$$

and $\blacksquare\phi$ is the vector of differences of ϕ

$$\blacksquare\phi = \begin{bmatrix} \phi_1 - \phi_0 \\ \phi_2 - \phi_0 \\ \vdots \\ \phi_J - \phi_0 \end{bmatrix} \tag{4.69}$$

Equation 4.66 represents J equations in two unknowns. Since in general J is larger than 2 this is an over-determined system. Physically, this means that we cannot assume a linear profile for ϕ around the cell $C0$ such that it exactly reconstructs the known solution at all of its neighbors. We can only hope to find a solution that fits the data in the best possible way, i.e., a solution for which the RMS value of the difference between the neighboring cell values and the reconstructed values is minimized. From Equation 4.65 we know that the difference in the reconstructed value and the cell value

for cell Cj is given by

$$R_j = \Delta x_j \left.\frac{\partial \phi}{\partial x}\right|_0 + \Delta y_j \left.\frac{\partial \phi}{\partial y}\right|_i - (\phi_j - \phi_0) \tag{4.70}$$

The sum of the squares of error over all the neighboring cells is

$$R = \sum_j R_j^2 \tag{4.71}$$

Let $\left.\frac{\partial \phi}{\partial x}\right|_0 = a$ and $\left.\frac{\partial \phi}{\partial y}\right|_0 = b$. Equation 4.71 can then be written as

$$R = \sum_j \left(a\Delta x_j + b\Delta y_j - (\phi_j - \phi_0) \right)^2 \tag{4.72}$$

Our objective is to find a and b such that R is minimized. Recall that the standard way of solving the problem is to differentiate R with respect to a and b and set the result to zero, ie.,

$$\begin{aligned} \frac{\partial R}{\partial a} &= 0 \\ \frac{\partial R}{\partial b} &= 0 \end{aligned} \tag{4.73}$$

This gives us two equations in the two unknowns, viz. the components of the gradient at cell $C0$. It is easy to show that this equation set is the same as that obtained by multiplying Equation 4.66 by the transpose of the matrix $\mathbf{M}$

$$\mathbf{M}^T \mathbf{M} \mathbf{d} = \mathbf{M}^T \blacksquare\phi \tag{4.74}$$

$\mathbf{M}^T\mathbf{M}$ is a 2×2 matrix that can be easily inverted analytically to yield the required gradient $\nabla\phi$. We should note here that since $\mathbf{M}$ is purely a function of geometry, the inversion only needs to be done once. In practical implementations, we would compute a matrix of weights for each cell. The gradient for any scalar can then be computed easily by multiplying the matrix with the difference vector $\blacksquare\phi$.

Mathematically, for the solution to exist the matrix $\mathbf{M}$ must be non-singular, i.e., it should have linearly independent columns and its rank must be greater than or equal to 2. Physically this implies that we must involve at least 3 non-collinear points to compute the gradient. Another way of understanding this requirement is to note that assuming a linear variation means that ϕ is expressed as

$$\phi = A + Bx + Cy \tag{4.75}$$

Since this involves three unknowns, we need at least three points at which ϕ is specified in order to determine the gradient.

The least squares approach is easily extended to three dimensions. We note that it places no restrictions on cell shape, and does not require a structured mesh. It also does not require us to choose only face neighbors for the reconstruction. At corner

boundaries, for example, it may not be possible to obtain a sufficient number of face neighbor cells, and we may be forced to reconstruct at cells sharing vertices with cell C0 in addition to the usual complement of face neighbors. In other cases, we may wish to involve more cells near cell $C0$ to get a better estimate of the gradient. This requires storing additional mesh connectivity information.

4.5 Influence of Secondary Gradients on Coefficients

We noted earlier that the presence of secondary gradient terms introduces the possibility that ϕ_P may not be bounded by its neighbors even when $S = 0$. Here we examine this assertion in somewhat greater detail.

For simplicity, let us consider the calculation of secondary gradient the term on the structured mesh shown in Figure 4.7. To computed the secondary gradient term, we must find the gradient $(\partial\phi/\partial\eta)_f$. As we noted before, one way to find this is to write

$$\left(\frac{\partial\phi}{\partial\eta}\right)_f = \frac{\phi_b - \phi_a}{\Delta\eta} \tag{4.76}$$

For a uniform non-orthogonal mesh with $\Delta\xi = \Delta\eta$, we may write

$$\begin{aligned}\phi_a &= \frac{\phi_P + \phi_E + \phi_{SE} + \phi_S}{4} \\ \phi_b &= \frac{\phi_P + \phi_E + \phi_{NE} + \phi_N}{4}\end{aligned} \tag{4.77}$$

so that

$$\left(\frac{\partial\phi}{\partial\eta}\right)_f = \frac{0.5\left(\phi_N + \phi_{NE}\right) - 0.5\left(\phi_S + \phi_{SE}\right)}{2\Delta\eta} \tag{4.78}$$

We note that $\phi_N, \phi_{NE}, \phi_S$ and ϕ_{SE} do not all have the same sign in the above expression. When $(\partial\phi/\partial\eta)_f$ is included in the cell balance as a part of the secondary gradient term for the face, it effectively introduces additional neighbors – ϕ_{NE} and ϕ_{SE}. These are not *face* neighbors; the corresponding cells share vertices with point P. These terms are hidden in $\mathscr{S}_f$. Notice that they do not all have positive coefficients. Consequently it is possible for an increase in one of the neighbor ϕ's to result in a decrease in ϕ_P. We are no longer guaranteed that ϕ_P is bounded by its neighbors when $S = 0$.

Even though we have adopted a particular gradient calculation method here, similar terms result from other calculation methods as well. The magnitude of the secondary gradient terms is proportional to $\mathbf{e}_\xi \cdot \mathbf{e}_\eta$. For most good quality meshes, this term is nearly zero, and the influence of neighbors with negative coefficients is relatively small. Thus, for good quality meshes, our discussions in the previous chapter about coefficient positivity and boundedness of ϕ hold in large measure. However, we no longer have the absolute guarantee of boundedness and positivity that we had with orthogonal meshes.

4.6 Implementation Issues

4.6.1 Data Structures

For both structured and unstructured meshes, the face flux $\mathbf{J}_f$ is most conveniently construed as belonging to the face f rather than to either of the cells sharing the face. Such an interpretation is in keeping with the conservation idea, whereby the diffusive flux of ϕ out of cell $C0$ enters cell $C1$ without modification. We can ensure this by thinking of $\mathbf{J}_f$ as belonging to the face, and using it for the cell balance in cells $C0$ and $C1$ in turn. This association of the flux with the *face* and not the *cell* makes a *face-based* data structure a convenient one for implementing finite volume schemes.

In a typical implementation, we would carry a linked list or array of faces. Each face would carry a pointer or index to each of the two cells that share it, i.e., cells $C0$ and $C1$. The influence of cell $C1$ on the equation for cell $C0$ is given by a_{01}; the influence of cell $C0$ on the equation for cell $C1$ is given by a_{10}. Thus, we compute the coefficient

$$a_{nb} = \frac{\Gamma_f}{\Delta\xi} \frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \tag{4.79}$$

for the face and make the assignment:

$$\begin{aligned} a_{01} &\leftarrow a_{nb} \\ a_{10} &\leftarrow a_{nb} \end{aligned} \tag{4.80}$$

A visit to all the faces in the list completes the calculation of the neighbor coefficients a_{nb} for all the cells in the domain.

If the cell gradient is available, the face value of the gradient may be found by averaging, as in Equation 4.33. The secondary gradient terms may then be computed during the visit to the face since they are also associated with the face. Each face contributes a secondary gradient term to the b term for cells $C0$ and $C1$:

$$\begin{aligned} b_0 &\leftarrow b_0 - \mathscr{S}_f \\ b_1 &\leftarrow b_1 + \mathscr{S}_f \end{aligned}$$

Notice that the secondary gradient term is added to one cell and subtracted from the other. This is because the face flux leaves one cell and enters the other.

It is also useful to carry a linked list or array of cells. Once the coefficient calculation is complete, a_P for all cells may be computed by visiting each cell, computing $S_P \Delta\mathscr{V}$ and summing a_{nb}. Similarly the S_C contribution to b may also be computed.

Not all gradient calculation procedures are amenable to a purely face-based implementation. Calculations based on the gradient theorem are amenable to a face-based calculation procedure. Here, the face value is computed using Equation 4.60 during the visit to the face, as well as the contribution to the sum in Equation 4.59 for each of the cells sharing the face. The reconstruction procedure described by Equation 4.61 may also be implemented in a face-based manner. The least-squares approach for gradient calculation may be implemented by a mixture of face and cell-based manipulations, depending on the calculation stencil chosen.

4.6.2 Overall Solution Loop

It is convenient to compute and store cell gradients prior to coefficient calculation. Using the gradient theorem to compute cell gradients, for example, the overall solution loop takes the following nominal form.

1. Guess ϕ at all cell centroids and at boundary face centroids as necessary.
2. For $\mathrm{f} = 1, \mathrm{n}_{\mathrm{faces}}$
   {
   Find ϕ_f by averaging neighbor cell values ϕ_0 and ϕ_1.
   Add gradient contribution to cells $C0$ and $C1$.
   }
   A visit to all faces completes the cell gradient calculation.
3. For $\mathrm{f} = 1, \mathrm{n}_{\mathrm{faces}}$
   {
   Find a_{01} and a_{10}.
   Find $\mathscr{S}_f$; find b_0 and b_1 by adding/subtracting secondary gradient contributions to $C0$ and $C1$.
   }
4. For $\mathrm{c} = 1, \mathrm{n}_{\mathrm{cells}}$
   {
   Find $a_P = \sum_{nb} a_{nb} - S_P \Delta \mathscr{V}$
   Find $b = b + S_C \Delta \mathscr{V}$
   }
   At this point the coefficient calculation is complete.
5. Solve for ϕ at cell centroids using a linear solver such as Gauss-Seidel iteration.
6. Check for convergence. If iterations are converged, stop. Else go to 2.

We refer to one pass through the above loop as an *outer* iteration. During an outer iteration, we make one call to a linear solver, such as a Gauss-Seidel solver. The Gauss-Seidel solver may perform a number of *inner* iterations to obtain the solution to the nominally linear system. For linear problems on orthogonal meshes, only one outer iteration (with sufficient inner iterations of the linear solver) would be required to obtain the converged solution. For non-linear problems, many outer iterations would be required. For non-orthogonal meshes, the above procedure employs a *deferred* computation of secondary gradient terms. Consequently, many outer iterations are required for convergence, even for linear problems.

A comment on the Scarborough criterion is appropriate here. Since the Scarborough criterion is satisfied by the primary diffusion terms when Dirichlet boundary conditions are present, we are guaranteed convergence of the Gauss-Seidel solver during any outer iteration. In our deferred calculation procedure, the secondary gradient terms remain fixed during an outer iteration. Therefore, even though the negative coefficients they introduce tend to violate the Scarborough criterion, they are not relevant since the secondary gradient terms are held constant during the Gauss-Seidel iteration.

4.7 Closure

In this chapter, we have seen how to discretize the diffusion equation on non-orthogonal meshes, both structured and unstructured. The overall idea is the same as for regular meshes, and involves a balance of diffusive fluxes on the faces of the cell with the source inside the cell. However, we have seen that the face fluxes may no longer be written purely in terms of the neighbor cell values if the mesh is non-orthogonal; an extra secondary gradient term appears. To compute this term, we require face gradients of ϕ, which may be averaged from cell gradients. We have seen how cell gradients may be computed for structured and unstructured meshes. Finally, we have seen how much of the calculation is amenable to a face-based data structure. In the next chapter, we address the discretization of the convective term, and therefore, the solution of a complete scalar transport equation.

Chapter 5

Convection

In this chapter, we turn to the remaining term in the general transport equation for the scalar ϕ, namely the convection term. We will see how to difference this term in the framework of the finite volume method, and the special problems associated with it. We will address both regular and non-orthogonal meshes. Once we have addressed this term, we will have developed a tool for solving the complete scalar transport equation, or the *convection-diffusion* equation, as it is sometimes called in the literature.

In the development that follows, we will assume that the fluid flow is known,i.e., the velocity vector $\mathbf{V}$ is known at all the requisite points in the domain. We seek to determine how the scalar ϕ is transported in the presence of a *given* fluid flow field. In reality, of course, the fluid flow would have to be computed. We will address the calculation of the fluid flow in later chapters.

5.1 Two-Dimensional Convection and Diffusion in A Rectangular Domain

Let us consider a two-dimensional rectangular domain such as that shown in Figure 5.1. The domain has been discretized using a regular Cartesian mesh. For the sake of clarity, let us assume that Δx and Δy are constant, i.e., the mesh is uniform in each of the directions x and y. As before, we store discrete values of ϕ at cell centroids.

The equation governing steady scalar transport in the domain is given by

$$\nabla \cdot \mathbf{J} = S \tag{5.1}$$

where $\mathbf{J}$ is given by

$$\mathbf{J} = \rho \mathbf{V} \phi - \Gamma \nabla \phi \tag{5.2}$$

Here, ρ is the density of the fluid and $\mathbf{V}$ is its velocity, and is given by

$$\mathbf{V} = u\mathbf{i} + v\mathbf{j} \tag{5.3}$$

As before, we integrate Equation 5.1 over the cell of interest, P, so that

$$\int_{\Delta\mathcal{V}} \nabla \cdot \mathbf{J} d\mathcal{V} = \int_{\Delta\mathcal{V}} S d\mathcal{V} \tag{5.4}$$

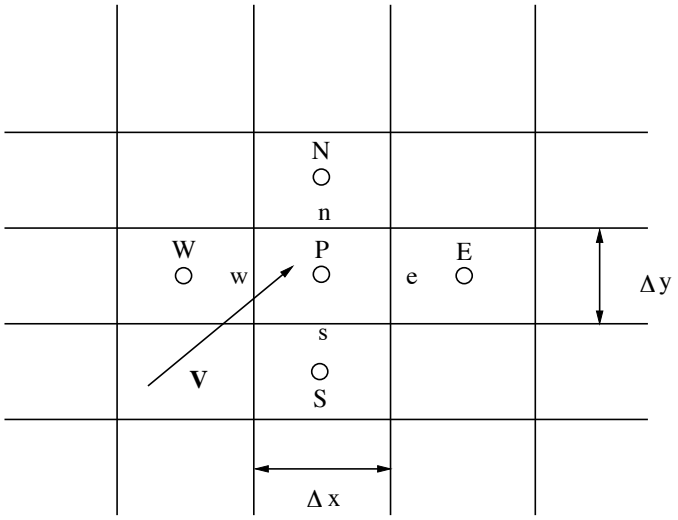


Figure 5.1: Convection on a Cartesian Mesh

Applying the divergence theorem, we write

$$\int_A \mathbf{J}\cdot d\mathbf{A} = \overline{S}\Delta\mathscr{V}_P \tag{5.5}$$

As usual, we assume that $\mathbf{J}$ is constant over each of the faces e, w, n and s of the cell P, and that the face centroid value is representative of the face average. Also, we assume $\overline{S} = S_C + S_P\phi_P$ as before. Thus

$$\mathbf{J}_e\cdot\mathbf{A}_e + \mathbf{J}_w\cdot\mathbf{A}_w + \mathbf{J}_n\cdot\mathbf{A}_n + \mathbf{J}_s\cdot\mathbf{A}_s = \left(S_C + S_P\phi_P\right)\Delta\mathscr{V}_P \tag{5.6}$$

with

$$\begin{aligned} \mathbf{A}_e &= \Delta y\,\mathbf{i} \\ \mathbf{A}_w &= -\Delta y\,\mathbf{i} \\ \mathbf{A}_n &= \Delta x\,\mathbf{j} \\ \mathbf{A}_s &= -\Delta x\,\mathbf{j} \end{aligned} \tag{5.7}$$

Thus far the discretization process is identical to our procedure in previous chapters. Let us now consider one of the transport terms, say, $\mathbf{J}_e\cdot\mathbf{A}_e$. This is given by

$$\mathbf{J}_e\cdot\mathbf{A}_e = (\rho u\phi)_e\Delta y - \Gamma_e\Delta y\left(\frac{\partial\phi}{\partial x}\right)_e \tag{5.8}$$

We see from Equation 5.8 that the convection component has the form

$$F_e\phi_e \tag{5.9}$$

where

$$F_e = (\rho u)_e \Delta y \tag{5.10}$$

is the mass flow rate through the face e. We note that the *face* value, ϕ_e, is required to determine the transport due to convection at the face. For the purposes of this chapter, we shall assume that F_e is given.

We have seen that the diffusive term on the face e may be written as

$$-D_e(\phi_E - \phi_P) \tag{5.11}$$

where

$$D_e = \Gamma_e \frac{\Delta y}{(\delta x)_e} \tag{5.12}$$

Similar diffusion terms can be written for other faces as well. The quantity

$$Pe = \frac{F}{D} = \frac{\rho u \delta x}{\Gamma} \tag{5.13}$$

is called the *Peclet* number, and measures the relative importance of convection and diffusion in the transport of ϕ. If it is based on a cell length scale, δx, it is referred to as the *cell* Peclet number.

We see that writing the face flux $\mathbf{J}_e$ requires two types of information: the face value ϕ_e, and the face gradient $(\partial\phi/\partial x)_e$. We already know how to write the face gradient. We turn now to different methods for writing the face value ϕ_e in terms of the cell centroid values. Once ϕ_e is determined, it is then simply a matter of doing the same operation on all the faces, collecting terms, and writing the discrete equation for the cell P.

5.1.1 Central Differencing

The problem of discretizing the convection term reduces to finding an interpolation for ϕ_e from the cell centroid values of ϕ. One approximation we can use is the *central-difference* approximation. Here, we assume that ϕ varies linearly between grid points. For a uniform mesh, we may write

$$\phi_e = \frac{\phi_E + \phi_P}{2} \tag{5.14}$$

so that the convective transport through the face is

$$F_e \frac{(\phi_E + \phi_P)}{2} \tag{5.15}$$

Similar expressions may be written for the convective contributions on other faces. We note with trepidation that ϕ_E and ϕ_P appear with the same sign in Equation 5.15.

Collecting the convection and diffusion terms on all faces, we may write the following discrete equation for the cell P:

$$a_P \phi_P = \sum_{nb} a_{nb} \phi_{nb} + b$$

where

$$\begin{aligned} a_E &= D_e - \frac{F_e}{2} \\ a_W &= D_w + \frac{F_w}{2} \\ a_N &= D_n - \frac{F_n}{2} \\ a_S &= D_s + \frac{F_s}{2} \\ a_P &= \sum_{nb} a_{nb} - S_P \Delta \mathscr{V}_P + (F_e - F_w + F_n - F_s) \\ b &= S_C \Delta \mathscr{V}_P \end{aligned} \tag{5.17}$$

In the above equations,

$$\begin{aligned} D_e &= \Gamma_e \frac{\Delta y}{(\delta x)_e} \\ D_w &= \Gamma_w \frac{\Delta y}{(\delta x)_w} \\ D_n &= \Gamma_n \frac{\Delta x}{(\delta y)_n} \\ D_s &= \Gamma_s \frac{\Delta x}{(\delta y)_s} \\ F_e &= (\rho u)_e \Delta y \\ F_w &= (\rho u)_w \Delta y \\ F_n &= (\rho v)_n \Delta x \\ F_s &= (\rho v)_s \Delta x \end{aligned} \tag{5.18}$$

The term

$$(F_e - F_w + F_n - F_s) \tag{5.19}$$

represents the net mass outflow from the cell P. If the underlying flow field satisfies the continuity equation, we would expect this term to be zero.

Let us consider the case when the velocity vector $\mathbf{V} = (u\mathbf{i} + v\mathbf{j})$ is such that $u > 0$ and $v > 0$. For $F_e > 2D_e$, we see that a_E becomes negative. Similarly for $F_n > 2D_n$, a_N becomes negative. (Similar behavior would occur with a_W and a_S if the velocity vector reverses sign). These negative coefficients mean that though $a_P = \sum_{nb} a_{nb}$ for $S = 0$, we are not guaranteed that ϕ_P is bounded by its neighbors. Furthermore, since the equation violates the Scarborough criterion, we are not guaranteed the convergence of the Gauss-Seidel iterative scheme.

For $u > 0$ and $v > 0$, we see that as long as $F_e < 2D_e$ and $F_n < 2D_n$, we are guaranteed positive coefficients, and physically plausible behavior. That is, the face Peclet numbers $Pe_e = F_e/D_e < 2$ and $Pe_n = F_n/D_n < 2$ are required for uniform meshes. For a given velocity field and physical properties we can meet this Peclet number criterion

by reducing the grid size sufficiently. For many practical situations, however, the resulting mesh may be very fine, and the storage and computational requirements may be too large to afford.

5.1.2 Upwind Differencing

When we examine the discretization procedure described above, we realize that the reason we encounter negative coefficients is the arithmetic averaging in Equation 5.14. We now consider an alternative differencing procedure call the *upwind* differencing scheme. In this scheme, the face value of ϕ is set equal to the *upwind* cell centroid value. Thus, for face e in Figure 5.1, we write

$$\begin{aligned} \phi_e &= \phi_P \quad \text{if } F_e \geq 0 \\ &= \phi_E \quad \text{if } F_e < 0 \end{aligned} \tag{5.20}$$

These expressions essentially say that the value of ϕ on the face is determined entirely by the mesh direction from which the flow is coming to the face. Similar expressions may be written on the other faces. Using Equation 5.20 in the cell balance for cell P, and the diffusion term discretization in Equation 5.11, we may write the following discrete equation for cell P:

$$a_P \phi_P = \sum_{nb} a_{nb} \phi_{nb} + b$$

where

$$\begin{aligned} a_E &= D_e + Max[-F_e, 0] \\ a_W &= D_w + Max[F_w, 0] \\ a_N &= D_n + Max[-F_n, 0] \\ a_S &= D_s + Max[F_s, 0] \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} - S_P \Delta \mathscr{V}_P + (F_e - F_w + F_n - F_s) \\ b &= S_C \Delta \mathscr{V}_P \end{aligned} \tag{5.22}$$

Here

$$\begin{aligned} Max[a,b] &= a \quad \text{if } a > b \\ &= b \quad \text{otherwise} \end{aligned} \tag{5.23}$$

We see that the upwind scheme yields positive coefficients, and $a_P = \sum_{\text{nb}} a_{\text{nb}}$ if the flow field satisfies the continuity equation and $S = 0$. Consequently, we are guaranteed that ϕ_P is bounded by its neighbors.

We will see in later sections that though the upwind scheme produces a coefficient matrix that guarantees physically plausible results and is ideally suited for iterative linear solvers, it can smear discontinuous profiles of ϕ even in the absence of diffusion. We will examine other *higher order* differencing schemes which do not have this characteristic.

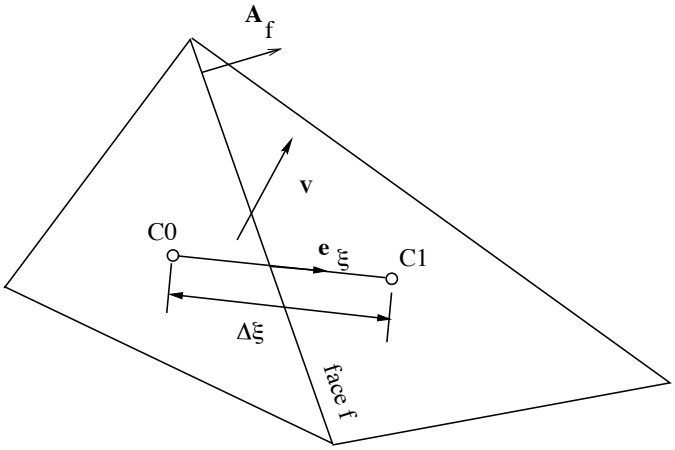


Figure 5.2: Convection on a Non-Orthogonal Mesh

5.2 Convection-Diffusion on Non-Orthogonal Meshes

For non-orthogonal meshes, both structured and unstructured, counterparts of the central difference and upwind schemes are easily derived. We start with the integration of the convection-diffusion equation as usual, integrate it over the cell C0 shown in Figure 5.2, and apply the divergence theorem to get

$$\int_A \mathbf{J}\cdot d\mathbf{A} = \left(S_C + S_P\phi_0\right)\Delta\mathscr{V}_0 \tag{5.24}$$

As before, we assume that the flux on the face may be written in terms of the face centroid value, so that

$$\sum_f \mathbf{J}_f\cdot\mathbf{A}_f = \left(S_C + S_P\phi_0\right)\Delta\mathscr{V}_0 \tag{5.25}$$

where the summation is over the faces f of the cell.The flux is given by

$$\mathbf{J}_f = (\rho\mathbf{V}\phi)_f - \Gamma_f(\nabla\phi)_f \tag{5.26}$$

The transport of ϕ at the face f may thus be written as

$$\mathbf{J}_f\cdot\mathbf{A}_f = (\rho\mathbf{V})_f\cdot\mathbf{A}_f\phi_f - \Gamma_f(\nabla\phi)_f\cdot\mathbf{A}_f \tag{5.27}$$

We define the face mass flow rate as

$$F_f = (\rho\mathbf{V})_f\cdot\mathbf{A}_f \tag{5.28}$$

This is the mass flow rate *out* of the cell C0.

We have already seen in the previous chapter that the diffusion transport at the face may be written as:

$$-\frac{\Gamma_f}{\Delta\xi}\frac{\mathbf{A}_f\cdot\mathbf{A}_f}{\mathbf{A}_f\cdot\mathbf{e}_\xi}\left(\phi_1 - \phi_0\right) + \mathscr{S}_f \tag{5.29}$$

Defining

$$D_f = \frac{\Gamma_f}{\Delta\xi}\frac{\mathbf{A}_f \cdot \mathbf{A}_f}{\mathbf{A}_f \cdot \mathbf{e}_\xi} \tag{5.30}$$

we write the net transport across the face f as

$$\mathbf{J}_f \cdot \mathbf{A}_f = F_f \phi_f - D_f\left(\phi_1 - \phi_0\right) + \mathscr{S}_f \tag{5.31}$$

We note that, as with regular meshes, the convective transport of ϕ at the face requires the evaluation of the face value ϕ_f.

5.2.1 Central Difference Approximation

As with regular meshes, we may find ϕ_f through either a central difference or an upwind approximation. The simplest central difference approximation is to write

$$\phi_f = \frac{\phi_0 + \phi_1}{2} \tag{5.32}$$

With this approximation, the following discrete equation is obtained:

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b$$

where

$$\begin{aligned} a_{\text{nb}} &= D_f - \frac{F_f}{2} \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} - S_P \Delta\mathscr{V}_0 + \sum_f F_f \\ b &= S_C \Delta\mathscr{V}_0 - \sum_{\text{nb}} \left(\mathscr{S}_f\right)_{\text{nb}} \end{aligned} \tag{5.34}$$

We see that, just as with structured meshes, it is possible to get negative coefficients using central differencing. If $F_f > 0$, we expect $a_{\text{nb}} < 0$ if $F_f/D_f > 2$, ie, if the Peclet number $Pe_f > 2$. As before, the quantity $\sum_f F_f$ is the sum of the outgoing mass fluxes for the cell, and is expected to be zero if the underlying flow field satisfies mass balance.

5.2.2 Upwind Differencing Approximation

Under the upwind differencing approximation,

$$\begin{aligned} \phi_f &= \phi_0 \quad \text{if } F_f > 0 \\ &= \phi_1 \quad \text{otherwise} \end{aligned} \tag{5.35}$$

Using this approximation in the discrete cell balance we get

$$a_P \phi_P = \sum_{\text{nb}} a_{\text{nb}} \phi_{\text{nb}} + b$$

where

$$\begin{aligned} a_{\text{nb}} &= D_f + Max[-F_f, 0] \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} - S_P \Delta \mathscr{V}_0 + \sum_f F_f \\ b &= S_C \Delta \mathscr{V}_0 - \sum_{\text{nb}} \left(\mathscr{S}_f \right)_{\text{nb}} \end{aligned} \tag{5.37}$$

As with structured meshes, we see that the a_{nb} is always guaranteed positive.

Thus, we see that the discretization for unstructured meshes yields a coefficient structure that is very similar to that for regular meshes. In both cases, the central difference scheme introduces the possibility of negative coefficients for cell Peclet numbers greater than 2 for uniform meshes. The upwind scheme produces positive coefficients, but, as we will see in the next section, this comes at the cost of accuracy.

5.3 Accuracy of Upwind and Central Difference Schemes

Let us consider the truncation error associated with the upwind and central difference schemes. Let us assume a uniform mesh, as shown in Figure 5.3. Using a Taylor series expansion about point e, we may write

$$\phi_P = \phi_e - \left(\frac{\Delta x}{2} \right) \left(\frac{d\phi}{dx} \right)_e + \frac{1}{2} \left(\frac{\Delta x}{2} \right)^2 \left(\frac{d^2\phi}{dx^2} \right)_e + O\left((\Delta x)^3 \right) \tag{5.38}$$

$$\phi_E = \phi_e + \left(\frac{\Delta x}{2} \right) \left(\frac{d\phi}{dx} \right)_e + \frac{1}{2} \left(\frac{\Delta x}{2} \right)^2 \left(\frac{d^2\phi}{dx^2} \right)_e + O\left((\Delta x)^3 \right) \tag{5.39}$$

From Equation 5.38, we see that

$$\phi_e = \phi_P + O(\Delta x) \tag{5.40}$$

Recall that we use Equation 5.40 when $F_e > 0$. We see that upwind differencing is only first order accurate.

Adding Equations 5.38 and 5.39, dividing by two, and rearranging terms, we get

$$\phi_e = \frac{\phi_P + \phi_E}{2} - \frac{(\Delta x)^2}{8} \left(\frac{d^2\phi}{dx^2} \right)_e + O\left((\Delta x)^3 \right) \tag{5.41}$$

We see that the central difference approximation is second-order accurate.

5.3.1 An Illustrative Example

Consider convection of a scalar ϕ over the square domain shown in Figure 5.4. The left and bottom boundaries are held at $\phi = 0$ and $\phi = 1$ respectively. The flow field in the domain is given by

$$\mathbf{V} = 1.0\mathbf{i} + 1.0\mathbf{j} \tag{5.42}$$

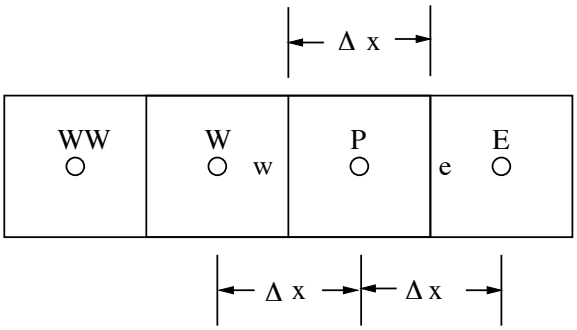


Figure 5.3: One-Dimensional Control Volume

so the velocity vector is aligned with the diagonal as shown. We wish to compute the distribution of ϕ in the domain using the upwind and central difference schemes for. The flow is governed by the domain Peclet number $Pe = \rho|\mathbf{V}|L/\Gamma$. For $\Gamma = 0$, i.e., $Pe \to \infty$, the solution is $\phi = 1$ below the diagonal and $\phi = 0$ above the diagonal. For other values of Pe, we expect a diffusion layer in which $0 < \phi < 1$. The diffusion layer is wider for smaller Pe. We compute the steady convection-diffusion problem in the domain using 13×16 quadrilateral cells to discretize the domain. We consider the case $Pe \to \infty$. Figure 5.5 shows the predicted ϕ values along the vertical centerline of the domain (x=0.5). We see that the upwind scheme smears the *phi* profile so that there is a diffusion layer even when there is no physical diffusion. The central difference scheme, on the other hand, shows unphysical oscillations in the value of *phi*. In this problem, it is not possible to control these oscillations by refining the mesh, since the cell Peclet number is infinite no matter how fine the mesh.

5.3.2 False Diffusion and Dispersion

We can gain greater insight into the behavior of the upwind and central difference schemes through the use of model equations. The main drawback of the first order upwind scheme is that it is very diffusive. To understand the reasons behind this we develop a *model equation* for the scheme. Consider case of steady two-dimensional convection with no diffusion:

$$\frac{\partial}{\partial x}(\rho u \phi) + \frac{\partial}{\partial y}(\rho v \phi) = 0 \tag{5.43}$$

Consider the case of a constant velocity field, with $u > 0, v > 0$, and ρ constant. Using upwind differencing, we obtain the following discrete form

$$\rho u \frac{\phi_P - \phi_W}{\Delta x} + \rho v \frac{\phi_P - \phi_S}{\Delta y} = 0 \tag{5.44}$$

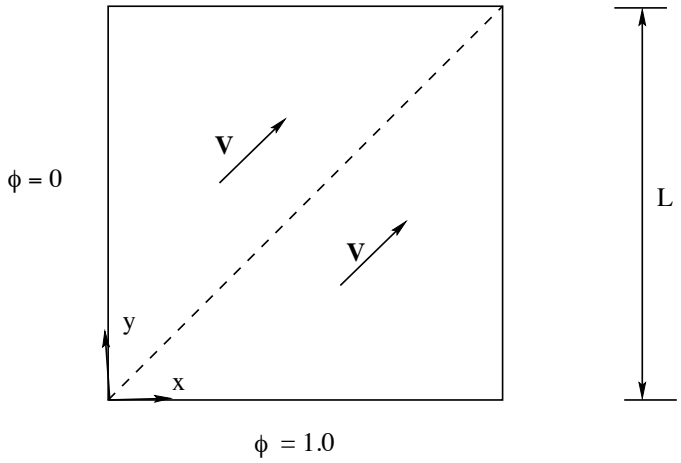


Figure 5.4: Convection and Diffusion in a Square Domain

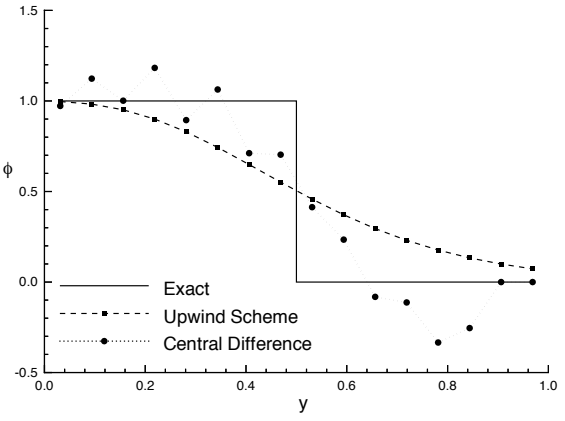


Figure 5.5: Variation of ϕ Along Vertical Centerline

Consider the computational domain shown in Figure 5.1.Expanding ϕ_W and ϕ_S about ϕ_P using a Taylor series, we get

$$\phi_W = \phi_P - \Delta x \frac{\partial \phi}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 \phi}{\partial x^2} - \frac{(\Delta x)^3}{3!}\frac{\partial^3 \phi}{\partial x^3} + \ldots \tag{5.45}$$

$$\phi_S = \phi_P - \Delta y \frac{\partial \phi}{\partial y} + \frac{(\Delta y)^2}{2!}\frac{\partial^2 \phi}{\partial y^2} - \frac{(\Delta y)^3}{3!}\frac{\partial^3 \phi}{\partial y^3} + \ldots \tag{5.46}$$

All derivatives in the above equations are evaluated at P. Rearranging,

$$\frac{\phi_P - \phi_W}{\Delta x} = \frac{\partial \phi}{\partial x} - \frac{(\Delta x)}{2!}\frac{\partial^2 \phi}{\partial x^2} + \frac{(\Delta x)^2}{3!}\frac{\partial^3 \phi}{\partial x^3} + \ldots \tag{5.47}$$

$$\frac{\phi_P - \phi_S}{\Delta y} = \frac{\partial \phi}{\partial y} - \frac{(\Delta y)}{2!}\frac{\partial^2 \phi}{\partial y^2} + \frac{(\Delta y)^2}{3!}\frac{\partial^3 \phi}{\partial y^3} + \ldots \tag{5.48}$$

Using the above expressions in Eq 5.44 and rearranging we obtain

$$\rho u \frac{\partial \phi}{\partial x} + \rho v \frac{\partial \phi}{\partial y} = \frac{\rho u \Delta x}{2}\frac{\partial^2 \phi}{\partial x^2} + \frac{\rho v \Delta x}{2}\frac{\partial^2 \phi}{\partial y^2} + O\left((\Delta x)^2\right) + O\left((\Delta y)^2\right) \tag{5.49}$$

For simplicity let us consider the case when $u = v$ and $\Delta x = \Delta y$. Equation 5.50 reduces to

$$\rho u \frac{\partial \phi}{\partial x} + \rho v \frac{\partial \phi}{\partial y} = \frac{\rho u \Delta x}{2}\left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}\right) + O\left((\Delta x)^2\right) \tag{5.50}$$

The differential equation derived in this manner is referred to as the *equivalent* or *modified* equation. It represents the continuous equation that our finite-volume numerical scheme is effectively modeling. The left hand side of Equation 5.50 is our original differential equation and the right hand side represents the truncation error.The leading term in Equation 5.50 is $O(\Delta x)$, as expected for a first-order scheme. It is interesting to compare Equation5.50 with the one dimensional form of the convection-diffusion equation (Equation 5.1). We find that the leading order error term in Equation 5.50 looks similar to the diffusion term in the convection-diffusion equation. Thus we see that although we are trying to solve a pure convection problem, applying the upwind scheme means that effectively we are getting the solution to a problem with some diffusion. This phenomenon is variously called *artificial*, *false* or *numerical* diffusion (or viscosity, in the context of momentum equations). In case of the upwind scheme, the artificial diffusion coefficient is proportional to the grid size so we would expect its effects to decrease as we refine the grid, but it is always present.

The same analysis for the two-dimensional central difference scheme starts with Equation 5.43 and the discrete equation

$$\rho u \frac{\phi_E - \phi_W}{2\Delta x} + \rho v \frac{\phi_N - \phi_S}{2\Delta y} = 0 \tag{5.51}$$

Expanding ϕ_W and ϕ_E about ϕ_P using a Taylor series, we get

$$\phi_W = \phi_P - \Delta x \frac{\partial \phi}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 \phi}{\partial x^2} - \frac{(\Delta x)^3}{3!}\frac{\partial^3 \phi}{\partial x^3} + \ldots \tag{5.52}$$

$$\phi_E = \phi_P + \Delta x \frac{\partial \phi}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 \phi}{\partial x^2} - \frac{(\Delta x)^3}{3!}\frac{\partial^3 \phi}{\partial x^3} + \ldots \tag{5.53}$$

Subtracting the two equations, we get

$$\frac{\phi_E - \phi_W}{2\Delta x} = \frac{\partial \phi}{\partial x} + \frac{(\Delta x)^2}{3!}\frac{\partial^3 \phi}{\partial x^3} + \ldots \tag{5.54}$$

A similar procedure in the y-direction yields

$$\frac{\phi_N - \phi_S}{2\Delta y} = \frac{\partial \phi}{\partial y} + \frac{(\Delta y)^2}{3!}\frac{\partial^3 \phi}{\partial y^3} + \ldots \tag{5.55}$$

As before, we consider the case when $u = v$ and $\Delta x = \Delta y$. Substituting Equations 5.54 and 5.55 into Equation 5.51 we get

$$\rho u \frac{\partial \phi}{\partial x} + \rho v \frac{\partial \phi}{\partial y} = -\frac{\rho u \Delta x^2}{3!}\left(\frac{\partial^3 \phi}{\partial x^3} + \frac{\partial^3 \phi}{\partial y^3}\right) + O\left((\Delta x)^3\right) \tag{5.56}$$

We see that the leading truncation term is $O(\Delta x^2)$, as expected in a second order scheme. Though we set out to solve a pure convection equation, the effective equation we solve using the central difference scheme contains a third derivative term on the right hand side. This term is responsible for *dispersion*, i.e., for the oscillatory behavior of ϕ. Thus, the upwind scheme leads to false or artificial diffusion which tends to smear sharp gradients, whereas the central-difference scheme tends to be dispersive.

5.4 First-Order Schemes Using Exact Solutions

A number of first-order schemes for the convection-diffusion equation have been published in the literature which treat the convection and diffusion terms together, rather than discretizing them separately. These schemes use local profile assumptions which are approximations to the exact solution to a local convection-diffusion equation. We present these here for historical completeness. Their behavior in multi-dimensional situations has the same characteristics as the upwind scheme.

5.4.1 Exponential Scheme

The one-dimensional convection diffusion equation with no source term may be written as

$$\frac{\partial}{\partial x}(\rho u \phi) - \frac{\partial}{\partial x}\left(\Gamma \frac{\partial \phi}{\partial x}\right) = 0 \tag{5.57}$$

The boundary conditions are

$$\begin{aligned} \phi &= \phi_0 \quad \text{atx} = 0 \\ \phi &= \phi_L \quad \text{atx} = \text{L} \end{aligned} \tag{5.58}$$

The exact solution to the problem is

$$\frac{\phi - \phi_0}{\phi_L - \phi_0} = \frac{exp((Pe)x/L) - 1}{exp(Pe) - 1} \tag{5.59}$$

where Pe is the Peclet number given by

$$Pe = \frac{\rho u L}{\Gamma} \tag{5.60}$$

We wiish to use this exact solution in making profile assumptions. Consider the one-dimensional mesh shown in Figure 5.3 and the convection-diffusion equation:

$$\frac{\partial}{\partial x}(\rho u \phi) - \frac{\partial}{\partial x}\left(\Gamma \frac{\partial \phi}{\partial x}\right) = S \tag{5.61}$$

We wish to obtain a discrete equation for point P. Integrating Equation 5.61 over the cell P yields

$$\mathbf{J}_e \cdot \mathbf{A}_e + \mathbf{J}_w \cdot \mathbf{A}_w = \left(S_C + S_P \phi_P\right) \Delta \mathcal{V}_P \tag{5.62}$$

Assuming, for this 1-D case that

$$\begin{aligned} \mathbf{A}_e &= \mathbf{i} \\ \mathbf{A}_w &= -\mathbf{i} \end{aligned} \tag{5.63}$$

we may write

$$\begin{aligned} \mathbf{J}_e \cdot \mathbf{A}_e &= (\rho u \phi)_e - \Gamma_e \left(\frac{d\phi}{dx}\right)_e \\ \mathbf{J}_w \cdot \mathbf{A}_w &= -(\rho u \phi)_w + \Gamma_w \left(\frac{d\phi}{dx}\right)_w \end{aligned} \tag{5.64}$$

As before, we must make profile assumptions to write ϕ_e, ϕ_w and the gradients $(d\phi/dx)_e$ and $(d\phi/dx)_w$. We assume that $\phi(x)$ may be taken from Equation 5.59. We will use this profile to evaluate *both* ϕ and $d\phi/dx$ at the face. Thus

$$\mathbf{J}_e \cdot \mathbf{A}_e = F_e \left(\phi_P + \frac{\phi_P - \phi_E}{exp(Pe_e) - 1}\right) \tag{5.65}$$

Here

$$Pe_e = \frac{(\rho u)_e \delta x_e}{\Gamma_e} = \frac{F_e}{D_e} \tag{5.66}$$

A similar expression may be written for the w face. Collecting terms yields the following discrete equation for ϕ_P:

$$a_P \phi_P = a_E \phi_E + a_W \phi_W + b$$

where

$$\begin{aligned} a_E &= \frac{F_e}{exp(F_e/D_e)-1} \\ a_W &= \frac{F_w exp(F_w/D_w)}{exp(F_w/D_w)-1} \\ a_P &= a_E + a_W - S_P \Delta \mathscr{V}_P + (F_e - F_w) \\ b &= S_C \Delta \mathscr{V}_P \end{aligned} \tag{5.68}$$

This scheme always yields positive coefficients and bounded solutions. For the case of $S = 0$, for one-dimensional situations, it will yield the exact solution regardless of mesh size or Peclet number. Of course, this is not true when a source term exists or when the situation is multi-dimensional. It is possible to show that for these general situations the scheme is only first-order accurate.

Because exponentials are expensive to compute, researchers have created schemes which approximate the behavior of the coefficients obtained using the exponential scheme. These include the hybrid and power law schemes which are described below.

5.4.2 Hybrid Scheme

The hybrid scheme seeks to approximate the behavior of the discrete coefficients from the exponential scheme by reproducing their limiting behavior correctly. The coefficient a_E in the exponential scheme may be written as

$$\frac{a_E}{D_e} = \frac{P_e}{exp(Pe_e)-1} \tag{5.69}$$

A plot of a_E/D_e is shown in Figure 5.6. It shows the following limiting behavior:

$$\begin{aligned} \frac{a_E}{D_e} &\rightarrow 0 \quad \text{for } Pe_e \rightarrow \infty \\ \frac{a_E}{D_e} &\rightarrow -Pe_e \quad \text{for } Pe_e \rightarrow -\infty \\ \frac{a_E}{D_e} &= 1 - \frac{Pe_e}{2} \quad \text{at } Pe_e = 0 \end{aligned} \tag{5.70}$$

The hybrid scheme models a_E/D_e using the three bounding tangents shown in Figure 5.6. Thus

$$\begin{aligned} \frac{a_E}{D_e} &= 0 \quad \text{for } Pe_e > 2 \\ \frac{a_E}{D_e} &= 1 - \frac{Pe_e}{2} \quad \text{for } -2 \geq Pe_e \leq 2 \\ \frac{a_E}{D_e} &= -Pe_e \quad \text{for } Pe_e < -2 \end{aligned} \tag{5.71}$$

The overall discrete equation for the cell P is given by

$$a_P \phi_P = a_E \phi_E + a_W \phi_W + b$$

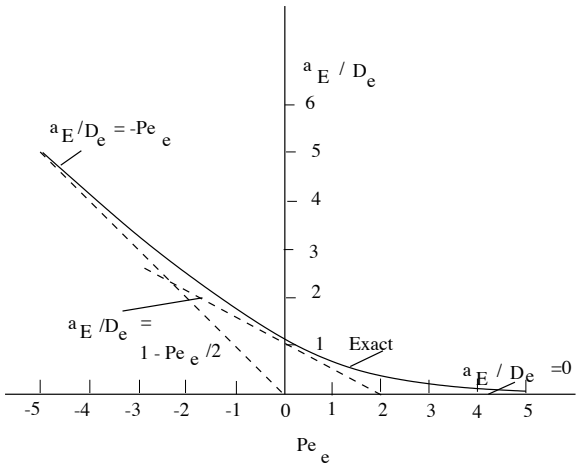


Figure 5.6: Variation of a_E/D_e (Adapted from Patankar(1980))

where

$$\begin{aligned} a_E &= Max[-F_e, D_e - \frac{F_e}{2}, 0] \\ a_W &= Max[F_w, D_w + \frac{F_w}{2}, 0] \\ a_P &= a_E + a_W - S_P \Delta \mathscr{V}_P + (F_e - F_w) \\ b &= S_C \Delta \mathscr{V}_P \end{aligned} \tag{5.73}$$

5.4.3 Power Law Scheme

Here, the objective is to curve-fit the a_E/D_e curve using a fifth-order polynomial, rather than to use the bounding tangents as the hybrid scheme does. The power-law expressions for a_E/D_e may be written as:

$$\frac{a_E}{D_e} = Max[0, \left(1 - \frac{0.1|F_e|}{D_e}\right)^5] + Max[0, -F_e] \tag{5.74}$$

This expression has the advantage that it is less expensive to compute that the exponential scheme, while reproducing its behavior closely.

5.5 Unsteady Convection

Let us now focus our attention on unsteady convection. For simplicity we will set ρ to be unity and $\Gamma = 0$. The convection-diffusion equation (Equation 5.1) then takes the

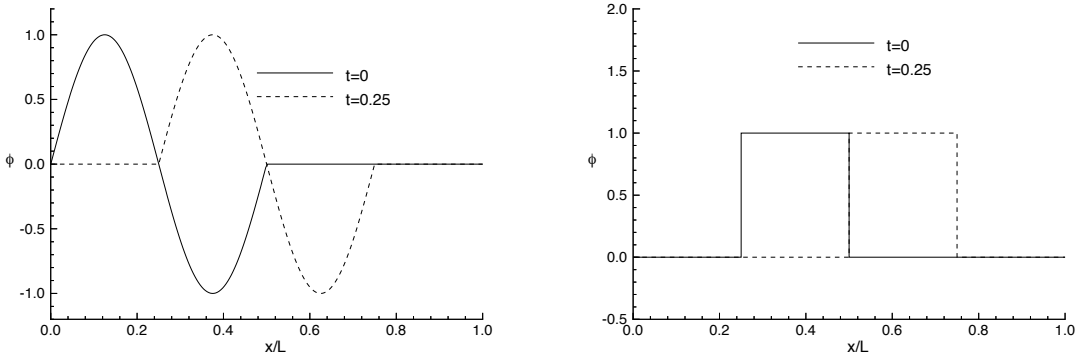


Figure 5.7: Exact solutions for linear convection of (a) sine wave (b) square wave

form

$$\frac{\partial \phi}{\partial t} + \frac{\partial u\phi}{\partial x} = 0 \tag{5.75}$$

Equation 5.75 is also known as the *linear advection* or *linear wave* equation. It is termed linear because the convection speed u is not a function of the convected quantity ϕ. We saw in a previous chapter that mathematically this is classified as a hyperbolic equation. To complete the specification of the problem we need to define the initial and boundary conditions. Let us consider a domain of length L and let the initial solution be given by a spatial function ϕ_0, i.e.,

$$\phi(x,0) = \phi_0(x) \tag{5.76}$$

The exact solution to this problem is given by

$$\phi(x,t) = \phi_0(x - ut) \tag{5.77}$$

In other words, the exact solution is simply the initial profile translated by a distance $-ut$.

Consider the convection of two initial profiles by a convection velocity $u = 1$ in a domain of length L, as shown in Figure 5.7. One is a single sine wave and the other is a square wave, The boundary conditions for both problems are $\phi(0,t) = \phi(L,t) = 0$. Figure 5.7 shows the initial solution as well as the solution at $t = 0.25$. We see that the profiles have merely shifted to the right by $0.25u$. We will use these examples to determine whether our discretization schemes are able to predict this translation accurately, without distorting or smearing the profile.

Even though Equation 5.75 appears to be quite simple, it is important because it provides a great deal of insight into the treatment of the more complex, non-linear, coupled equations that govern high speed flows. For these reasons, historically it has been one of the most widely studied equations in CFD. Many different approaches

have been developed for its numerical solution but in this book we will concentrate mostly on modern techniques that form the basis of solution algorithms for the Euler and Navier-Stokes equations.

5.5.1 1D Finite Volume Discretization

For simplicity let us examine the discretization of Equation 5.75. We assume a uniform mesh with a cell width Δx as shown in Figure 5.3. Integrating Equation 5.75 over the cell P yields

$$\left(\frac{\partial \phi}{\partial t}\right)_P \Delta x + (u_e \phi_e - u_w \phi_w) = 0 \tag{5.78}$$

Since u is constant, we may rearrange the equation to write

$$\left(\frac{\partial \phi}{\partial t}\right)_P + \frac{u}{\Delta x}(\phi_e - \phi_w) = 0 \tag{5.79}$$

Since for the linear problem the velocity u is known everywhere, the problem of determination of the face flux simply reduces to the determination of the face values ϕ.

In general, we are interested in two kinds of problems. In some instances we might be interested in the transient evolution of the solution. Since the equation is hyperbolic in the time coordinate and the solution only depends on the past and not the future, it would seem logical to devise methods that yield the solution at successive instants in time, starting with the initial solution, using *time marching*. Often, however, only the steady-state solution (i.e., the solution as $\frac{\partial \phi}{\partial t} \to 0$) is of interest. In the previous chapter we saw that we could obtain the steady-state solution by solving a sparse system of nominally linear algebraic equations, iterating for non-linearities. An alternative to iterations is to use time-marching, and to obtain the steady state solution as the culmination of an unsteady process. With the general framework in hand, let's look at some specific schemes.

5.5.2 Central Difference Scheme

Using the explicit scheme we developed in a previous chapter, we may write

$$\frac{\phi_P - \phi_P^0}{\Delta t} + u\frac{(\phi_E^0 - \phi_W^0)}{2\Delta x} = 0 \tag{5.80}$$

where, as per our convention, the un-superscripted values denote the values at the current time, and the terms carrying the superscript "0" denote the value at the previous time level. As we noted earlier, in the explicit scheme, ϕ_P for every cell is only a function of the (known) solution at the previous time level.

We have shown from a truncation error analysis that the central difference scheme is second-order accurate in space; explicit time discretization is first order accurate in time. Thus the scheme described above is second-order accurate in space and first-order accurate in time. Applying the von-Neumann stability analysis to this scheme, however, shows that it is unconditionally unstable. As such it is not usable but there are several important lessons one can learn from this.

The first point to note is that the instability is not caused by either the spatial or the temporal discretization alone but by the combination of the two. Indeed, we get very different behavior if we simply using an implicit temporal discretization while retaining the central difference scheme for the spatial term:

$$\frac{\phi_P - \phi_P^0}{\Delta t} + u\frac{(\phi_E - \phi_W)}{2\Delta x} = 0 \tag{5.81}$$

It is easy to show that the resulting implicit scheme is unconditionally stable. However, it is important to realize that stability does not guarantee physical plausibility in the solution. We may rewrite the scheme in the following form:

$$\phi_P = u\Delta t\frac{(\phi_E - \phi_W)}{2\Delta x} + \phi_P^0 \tag{5.82}$$

We see that a_W is negative for $u > 0$ and a_E is negative for $u < 0$. The solution is therefore not guaranteed to be bounded by the spatial and temporal neighbor values. Since the scheme is implicit, it requires the solution of a linear equation set at each time step. However, the Scarborough criterion is not satisfied, making it difficult to use iterative solvers.

5.5.3 First Order Upwind Scheme

Using the upwind difference scheme for spatial discretization and an explicit time discretization, we obtain the following scheme

$$\frac{\phi_P - \phi_P^0}{\Delta t} + u\frac{(\phi_P^0 - \phi_W^0)}{\Delta x} = 0 \tag{5.83}$$

We have shown that the upwind differencing scheme is only first-order accurate and that the explicit scheme is also only first-order accurate. Stability analysis reveals that the scheme is stable as long as

$$0 \leq u\frac{\Delta t}{\Delta x} \leq 1 \tag{5.84}$$

The quantity $\nu = u\frac{\Delta t}{\Delta x}$ is known as the *Courant* or *CFL* number (after Courant, Friedrichs and Lewy [1, 2]) who first analyzed the convergence characteristics of such schemes. Explicit schemes usually have a stability limit which dictates the maximum Courant number that can be used. This limits the time step and makes the use of time marching with explicit schemes undesirable for steady state problems.

It is interesting to note that our heuristic requirement of all spatial and temporal neighbor coefficients being positive is also met when the above condition is satisfied. This is easily seen by writing Equation 5.83 in the form

$$\phi_P = (1 - \nu)\phi_P^0 + \nu\phi_W \tag{5.85}$$

We therefore the expect the scheme to also be monotone when it is stable.

To see how well the upwind scheme performs for unsteady problems let us apply it to the problem of convection of single sine and square waves we described above. To

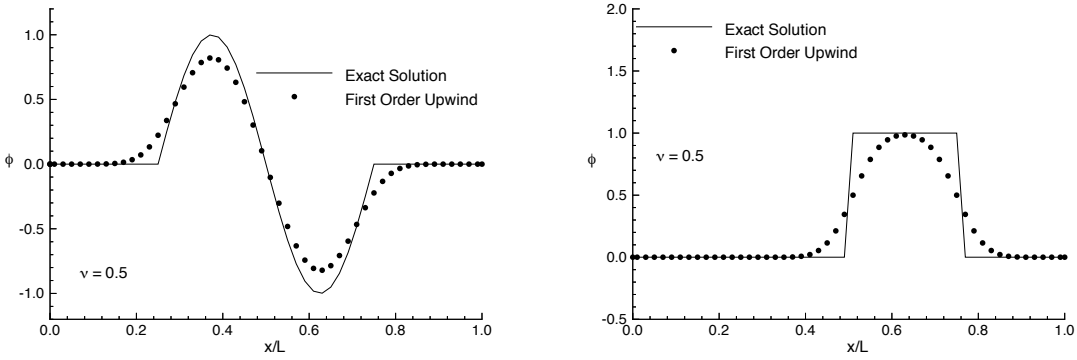


Figure 5.8: First order upwind solutions for linear convection of (a) sine wave (b) square wave

compute this numerical solution we use an equispaced solution mesh of 50 cells. Using $\nu = 0.5$, we apply the explicit scheme for 25 time steps. We compare the resulting solutions at $t = 0.25$ with the exact solutions in Figure 5.8. The exact solution, as expected, is the initial profile translated to the right by a distance of 0.25. Our numerical solution is similarly shifted but we note that the sharp discontinuities in either the slope of the variable itself have been smoothened out considerably. In case of the sine wave, the peak amplitude has decreased but nowhere has the solution exceeded the initial bounds. We also observe that the profiles in both examples are *monotonic*.

5.5.4 Error Analysis

We can develop a model equation for the transient form of the upwind scheme (Equation 5.83) using techniques similar those we used for steady state to obtain

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = \frac{u\Delta x}{2}(1-\nu)\frac{\partial^2 \phi}{\partial x^2} + \ldots \tag{5.86}$$

We see that, just like steady state, the transient form of the upwind scheme also suffers from numerical dissipation, which is now a function of the Courant number.

We know that physically the effect of diffusion (or viscosity) is to smoothen out the gradients. The numerical diffusion present in the upwind scheme acts in a similar manner and this is why we find the profiles in Figure 5.8 are smoothened. Although this results in a loss of accuracy, this same artificial dissipation is also responsible for the stability of our scheme. This is because the artificial viscosity also damps out any errors that might arise during the course of time marching (or iterations) and prevents these errors from growing. Note also that for $\nu > 1$ the numerical viscosity of the upwind scheme would be *negative*. We can now appreciate the physical reason behind

the CFL condition; when it is violated the numerical viscosity will be negative and thus cause any errors to grow.

Let us now analyze the explicit central difference scheme (Equation 5.80) we saw earlier. The corresponding modified equation is

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = -\frac{u^2 \Delta t}{2}\frac{\partial^2 \phi}{\partial x^2} + \ldots \tag{5.87}$$

This scheme has a *negative* artificial viscosity in all cases and therefore it is not very surprising that it is unconditionally unstable. It can also be shown that the implicit version (Equation 5.81 has the following modified equation

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = \frac{u^2 \Delta t}{2}\frac{\partial^2 \phi}{\partial x^2} + \ldots \tag{5.88}$$

Thus this scheme is stable but also suffers from artificial diffusion.

5.5.5 Lax-Wendroff Scheme

A large number of schemes have been developed to overcome the shortcomings of the explicit central difference scheme that we discussed in the previous section. Of these, the most important is the Lax-Wendroff scheme since it forms the basis of several well-known schemes used for solution of Euler and compressible Navier-Stokes equations.

The principle idea is to remove the negative artificial diffusion of the explicit central difference scheme by adding an equal amount of positive diffusion. That is, we seek to solve

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = \frac{u^2 \Delta t}{2}\frac{\partial^2 \phi}{\partial x^2} \tag{5.89}$$

rather than the original convection equation. Discretizing the second derivative using linear profiles assumptions, as in previous chapters, we obtain the following explicit equation for cell P

$$\frac{\phi_P - \phi_P^0}{\Delta t} + u\frac{(\phi_E^0 - \phi_W^0)}{2\Delta x} - \frac{u^2 \Delta t}{2}\frac{(\phi_E^0 - 2\phi_P^0 + \phi_W^0)}{(\Delta x)^2} = 0 \tag{5.90}$$

It is possible to show that this scheme is second order accurate in both space and time. Stability analysis shows that it is stable for $|\nu| \leq 1$. Applying it to the sine wave convection problem, we see that it resolves the smoothly varying regions of the profile much better than the upwind scheme (see Figure 5.9(a)). However, in regions of slope discontinuity we see spurious "wiggles". Such non-monotonic behavior is even more pronounced in the presence of discontinuities, as shown in Figure 5.9(b). We also note that the solution in this case exceeds the initial bounds. At some locations it is higher than one while in other places it is negative. If ϕ is a physical variable such as species concentration or the turbulence kinetic energy that is always supposed to be positive, such behavior could cause a lot of difficulties in our numerical procedure.

The reasons for this behavior can once again be understood by examining the truncation error. The modified equation for the Lax-Wendroff scheme is

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial t} = -u\frac{(\Delta x)^2}{6}(1-\nu^2)\frac{\partial^3 \phi}{\partial x^3} + O(\Delta x)^3 \tag{5.91}$$

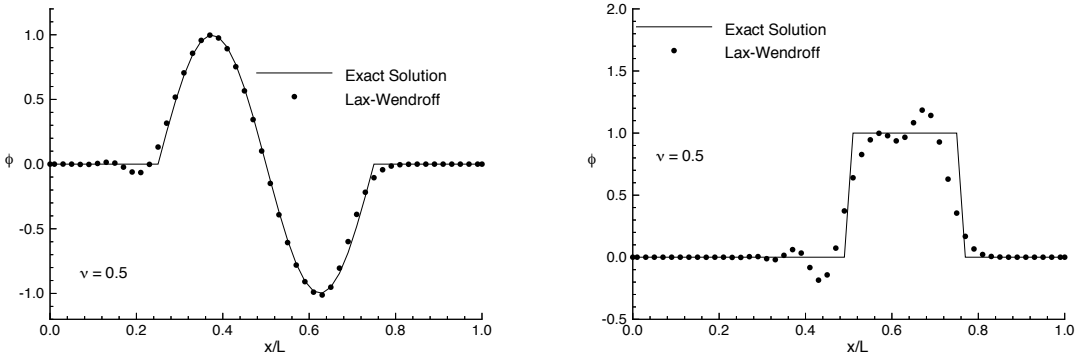


Figure 5.9: Lax-Wendroff solutions for linear convection of (a) sine wave (b) square wave

The leading order term is proportional to the *third* derivative of ϕ and there is no diffusion like term. This is characteristic of second-order schemes and is the reason why wiggles in the solution do not get damped. The errors produced by second-order terms are *dispersive* in nature whereas those produced by first-order schemes are dissipative. In wave-mechanics terms, dispersion refers to the phenomenon that alters the frequency content of a signal and dissipation refers to the reduction in amplitude. For smooth profiles that contain few frequencies, second-order schemes work very well; the lack of numerical diffusion preserves the amplitude. However, in case of discontinuities (which are composed of many frequencies), the effect of numerical dispersion is to cause phase errors between the different frequencies. The first order upwind scheme, on the other hand, does not alter the phase differences but damps all modes.

All the numerical schemes we saw in previous sections were derived by *separate* profile assumptions for the spatial and temporal variations. The Lax-Wendroff scheme is different in that the spatial profile assumption is tied to the temporal discretization. This becomes more clear if we re-arrange Equation 5.90 in the following form

$$\frac{\phi_P - \phi_P^0}{\Delta t} + \frac{u}{\Delta x}\left\{\left[\frac{\phi_E^0 + \phi_P^0}{2} - \frac{u\Delta t}{2\Delta x}(\phi_E^0 - \phi_P^0)\right] - \left[\frac{\phi_P^0 + \phi_W^0}{2} - \frac{u\Delta t}{2\Delta x}(\phi_P^0 - \phi_W^0)\right]\right\} = 0$$

We recognize the terms in square brackets as the face flux definitions for the e and w faces respectively. If we are using this scheme for computing a steady state solution, the equation that is satisfied at convergence is given by

$$\left[\frac{\phi_E + \phi_P}{2} - \frac{u\Delta t}{2\Delta x}(\phi_E - \phi_P)\right] - \left[\frac{\phi_P + \phi_W}{2} - \frac{u\Delta t}{2\Delta x}(\phi_P - \phi_W)\right] = 0 \qquad (5.92)$$

The consequence of this is that the final answer we obtain depends on the time-step size Δt ! Although the solution still has a spatial truncation error of $O(\Delta x)^2$), this

dependence on the time step is clearly not physical. In this particular case, the resulting error in the final solution maybe small since stability requirements restrict the time step size. We will see in later chapters that similar path dependence of the converged solution can occur even in iterative schemes if we are not careful.

5.6 Higher-Order Schemes

We have seen thus far that both the upwind and central difference schemes have severe limitations, the former due to artificial diffusion, and the latter due to dispersion. Therefore there has been a great deal of research to improve the accuracy of the upwind scheme, by using higher-order interpolation. These higher-order schemes aim to obtain at least a second-order truncation error, while controlling the severity of spatial oscillations.

Thus far, we have assume that, for the purposes of writing the face value ϕ_e, the profile of ϕ is essentially constant. That is, for $F_e > 0$,

$$\phi_e = \phi_P \tag{5.93}$$

Instead of assume a constant profile assumption for ϕ, we may use higher-order profile assumptions, such as linear or quadratic, to derive a set of *upwind weighted* higher-order schemes. If $F_e > 0$, we write a Taylor series expansion for ϕ in the neighborhood of the *upwind* point P:

$$\phi(x) = \phi_P + (x - x_P)\frac{\partial \phi}{\partial x} + \frac{(x - x_P)^2}{2!}\frac{\partial^2 \phi}{\partial x^2} + O(\Delta x)^3 \tag{5.94}$$

5.6.1 Second-Order Upwind Schemes

We may derive a second-order upwind scheme by making a linear profile assumption. This is equivalent to retaining the first two terms of the expansion. Evaluating Equation 5.94 at $x_e = x_P + (\Delta x)/2$, we obtain

$$\phi_e = \phi_P + \frac{\Delta x}{2}\frac{\partial \phi}{\partial x} \tag{5.95}$$

This assumption has a truncation error of $O(\Delta x)^2$. In order to write ϕ_e in terms of cell centroid values, we must write $\frac{\partial \phi}{\partial x}$ in terms of cell centroid values. On our one-dimensional grid we can represent the derivative at P using either a forward, backward or central difference formula to give us three different second-order schemes. If we write $\frac{\partial \phi}{\partial x}$ using

$$\frac{\partial \phi}{\partial x} = \frac{\phi_E - \phi_W}{2\Delta x} \tag{5.96}$$

we obtain

$$\phi_e = \phi_P + \frac{(\phi_E - \phi_W)}{4} \tag{5.97}$$

or, adding and subtracting $\phi_P/4$, we get

$$\phi_e = \phi_P + \frac{\left(\phi_P - \phi_W\right)}{4} + \frac{\left(\phi_E - \phi_P\right)}{4} \tag{5.98}$$

This scheme is referred to as the Fromm scheme in the literature.

If we write $\frac{\partial \phi}{\partial x}$ using

$$\frac{\partial \phi}{\partial x} = \frac{\phi_P - \phi_W}{\Delta x} \tag{5.99}$$

we obtain

$$\phi_e = \phi_P + \frac{\left(\phi_P - \phi_W\right)}{2} \tag{5.100}$$

This scheme is referred to in the literature as the Beam-Warming scheme.

5.6.2 Third-Order Upwind Schemes

We may derive third-order accurate schemes by retaining the second derivative in the Taylor series expansion:

$$\phi(x) = \phi_P + (x - x_P)\frac{\partial \phi}{\partial x} + \frac{(x - x_P)^2}{2!}\frac{\partial^2 \phi}{\partial x^2} \tag{5.101}$$

and using cell-centroid values to write the derivatives $\frac{\partial \phi}{\partial x}$ and $\frac{\partial^2 \phi}{\partial x^2}$. Using

$$\frac{\partial \phi}{\partial x} = \frac{\left(\phi_E - \phi_W\right)}{2\Delta x} + O(\Delta x^2) \tag{5.102}$$

and

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\left(\phi_E + \phi_W - 2\phi_P\right)}{(\Delta x)^2} + O(\Delta x^2) \tag{5.103}$$

we may write

$$\phi_e = \phi_P + \frac{\left(\phi_E - \phi_W\right)}{4} + \frac{\left(\phi_E + \phi_W - 2\phi_P\right)}{8} \tag{5.104}$$

Re-arranging, we may write

$$\phi_e = \frac{(\phi_E + \phi_P)}{2} - \frac{\left(\phi_E + \phi_W - 2\phi_P\right)}{8} \tag{5.105}$$

This scheme is called the QUICK scheme (Quadratic Upwind Interpolation for Convective Kinetics) [3]. This scheme may be viewed as a parabolic correction to linear interpolation for ϕ_e. We can emphasize this by introducing a *curvature factor* C such that

$$\phi_e = \frac{1}{2}(\phi_E + \phi_P) - C(\phi_E + \phi_W - 2\phi_P) \tag{5.106}$$

and C=1/8.

The second- and third-order schemes we have seen here may be combined into a single expression for ϕ_e using

$$\phi_e = \phi_P + \frac{(1-\kappa)}{4}(\phi_P - \phi_W) + \frac{(1+\kappa)}{4}(\phi_E - \phi_P) \tag{5.107}$$

Here, $\kappa = -1$ yields the Beam-Warming scheme, $\kappa = 0$ the Fromm scheme and $\kappa = 1/2$ the QUICK scheme. For $\kappa = 1$ we get the familiar central difference scheme.

We see from the signs of the terms in Equation 5.107 that it is possible to produce negative coefficients in our discrete equation using these higher-order schemes. However, the extent of the resulting spatial oscillations is substantially smaller than those obtained through the central difference scheme, while retaining at least second-order accuracy.

5.6.3 Implementation Issues

If iterative solvers are used to solve the resulting set of discrete equations, it is important to ensure that the Scarborough criterion is satisfied by the nominally linear equations presented to the iterative solver. Consequently, typical implementations of higher-order schemes use deferred correction strategies whereby the higher order terms are included as corrections to an upwind flux. For the QUICK scheme, for example, the convective transport $F_e\phi_e$ for $F_e > 0$ is written as

$$F_e\phi_e = F_e\phi_P + F_e\left(\frac{(\phi_E^* + \phi_P^*)}{2} - \frac{(\phi_E^* + \phi_W^* - 2\phi_P^*)}{8} - \phi_P^*\right) \tag{5.108}$$

Here, the first term on the right hand side represents the upwind flux. The second term is a correction term, and represents the difference between the QUICK and upwind fluxes. The upwind term is included in the calculation of the coefficients a_p and a_{nb} while the correction term is included in the b term. It is evaluated using the prevailing value of ϕ. At convergence, $\phi_P = \phi_P^*$, and the resulting solution satisfies the QUICK scheme. Since the upwind scheme gives us coefficients that satisfy the Scarborough criterion, we are assured that the iterative solver will converge every outer iteration. Just as with non-linear problems, we have no guarantee that the outer iterations will themselves converge. It is sometimes necessary to use good initial guesses and under-relaxation strategies to obtain convergence.

5.7 Higher-Order Schemes for Unstructured Meshes

All the higher-order schemes we have seen so far assume line structure. In order to write ϕ_e, we must typically know the values ϕ_W, ϕ_P and ϕ_E; a similar stencil is required for ϕ_w, and involves the values ϕ_{WW}, ϕ_W, and ϕ_P in Figure 5.3. Thus, we can no longer write the face value purely in terms of cell centroid values on either side of the face. This presents a big problem for unstructured meshes since no such line structure exists. Higher-order schemes for unstructured meshes are an area of active research and new ideas continue to emerge . We present here a second-order accurate upwind scheme suitable for unstructured meshes.

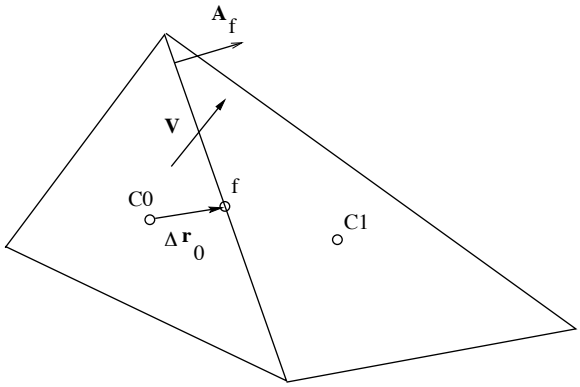


Figure 5.10: Schematic for Second-Order Scheme

Our starting point is the multi-dimensional equivalent of Equation 5.95. If $F_f > 0$, referring to Figure 5.10, we may write ϕ using a Taylor series expansion about the *upwind* cell centroid:

$$\phi(x,y) = \phi_0 + (\nabla\phi)_0 \cdot \Delta\mathbf{r} + O(|\Delta\mathbf{r}|^2) \tag{5.109}$$

Here, $\Delta\mathbf{r}$ is given by

$$\Delta\mathbf{r} = (x - x_0)\,\mathbf{i} + (y - y_0)\,\mathbf{j} \tag{5.110}$$

To find the face value ϕ_f, we evaluate Equation 5.109 at $\Delta\mathbf{r} = \Delta\mathbf{r}_0$, as shown in Figure 5.10

$$\phi_f = \phi_0 + (\nabla\phi)_0 \cdot \Delta\mathbf{r}_0 + O(|\Delta\mathbf{r}_0|^2) \tag{5.111}$$

As with structured meshes, the problem now turns to the evaluation of $(\nabla\phi)_0$. We have already seen in the previous chapter several methods for the calculation of the cell centered gradient. Any of these methods may be used to provide $(\nabla\phi)_0$.

5.8 Discussion

We have touched upon a number of different first and second-order schemes for discretizing the convection term. We have seen that all the schemes we have discussed here have drawbacks. The first-order schemes are diffusive, whereas the central difference and higher-order differencing schemes exhibit non-monotonicity to varying degrees. All the schemes we have seen employ *linear* coefficients, i.e., the discrete coefficients are independent of ϕ. Thus, for linear problems, we do not, in principle, require outer iterations unless a deferred correction strategy is adopted.

A number of researchers have sought to control the spatial oscillations inherent in higher-order schemes by *limiting* cell gradients so as to ensure monotonicity. These schemes typically employ non-linear coefficients which are adjusted to ensure adjacent

cell values are smoothly varying [4]. Though we do not address this class of discretization scheme here, they nevertheless represent an important arena of research.

5.9 Boundary Conditions

When dealing with the diffusion equation, we classified boundaries according to what information was specified. At Dirichlet boundaries, the value of ϕ itself was specified whereas at Neumann boundaries, the gradient of ϕ was specified. For convection problems we must further distinguish between *flow* boundaries where flow enters or leaves the computational domain, and *geometric* boundaries. Flow boundaries occur in a problem because we cannot include the entire universe in our computational domain and are forced to consider only a subset. We must then supply the appropriate information that represents the part of the universe that we are not considering but that is essential to solve the problem we are considering. For example, while analyzing the exhaust manifold of an automobile we might not include the combustion chamber and external airflow but then we must specify information about the temperature, velocity etc. of the flow as it leaves the combustion chamber and enters our computational domain. The geometric boundaries in such a problem would be the external walls of the manifold as well the surfaces of any components inside the manifold.

Flow boundaries may further be classified as *inflow* and *outflow* boundaries. We consider each in turn.

5.9.1 Inflow Boundaries

At inflow boundaries, we are given the inlet velocity distribution, as well the value of ϕ

$$\begin{aligned} \mathbf{V} &= \mathbf{V}_b; \quad \mathbf{V}_b \cdot \mathbf{A}_b \leq 0 \\ \phi &= \phi_{\text{given}} \end{aligned} \tag{5.112}$$

Consider the boundary cell shown in Fig 5.11. The dashed line shows the inflow boundary. The discrete equation for the cell is given by

$$\mathbf{J}_b \cdot \mathbf{A}_b + \sum_f \mathbf{J}_f \cdot \mathbf{A}_f = \left(S_C + S_P \phi_0\right) \Delta \mathscr{V}_0 \tag{5.113}$$

The summation in the second term is over the *interior* faces of the cell. We have already seen how to deal with the interior fluxes $\mathbf{J}_f$. The boundary flux may $\mathbf{J}_b$ is given by

$$\mathbf{J}_b \cdot \mathbf{A}_b = \rho \mathbf{V}_b \cdot \mathbf{A}_b \phi_b - \Gamma_b \left(\nabla \phi\right)_b \cdot \mathbf{A}_b \tag{5.114}$$

Using $\phi_b = \phi_{\text{given}}$, and writing the diffusion flux as in the previous chapter,

$$\mathbf{J}_b \cdot \mathbf{A}_b = \rho \mathbf{V}_b \cdot \mathbf{A}_b \phi_{\text{given}} - \frac{\Gamma_b}{\Delta \xi} \frac{\mathbf{A}_b \cdot \mathbf{A}_b}{\mathbf{A}_b \cdot \mathbf{e}_\xi} \left(\phi_0 - \phi_{\text{given}}\right) + \mathscr{S}_b \tag{5.115}$$

This boundary flux may be incorporated into the cell balance for the cell $C0$.

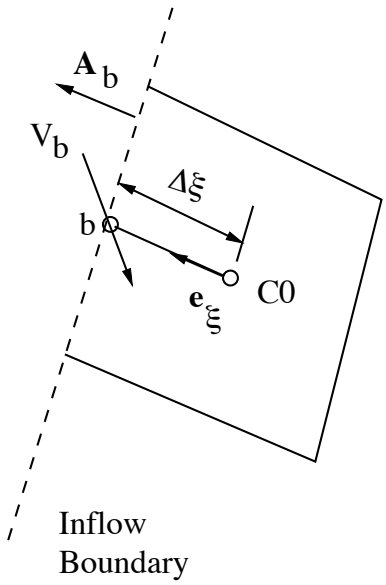


Figure 5.11: Cell Near Inflow Boundary

5.9.2 Outflow Boundaries

At outflow boundaries, we assume

$$-\Gamma_b (\nabla\phi)_b \cdot \mathbf{A}_b = 0 \tag{5.116}$$

That is, the diffusive component of the flux normal to the boundary is zero. Thus, the net boundary flux at the outflow boundary is

$$\mathbf{J}_b \cdot \mathbf{A}_b = \rho \mathbf{V}_b \cdot \mathbf{A}_b \phi_b; \quad \mathbf{V}_b \cdot \mathbf{A}_b > 0 \tag{5.117}$$

The cell balance for cell near an outflow boundary, such as that shown in Figure 5.12 is given by Equation 5.113. Using a first-order upwind scheme, we may write $\phi_b = \phi_0$, so that

$$\mathbf{J}_b \cdot \mathbf{A}_b = \rho \mathbf{V}_b \cdot \mathbf{A}_b \phi_0 \tag{5.118}$$

Thus, at outflow boundaries, we do not require the value of ϕ to be specified – it is determined by the physical processes in the domain, and convected to the boundary by the exiting flow. This result makes physical sense. For example, if we were solving for temperature in an exhaust manifold where the fluid was cooled because of conduction through the walls, we would certainly need to know the temperature of the fluid where it entered the domain but the temperature at the outlet would be determined as part of the solution and thus cannot be specified *a priori*.

There are two key implicit assumptions in writing Equation 5.118. The first is that convective flux is more important than diffusive flux. Indeed, we have assumed that the

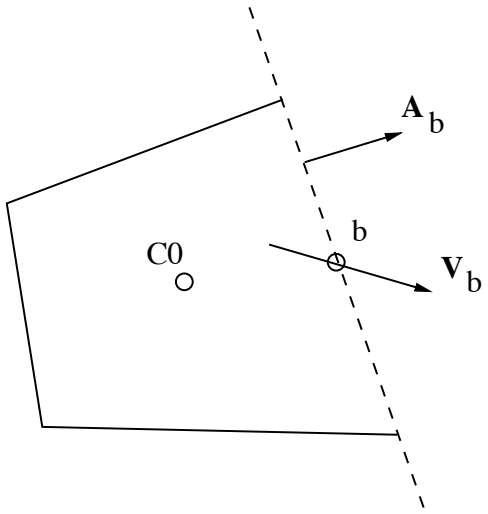


Figure 5.12: Cell Near Outflow Boundary

local grid Peclet number at the outflow face is infinitely large. For the exhaust manifold example, this means that the flow rate is high enough so that conditions downstream of our boundary do not affect the solution inside the domain. If the flow rate was not sufficiently high compared to the diffusion, then we would expect that a lower temperature downstream of the our domain would cause a lower temperature inside the domain as well. The second assumption is that the flow is directed out of the domain at all points on the boundary. Consider the flow past a backward-facing step, as shown in Figure 5.13. If we choose location A as the the outflow boundary, we cut across the recirculation bubble. In this situation, we would have to specify the ϕ values associated with the *incoming* portions of the flow for the problem to be well-posed. These are not usually available to us. Location B, located well past the recirculation zone, is a much better choice for an outflow boundary. It is very important to place flow boundaries at the appropriate location. Inflow boundaries should be placed at locations where we have sufficient data, either from another numerical simulation or from experimental observations. Outlet boundaries should be placed such that the conditions downstream have no influence on the solution.

5.9.3 Geometric Boundaries

At geometric boundaries, such as the external walls of an exhaust manifold, the normal component of the velocity is zero:

$$\mathbf{V}_b \cdot \mathbf{A}_b = 0 \tag{5.119}$$

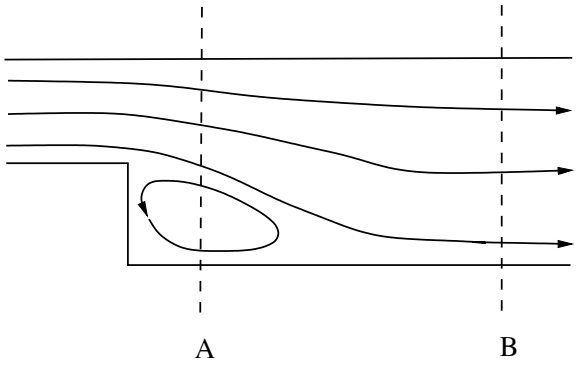


Figure 5.13: Recirculation At Domain Boundary

Consequently the boundary flux $\mathbf{J}_b$ is purely diffusive and given by

$$\mathbf{J}_b = -\Gamma_b (\nabla\phi)_b \tag{5.120}$$

At geometric boundaries, we are typically given either Dirichlet, Neumann or mixed boundary conditions. We have already shown how these may be discretized in a previous chapter.

5.10 Closure

In this chapter, we have addressed the discretization of the convection-diffusion equation. We have seen that the convection term requires the evaluation of ϕ at the faces of the cell for both structured and unstructured meshes. If the face value is interpolated using a central difference scheme, we have seen that our solution may have spatial oscillations for high Peclet numbers. The upwind scheme on the other hand, smears discontinuities, though the solution is bounded. We have also examined a class of upwind-weighted second-order and third–order schemes. All these higher-order schemes yield solutions which can have spatial oscillations. In developing these schemes for convection, we have assumed that the flow field is known. In the next chapter we shall turn to the task of computing the flow and pressure fields.

Chapter 6

Fluid Flow: A First Look

We have thus far considered convection and diffusion of a scalar in the presence of a *known* flow field. In this chapter, we shall examine the particular issues associated with the computation of the flow field. The momentum equations have the same form as the general scalar equation, and as such we know how to discretize them. The primary obstacle is the fact that the pressure field is unknown. The extra equation available for its determination is the continuity equation. The calculation of the flow field is complicated by the coupling between these two equations. We shall examine how to deal with this coupling, especially for incompressible flows. We shall also see how to develop formulations suitable for unstructured meshes.

6.1 Discretization of the Momentum Equation

Consider the two-dimensional rectangular domain shown in Figure 6.1. Let us assume for the moment that the velocity vector $\mathbf{V}$ and the pressure p are stored at the cell centroids. Let us, for simplicity, assume a Newtonian fluid though the issues raised here apply to other rheologies as well. Let us also assume steady state. The momentum equations in the x and y directions may be written as:

$$\nabla\cdot(\rho\mathbf{V}u) = \nabla\cdot(\mu\nabla u) - \nabla p\cdot\mathbf{i} + S_u \tag{6.1}$$

$$\nabla\cdot(\rho\mathbf{V}v) = \nabla\cdot(\mu\nabla v) - \nabla p\cdot\mathbf{j} + S_v \tag{6.2}$$

In the above equations, the stress tensor term has been split so that a portion of the normal stress appears in the diffusion term, and the rest is contained in S_u and S_v. The reader may wish to confirm that

$$S_u = f_u + \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial x}\right) - \frac{2}{3}\frac{\partial}{\partial x}(\mu\nabla\cdot\mathbf{V}) \tag{6.3}$$

and

$$S_v = f_v + \frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right) + \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial y}\right) - \frac{2}{3}\frac{\partial}{\partial y}(\mu\nabla\cdot\mathbf{V}) \tag{6.4}$$

Here, f_u and f_v contain the body force components in the x and y directions respectively. We see that Equations 6.1 and 6.2 have the same form as the general scalar transport equation, and as such we know how to discretize most of the terms in the equation. Each momentum equation contains a pressure gradient term, which we have written separately, as well as a source term (S_u or S_v) which contains the body force term, as well as remnants of the stress tensor term.

Let us consider the pressure gradient term. In deriving discrete equations, we integrate the governing equations over the cell volume. This results in the integration of the pressure gradient over the control volume. Applying the gradient theorem, we get

$$\int_{\Delta\mathcal{V}} \nabla p d\mathcal{V} = \int_A p d\mathbf{A} \tag{6.5}$$

Assuming that the pressure at the face centroid represents the mean value on the face, we write

$$\int_A p d\mathbf{A} = \sum_f p_f \mathbf{A}_f \tag{6.6}$$

The face area vectors are

$$\begin{aligned} \mathbf{A}_e &= \Delta y \mathbf{i} \\ \mathbf{A}_w &= -\Delta y \mathbf{i} \\ \mathbf{A}_n &= \Delta x \mathbf{j} \\ \mathbf{A}_s &= -\Delta x \mathbf{j} \end{aligned} \tag{6.7}$$

Thus, for the discrete u- momentum equation, the pressure gradient term is

$$-\mathbf{i} \cdot \int_{\Delta\mathcal{V}} \nabla p d\mathcal{V} = -\mathbf{i} \cdot \sum_f p_f \mathbf{A}_f \tag{6.8}$$

which in turn is given by

$$-\mathbf{i} \cdot \sum_f p_f \mathbf{A}_f = (p_w - p_e)\Delta y \tag{6.9}$$

Similarly, for the discrete v-momentum equation, the pressure gradient term is

$$-\mathbf{j} \cdot \sum_f p_f \mathbf{A}_f = (p_s - p_n)\Delta x \tag{6.10}$$

Completing the discretization, the discrete u- and v-momentum equations may be written as

$$\begin{aligned} a_P u_P &= \sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + (p_w - p_e)\Delta y + b_u \\ a_P v_P &= \sum_{\text{nb}} a_{\text{nb}} v_{\text{nb}} + (p_s - p_n)\Delta x + b_v \end{aligned} \tag{6.11}$$

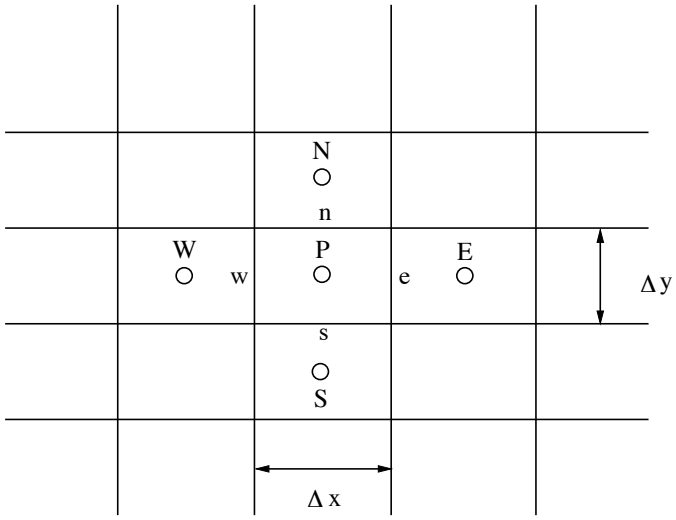


Figure 6.1: Orthogonal Mesh

The next step is to find the face pressures p_e, p_w, p_n and p_s. If we assume that the pressure varies linearly between cell centroids, we may write, for a uniform grid

$$\begin{aligned} p_e &= \frac{p_E + p_P}{2} \\ p_w &= \frac{p_W + p_P}{2} \\ p_n &= \frac{p_N + p_P}{2} \\ p_s &= \frac{p_S + p_P}{2} \end{aligned} \tag{6.12}$$

Therefore the pressure gradient terms in the momentum equations become

$$\begin{aligned} (p_w - p_e)\Delta y &= (p_W - p_E)\Delta y \\ (p_s - p_n)\Delta x &= (p_S - p_N)\Delta x \end{aligned} \tag{6.13}$$

Given a pressure field, we thus know how to discretize the momentum equations. However, the pressure field must be computed, and the extra equation we need for its computation is the continuity equation. Let us examine its discretization next.

6.2 Discretization of the Continuity Equation

For steady flow, the continuity equation, which takes the form

$$\nabla \cdot (\rho \mathbf{V}) = 0 \tag{6.14}$$

Integrating over the cell P and applying the divergence theorem, we get

$$\int_{\Delta\mathcal{V}} \nabla\cdot(\rho\mathbf{V})\, d\mathcal{V} = \int_A \rho\mathbf{V}\cdot d\mathbf{A} \tag{6.15}$$

Assuming that the $\rho\mathbf{V}$ on the face is represented by its face centroid value, we may write

$$\int_A \rho\mathbf{V}\cdot d\mathbf{A} = \sum_f (\rho\mathbf{V})_f \cdot \mathbf{A}_f \tag{6.16}$$

Using $\mathbf{V} = u\mathbf{i} + v\mathbf{j}$ and Equations 6.7, we may write the discrete continuity equation as

$$(\rho u)_e \Delta y - (\rho u)_w \Delta y + (\rho v)_n \Delta x - (\rho v)_s \Delta x = 0 \tag{6.17}$$

We do not have the face velocities available to us directly, and must interpolate the cell centroid values to the face. For a uniform grid, we may assume

$$\begin{aligned}
(\rho u)_e &= \frac{(\rho u)_P + (\rho u)_E}{2} \\
(\rho u)_w &= \frac{(\rho u)_W + (\rho u)_P}{2} \\
(\rho v)_n &= \frac{(\rho v)_P + (\rho v)_N}{2} \\
(\rho v)_s &= \frac{(\rho v)_S + (\rho v)_P}{2}
\end{aligned} \tag{6.18}$$

Gathering terms, the discrete continuity equation for the cell P is

$$(\rho u)_E \Delta y - (\rho u)_W \Delta y + (\rho v)_N \Delta x - (\rho v)_S \Delta x = 0 \tag{6.19}$$

We realize that continuity equation for cell P does not contain the velocity for cell P. Consequently, a checkerboard velocity pattern of the type shown in Figure 6.2 can be sustained by the continuity equation. If the momentum equations can sustain this pattern, the checkerboarding would persist in the final solution. Since the pressure gradient is not given *a priori*, and is computed as a part of the solution, it is possible to create pressure fields whose gradients exactly compensate the checkerboarding of momentum transport implied by the checkerboarded velocity field. Under these circumstances, the final pressure and velocity fields would exhibit checkerboarding.

We see also that the pressure gradient term in the u-momentum equation (Equation 6.13) involves pressures that are $2\Delta x$ apart on the mesh, and does not involve the pressure at the point P. The same is true for the v-momentum equation. This means that if a checkerboarded pressure field were imposed on the mesh during iteration, the momentum equations would not be able to distinguish it from a completely uniform pressure field. If the continuity equation were consistent with this pressure field as well, it would persist at convergence.

In practice, perfect checkerboarding is rarely encountered because of irregularities in the mesh, boundary conditions and physical properties. Instead, the tendency towards checkerboarding manifests itself in unphysical wiggles in the velocity and pressure fields. We should emphasize that these wiggles are a property of the spatial discretization and would be obtained regardless of the method used to solve the discrete equations.

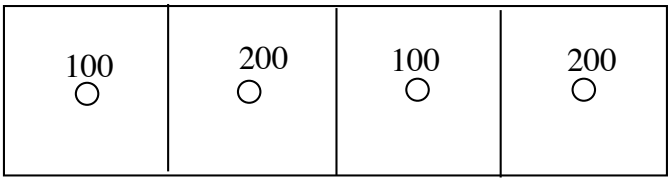


Figure 6.2: Checkerboard Velocity Field

6.3 The Staggered Grid

A popular remedy for checkerboarding is the use of a staggered mesh. A typical staggered mesh arrangement is shown in Figure 6.3. We distinguish between the *main* cell or control volume and the *staggered* cell or control volume. The pressure is stored at centroids of the *main* cells. The velocity components are stored on the faces of the main cells as shown, and are associated with the *staggered* cells. The u velocity is stored on the e and w faces and the v velocity is stored on the n and s faces. Scalars such as enthalpy or species mass fraction are stored at the centroids of the cell P as in previous chapters. All properties, such as density and Γ, are stored at the main grid points.

The cell P is used to discretize the continuity equation as before:

$$(\rho u)_e \Delta y - (\rho u)_w \Delta y + (\rho v)_n \Delta x - (\rho v)_s \Delta x = 0 \tag{6.20}$$

However, no further interpolation of velocity is necessary since discrete velocities are available directly where required. Thus the possibility of velocity checkerboarding is eliminated.

For the momentum equations, the staggered control volumes are used to write momentum balances. The procedure is the same as above, except that the pressure gradient term may be written directly in terms of the pressures on the faces of the momentum control volumes directly, without interpolating as in Equation 6.12. Thus for the discrete momentum equation for the velocity u_e, the pressure gradient term is

$$(p_P - p_E)\Delta y \tag{6.21}$$

Similarly, for the velocity v_n, the pressure gradient term is

$$\left(p_P - p_N\right)\Delta y \tag{6.22}$$

Thus, we no longer have a dependency on pressure values that are $2\Delta x$ apart.

We note that the mesh for the u-momentum equation consists of non-overlapping cells which fill the domain completely. This is also true for the v-momentum equation and the continuity equation. The control volumes for u and v overlap each other and the cell P, but this is of no consequence.

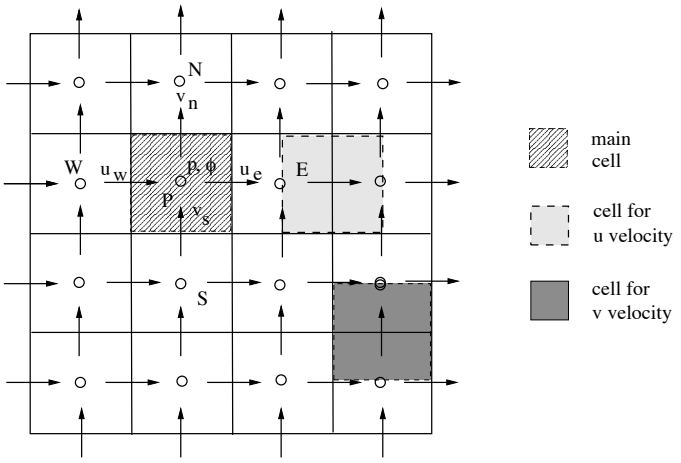


Figure 6.3: Staggered Mesh

We note further that the mass flow rates

$$\begin{aligned} F_e &= (\rho u)_e \Delta y \\ F_w &= (\rho u)_w \Delta y \\ F_n &= (\rho v)_n \Delta x \\ F_s &= (\rho v)_s \Delta x \end{aligned} \tag{6.23}$$

are available at the main cell faces, where they are needed for the discretization of the convective terms in scalar transport equations.

6.4 Discussion

The staggered mesh provides an ingenious remedy for the checkerboarding problem by locating discrete pressures and velocities exactly where required. At this point, our discretization of the continuity and momentum equations is essentially complete. The u_e momentum equation may be written as

$$a_e u_e = \sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + \Delta y (p_P - p_E) + b_e \tag{6.24}$$

Similarly the equation for v_n may be written as:

$$a_n v_n = \sum_{\text{nb}} a_{\text{nb}} v_{\text{nb}} + \Delta x \left(p_P - p_N\right) + b_n \tag{6.25}$$

Here, *nb* refers to the momentum control volume neighbors. For the *e* momentum equation, the neighbors *nb* would involve the *u* velocities at points *ee*, *nne w* and *sse* shown in Figure 6.4. A similar stencil influences v_n. The discrete continuity equation

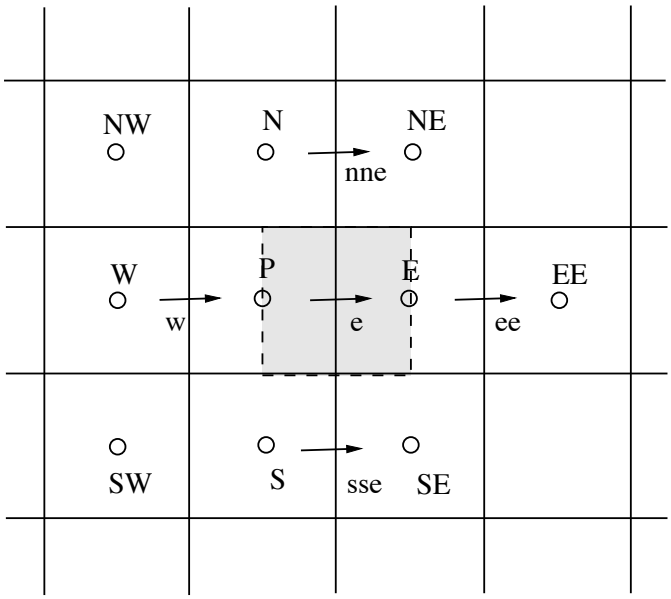


Figure 6.4: Neighbors for u_e Momentum Control Volume

is given by Equation 6.20. (Note that because of grid staggering, the coefficients a_p and a_{nb} are different for the u and v equations). It is not yet clear how this equation set is to be solved to obtain u, v and p. We turn to this matter next.

6.5 Solution Methods

Thus far, we have examined issues related to *discretization* of the continuity and momentum equations. The discretization affects the accuracy of the final answer we obtain. We now turn to issues related to the solution of these equations. The solution path determines whether we obtain a solution and how much computer time and memory we require to obtain the solution. For the purposes of this book, the final solution we obtain is considered independent of the path used to obtain it, and only dependent on the discretization. (This is not true in general for non-linear problems, where the solution path may determine which of several possible solutions is captured).

Thus far, our solution philosophy has been to so solve our discrete equations iteratively. Though we have not emphasized this, when solving multiple differential equations, a convenient way is to solve them sequentially. That is, when computing the transport of N_s chemical species, for example, one option is to employ a solution loop of the type:

1. for species i =1, N_s
    - Discretize governing equation for species i

- Solve for mass fractions of species i at cell centroids, assuming prevailing values to resolve non-linearities and dependences on the mass fractions of species j

2. If all species mass fractions have converged, stop; else go to step 1.

Other alternatives are possible. If we are not worried about storage or computational time, we may wish to solve the entire problem directly using a linear system of the type

$$\mathbf{M}\phi = \mathbf{b} \tag{6.26}$$

where $\mathbf{M}$ is a coefficient matrix of size $(N \times N_s) \times (N \times N_s)$, where N is the number of cells and N_s is the number of species, ϕ is a column vector of size $N \times N_s$, and $\mathbf{b}$ is a column vector also of size $N \times N_s$. That is, our intent is to solve for the entire set of $N \times N_s$ species mass fractions in the calculation domain simultaneously. For most practical applications, this type of simultaneous solution is still not affordable, especially for non-linear problems where the $\mathbf{M}$ matrix (or a related matrix) would have to be recomputed every iteration.

For practical CFD problems, sequential iterative solution procedures are frequently adopted because of low storage requirements and reasonable convergence rate. However there is a difficulty associated with the sequential solution of the continuity and momentum equations for incompressible flows. In order to solve a set of discrete equations iteratively, it is necessary to associate the discrete set with a particular variable. For example, we use the discrete energy equation to solve for the temperature. Similarly, we intend to use the discrete u-momentum equation to solve for the u-velocity. If we intend to use the continuity equation to solve for pressure, we encounter a problem for incompressible flows because the pressure does not appear in the continuity equation directly. The density does appear in the continuity equation, but for incompressible flows, the density is unrelated to the pressure and cannot be used instead. Thus, if we want to use sequential, iterative methods, it is necessary to find a way to introduce the pressure into the continuity equation. Methods which use pressure as the solution variable are called *pressure-based* methods. They are very popular in the incompressible flow community.

There are a number of methods in the literature [5] which use the density as a primary variable rather than pressure. This practice is especially popular in the compressible flow community. For compressible flows, pressure and density are related through an equation of state. It is possible to find the density using the continuity equation, and to deduce the pressure from it for use in the momentum equations. Such methods are called *density-based* methods. For incompressible flows, a class of methods called the *artificial compressibility* methods have been developed which seek to ascribe a small (but finite) compressibility to incompressible flows in order to facilitate numerical solution through density-based methods [6]. Conversely, pressure-based methods have also been developed which may be used for compressible flows [7].

It is important to realize that the necessity for pressure- and density-based schemes is directly tied to our decision to solve our governing equations sequentially and iteratively. It is this choice that forces us to associate each governing differential equation with a solution variable. If we were to use a direct method, and solve for the discrete

velocities and pressure using a domain-wide matrix of size $3N \times 3N$, (three variables - u,v,p - over N cells), no such association is necessary. However, even with the power of today's computers, direct solutions of this type are still out of reach for most problems of practical interest. Other methods, including local direct solutions at each cell, coupled to an iterative sweep, have been proposed [8] but are not pursued here.

In this chapter, we shall concentrate on developing pressure-based methods suitable for incompressible flows, but which can be extended to compressible flows as well. These methods seek to create an equation for pressure by using the *discrete* momentum equations. They then solve for the continuity and momentum equations sequentially, with each discrete equation set being solved using iterative methods. We emphasize here that these methods define the *path to solution* and not the discretization technique.

6.6 The SIMPLE Algorithm

The SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm and its variants are a set of pressure-based methods widely used in the incompressible flow community [9]. The primary idea behind SIMPLE is to create a discrete equation for pressure (or alternatively, a related quantity called the pressure correction) from the discrete continuity equation (Equation 6.20). Since the continuity equation contains discrete face velocities, we need some way to relate these discrete velocities to the discrete pressure field. The SIMPLE algorithm uses the discrete momentum equations to derive this connection.

Let u^* and v^* be the discrete u and v fields resulting from a solution of the discrete u and v momentum equations. Let p^* represent the discrete pressure field which is used in the solution of the momentum equations. Thus, u_e^* and v_n^* satisfy

$$\begin{aligned} a_e u_e^* &= \sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}}^* + \Delta y \left(p_P^* - p_E^*\right) + b_e \\ a_n v_n^* &= \sum_{\text{nb}} a_{\text{nb}} v_{\text{nb}}^* + \Delta x \left(p_S^* - p_P^*\right) + b_n \end{aligned} \tag{6.27}$$

Similar expressions may be written for u_w^* and v_s^*. If the pressure field p^* is only a guess or a prevailing iterate, the discrete u^* and v^* obtained by solving the momentum equations will not, in general, satisfy the discrete continuity equation (Equation 6.20). We propose a correction to the starred velocity field such that the corrected values satisfy Equation 6.20:

$$\begin{aligned} u &= u^* + u' \\ v &= v^* + v' \end{aligned} \tag{6.28}$$

Correspondingly, we wish to correct the existing pressure field p^* with

$$p = p^* + p' \tag{6.29}$$

If we subtract Equations 6.27 from Equations 6.24 and 6.25 we obtain

$$\begin{aligned} a_e u_e' &= \sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}}' + \Delta y \left(p_P' - p_E'\right) \\ a_n v' &= \sum_{\text{nb}} a_{\text{nb}} v_{\text{nb}}' + \Delta x \left(p_S' - p_P'\right) \end{aligned} \tag{6.30}$$

Similar expressions may be written for u'_w and v'_s. Equations 6.30 represent the dependence of the velocity corrections u' and v' on the pressure corrections p'. In effect, they tell us how the velocity field will respond when the pressure gradient is increased or decreased.

We now make an important simplification. We approximate Equations 6.30 as

$$\begin{aligned} a_e u'_e &\approx \Delta y \left(p'_P - p'_E\right) \\ a_n v'_n &\approx \Delta x \left(p'_S - p'_P\right) \end{aligned} \tag{6.31}$$

or, defining

$$\begin{aligned} d_e &= \frac{\Delta y}{a_e} \\ d_n &= \frac{\Delta x}{a_n} \end{aligned} \tag{6.32}$$

we write Equations 6.31 as

$$\begin{aligned} u'_e &= d_e \left(p'_P - p'_E\right) \\ v'_n &= d_n \left(p'_P - p'_S\right) \end{aligned} \tag{6.33}$$

so that

$$\begin{aligned} u_e &= u^*_e + d_e \left(p'_P - p'_E\right) \\ v_n &= v^*_n + d_n \left(p'_P - p'_S\right) \end{aligned} \tag{6.34}$$

Further, using Equations 6.23 we may write the face flow rates obtained after the solution of the momentum equations as

$$\begin{aligned} F^*_e &= \rho_e u^*_e \Delta y \\ F^*_n &= \rho_n v^*_n \Delta x \end{aligned} \tag{6.35}$$

The corrected face flow rates are given by

$$\begin{aligned} F_e &= F^*_e + F'_e \\ F_n &= F^*_n + F'_n \end{aligned} \tag{6.36}$$

with

$$\begin{aligned} F'_e &= \rho_e d_e \Delta y \left(p'_P - p'_E\right) \\ F'_n &= \rho_n d_n \Delta x \left(p'_P - p'_S\right) \end{aligned} \tag{6.37}$$

Similar expressions may be written for F_w, F'_w, F_s and F'_s.

Thus far, we have derived expressions describing how the face flow rates vary if the pressure difference across the face is changed. We now turn to the task of creating an equation for pressure from the discrete continuity equation.

6.6.1 The Pressure Correction Equation

We now consider the discrete continuity equation. The starred velocities u^* and v^*, obtained by solving the momentum equations using the prevailing pressure field p^* do not satisfy the discrete continuity equation. Thus

$$(\rho u^*)_e \Delta y - (\rho u^*)_w \Delta y + (\rho v^*)_n \Delta x - (\rho v^*)_s \Delta x \neq 0 \tag{6.38}$$

or, in terms of F, we have

$$F_e^* - F_w^* + F_n^* - F_s^* \neq 0 \tag{6.39}$$

We require our corrected velocities, given by Equations 6.28, to satisfy continuity. Alternately, the corrected face flow rates, given by Equation 6.36, must satisfy continuity. Thus,

$$F_e^* + F_e' - F_w^* - F_w' + F_n^* + F_n' - F_s^* - F_s' = 0 \tag{6.40}$$

or using Equations 6.37

$$F_e^* + \rho_e d_e \Delta y \left(p_P' - p_E'\right) - F_w^* - \rho_w d_w \Delta y \left(p_W' - p_P'\right) \tag{6.41}$$

$$+F_n^* + \rho_n d_n \Delta x \left(p_P' - p_N'\right) - F_s^* - \rho_s d_s \Delta x \left(p_S' - p_P'\right) = 0 \tag{6.42}$$

Rearranging terms, we may write an equation for the pressure correction p'_P as:

$$a_P p_P' = \sum_{\text{nb}} p_{\text{nb}}^{'} + b$$

where

$$\begin{aligned} a_E &= \rho_e d_e \Delta y \\ a_W &= \rho_w d_w \Delta y \\ a_N &= \rho_n d_n \Delta x \\ a_S &= \rho_s d_s \Delta x \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} \\ b &= F_w^* - F_e^* + F_s^* - F_n^* \end{aligned} \tag{6.44}$$

We note that the source term in the pressure correction equation is the *mass source* for the cell P. If the face flow rates F^* satisfy the discrete continuity equation (i,e, b is zero), we see that $p' =$ constant satisfies Equation 6.44. Thus, the pressure correction equation yields non-constant corrections only as long as the velocity fields produced by the momentum equations do not satisfy continuity. Once these velocity fields satisfy the discrete continuity equations, the pressure correction equation will yield a constant correction. In this limit, differences of p' are zero and no velocity corrections are obtained. If the constant correction value is chosen to be zero (we will see why this is possible in a later section), the pressure will not be corrected. Thus, convergence is obtained once the velocities predicted by the momentum equations satisfy the continuity equation.

6.6.2 Overall Algorithm

The overall procedure for the SIMPLE algorithm is the following:

1. Guess the pressure field p^*.
2. Discretize and solve the momentum equations using the guessed value p^* for the pressure source terms. This yields the u^* and v^* fields.
3. Find the mass flow rates F^* using the starred velocity fields. Hence find the pressure correction source term b.
4. Discretize and solve the pressure correction equation, and obtain the p' field.
5. Correct the pressure field using Equation 6.29 and the velocities using Equation 6.28. The corrected velocity field satisfies the discrete continuity equation exactly.
6. Solve the discrete equations for scalar ϕ if desired, using the continuity-satisfying velocity field for the convection terms.
7. If the solution is converged, stop. Else go to step 2.

6.6.3 Discussion

The pressure correction equation is a vehicle by which the velocity and pressure fields are nudged towards a solution that satisfies both the discrete continuity and momentum equations. If we start with an arbitrary guess of pressure and solve the momentum equations, there is no guarantee that the resulting velocity field will satisfy the continuity equation. Indeed the b term in the continuity equation is a measure of the resulting mass imbalance. The pressure correction equation corrects the pressure and velocity fields to ensure that the resulting field annihilates this mass imbalance. Thus, once we solve the pressure correction equation and correct the u^* and v^* fields using Equations 6.28, the corrected velocity fields will satisfy the discrete continuity equations *exactly*. It will no longer satisfy the discrete momentum equations, and the iteration between the two equations continues until the pressure and velocity fields satisfy both equations.

It is important to realize that because we are solving for the pressure correction rather that the pressure itself, the omission of the $\sum_{nb} a_{nb} u'_{nb}$ and $\sum_{nb} a_{nb} v'_{nb}$ terms in deriving the pressure correction equation is of no consequence as far as the final answers are concerned. This is easily seen if we consider what happens in the final iteration. In the final iteration, u^*, v^* and p^* satisfy the momentum equations, and the b term in the p' equation is zero. As discussed earlier, the p' equation does not generate any corrections, and $u = u^*$, $v = v^*$, $p = p^*$ holds true. Thus, the p' equation plays no role in the final iteration. Only discrete momentum equation, and the discrete continuity equation (embodied in the b term) determine the final answer.

The dropping of the $\sum_{nb} a_{nb} u'_{nb}$ and $\sum_{nb} a_{nb} v'_{nb}$ terms does have consequences for the *rate* of convergence, however. The u-velocity correction in Equation 6.30, for example, is a function of both the velocity correction term $\sum_{nb} a_{nb} u'_{nb}$ and the pressure

correction term. If we drop the $\sum_{nb} a_{nb} u'_{nb}$ term, we place the entire burden of correcting the u-velocity upon the pressure correction. The resulting corrected velocity will satisfy the continuity equation all the same, but the resulting pressure is over-corrected. Indeed, because of this over-correction of pressure, the SIMPLE algorithm is prone to divergence unless underrelaxation is used. We underrelax the momentum equations using underrelaxation factors α_u and α_v in the manner outlined in previous chapters. In addition, the pressure correction is not applied in its entirety as in Equation 6.29. Instead, only a part of the pressure correction is applied:

$$p = p^* + \alpha_p p' \tag{6.45}$$

The underrelaxation factor α_p is chosen to be less than one in order to correct for the over-correction of pressure. It is important to emphasize that we *do not* underrelax the velocity corrections in the manner of Equation 6.45. The entire point of the pressure correction procedure is to create velocity corrections such that the corrected velocity fields satisfy continuity. Underrelaxing the velocity corrections would destroy this feature.

Thus, the SIMPLE algorithm approaches convergence through a set of intermediate continuity-satisfying fields. The computation of transported scalars such as enthalpy or species mass fraction is therefore done soon after the velocity correction step (Step 6). This ensures that the face flow rates used in discretizing the ϕ transport equation are *exactly* continuity satisfying every single iteration.

6.6.4 Boundary Conditions

We have already dealt with the boundary conditions for scalar transport in the previous chapter. These apply to the momentum equations as well. We turn now to boundary conditions for pressure. Two common boundary conditions are considered here: given normal velocity and given static pressure. A third condition, given stagnation pressure and flow angle, is also encountered, but we will not address it here.

At a given-velocity boundary, we are given the normal component of the velocity vector $\mathbf{V}_b$ at the boundary. This type of boundary could involve inflow or outflow boundaries or boundaries normal to which there is no flow, such a walls.

Consider the near-boundary cell shown in Figure 6.5. Out objective is to derive the pressure correction equation for C0. Integrating the continuity equation for the cell C0 in the usual fashion, we have

$$F_e - F_b + F_n - F_s = 0 \tag{6.46}$$

We know how to write the interior face flow rates F_e, F_n and F_s in terms of the the pressure corrections. For F_b, no such expansion in terms of pressure correction is necessary because F_b is known, and is given by

$$F_b = \rho_b u_b \Delta y$$

This known flow rate is incorporated directly into the mass balance equation for cell C0. When all boundaries are given-velocity boundaries, we must ensure that the specified

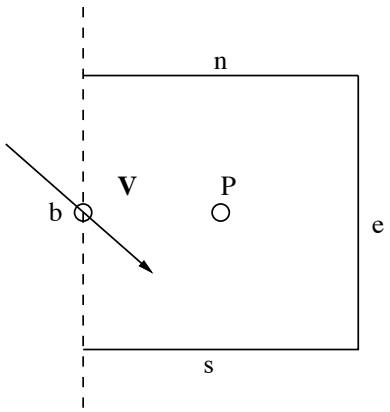


Figure 6.5: Near-Boundary Cell for Pressure

boundary velocities satisfy overall mass balance over the entire computational domain; otherwise the problem would not be well-posed.

At a given-pressure boundary, the pressure correction p'_b is set equal to zero.

6.6.5 Pressure Level and Incompressibility

For incompressible flows,where density is not a function of pressure, it is common to encounter situations which are best modeled with given-velocity boundary conditions on all boundaries. In such a case, the level of pressure in the domain is not set. Differences in pressure are unique, but the individual pressure values themselves are not. The reader may verify that p and $p+C$ are solutions to the governing differential equations.

From a computational viewpoint we may interpret this situation in the following way. We are given velocity boundary conditions that are continuity-satisfying in an overall sense. Thus, if we divide the computational domain into N cells, and impose a mass balance on them, only $N-1$ unique equations can result. Therefore we do not have enough equations for N pressure (or pressure correction) unknowns. This situation may be remedied by setting the pressure at one cell centroid arbitrarily; alternatively, we may set $p'=0$ in one cell in the domain.

We should emphasize this situation only occurs if all boundaries are given-velocity boundaries. When the static pressure p is given on a boundary, the pressure is made unique, and the problem does not arise. For compressible flows, where the density is a function of pressure, it is necessary to specify pressure boundary conditions on at least part of the domain boundary.

Let us go back to the pressure correction equation, Equation 6.44 and its behavior at convergence. We have said that the pressure correction becomes a constant at convergence. If all boundaries are given-velocity boundaries, the pressure correction has an arbitrary level, which we are free to set equal to zero. Thus, the pressure p^* sees no corrections at convergence.(Even if we did not set p' to zero, the result is still converged; the pressure level would rise by a constant every iteration, but this is acceptable since

only differences of pressure are relevant with all given-velocity boundaries). If one or more given-pressure boundaries are present, $p' = 0$ is set at at least one boundary face. Thus the constant value predicted by the pressure correction will come out to be zero. In this case also, the pressure correction predicts zero corrections at convergence.

6.7 The SIMPLER Algorithm

The SIMPLE algorithm has been widely used in the literature. Nevertheless, there have been a number of attempts to accelerate its convergence, and one such algorithm is SIMPLER (SIMPLE-Revised) [10]. One of the drawbacks of the SIMPLE algorithm is the approximate nature of the pressure correction equation. Because the $\sum_{\text{nb}} a_{\text{nb}} u'_{\text{nb}}$ and $\sum_{\text{nb}} a_{\text{nb}} v'_{\text{nb}}$ terms are dropped in its derivation, the pressure corrections resulting from it are too large, and require under-relaxation. This slows down convergence, since optimal values are problem dependent and rarely know *a priori*. The velocity corrections, however, are good, and guarantee that the corrected velocities satisfy the continuity equation. Consequently, it would seem appropriate to use the pressure correction equation to correct velocities, while finding another way to compute the pressure.

A good way of understanding this is to consider what the SIMPLE algorithm does when we know the velocity field, but do not know the pressure field. If we solve the momentum equations with a guessed pressure field p^*, we destroy the original (good) velocity field, and then embark on a long iterative process to recover it. A good guess of the velocity field is no use when using the SIMPLE algorithm, unless accompanied by a good pressure guess. We would prefer an algorithm which can recover the correct pressure field immediately if the exact velocity field is known.

With SIMPLER, we derive the pressure equation by re-arranging the momentum equations as follows:

$$\begin{aligned} u_e &= \frac{\sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + b_e}{a_e} + d_e \left(p_P - p_E\right) \\ v_n &= \frac{\sum_{\text{nb}} a_{\text{nb}} v_{\text{nb}} + b_n}{a_n} + d_n \left(p_P - p_N\right) \end{aligned} \tag{6.48}$$

By defining

$$\begin{aligned} \hat{u}_e &= \frac{\sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + b_e}{a_e^u} \\ \hat{v}_n &= \frac{\sum_{\text{nb}} a_{\text{nb}} v_{\text{nb}} + b^n}{a_e} \end{aligned} \tag{6.49}$$

we may write

$$\begin{aligned} u_e &= \hat{u}_e + d_e \left(p_P - p_E\right) \\ v_n &= \hat{v}_n + d_n \left(p_P - p_N\right) \end{aligned} \tag{6.50}$$

Furthermore, we may define

$$\begin{aligned} \hat{F}_e &= \rho_e \hat{u}_e \Delta y \\ \hat{F}_n &= \rho_n \hat{v}_n \Delta x \end{aligned} \tag{6.51}$$

so that

$$\begin{aligned} F_e &= \hat{F}_e + \rho_e d_e \Delta y \left(p_P - p_E\right) \\ F_n &= \hat{F}_n + \rho_n d_n \Delta x \left(p_P - p_N\right) \end{aligned} \tag{6.52}$$

Substituting Equations 6.52 into the discrete continuity equation (Equation 6.20) we obtain the following equation for the pressure:

$$a_P p_P = \sum_{\text{nb}} a_{\text{nb}} p_{\text{nb}} + b$$

where

$$\begin{aligned} a_E &= \rho_e d_e \Delta y \\ a_W &= \rho_w d_w \Delta y \\ a_N &= \rho_n d_n \Delta x \\ a_S &= \rho_s d_s \Delta x \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} \\ b &= \hat{F}_w - \hat{F}_e + \hat{F}_s - \hat{F}_n \end{aligned} \tag{6.54}$$

The form of the pressure equation is identical to that of the pressure correction equation, and the a_P and a_{nb} coefficients are identical to those governing the pressure correction. The b term, however, is different, and involves the velocities $\hat{u}$ and $\hat{v}$ rather than the prevailing velocities u^* and v^*. We should emphasize that although the b term looks similar in form to that in the p' equation, it does not represent the mass imbalance. Another important difference is that no approximations have been made in deriving the pressure equation. Thus, if the velocity field is exact, the correct pressure field will be recovered.

6.7.1 Overall Algorithm

The SIMPLER solution loop takes the following form:

1. Guess the velocity field.
2. Compute $\hat{u}$ and $\hat{v}$.
3. Solve the pressure equation (Equation 6.54) using the guessed field and obtain the pressure.
4. Solve the momentum equations using the pressure field just computed, to obtain u^*, and v^*.
5. Compute the mass source term b in the pressure correction equation.
6. Solve the pressure correction equation to obtain p'.
7. Correct u^* and v^* using Equations 6.28. *Do not correct the pressure !*

8. At this point we have a continuity satisfying velocity field. Solve for any scalar ϕ's of interest.

9. Check for convergence. If converged, stop. Else go to 2.

6.7.2 Discussion

The SIMPLER algorithm has been shown to perform better than SIMPLE. This is primarily because the SIMPLER algorithm does not require a good pressure guess (which is difficult to provide in any case). It generates the pressure field from a good guess of the velocity field, which is easier to guess. Thus, the SIMPLER algorithm does not have the tendency to destroy a good velocity field guess like the SIMPLE algorithm.

The SIMPLER algorithm solves for two pressure variables - the actual pressure and the pressure correction. Thus, it uses one extra equation, and therefore involves more computational effort. The pressure correction solver in the SIMPLE loop typically accounts for half the computational effort during an iteration. This is because the pressure correction equation is frequently solved with given-velocity boundary conditions. The lack of Dirichlet boundary conditions for p' causes linear solvers to converge slowly. The same is true of pressure. Thus, adding an extra pressure equation in the SIMPLER algorithm increases the computational effort by about 50%. The momentum equation coefficients are needed in two places – to find $\hat{u}$ and $\hat{v}$ for the pressure equation, and later, to solve the momentum equations. To avoid computing them twice, storage for each of the coefficient sets is required. This is also true for the pressure coefficients.

Because the pressure correction p' is not used to correct the pressure, no underrelaxation of the pressure correction in the manner of Equation 6.45 is required. The pressure equation may itself be underrelaxed if desired, but this is not usually required. The momentum equations must be underrelaxed to account for non-linearities and also to account for the sequential nature of the solution procedure.

6.8 The SIMPLEC Algorithm

The SIMPLE-Corrected (SIMPLEC) algorithm [11] attempts to cure the primary failing of the SIMPLE algorithm, i.e., the neglect of the $\sum_{nb} a_{nb} u'_{nb}$ and $\sum_{nb} a_{nb} v'_{nb}$ terms in writing Equations 6.33. Instead of ignoring these completely, the SIMPLEC algorithm attempts to approximate the neighbor corrections by using the cell correction as:

$$\begin{aligned} \sum_{nb} a_{nb} u'_{nb} &\approx u'_e \sum_{nb} a_{nb} \\ \sum_{nb} a_{nb} v'_{nb} &\approx v'_e \sum_{nb} a_{nb} \end{aligned} \tag{6.55}$$

Thus, the velocity corrections take the form

$$\begin{aligned}\left(a_e - \sum_{\text{nb}} a_{\text{nb}}\right) u'_e &= \Delta y \left(p'_P - p'_E\right) \\ \left(a_n - \sum_{\text{nb}} a_{\text{nb}}\right) v'_n &= \Delta x \left(p'_P - p'_N\right)\end{aligned} \tag{6.56}$$

Redefining d_e and d_n as

$$\begin{aligned}d_e &= \frac{\Delta y}{\left(a_e - \sum_{\text{nb}} a_{\text{nb}}\right)} \\ d_n &= \frac{\Delta x}{\left(a_n - \sum_{\text{nb}} a_{\text{nb}}\right)}\end{aligned} \tag{6.57}$$

The rest of the procedure is the same as that for the SIMPLE algorithm except for the fact that the pressure correction need not be underrelaxed as in Equation 6.45. However, we should note that the d_e and d_n definitions require the momentum equations to be underrelaxed to prevent the denominator from going to zero. The SIMPLEC algorithm has been shown to converge faster than the SIMPLE and does not have the computational overhead of the SIMPLER algorithm. However, it does share with SIMPLE the property that a good velocity field guess would be destroyed in the initial iterations unless accompanied by a good guess of the pressure field.

6.9 Optimal Underrelaxation for SIMPLE

It is possible to make SIMPLE duplicate the speed-up exhibited by SIMPLEC through the judicious choice of underrelaxation factors. The SIMPLE procedure employs a pressure correction of the type

$$p = p^* + \alpha_p p' \tag{6.58}$$

whereas the SIMPLEC algorithm employs a pressure correction of the type

$$p = p^* + p' \tag{6.59}$$

Rather than solve for p', let us make the SIMPLE algorithm solve for a variable $\hat{p}'$ defined as

$$\hat{p}' = \alpha_p p' \tag{6.60}$$

Then its correction equation takes the form of Equation 6.59. We may think of SIMPLEC as solving for $\hat{p}'$ rather than p'.

Now let us examine the p' equation (or $\hat{p}'$ equation) solved by SIMPLEC. The equation has the form

$$a_P \hat{p}'_P = \sum_{\text{nb}} a_{\text{nb}} \hat{p}'_{\text{nb}} + b \tag{6.61}$$

with the coefficients a_{nb} having the form

$$a_{\text{nb}} = \frac{\rho \Delta y^2}{a_e - \sum_{\text{nb}} a_{\text{nb}}} \tag{6.62}$$

Let us assume for simplicity that the momentum equation coefficients have no S_P terms, so that

$$a_e = \frac{\sum_{\text{nb}} a_{\text{nb}}}{\alpha_u} \tag{6.63}$$

Thus, the pressure correction coefficients in Equation 6.62 may be written as

$$a_{\text{nb}} = \frac{\rho \Delta y^2}{\frac{1-\alpha_u}{\alpha_u} \sum_{\text{nb}} a_{\text{nb}}} \tag{6.64}$$

Now we turn to the SIMPLE algorithm. If we cast its p' equation in $\hat{p}'$ form, we get

$$a_P \hat{p}'_P = \sum_{\text{nb}} a_{\text{nb}} \hat{p}'_{\text{nb}} + b \tag{6.65}$$

with the coefficients a_{nb} having the form

$$a_{\text{nb}} = \frac{\rho \Delta y^2}{\frac{\alpha_p}{\alpha_u} \sum_{\text{nb}} a_{\text{nb}}} \tag{6.66}$$

If we require the a_{nb} coefficient in Equation 6.64 to be equal to that in Equation 6.66, we conclude that

$$\alpha_p = 1 - \alpha_u \tag{6.67}$$

Thus, we see that if we use Equation 6.67 in underrelaxing the momentum equations and the pressure correction, we would essentially reproduce the iterations computed by SIMPLEC.

6.10 Discussion

We have seen three different possibilities for a sequential solution of the continuity and momentum equations using a pressure-based scheme. A number of other variants and improvements of the basic SIMPLE procedure are also available in the literature. These are, however, variations on the basic theme of sequential solution, and the same basic advantages and disadvantages obtain. All these procedures have the advantage of low storage, and reasonably good performance over a broad range of problems. However, they are known to require a large number of iterations in problems with large body forces resulting from buoyancy, swirl and other agents; in strongly non-linear cases, divergence may occur despite underrelaxation. Many of these difficulties are a result of the sequential nature of the momentum and continuity solutions, which are strongly coupled through the pressure gradient term when strong body forces are present. For such problems, the user would not find much difference in the performance of the different SIMPLE variants. A variety of coupled solvers have been developed which seek

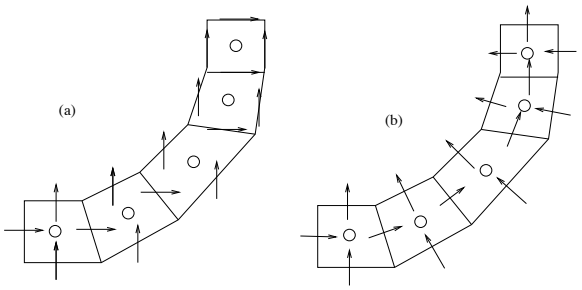


Figure 6.6: Velocity Systems for Structured Bodyfitted Meshes: (a) Cartesian, and (b) Grid-Following

to replace the sequential solution of the momentum and continuity equations through tighter coupling of the two equations (see [8] for example). These procedures usually incur a storage penalty but exhibit substantial convergence acceleration, at least for laminar flow problems. This area continues to be active area for new research, especially in connection with unstructured meshes.

6.11 Non-Orthogonal Structured Meshes

When structured non-orthogonal meshes are used, the staggered mesh procedures described above may be used in principle. However, great care must be taken in the choice of velocity components. For example, it is not possible to use Cartesian velocity components with a staggered mesh, as illustrated in Figure 6.6(a). Here we consider a 90° elbow. If we store the u and v velocities on the faces as shown, similar to our practice on regular meshes, it is possible to encounter cells where the stored face velocity component is tangential to the face, making it difficult to discretize the continuity equation correctly. If staggered meshes are used, it is necessary to use *grid following* velocities, i.e., velocities whose orientation is defined with respect to the local face. In Figure 6.6(b), for example, we use velocity components normal to the face in question. These velocity components are guaranteed to never become tangential to the face because they turn as the mesh turns. Any velocity set with a fixed (non-zero) angle to the local face would do as well.(Other options, such as storing *both* components of the Cartesian velocities at all faces, have been tried; though formulations of this type can be worked out, the result is not very elegant; for example, overlapping momentum control volumes result).

Two primary grid-following or curvilinear velocity systems have been used in the literature for this purpose. These are the *covariant* velocity and the *contravariant* velocity systems. Consider the cells P and E is Figure 6.7(a). The vectors $\mathbf{e}_\xi$ and $\mathbf{e}_\eta$ are called the covariant basis vectors. The $\mathbf{e}_\xi$ vector is aligned along the line joining the cell centroids. The $\mathbf{e}_\eta$ vector is aligned tangential to the face. The velocity components along these basis directions are called the covariant velocity components. On each face, the component along the line joining the centroids is stored. Thus on Karki

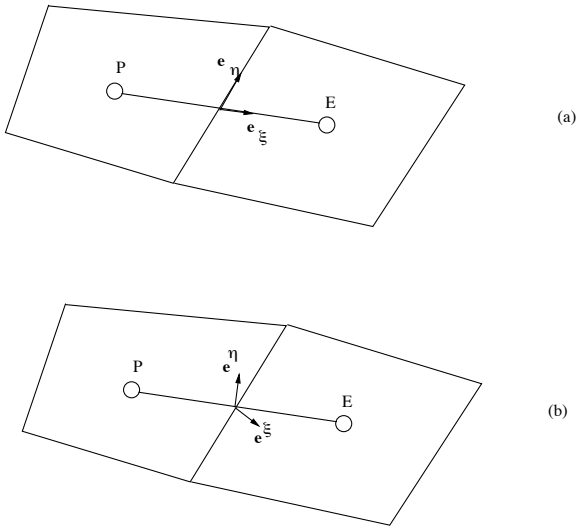


Figure 6.7: Curvilinear Velocity Components (a) Covariant, and (b) Contravariant

and Patankar [7] have use a covariant velocity formulation to develop a staggered mesh SIMPLE algorithm for body-fitted structured meshes.

Alternatively, *contravariant* basis vectors may be used, as shown in Figure 6.7(b). Here, the basis vector $\mathbf{e}^{\xi}$ is perpendicular to the face, and the basis vector $\mathbf{e}^{\eta}$ is perpendicular to the line joining the cell centroids. The contravariant velocities are aligned along these basis directions. Each face stores one contravariant component, the component perpendicular to the face.

The SIMPLE family of algorithms may be developed for a staggered mesh discretization using either of these two coordinate systems. Though these efforts have generally been successful, curvilinear velocities are not particularly easy entities to deal with. Since Newton's laws of motion conserve *linear* momentum, the momentum equations written in Cartesian coordinates may always be cast in conservative form. Curvilinear coordinates do not have this property. Since covariant and contravariant basis vectors change direction with respect to an invariant Cartesian system, the momentum equations written in these curvilinear directions cannot be written in conservative form; momentum in the curvilinear directions is not conserved. As a result, additional curvature terms appear, just as they do in cylindrical-polar or spherical coordinates. Thus we are not guaranteed conservation. (Researchers have proposed clever cures for this problem; see Karki and Patankar [7], for example). Furthermore, velocity gradients are required in other equations, for example for production terms in turbulence models, or for strain rates in non-Newtonian rheologies. These quantities are extremely cumbersome to derive in curvilinear coordinates. To overcome this, researchers have computed the flow field in curvilinear coordinates and stored both Cartesian and curvilinear velocities. They use Cartesian velocity components for all such manipulations. Despite these workarounds, curvilinear velocities remain cumbersome and are difficult

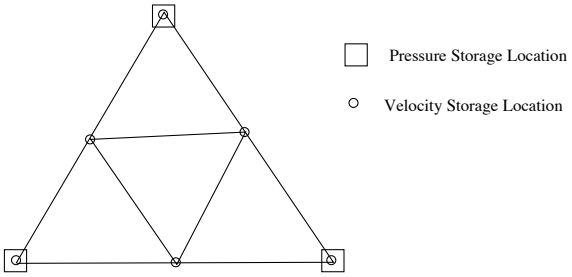


Figure 6.8: Storage Arrangement for Node-Based Unequal Order Finite Volume Scheme

to interpret and visualize in complex domains.

6.12 Unstructured Meshes

For unstructured meshes, the staggered mesh discretization method we have derived so far is almost entirely useless since no obvious mesh staggering is possible. In the finite element community, a class of *unequal order* methods have been developed which interpolate pressure to lower order than velocity. This has been shown to circumvent checker-boarding. Node-based finite-volume schemes [12] have been developed which employ the same unequal-order idea. In the work by Baliga and Patankar, for example, triangular macro-elements are employed, as shown in Figure 6.8. Pressure is stored at the nodes of the macroelement. The macroelement is subdivided into 4 sub-elements, and velocity is stored on the vertices of the subelements. Though this arrangement has been shown to prevent checkerboarding, pressure is resolved to only one-fourth the mesh size as the velocity in two dimensions, prompting concerns about accuracy. For cell based schemes there is no obvious counterpart of this unequal order arrangement.

6.13 Closure

In this chapter, we have developed staggered mesh based discretization techniques for the continuity and momentum equations. Staggering was shown to be necessary to prevent checkerboarding in the velocity and pressure fields. We then developed sequential and iterative pressure-based techniques called SIMPLE, SIMPLER and SIMPLEC for solving this set of discrete equations.

We see that the staggered discretization developed in this chapter is not easily extended to body-fitted and unstructured meshes. The use of staggered mesh methods based on curvilinear velocities is cumbersome and uninviting for structured meshes. For unstructured meshes, it is not easy to identify a workable staggered mesh, even if we could use curvilinear velocity components. Because of these difficulties, efforts have been underway to do away with staggering altogether and to develop techniques known variously as *non-staggered* , *equal-order*, or *co-located* methods. These tech-

niques store pressure and velocity at the same physical location and attempt to solve the problem of checkerboarding through clever interpolation techniques. They also do away with the necessity for curvilinear velocity formulations, and use Cartesian velocities in the development. We will address this class of techniques in the next chapter.

We should emphasize that the material to be presented in the next chapter changes only the discretization practice. We may still use sequential and iterative techniques such as SIMPLE to solve the resulting set of discrete equations.

Chapter 7

Fluid Flow: A Closer Look

In this chapter, we turn to the problem of discretizing the continuity and momentum equations using a non-staggered or co-located mesh. We saw in the last chapter that storing pressure and velocity at the same location, i.e., at the cell centroids, leads to checkerboarding in the pressure and velocity fields. We circumvented this in Cartesian meshes by using staggered storage of pressure and velocity. We also used Cartesian velocity components as our primary variables.

For unstructured meshes, it is not obvious how to define staggered pressure and velocity control volumes. Furthermore, staggered meshes are somewhat cumbersome to use. For Cartesian meshes, staggering requires the storage of geometry information for the main and staggered u and v control volumes, as well increased coding complexity. For body-fitted meshes, we saw in the previous chapter that staggered meshes could only be used if grid-following velocities were used; we saw that this option is also not entirely optimal. As a result, recent research has focused on developing formulations which employ Cartesian velocity components, storing both pressure and velocity at the cell centroid. Specialized interpolation schemes are used to prevent checkerboarding.

The change to a co-located or non-staggered storage scheme is a change in the discretization practice. The iterative methods used to solve the resulting discrete set are the same as those in the previous chapter, albeit with a few minor changes to account for the change in storage scheme. For the purposes of this chapter, we will continue to use the SIMPLE family of algorithms for the solution of the discrete equations.

7.1 Velocity and Pressure Checkerboarding

Co-located or non-staggered methods store pressure and velocity at the cell centroid. Furthermore, we use Cartesian velocity components u and v, defined in a global coordinate system. Thus, in Figure 7.1, u, v and p are stored at the cell centroid P, as are other scalars ϕ. The principle of conservation is enforced on the cell P.

In the discussion that follows, let us assume an orthogonal one-dimensional uniform mesh. We will address two-dimensional and non-orthogonal meshes in a later section, once the basic idea is clear. The mesh and associated nomenclature are shown

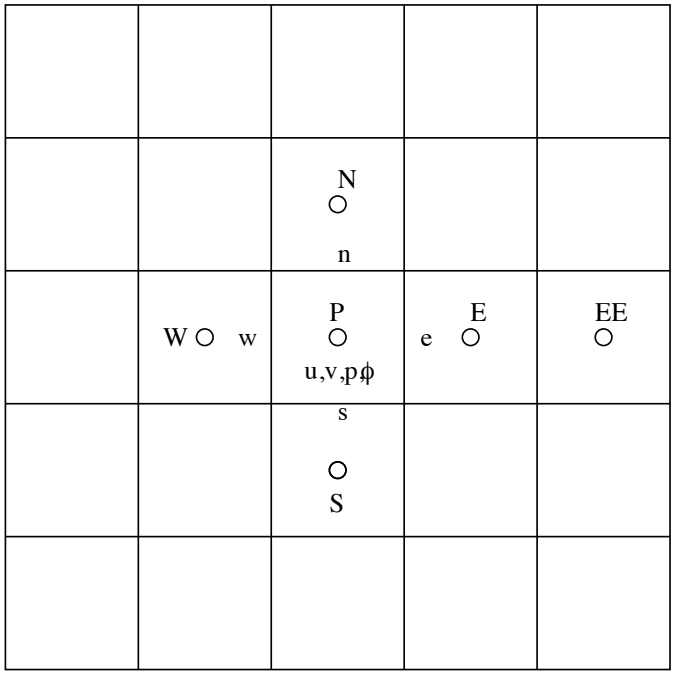


Figure 7.1: Co-Located Storage of Pressure and Velocity

in Figure 7.2.

7.1.1 Discretization of Momentum Equation

The discretization of the u-momentum equation for the cell P follows the principles outlined in previous chapters and yields the following discrete equation:

$$a_P u_P = \sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + b_P^u + (p_w - p_e) \tag{7.1}$$

Here, a unit area of cross section $\Delta y = 1$ has been assumed. The neighbors nb for this co-located arrangement include the velocities at the points E and W. The pressure at the faces e and w are not known since the pressure is stored at the cell centroids.

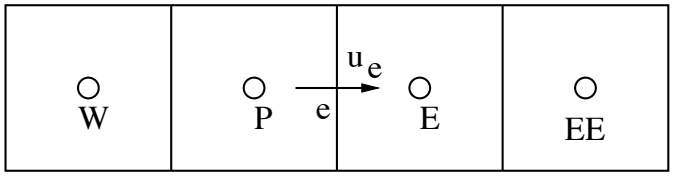


Figure 7.2: Control Volumes for Velocity Interpolation

Consequently, interpolation is required. Adopting a linear interpolation, and a uniform mesh, we may write

$$a_P u_P = \sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + b_P^u + \frac{(p_W - p_E)}{2} \tag{7.2}$$

Similarly, for u_E, we may write

$$a_E u_E = \sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + b_E^u + \frac{(p_P - p_{EE})}{2} \tag{7.3}$$

We see that the u-momentum equation at point P does not involve the pressure at the point P; it only involves pressures at cells on either side. The same is true for u_E. Thus, the u-momentum equation can support a checkerboarded solution for pressure. If we retain this type of discretization for the pressure term in the momentum equation, we must make sure that the discretization of the continuity equation somehow disallows pressure checkerboarding.

7.1.2 Discretization of Continuity Equation

In order to discretize the continuity equation, we integrate it over the cell P as before and apply the divergence theorem. This yields

$$F_e - F_w = 0 \tag{7.4}$$

where

$$\begin{aligned} F_e &= \rho_e u_e \Delta y \\ F_w &= \rho_w u_w \Delta y \end{aligned} \tag{7.5}$$

As we discussed in the previous chapter, we must interpolate u from the cell centroid values to the face in order to find u_e and u_w. If we use a linear interpolation

$$\begin{aligned} u_e &= \frac{u_P + u_E}{2} \\ u_w &= \frac{u_W + u_P}{2} \end{aligned} \tag{7.6}$$

Substituting these relations into Equation 7.4 and assuming unit Δy yields

$$\rho_e u_E - \rho_w u_W = 0 \tag{7.7}$$

7.1.3 Pressure Checkerboarding

Let us now consider the question of whether Equation 7.7 can support a checkerboarded pressure field. Dividing Equations 7.2 and 7.3 by their respective center coefficients a_P and a_E, we have

$$\begin{aligned} u_P &= \hat{u}_P + \frac{1}{a_P} \frac{(p_W - p_E)}{2} \\ u_E &= \hat{u}_E + \frac{1}{a_E} \frac{(p_P - p_{EE})}{2} \end{aligned} \tag{7.8}$$

where

$$\begin{aligned} \hat{u}_P &= \frac{\sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + b_P^u}{a_P} \\ \hat{u}_E &= \frac{\sum_{\text{nb}} a_{\text{nb}} u_{\text{nb}} + b_E^u}{a_E} \end{aligned} \tag{7.9}$$

If we average u_P and u_E using Equation 7.8 to find u_e in Equation 7.7 we obtain

$$u_e = \frac{\hat{u}_P + \hat{u}_E}{2} + \frac{1}{a_P}\left(\frac{p_W - p_E}{4}\right) + \frac{1}{a_E}\left(\frac{p_P - p_{EE}}{4}\right) \tag{7.10}$$

A similar equation can be written for u_w:

$$u_w = \frac{\hat{u}_W + \hat{u}_P}{2} + \frac{1}{a_W}\left(\frac{p_{WW} - p_P}{4}\right) + \frac{1}{a_P}\left(\frac{p_W - p_E}{4}\right) \tag{7.11}$$

We see right away that any checkerboarded pressure field which sets $p_W = p_E$ and $p_P = p_{EE} = p_{WW}$ will be seen as a uniform pressure field by u_e and u_w. These face velocities are used to write the one-dimensional discrete continuity equation (Equation 7.7). Since the same type of checkerboarding is supported by the discrete momentum equations, a checkerboarded pressure field can persist in the final solution if boundary conditions permit. Our discrete continuity equation does nothing to filter spurious oscillatory modes in the pressure field supported by the momentum equation.

7.1.4 Velocity Checkerboarding

In addition to pressure checkerboarding, the linear interpolation of cell velocities also introduces checkerboarding. As we have seen in the previous chapter, the resulting discrete continuity equation, Equation 7.7, does not involve the cell-centered velocities u_P. Thus, the continuity equation supports a checkerboarded velocity field. Such a checkerboarded velocity field implies checkerboarded momenta in the cell momentum balance. If we enforce momentum balance on the cell, we will in effect create a pressure field to offset these checkerboarded momenta; this pressure field must of necessity be checkerboarded. In order to prevent checkerboarding in the final solution, we must ensure that the discretization of either the momentum or the continuity equation provides a filter to remove these oscillatory modes.

7.2 Co-Located Formulation

Co-located formulations prevent checkerboarding by devising interpolation procedures which express the face velocities u_e and u_w in terms of *adjacent* pressure values rather than *alternate* pressure values. Furthermore the face velocity u_e is not defined purely as a linear interpolant of the two adjacent cell values; an additional term, called the *added dissipation* prevents velocity checkerboarding.

As we saw in the previous section, the discrete one-dimensional momentum equations for cells P and E yield

$$\begin{aligned} u_P &= \hat{u}_P + d_P \frac{(p_W - p_E)}{2} \\ u_E &= \hat{u}_E + d_E \frac{(p_P - p_{EE})}{2} \end{aligned} \tag{7.12}$$

where

$$\begin{aligned} d_P &= \frac{1}{a_P} \\ d_E &= \frac{1}{a_E} \end{aligned} \tag{7.13}$$

Writing the continuity equation for cell P we have

$$F_e - F_w = 0 \tag{7.14}$$

or equivalently

$$\rho_e u_e - \rho_w u_w = 0 \tag{7.15}$$

As before, unit cross-sectional area is assumed. If we interpolate u_e linearly, we get

$$u_e = \frac{u_P + u_E}{2} = \frac{\hat{u}_P + \hat{u}_E}{2} + d_P \left(\frac{p_W - p_E}{4} \right) + d_E \left(\frac{p_P - p_{EE}}{4} \right) \tag{7.16}$$

Instead of interpolating linearly, we use

$$\begin{aligned} u_e &= \frac{u_P + u_E}{2} - d_P \left(\frac{p_W - p_E}{4} \right) - d_E \left(\frac{p_P - p_{EE}}{4} \right) + d_e (p_P - p_E) \\ &= \frac{\hat{u}_P + \hat{u}_E}{2} + d_e (p_P - p_E) \end{aligned} \tag{7.17}$$

where

$$d_e = \frac{d_P + d_E}{2} \tag{7.18}$$

A similar expression may be written for u_w.

It is important to understand the manipulation that has been done in obtaining Equation 7.17. We have removed the pressure gradient term resulting from linear interpolation of velocities (which involves the pressures p_W, p_E, p_P and p_{EE}) and added in a new pressure gradient term written in terms of the pressure difference $(p_P - p_E)$. Another way of looking at this is to say that in writing u_e, we interpolate the $\hat{u}$ component linearly between P and E, but write the pressure gradient term directly in terms of the *adjacent* cell-centroid pressures p_P an p_E.

This type of interpolation is sometimes referred to as *momentum interpolation* in the literature. It is also sometimes referred to as an *added dissipation* scheme. It was proposed, with small variations, by different researchers in the early 1980's [13, 14, 15]

for use with pressure-based solvers. Ideas similar to it have also been used in the compressible flow community with density-based solvers. Momentum interpolation prevents checkerboarding of the velocity field by not interpolating the velocities linearly. The face velocities u_e and u_w are used to write the discrete continuity equation for cell P. Since they are written in terms of *adjacent* pressures rather that *alternate* ones, a continuity-satisfying velocity field would not be able to ignore a checkerboarded pressure field. Thus, even though the momentum equation contains a pressure gradient term that can support a checkerboarded pattern, the continuity equation does not permit such a pressure field to persist.

Another useful way to think about momentum interpolation is to consider the face velocity u_e to be a sort of staggered velocity. The momentum interpolation formula, Equation 7.17, may be interpreted as a momentum equation for the staggered velocity u_e. Instead of deriving the staggered momentum equation from first principles, the momentum interpolation procedure derives it by interpolating $\hat{u}$ linearly, and adding the pressure gradient appropriate for the staggered cell. (Recall that the quantity $\hat{u}$ contains the convection, diffusion and source contributions of the momentum equation.) By not using an actual staggered-cell discretization, momentum interpolation avoids the creation of staggered cell geometry and makes it possible to use the idea for unstructured meshes.

7.3 The Concept of Added Dissipation

It is useful to understand why momentum interpolation is also referred to as the added dissipation scheme. For simplicity, let us assume that $d_P = d_E = d_e$. This would be the case if the mesh were uniform, and the flow field, diffusion coefficients and source terms were constant for the cells P and E. Let us consider the first of the expressions in Equation 7.17:

$$u_e = \frac{u_P + u_E}{2} - d_P \left(\frac{p_W - p_E}{4} + \frac{p_P - p_{EE}}{4} \right) + d_P (p_P - p_E)$$

Let us look at the pressure terms

$$-d_P \left(\frac{p_W - p_E}{4} + \frac{p_P - p_{EE}}{4} \right) + d_P (p_P - p_E)$$

Rearranging, we have

$$\begin{aligned} & -d_P \left(\frac{p_W - p_E}{4} + \frac{p_P - p_{EE}}{4} - (p_P - p_E) \right) \\ = \quad & -\frac{d_P}{4} \left((p_W + p_E - 2p_P) - (p_P + p_{EE} - 2p_E) \right) \end{aligned} \tag{7.21}$$

Using a Taylor series expansion, we can show that

$$\left(\frac{\partial^2 p}{\partial x^2} \right)_P = \frac{(p_W + p_E - 2p_P)}{\Delta x^2} + O\left(\Delta x^2\right) \tag{7.22}$$

Similarly

$$\left(\frac{\partial^2 p}{\partial x^2}\right)_E = \frac{(p_P + p_{EE} - 2p_E)}{\Delta x^2} + O\left(\Delta x^2\right) \tag{7.23}$$

The pressure term in the momentum interpolation scheme may thus be written as

$$u_e = \frac{u_P + u_E}{2} - \frac{d_P}{4}\left(\left(\frac{\partial^2 p}{\partial x^2}\right)_P - \left(\frac{\partial^2 p}{\partial x^2}\right)_E\right)\Delta x^2 \tag{7.24}$$

or, dividing and multiplying the pressure term by Δx, we may write

$$u_e = \frac{u_P + u_E}{2} - \frac{d_P}{4}\left(\frac{\partial^3 p}{\partial x^3}\right)_e \Delta x^3 \tag{7.25}$$

A similar expression may be written for u_w:

$$u_w = \frac{u_W + u_P}{2} - \frac{d_P}{4}\left(\frac{\partial^3 p}{\partial x^3}\right)_w \Delta x^3 \tag{7.26}$$

Now, let us look at the continuity equation. If we write the continuity equation for constant ρ, and dividing through by Δx we get

$$\frac{u_e - u_w}{\Delta x} = 0 \tag{7.27}$$

Substituting for u_e and u_w from Equations 7.25 and 7.26 we get

$$\frac{u_E - u_W}{2\Delta x} - \frac{d_P}{4}\left(\frac{\partial^4 p}{\partial x^4}\right)_P \Delta x^3 \tag{7.28}$$

Using a Taylor series expansion, we may show that

$$\frac{u_E - u_W}{2\Delta x} = \left(\frac{\partial u}{\partial x}\right)_P + O(\Delta x^2) \tag{7.29}$$

so that Equation 7.28 may be written as

$$\left(\frac{\partial u}{\partial x}\right)_P - \frac{d_P}{4}\left(\frac{\partial^4 p}{\partial x^4}\right)_P \Delta x^3 = 0 \tag{7.30}$$

We see that momentum interpolation is equivalent to solving a continuity equation with an added fourth derivative of pressure. Even derivatives are frequently referred to in the literature as *dissipation*; hence the name *added dissipation* scheme.

7.4 Accuracy of Added Dissipation Scheme

Let us now examine the accuracy of the momentum interpolation or the added dissipation scheme. The interpolation scheme may be written using Equation 7.21

$$u_e = \frac{u_P + u_E}{2} - \frac{d_P}{4}\left(\left(p_W + p_E - 2p_P\right) - \left(p_P + p_{EE} - 2p_E\right)\right) \tag{7.31}$$

Let us consider the first term $(u_P + u_E)/2$. Using a Taylor series expansion about e, we may write

$$\begin{aligned} u_P &= u_e - \left(\frac{\partial u}{\partial x}\right)_e \frac{\Delta x}{2} + \left(\frac{\partial^2 u}{\partial x^2}\right)_e \frac{\Delta x^2}{8} + O(\Delta x^3) \\ u_E &= u_e + \left(\frac{\partial u}{\partial x}\right)_e \frac{\Delta x}{2} + \left(\frac{\partial^2 u}{\partial x^2}\right)_e \frac{\Delta x^2}{8} + O(\Delta x^3) \end{aligned} \tag{7.32}$$

Adding the two equations and dividing by two yields

$$u_e = \frac{u_P + u_E}{2} + O(\Delta x^2) \tag{7.33}$$

Thus the truncation error in writing the first term in Equation 7.31 is $O(\Delta x^2)$. We already saw from Equation 7.22 that the pressure term may be written as

$$-\frac{d_P}{4}\left(\left(p_W + p_E - 2p_P\right) - \left(p_P + p_{EE} - 2P_E\right)\right) = -\frac{d_P}{4}\left(\frac{\partial^3 p}{\partial x^3}\right)_e \Delta x^3 \tag{7.34}$$

so that the total expression for u_e is

$$u_e = \frac{u_P + u_E}{2} - \frac{d_P}{4}\left(\frac{\partial^3 p}{\partial x^3}\right)_e \Delta x^3 + O(\Delta x^2) \tag{7.35}$$

We see that the pressure term we have added is $O(\Delta x^3)$, which is of higher order than the second-order truncation error of the linear interpolation. Consequently, the added dissipation term does not change the formal second-order accuracy of the underlying scheme.

7.5 Discussion

Thus far we have been looking at how to interpolate the face velocity in order to circumvent the checkerboarding problem for co-located arrangements. We have seen that momentum interpolation is equivalent to solving the continuity equation with an extra dissipation term for pressure. Adding this dissipation term does not change the formal accuracy of our discretization scheme since the term added has a dependence $O(\Delta x^3)$ whereas the other terms have a truncation error of $O(\Delta x^2)$. Our intent is to write the continuity equation using this interpolation for the face velocity. The discretization of the momentum equations retains the form of Equation 7.1.

An extremely important point to be made is that the discrete continuity equation (Equation 7.14) is written in terms of the face velocities u_e and u_w. It is not written directly in terms of the cell-centered velocities u_P and u_E. Thus, at convergence, it is the *face* velocities that directly satisfy the discrete continuity equation, not the cell-centered velocities. In a co-located formulation, the cell-centered velocities directly satisfy the discrete momentum equations. They satisfy the continuity equations only indirectly

through their role in defining u_e and u_w. Conversely, in a co-located formulation, the face velocities u_e and u_w directly satisfy the continuity equation, but do not satisfy any discrete momentum equation since no direct momentum conservation over a staggered cell is ever written for them. They satisfy momentum conservation only indirectly, in the sense that they satisfy the momentum interpolation formula.

To complete the development, let us look at regular two-dimensional meshes and derive the equivalent forms for the face velocity interpolation. Though the properties of the momentum interpolation scheme are less clearly evident in 2D, everything we have said about the one-dimensional case is also true in two dimensions. We will use this development as a stepping stone to developing a SIMPLE algorithm for solving the discrete set of equations.

7.6 Two-Dimensional Co-Located Variable Formulation

Let us generalize our development to two-dimensional regular meshes before considering how to solve our discrete set of equations. We consider the cell P in Figure 7.1.

7.6.1 Discretization of Momentum Equations

Using the procedures described earlier, we may derive the discrete u- and v-momentum equations for the velocities u_P and v_P:

$$\begin{aligned} a_P^u u_P &= \sum_{\text{nb}} a_{\text{nb}}^u u_{\text{nb}} + b_P^u + \Delta y \frac{(p_W - p_E)}{2} \\ a_P^v v_P &= \sum_{\text{nb}} a_{\text{nb}}^v v_{\text{nb}} + b_P^v + \Delta x \frac{(p_S - p_N)}{2} \end{aligned} \tag{7.36}$$

Here, the coefficients a_P^u and a_{nb}^u are the coefficients of the u-momentum equation for cell P. Similarly, a_P^v and a_{nb}^v are the coefficients of the v-momentum equation. We note that the pressure gradient terms involve pressures $2\Delta x$ and $2\Delta y$ apart respectively. Similar discrete equations may be written for the neighboring cells.

7.6.2 Momentum Interpolation

Consider the face e in Figure 7.1, and the u momentum equations for the cells P and E. Dividing the discrete momentum equations by the center coefficients we obtain:

$$\begin{aligned} u_P &= \hat{u}_P + d_P^u \frac{(p_W - p_E)}{2} \\ u_E &= \hat{u}_E + d_E^u \frac{(p_P - p_{EE})}{2} \end{aligned} \tag{7.37}$$

where

$$\begin{aligned} d_P^u &= \frac{\Delta y}{a_P^u} \\ d_E^u &= \frac{\Delta y}{a_E^u} \end{aligned} \tag{7.38}$$

The factor Δy appears because the pressure gradient term is multiplied by it in a two-dimensional mesh. By analogy we may define equivalent quantities at other faces:

$$\begin{aligned} d_W^u &= \frac{\Delta y}{a_W^u} \\ d_N^v &= \frac{\Delta x}{a_N^v} \\ d_S^v &= \frac{\Delta x}{a_S^v} \end{aligned} \tag{7.39}$$

Using momentum interpolation for the face velocity u_e, we obtain

$$u_e = \hat{u}_e + d_e \left(p_P - p_E\right) \tag{7.40}$$

where

$$\begin{aligned} \hat{u}_e &= \frac{\hat{u}_P + \hat{u}_E}{2} \\ d_e &= \frac{d_P^u + d_E^u}{2} \end{aligned} \tag{7.41}$$

By analogy, we may write face velocities for the other faces as:

$$\begin{aligned} u_w &= \hat{u}_w + d_w \left(p_W - p_P\right) \\ v_n &= \hat{v}_n + d_n \left(p_P - p_N\right) \\ v_s &= \hat{v}_s + d_s \left(p_S - p_P\right) \end{aligned} \tag{7.42}$$

with

$$\begin{aligned} d_w &= \frac{d_W^u + d_P^u}{2} \\ d_n &= \frac{d_P^v + d_N^v}{2} \\ d_s &= \frac{d_S^v + d_P^v}{2} \end{aligned} \tag{7.43}$$

7.7 SIMPLE Algorithm for Co-Located Variables

Having defined the face velocities as momentum-interpolants of cell centered values, we now turn to the question of how to write the discrete continuity equation, and how

to solve the discrete set. In keeping with our philosophy of using sequential iterative solutions, we wish to use the SIMPLE algorithm. We must now devise a way to formulate a pressure correction equation that can be used with the co-located variables. The procedure is similar to that adopted in the previous chapter, albeit with a few small changes.

7.7.1 Velocity and Pressure Corrections

As before, let u^* and v^* denote the solution to the discrete momentum equations using a guessed pressure field p^*. The face velocities u_e, u_w, v_n and v_s found by interpolating the u^* and v^* to the face using momentum interpolation are not guaranteed to satisfy the discrete continuity equation. Thus,

$$F_e^* - F_w^* + F_n^* - F_s^* \neq 0 \tag{7.44}$$

The face mass flow rates F are defined in terms of the momentum-interpolated face velocities as

$$\begin{aligned} F_e^* &= \rho_e u_e^* \Delta y \\ F_n^* &= \rho_n v_n^* \Delta x \end{aligned} \tag{7.45}$$

Similar expressions may be written for F_w^* and F_s^*. We wish to correct the face velocities (and the face flow rates) such that the corrected flow rates satisfy the discrete continuity equation. Thus, we propose *face velocity* corrections

$$\begin{aligned} u_e &= u_e^* + u_e' \\ v_n &= v_n^* + v_n' \end{aligned} \tag{7.46}$$

and the *cell* pressure corrections

$$p_P = p_P^* + p_P' \tag{7.47}$$

Correspondingly, we may write face flow rate corrections

$$\begin{aligned} F_e &= F_e^* + F_e' \\ F_n &= F_n^* + F_n' \end{aligned} \tag{7.48}$$

where

$$\begin{aligned} F_e' &= \rho_e \Delta y u_e' \\ F_n' &= \rho_n \Delta x v_n' \end{aligned} \tag{7.49}$$

We now seek to express u' and v' in terms of the cell pressure corrections p', and to use these expressions to derive a pressure correction equation. From our face velocity definitions, we write

$$\begin{aligned} u_e' &= \hat{u}_e' + d_e \left(p_P' - p_E'\right) \\ u_w' &= \hat{u}_w' + d_w \left(p_W' - p_P'\right) \\ v_n' &= \hat{v}_n' + d_n \left(p_P' - p_N'\right) \\ v_s' &= \hat{v}_s' + d_s \left(p_S' - p_P'\right) \end{aligned} \tag{7.50}$$

In keeping with the SIMPLE algorithm, we approximate Equations 7.50 as

$$\begin{aligned} u'_e &= d_e\left(p'_P - p'_E\right) \\ u'_w &= d_w\left(p'_W - p'_P\right) \\ v'_n &= d_n\left(p'_P - p'_N\right) \\ v'_s &= d_s\left(p'_S - p'_P\right) \end{aligned} \tag{7.51}$$

by dropping the $\hat{u}'$ and $\hat{v}'$ terms. This is analogous to dropping the $\sum_{\text{nb}} a_{\text{nb}} u'_{\text{nb}}$ terms in the staggered formulation.

The corresponding face flow rate corrections are

$$\begin{aligned} F'_e &= \rho_e \Delta y d_e\left(p'_P - p'_E\right) \\ F'_w &= \rho_w \Delta y d_w\left(p'_W - p'_P\right) \\ F'_n &= \rho_n \Delta x d_n\left(p'_P - p'_N\right) \\ F'_s &= \rho_s \Delta x d_s\left(p'_S - p'_P\right) \end{aligned} \tag{7.52}$$

We notice that the velocity corrections in Equations 7.51 correct the *face* velocities, but not the *cell-centered* velocities. For later use, we write the cell-centered velocity corrections by analogy:

$$\begin{aligned} u'_P &= d_P^u \frac{\left(p'_W - p'_E\right)}{2} \\ v'_P &= d_P^v \frac{\left(p'_S - p'_N\right)}{2} \end{aligned} \tag{7.53}$$

7.7.2 Pressure Correction Equation

To derive the pressure correction equation for the co-located formulation, we write the continuity equation in terms of the corrected face flow rates F as before:

$$F_e^* + F'_e - F_w^* - F'_w + F_n^* + F'_n - F_s^* - F'_s = 0 \tag{7.54}$$

Substituting from Equations 7.51 and 7.49 for the F' values, we obtain the pressure correction equation:

$$a_P p'_P = \sum_{\text{nb}} a_{\text{nb}} p'_{\text{nb}} + b$$

with

$$\begin{aligned} a_E &= \rho_e d_e \Delta y \\ a_W &= \rho_w d_w \Delta y \\ a_N &= \rho_n d_n \Delta x \\ a_S &= \rho_s d_s \Delta x \\ a_P &= a_E + a_W + a_N + a_S \\ b &= F_w^* - F_e^* + F_n^* - F_s^* \end{aligned} \tag{7.56}$$

We see that the pressure correction equation has the same structure as that for the staggered grid formulation. The source term in the pressure correction equation is the mass imbalance resulting from the solution of the momentum equations.

7.7.3 Overall Solution Procedure

The overall SIMPLE solution procedure for co-located meshes takes the following form:

1. Guess the pressure field p^*.
2. Solve the u and v momentum equations using the prevailing pressure field p^* to obtain u^* and v^* at cell centroids.
3. Compute the face mass flow rates F^* using momentum interpolation to obtain face velocities.
4. Solve the p' equation.
5. Correct the face flow rates using Equation 7.48.
6. Correct the cell-centered velocities u_P^* and v_P^* using Equation 7.50.
7. Correct the cell pressure using Equation 7.47. In keeping with the SIMPLE algorithm, underrelax the pressure correction as:
$$p = p^* + \alpha_p p'$$
8. Solve for other scalars ϕ if desired.
9. Check for convergence. If converged, stop. Else go to 2.

7.7.4 Discussion

We see that the overall SIMPLE procedure is very similar to that for staggered meshes. However, we should note a very important difference. The pressure correction equation contains a source term b which is the mass imbalance in the cell P. The computed pressure corrections are designed to annihilate this mass imbalance. Thus, we are assured corrected face flow rates in step 5 will satisfy the discrete continuity equation *identically* at each iteration of the SIMPLE procedure. However, the cell-centered velocities either before or after the correction in step 6 are *never* guaranteed to satisfy the discrete continuity equation. This is because the flow rates F are not written directly using u_P and v_P; the momentum-interpolated values are used instead. Thus, in a co-located formulation, the cell-centered velocities satisfy the discrete momentum equations, but not the discrete continuity equation. We should also note that the cell-velocity correction in step 7 is designed to speed up convergence, but does nothing to make u_P and v_P satisfy the discrete continuity equation. By the same token, the face flow rates (and by implication the face velocities) satisfy the discrete continuity equation at step 5 in each iteration, and also at convergence. However, they do not satisfy a discrete momentum equation directly. This curious disconnect between the cell-center and face velocities is an inherent property of co-located schemes.

The solution of passive scalars ϕ in step 8 employs the continuity-satisfying face flow rates F in discretizing the convective terms. The cell-centered velocities are never used for this purpose since they do not satisfy the discrete continuity equation for the cell.

7.8 Underrelaxation and Time-Step Dependence

We have thus far said very little about the role of underrelaxation in developing our co-located formulation. Majumdar [16] has shown that unless care is taken in underrelaxing the face velocities, the resulting co-located formulation is underrelaxation dependent. That is, the final solution depends on the underrelaxation employed, and different underrelaxation factors may lead to different solutions. This is clearly extremely undesirable.

Consider the face velocity u_e, which may be written as

$$u_e = \hat{u}_e + d_e (p_P - p_E) \tag{7.58}$$

Recall that d_e involves the averages of $1/a_P^u$ and $1/a_E^u$, the center coefficients of the momentum equations at points P and E. Similarly, $\hat{u}_e$ also contains a_P^u and a_E^u in the denominator. Let us say that $\hat{u}_e$ and d_e in Equation 7.40 corresponds to *un-underrelaxed* values of a_P^u and a_E^u.

If the momentum equations are underrelaxed, the cell-centered velocities satisfy

$$\begin{aligned} u_P &= \alpha \left(\hat{u}_P + d_P^u \frac{(p_W - p_E)}{2} \right) + (1-\alpha) u_P^* \\ u_E &= \alpha \left(\hat{u}_E + d_E^u \frac{(p_E - p_{EE})}{2} \right) + (1-\alpha) u_E^* \end{aligned} \tag{7.59}$$

Using momentum interpolation as before, we obtain

$$u_e = \alpha (\hat{u}_e + d_e (p_P - p_E)) + (1-\alpha) \frac{u_P^* + u_E^*}{2} \tag{7.60}$$

In order for a variable ϕ to be underrelaxation-independent at convergence, the underrelaxation expression must have the form

$$\alpha \phi + (1-\alpha) \phi^* \tag{7.61}$$

At convergence, $\phi = \phi^*$, and the above expression recovers ϕ regardless of what underrelaxation is used.

We see that the underrelaxed value of u_e does not have this form. The underrelaxed face velocity has the form

$$\alpha u_e + (1-\alpha) u_{\text{linear}}^* \tag{7.62}$$

where u_{linear}^* is the prevailing linearly interpolated face value. Since u_e is never equal to u_{linear}^*, not even at convergence, the value of u_e is underrelaxation dependent.

The remedy is to use an underrelaxation of the form

$$u_e = \alpha (\hat{u}_e + d_e (p_P - p_E)) + (1-\alpha) u_e^* \tag{7.63}$$

Here $\hat{u}_e$ and d_e are computed using un-underrelaxed momentum equations for cells P and E. The face velocity is then underrelaxed separately to obtain the desired form. We note that the interpolation requires the storage of the face velocity u_e, since we cannot underrelax u_e without storing it.

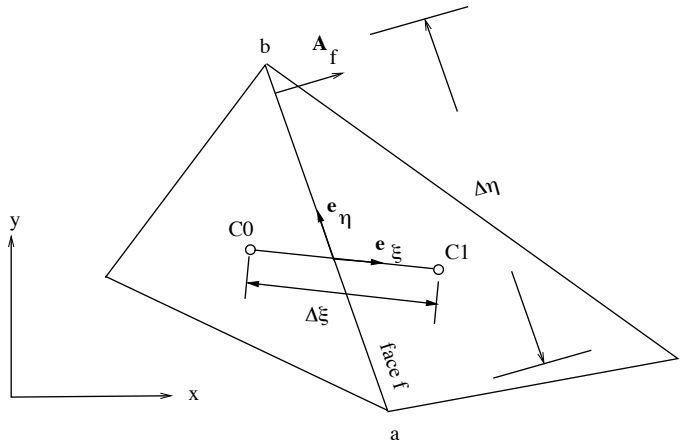


Figure 7.3: Cell Cluster for Unstructured Mesh

Similar arguments may be made about time-step dependence of co-located schemes. Unless remedied, the steady state obtained by co-located formulations will depend on the time step taken during the preceding unsteady process. Recall that the unsteady schemes we used for scalar transport (and indeed all reasonable time-stepping schemes) yield steady state solutions that are independent of the time-steps taken in getting to steady state. A remedy similar to that for underrelaxation may be devised.

We should note that the difference between the momentum-interpolated and linearly-interpolated face value decreases as Δx^3, as shown by our error analysis of the added dissipation scheme. Thus, even if we did not take steps to remedy the situation, we expect the dependence to disappear progressively as the mesh is refined.

7.9 Co-Located Formulation for Non-Orthogonal and Unstructured Meshes

The co-located formulation presented above can be applied readily to non-orthogonal and unstructured meshes. Consider the cells $C0$ and $C1$ in Figure 7.3. In keeping with the co-located formulation, we store the Cartesian velocities u and v and the pressure p at the cell centroids. We note that the direction $\mathbf{e}_\xi$ is aligned with the line joining the centroids, and for general non-orthogonal meshes, is not parallel to the face area vector $\mathbf{A}_f$. The vector $\mathbf{e}_\eta$ is any direction tangential to the face.

The procedures for discretizing the u and v momentum equations on the cell are similar to those adopted in previous chapters for the convection-diffusion equation. The only term that needs special consideration is the pressure gradient term. Since the cell momentum equations are derived by integrating the governing equation over the cell, the pressure gradient term is also integrated over the cell. Applying the gradient

theorem, we may write

$$-\int_{\Delta\mathcal{V}_0} \nabla p d\mathcal{V} = -\int_A p d\mathbf{A} \tag{7.64}$$

Assuming that the pressure at the face centroid prevails over the face, the pressure gradient term in the u and v momentum equations may be written as

$$-\sum_f p_f \mathbf{A}_f \tag{7.65}$$

The face area vector is given by

$$\mathbf{A}_f = A_x\mathbf{i} + A_y\mathbf{j} \tag{7.66}$$

The pressure gradient terms in the u and v momentum equations are

$$\begin{aligned} -\mathbf{i}\cdot\int_{\Delta\mathcal{V}_0} \nabla p d\mathcal{V} &= -\mathbf{i}\cdot\sum_f p_f \mathbf{A}_f = -\sum_f p_f A_x \\ -\mathbf{j}\cdot\int_{\Delta\mathcal{V}_0} \nabla p d\mathcal{V} &= -\mathbf{j}\cdot\sum_f p_f \mathbf{A}_f = -\sum_f p_f A_y \end{aligned} \tag{7.67}$$

In keeping with our co-located mesh technique, p_f is interpolated linearly to the face. For a uniform mesh this interpolation would take the form

$$p_f = \frac{p_0 + p_1}{2} \tag{7.68}$$

For non-uniform meshes, we may use the reconstructed value

$$p_f = \frac{p_0 + \nabla p_0\cdot\mathbf{r}_0 + p_1 + \nabla p_1\cdot\mathbf{r}_1}{2} \tag{7.69}$$

where $\mathbf{r}_0$ and $\mathbf{r}_1$ are the distances from the cell centroids of cells $C0$ and $C1$ to the face centroid. In either event, p_f may be written in terms of the cell-centroid values of the pressure. Since

$$\sum_f \mathbf{A}_f = 0 \tag{7.70}$$

it is clear that the summation in Equation 7.65 eliminates the cell pressure p_0. Thus, as with regular meshes, the momentum equations can support a checkerboarded pressure field. If ∇p_0 and ∇p_1 are computed using the same type of linear assumptions, the reconstructed pressure value from Equation 7.69 will also behave in the same way.

For future use, let us write the pressure gradient term in the cell as

$$-\sum_f p_f \mathbf{A}_f = -\nabla p_0 \Delta\mathcal{V}_0 \tag{7.71}$$

where ∇p_0 denotes the average pressure gradient in the cell.

Having discretized the pressure gradient term, the discrete momentum equations for the cell $C0$ may now be written:

$$\begin{aligned} a_0^u u_0 &= \sum_{nb} a_{nb}^u u_{nb} + b_0^u - \nabla p_0\cdot\mathbf{i}\Delta\mathcal{V}_0 \\ a_0^v v_0 &= \sum_{nb} a_{nb}^v v_{nb} + b_0^v - \nabla p_0\cdot\mathbf{j}\Delta\mathcal{V}_0 \end{aligned} \tag{7.72}$$

Here, a_0^u and a_0^v denote the center coefficients of the u and v momentum equations and b_0^u and b_0^v the source terms. The pressure gradient terms sum the face pressures on all the faces of the cell.

7.9.1 Face Normal Momentum Equation

Let us consider the face f between the cells $C0$ and $C1$. Since it is the face normal velocity that appears in the discrete continuity equation for cells $C0$ and $C1$, it is necessary to understand the form taken by the momentum equation for the velocity in the face normal direction. The face normal vector $\mathbf{n}$ is given by

$$\mathbf{n} = \frac{\mathbf{A}_f}{|\mathbf{A}|_f} = n_x\mathbf{i} + n_y\mathbf{j} \tag{7.73}$$

Let V_0^n denote the component of the cell-centered velocity at cell $C0$ in the direction of the face normal $\mathbf{n}$. This velocity is given by

$$V_0^n = \mathbf{V}_0 \cdot \mathbf{n} = u_0 n_x + v_0 n_y \tag{7.74}$$

In a co-located variable formulation, the coefficients of the momentum equations are equal to each other when there are no body forces present, and for most boundary conditions. This is because the flow rates governing convection are the same for all ϕ's. The diffusion coefficient for the u and v momentum equations is the same, and is equal to viscosity μ. Away from the boundaries, the only difference between the center coefficients a_0^u and a_0^v occurs because of source terms with S_p components which act preferentially in the x or y directions. At Dirichlet boundaries, the coefficient modifications for both velocity directions are the same; the same is true at inlet and outflow boundaries. The main difference occurs at symmetry boundaries aligned with either the x or y directions. But for these exceptions, the u momentum coefficient set (a_0^u and a_{nb}^u) are equal to the coefficients of the v momentum equation (a_0^v and a_{nb}^v).

Under these circumstances, the momentum equation in the cell $C0$ for the velocity in the *face normal* direction may be written as:

$$a_0^n V_0^n = \sum_{\mathrm{nb}} a_{\mathrm{nb}}^n V_{\mathrm{nb}}^n + b_0^n - \nabla p_0 \cdot \mathbf{n} \tag{7.75}$$

where

$$\begin{aligned} a_0^n &= a_0^u = a_0^v \\ a_{\mathrm{nb}}^n &= a_{\mathrm{nb}}^u = a_{\mathrm{nb}}^v \end{aligned} \tag{7.76}$$

The pressure gradient term may be written as

$$-\sum_{\mathrm{f}} p_f A_f = -\left(\frac{\partial p}{\partial n}\right)_0 \Delta\mathscr{V}_0 = -\nabla p_0 \cdot \mathbf{n} \Delta\mathscr{V}_0 \tag{7.77}$$

We note further that

$$b_0^n = b_0^u n_x + b_0^v n_y \tag{7.78}$$

Dividing Equation 7.75 by a_0^n, we may write

$$V_0^n = \hat{V}_0^n - \frac{\Delta\mathscr{V}_0}{a_0^n}\nabla p_0 \cdot \mathbf{n} \tag{7.79}$$

Using similar procedures, we may write, for cell $C1$

$$V_1^n = \hat{V}_1^n - \frac{\Delta\mathscr{V}_1}{a_1^n}\nabla p_1 \cdot \mathbf{n} \tag{7.80}$$

Here

$$\begin{aligned} \hat{V}_0^n &= \hat{u}_0 n_x + \hat{v}_0 n_y \\ \hat{V}_1^n &= \hat{u}_1 n_x + \hat{v}_1 n_y \end{aligned} \tag{7.81}$$

7.9.2 Momentum Interpolation for Face Velocity

The momentum interpolation procedure is applied to the *normal* velocity at the face. Let the linearly interpolated face normal velocity be given by $\overline{V}_f$. On a uniform mesh

$$\overline{V}_f = \frac{V_0^n + V_1^n}{2} \tag{7.82}$$

For non-uniform meshes, face values of u and v may be reconstructed to the face and averaged in the manner of Equation 7.69, and a face normal velocity found using Equation 7.74.

The momentum-interpolated face normal velocity is given by

$$V_f = \overline{V}_f + \frac{\Delta\mathscr{V}_f}{a_f^n}\left(\overline{\nabla} p \cdot \mathbf{n} - \left(\frac{\partial p}{\partial n}\right)_f\right) \tag{7.83}$$

Here, the quantities $\Delta\mathscr{V}_f$ and a_f^n represent the *cell volume* and *center coefficient* associated with the face. These may be chosen in a number of different ways, as long as the associated truncation error is kept $O(\Delta x^3)$. For the purposes of this chapter, we choose them as

$$\begin{aligned} \Delta\mathscr{V}_f &= \frac{\Delta\mathscr{V}_0 + \Delta\mathscr{V}_1}{2} \\ a_f^n &= \frac{a_0^n + a_1^n}{2} \end{aligned} \tag{7.84}$$

The pressure gradient $\overline{\nabla} p$ is the mean pressure gradient at the face and is given by

$$\overline{\nabla} p = \frac{\nabla p_0 + \nabla p_1}{2} \tag{7.85}$$

The quantity $\left(\partial p/\partial n\right)_f$ is the face value of the pressure gradient. In writing Equation 7.83 we are removing the mean normal pressure gradient, and adding in a face

pressure gradient term written. We intend to write this face pressure gradient in terms of the adjacent pressure values p_0 and p_1, just like we did for regular meshes.

We realize however, that the normal gradient of pressure cannot be written purely in terms of p_0 and p_1 for general non-orthogonal meshes; other neighboring values would be involved. Only the gradient $\partial p/\partial \xi$ may be written in terms of p_0 and p_1 alone. Thus, we decompose the normal gradient into the directions ξ and η to obtain, as in previous chapters:

$$\left(\frac{\partial p}{\partial n}\right)_f = \frac{\mathbf{n}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{e}_\xi}\left(\frac{\partial p}{\partial \xi}\right)_f + \frac{\mathbf{n}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{e}_\xi}\mathbf{e}_\xi\cdot\mathbf{e}_\eta\left(\frac{\partial p}{\partial \eta}\right)_f \tag{7.86}$$

We now write the gradient $(\partial p/\partial \eta)_f$ in terms of the mean pressure gradient:

$$\left(\frac{\partial p}{\partial n}\right)_f = \frac{\mathbf{n}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{e}_\xi}\frac{p_1-p_0}{\Delta\xi} + \frac{\mathbf{n}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{e}_\xi}\mathbf{e}_\xi\cdot\mathbf{e}_\eta\overline{\nabla}p\cdot\mathbf{e}_\eta \tag{7.87}$$

Using

$$\overline{\nabla}p\cdot\mathbf{e}_\eta = \overline{\nabla}p\cdot\mathbf{n} - \overline{\nabla}p\cdot\mathbf{e}_\xi \tag{7.88}$$

and combining Equations 7.83 and 7.87, we get

$$V_f = \overline{V}_f + \frac{\Delta\mathscr{V}_f}{a_f^n}\frac{\mathbf{n}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{e}_\xi}\left(\overline{\nabla}p\cdot\mathbf{e}_\xi - \frac{p_1-p_0}{\Delta\xi}\right) \tag{7.89}$$

Thus, our manipulation results in adding a dissipation associated with the gradient $\partial p/\partial \xi$ rather than $\partial p/\partial n$, since this the only gradient that can be directly associated with the adjacent pressure difference $p_1 - p_0$.

Rearranging terms, we may write

$$V_f = \hat{V}_f + d_f\left(p_0 - p_1\right) \tag{7.90}$$

where

$$\begin{aligned}\hat{V}_f &= \overline{V}_f + d_f\overline{\nabla}p\cdot\mathbf{e}_\xi\Delta\xi \\ d_f &= \frac{\Delta\mathscr{V}_f}{\Delta\xi a_f^n}\frac{\mathbf{n}\cdot\mathbf{n}}{\mathbf{n}\cdot\mathbf{e}_\xi}\end{aligned} \tag{7.91}$$

7.10 The SIMPLE Algorithm for Non-Orthogonal and Unstructured Meshes

The procedure for deriving the pressure correction closely parallels that for regular meshes. The face flow rate is defined as

$$F_f = \rho_f\mathbf{V}_f\cdot\mathbf{A}_f = \rho_f V_f A_f \tag{7.92}$$

and represents the *outflow* from cell $C0$. As before, let u^* and v^* be the solutions to the cell momentum equations using a guessed or prevailing pressure field p^*. As

before, the discrete continuity equation for the cell $C0$ is not satisfied by u^* and v^*. We postulate face flow rate corrections F' such that

$$\sum_{\text{f}} F_f^* + F_f' = 0 \tag{7.93}$$

where F^* are the face mass flow rates computed from the momentum-satisfying velocities u^* and v^*. As before, we postulate face normal velocity corrections

$$V_f' = d_f \left(p_0' - p_1' \right) \tag{7.94}$$

Here the correction to $\hat{V}_f$ has been dropped in keeping with the SIMPLE algorithm. The corresponding face flow rate corrections are

$$F_f = \rho_f d_f A_f \left(p_0' - p_1' \right) \tag{7.95}$$

We also postulate a cell pressure correction

$$p_0' = p_0^* + p_0' \tag{7.96}$$

As with regular meshes, we define cell velocity corrections

$$\begin{aligned} u_0' &= \frac{\Delta \mathscr{V}_0}{a_0^u} - \sum_{\text{f}} p_f' A_x \\ v_0' &= \frac{\Delta \mathscr{V}_0}{a_0^v} - \sum_{\text{f}} p_f' A_y \end{aligned} \tag{7.97}$$

with face pressure corrections

$$p_f' = \frac{p_0' + p_1'}{2} \tag{7.98}$$

Substituting Equations 7.94 and 7.96 into the discrete continuity equation (Equation 7.93) yields a pressure correction equation for the cell center pressure. We may write this in the form

$$a_P p_P' = \sum_{\text{nb}} p_{\text{nb}}' + b$$

where

$$\begin{aligned} a_{\text{nb}} &= \rho_f d_f A_f \\ a_P &= \sum_{\text{nb}} a_{\text{nb}} \\ b &= -\sum_{\text{f}} F_f^* \end{aligned} \tag{7.100}$$

7.10.1 Discussion

The broad structure of the pressure correction equation is the same as for regular meshes. The overall SIMPLE algorithm takes the same form, and is not repeated here. However a few important points must be made.

In writing our added dissipation term, we have chosen to add a dissipation involving the term $\partial p/\partial \xi$, and to write this gradient explicitly in terms of $(p_0 - p_1)$. The total gradient driving the face normal velocity, however, also contains a pressure gradient tangential to the face. But because it is not easy to write this explicitly, we have chosen to leave it embedded in the $\hat{V}_f$ term in Equation 7.91. The accuracy of this omission is not a concern since the added dissipation scheme is $O(\Delta\xi^3)$ accurate. However, we should note that this choice does have consequences for convergence.

The primary consequence of this choice is that the pressure correction equation ignores pressure corrections due to $\partial p/\partial \eta$. The $\partial p/\partial \eta$ term is proportional to the non-orthogonality of the mesh. For orthogonal meshes ($\mathbf{e}_\eta \cdot \mathbf{e}_\xi = 0$), the term drops out altogether. But when the mesh is not orthogonal, the pressure correction equation attributes to $\partial p/\partial \xi$ the corrections that should have been attributed to $\partial p/\partial \eta$. The final answer is the same whether we include the corrections due to $\partial p/\partial \eta$ or not; only the rate of convergence changes. Our experience shows that this approximation in the pressure correction equation is tolerable for most reasonable meshes. Since we are dropping the corrections to $\hat{V}_f$ in keeping with the SIMPLE algorithm anyway, we may think of this as an additional approximation to the coefficients of pressure correction equation.

In the interest of clarity, one important aspect has been pushed to the background: the linear interpolation of face pressure. For many flows, the pressure field is smooth and a linear interpolation is adequate. In other cases, the presence of strong body force terms, such as in swirling or buoyant flows, means that the cell pressure gradient is steeper than that implied by linear interpolation. Since a linear interpolation underpredicts the cell pressure gradient, the flow field must distort itself to provide the extra momentum sources required to balance the body force. This can lead to distortions in the cell-centered velocities. Improvement of co-located schemes for large-body force problems continues to be an active area of research.

7.11 Closure

In this chapter, we have developed a co-located formulation for structured and unstructured meshes. We have seen that the primary difficulty has to do with the computation of the face normal velocity, which is used to write the discrete continuity equation. To circumvent checkerboarding resulting from linear interpolation of the face normal velocity, we developed a momentum interpolation or added dissipation scheme. We saw that the idea is easily extended to unstructured meshes and that a SIMPLE algorithm may be developed using it.

At this point, we have a complete procedure capable of computing the convection and diffusion of scalars, as well the underlying flow field. The development has been done for general orthogonal and non-orthogonal meshes, both structured and unstructured. The scheme preserves the basic conservation principle regardless of cell shape. Indeed we have made no assumptions about cell shape save that the cell be an arbitrary convex polyhedron. We turn now to the problem of solving general unstructured sets of algebraic equations which result from the unstructured mesh discretizations we have seen in this and preceding chapters.

Chapter 8

Linear Solvers

As we have seen in the earlier chapters, implicit schemes result in a system of linear equations of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{8.1}$$

Here $\mathbf{A}$ is a $N \times N$ matrix and $\mathbf{x}$ is a vector of the unknowns. The efficient solution of such systems is an important component of any CFD analysis.

Linear systems also arise in numerous other engineering and scientific applications and a large number of techniques have been developed for their solution. However, the systems of equations that we deal with in CFD have certain distinguishing characteristics that we need to bear in mind while selecting the appropriate algorithms.

One important characteristic of our linear systems is that they are very *sparse*, i.e., there are a large number of zeroes in the matrix $\mathbf{A}$. Recall that the discrete equation at a cell has non-zero coefficients for only the neighboring cells. Thus for a two dimensional structured quadrilateral grid, for example, out of the N^2 entries in the matrix, only about $5 * N$ of them are non-zero. It would seem to be a good idea to seek solution methods that take advantage of the sparse nature of our matrix.

Depending on the structure of the grid, the matrix might also have specific *fill pattern*, i.e the pattern of location of the non-zero entries. The system of equations resulting from a one-dimensional grid, for example, has non-zero entries only on the diagonal and two adjacent "lines" on either side. For a mesh of 5 cells, the matrix has the form

$$\mathbf{A} = \begin{bmatrix} x & x & 0 & 0 & 0 \\ x & x & x & 0 & 0 \\ 0 & x & x & x & 0 \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \end{bmatrix} \tag{8.2}$$

Here x denote the non-zero entries. As we shall see shortly, linear systems involving such matrices, known as tri-diagonal matrices, can be solved easily and form the basis of some solution methods for more general matrices as well. We also note that two and three-dimensional structured grids similarly result in banded matrices, although the exact structure of these bands depends on how the cells are numbered. Once again,

it would seem advantageous to exploit the band structure of our matrix, both for storage and solution techniques.

Another important characteristic of our linear systems is that in many instances they are *approximate*. By this we mean that the coefficients of the matrix and/or the source vector **b** are themselves subject to change after we have solved the equation. This maybe because of coupling between different equations (e.g., the mass flux appearing in convective coefficients of the energy equation), variable properties (temperature dependent thermal conductivity, for example) or other non-linearities in the governing equations. Whatever the underlying reason, the implication for the linear solver is that it may not be really worthwhile to solve the system to machine accuracy. Since we are going to recompute the coefficient matrix and solve the linear problem once again, it is usually sufficient if we obtain only an approximate solution to any given linear system. Also, as we are iterating, we usually have a good initial guess for the unknown and linear solvers that can take advantage of this are obviously desirable.

8.1 Direct vs Iterative Methods

Linear solution methods can broadly be classified into two categories, direct or iterative. Direct methods, such as Gauss elimination, LU decomposition etc., typically do not take advanatage of matrix sparsity and involve a fixed number of operations to obtain the final solution which is determined to machine accuracy. They also do not take advantage of any initial guess of the solution. Given the characteristics of the linear systems outlined above, it is easy to see why they are rarely used in CFD applications.

Iterative methods on the other hand, can easily be formulated to take advantage of the matrix sparsity. Since these methods successively improve the solution by the application of a fixed number of operations, we can stop the process when the solution at any given outer iteration [1] has been obtained to a sufficient level of accuracy and not have to incur the expense of obtaining the machine-accurate solution. As the outer iterations progress and we have better initial guesses for the iterations of the linear solver, the effort required during the linear solution also decreases. Iterative methods are therefore preferred and we shall devote the bulk of this chapter to such methods.

8.2 Storage Strategies

As we have already noted, a large number of the entries of our coefficient matrix are zero. Consequently, it is very inefficient to use a two dimensional array structure to store our matrix. In this section we consider some smarter ways of storing only the non-zero entries that will still allow us to perform any of the matrix operations that the solution algorithm might require. The exact way of doing this will depend on the nature of the grid.

[1] To distinguish the iterations being performed because of equation non-linearity and/or inter equation coupling from the iterations being performed to obtain the solution for a given linear system, we use the term "outer iteration" for the former and "inner iteration" or simply iteration for the latter.

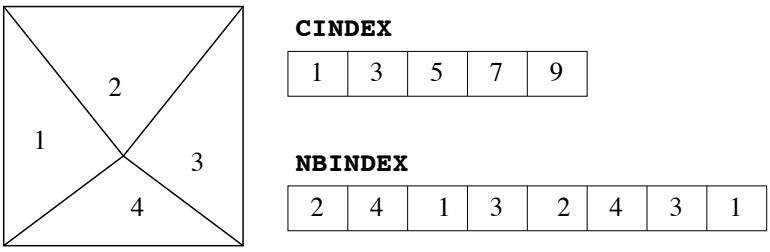


Figure 8.1: Storage Scheme for Unstructured Mesh Coefficient Matrix

For a one-dimensional grid of N cells, for example, we could store the diagonal and the two lines parallel to it using three one-dimensional arrays of length N. Following the notation we have used in the previous chapters, we label these arrays `AP, AE` and `AW`, respectively. The non-zero entries of the matrix $\mathbf{A}$ can then be obtained as

$$\mathbf{A}(i,i) = \texttt{AP(i)} \tag{8.3}$$
$$\mathbf{A}(i,i-1) = -\texttt{AW(i)} \tag{8.4}$$
$$\mathbf{A}(i,i+1) = -\texttt{AE(i)} \tag{8.5}$$

For a two dimensional structured grid of $NI \times NJ$ cells, it is usually convenient to refer to the cells using a double index notation and therefore we could use 5 two dimensional arrays of dimension (NI,NJ) to store the `AP, AE, AW, AN` and `AS` coefficients. Alternatively, one might prefer to number the cells using a single index notation and store coefficients using 5 one-dimensional arrays of size $NI * NJ$ instead. In either case, because of the grid structure we implicitly know the indices of the neighboring cells and thus the position of these coefficients in the matrix $\mathbf{A}$. It is therefore easy to interpret any matrix operation involving the coefficient matrix $\mathbf{A}$ in terms of these coefficient arrays.

For unstructured grids however, the connectivity of the matrix must be stored explicitly. Another difficulty is caused by the fact that the number of neighbors is not fixed. Therefore we cannot use the approach mentioned above of storing coefficients as a_P, a_E, a_W etc. arrays. We will look at one strategy that is often used in these cases.

Consider a mesh of N cells and let n_i represent the number of neighbors of cell i. The total number of neighbor coefficients that we need to store is then given by

$$B = \sum_{i=1}^{N} n_i \tag{8.6}$$

We allocate two arrays of length B, one of integers (labelled `NBINDEX`) and one of floating point numbers (labelled `COEFF`). We also allocate one other integer array of length $N+1$, labelled `CINDEX` which is defined as

$$\texttt{CINDEX(1)} = 1 \tag{8.7}$$
$$\texttt{CINDEX(i)} = \texttt{CINDEX(i-1)} + n_{i-1} \tag{8.8}$$

```
TDMA(AP,AE,AW,B,X)
  {
    for i = 2 to N
      {
        r = AW(i)/AP(i-1);
        AP(i) = AP(i) - r*AE(i-1);
        B(i) = B(i) - r*B(i-1);
      }
    X(N) = B(N)/AP(N);
    for i = N-1 down to 1
      {
        X(i) = (B(i) - AW(i)*X(i+1))/AP(i);
      }
  }
```

Figure 8.2: Tri-Diagonal Matrix Algorithm

The idea is that the indices of the neighbours of cell i will be stored in the array `NBINDEX` at locations locations j that are given by `CINDEX(i)` $<= j <$ `CINDEX(i+1)`. The corresponding coefficients for these neighbors are stored in the corresponding locations in the `COEFF` array. Finally the center coefficient is stored in a separate array `AP` of length N. This is illustrated in Fig. 8.1 which shows the contents of the `CINDEX` and `NBINDEX` for a two dimensional unstructured grid.

8.3 Tri-Diagonal Matrix Algorithm

Although the bulk of this chapter is concerned with iterative solution techniques, for the tridiagonal linear system arising out of a one-dimensional problem there is a particularly simple direct solution method that we consider first. The idea is essentially the same as Gaussian elimination; however the sparse, tri-diagonal pattern of the matrix allows us to obtain the solution in $O(N)$ operations. This is accomplished in two steps. First, the matrix is *upper-triangularized*, i.e., the entries below the diagonal are successively eliminated starting with the second row. The last equation thus has ony one unknown and can be solved. The solution for the other equations can then be obtained by working our way back from the last to the first unknown, in a process known as *back-substitution*. Using the storage strategy described by Eq. 8.3, the algorithm can be written in the form shown in Fig. 8.3

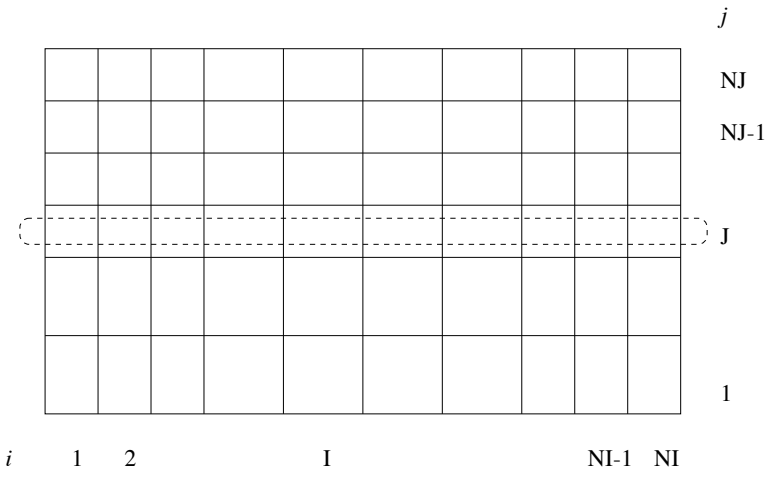


Figure 8.3: Structured Grid for Line by Line TDMA

8.4 Line by line TDMA

Linear systems arising from two or three dimensional structured grids also have a regular fill pattern. Unfortunately, there are no simple methods analogous to the TDMA that we saw in the previous section for the direct solution of such systems. However, using the TDMA we can devise *iterative* methods. Consider, for example, the two dimensional structured grid shown in Fig. 8.3. We will assume that the coefficient matrix is stored using the strategy discussed in Sec. 8.2, i.e, in 5 two dimensional arrays `AP, AE, AW, AN` and `AS`. The equation at point (I,J) is then given by

```
AP(I,J) X(I,J)+AE(I,J) X(I+1,J)+AW(I,I) X(I-1,J)
+AN(I,J) X(I,J+1)+AS(I,J) X(I,J-1)=B(I,J)        (8.9)
```

We also assume that we have a guess for the solution everywhere. We rewrite this equation as

```
AP(I,J) X(I,J)+AE(I,J) X(I+1,J)+AW(I,I) X(I-1,J)=
=B(I,J)-AN(I,J) X*(I,J+1)-AS(I,J) X*(I,J-1)      (8.10)
```

where the superscript denotes guessed values. The right hand side of Eq. 8.10 is thus considered to be known and only `X(I,J), X(I+1,J)` and `X(I-1,J)` are considered to be unknowns. Writing similar equations for all the cells $(i,J), i = 1,NI$ (shown by the dotted oval in Fig. 8.3 we obtain a system which has the same form as the tri-diagonal system which we can then solve using TDMA. This gives us values for `X(i)` for all cells i along $j = J$ line. However, unlike the one-dimensional problem, this is not the *exact* solution but only an approximate one since we had to guess for values of `X(i,J+1)` and `X(i,J-1)` in building up the tri-diagonal system. We can now

```
for j = 1 to NJ
  {
    for i = 1 to NI
      {
        AP1D(i) = AP(i,j);
        AE1D(i) = AE(i,j);
        AW1D(i) = AW(i,j);
        B1D(i) = B(i,j);
        if (j > 1) B1D(i) = B1D(i) - AS(i,j)*X(i,j-1);
        if (j < NJ) B1D(i) = B1D(i) - AN(i,j)*X(i,j+1);
      }
    TDMA(AP1D,AE1D,AW1D,B1D,X1D);
    for i = 1 to NI
      {
        X(i,j) = X1D(i);
      }
  }
```

Figure 8.4: Line By Line TDMA Algorithm along j lines

apply the same process along the next line, $j = J + 1$. In doing so we will use the recently computed values whenever `X(i,J)`'s are required. The overall procedure can be described with the pseudocode shown in Fig. 8.4

Once we have applied the process for all the j lines, we will have updated the value of each $X(i, j)$. As noted above these are only approximate values but hopefully they are better approximations than our initial guess. As in all iterative methods, we will try to improve the solution by repeating the process. To this end, we could apply the algorithm in Fig. 8.4 again. However, we notice that visiting j lines in sequence from 1 to NJ means that all cells have seen the influence of the boundary at $y = 0$ but only the cells at $j = NJ$ have seen influence of $y = 1$ boundary. If we repeat the process in Fig. 8.4 again, this time the cells at $j = NJ - 1$ will see this influence (since they will use the values obtained at $j = NJ$ during the present update) but it will takes several repetitions before cells near $J = 1$ see any influence of the boundary at $y = 1$. One easy way of removing this bias is to visit the j lines in the reverse order during the second update. With this symmetric visiting sequence we ensure that all cells in the domain see the influence of both boundaries as soon as possible.

The sequence of operations whereby all the values of $X(i, j)$ are updated once is referred to as a *sweep*. An *iteration* is the sequence that is repeated multiple number of times. Thus for the line by line TDMA, an iteration may consist of two sweeps, first visiting all j lines in increasing order of their index and then in decreasing order as described above. Of course, it is not mandatory to apply the TDMA for a constant j line. We could apply the same process along a constant i, considering only the j

direction neighbours implicitly. Depending on the coordinate direction along which information transfer is most critical, sweeping by visiting i or j lines might be the most optimal. In general cases, however, it is useful to combine both. Thus one iteration of the line by line TDMA would consist of visiting, say each of the i lines first in increasing order and then in decreasing order followed by similar symmetric sweeps along j lines (and k lines in three dimensional problems).

8.5 Jacobi and Gauss Seidel Methods

For matrices resulting from unstructured grids, like the one shown in Fig. 8.1, of course, no line-by-line procedure is possible. Instead, we must use more general update methods. The simplest of these are the Jacobi and Gauss-Seidel methods. In both cases, the cells are visited in sequence and at each cell i the value of x_i is updated by writing its equation as

$$A_{i,i}x_i = b - \sum A_{i,\mathrm{nb}}x_{\mathrm{nb}} \tag{8.11}$$

where the summation is over all the neighbors of cell i. The two methods differ in the values of the neighboring x_i that are employed. In case of the Jacobi method, the "old" values of x_i are used for all the neighbors whereas in the Gauss-Seidel method, the latest values of x_i are used at all the neighbours that have already been updated during the current sweep and old values are used for the neighbours that are yet to be visited. As in the case of the line by line TDMA, the order of visiting the cell is reversed for the next sweep so as to avoid directional bias.

In general the Gauss-Seidel method has better convergence characteristics than the Jacobi method and is therefore most widely used although the latter is sometimes used on vector and parallel hardware. Using the storage scheme outlined in Sec. 8.2, one iteration of the Gauss-Seidel method can be expressed in the pseudo-code shown in Fig. 8.5.

8.6 General Iterative Methods

The general principle in all iterative methods is that given an approximate solution $\mathbf{x}^k$, we seek to obtain a better approximation $\mathbf{x}^{k+1}$ and then repeat the whole process. We define the error at any given iteration as

$$\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k \tag{8.12}$$

where $\mathbf{x}$ is the exact solution. Of course, since we don't know the exact solution, we also don't know the error at any iteration. However, it is possible to check how well any given solution satisfies the equation by examining the residual $\mathbf{r}$, defined as

$$\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k \tag{8.13}$$

As the residual approaches zero, the solution approaches the exact solution. We can determine the relation between them using Eqs. 8.12 and 8.13 as

$$\mathbf{A}\mathbf{e}^k = \mathbf{r}^k \tag{8.14}$$

```
Gauss_Seidel(AP,COEFF,CINDEX,NBINDEX,B,X)
  {
    for sweep = 1 to 2
      {
        if (sweep = 1)
          IBEG = 1, IEND = N, ISTEP = 1;
        else
          IBEG = N, IEND = 1, ISTEP = -1;
        for i = IBEG to IEND stepping by ISTEP
          {
            r = B(i);
            for n = CINDEX(i) to CINDEX(i+1)-1
              {
                j = NBINDEX(n);
                r = r - COEFF(n)*X(j);
              }
            X(i) = r/AP(i);
          }
      }
  }
```

Figure 8.5: Symmetric Gauss-Seidel Sweep

Thus the error satisfies the same set of equations as the solution, with the residual $\mathbf{r}$ replacing the source vector $\mathbf{b}$. This is an important property that we will make use of in devising multigrid schemes.

Using Eq. 8.14 and the definition of the error we obtain

$$\mathbf{x} - \mathbf{x}^k = \mathbf{A}^{-1}\mathbf{r}^k \tag{8.15}$$

Most iterative methods are based on approximating this expression as

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{B}\mathbf{r}^k \tag{8.16}$$

where $\mathbf{B}$ is some approximation of the inverse of A that can be computed inexpensively. For example, it can be shown that the Jacobi method is obtained when $\mathbf{B} = \mathbf{D}^{-1}$, where $\mathbf{D}$ is the diagonal part of the matrix $\mathbf{A}$.

Another way of expressing any iterative scheme that we will find useful in later analysis is

$$\mathbf{x}^{k+1} = \mathbf{P}\mathbf{x}^k + \mathbf{g}^k \tag{8.17}$$

where $\mathbf{P}$ is known as the *iteration matrix*. For the Jacobi method the iteration matrix is given by $\mathbf{P} = \mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})$ while for the Gauss-Seidel method it is $\mathbf{P} = (\mathbf{D}-\mathbf{L})^{-1}\mathbf{U}$. Here $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ are the diagonal, strictly lower and upper triangular parts of $\mathbf{A}$ obtained by splitting it as

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U} \tag{8.18}$$

8.7 Convergence of Jacobi and Gauss Seidel Methods

Although the Jacobi and Gauss-Seidel methods are very easy to implement and are applicable for matrices with arbitrary fill patterns their usefulness is limited by their slow convergence characteristics. The usual observation is that residuals drop quickly during the first few iterations but afterwards the iterations "stall". This is specially pronounced for large matrices.

To demonstrate this behavior, let us consider the following 1D Poisson equation over a domain of length L.

$$\frac{\partial^2 \phi}{\partial x^2} = s(x) \tag{8.19}$$

and specified Dirichlet boundary conditions $\phi(0) = \phi_0$ and $\phi(L) = \phi_L$. Recall that this equation results from our 1d scalar transport equation in the pure diffusion limit if we choose a diffusion coefficient of unity. If we discretize this equation on a grid of N equispaced control volumes using the method outlined in Chapter 3 we will obtain a linear system of the form

$$\frac{1}{h}\begin{bmatrix} 3 & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{N-1} \\ \phi_N \end{bmatrix} = \begin{bmatrix} hs_1 + \frac{2\phi_0}{h} \\ hs_2 \\ \vdots \\ hs_{N-1} \\ hs_N + \frac{2\phi_L}{h} \end{bmatrix} \tag{8.20}$$

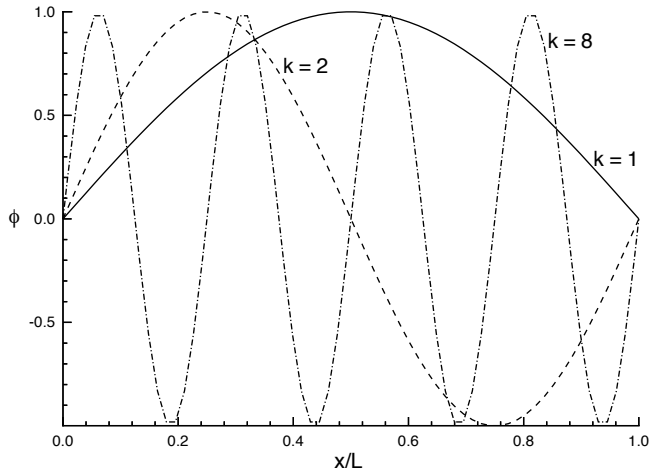


Figure 8.6: Fourier Modes on $N = 64$ grid

where $h = \frac{L}{N}$ and hs_i represents the source term integrated over the cell.

Another simplification we make is to choose $\phi_0 = \phi_L = 0$ as well as $s(x) = 0$. Thus the exact solution to this problem is simply $\phi(x) = 0$. We can now study the behavior of iterative schemes by starting with arbitrary initial guesses; the error at any iteration is then simply the current value of the variable ϕ. In order to distinguish the convergence characteristics for different error profiles we will solve the problem with initial guesses given by

$$\phi_i = \sin\left(\frac{k\pi x_i}{L}\right) \tag{8.21}$$

Equation 8.21 represents *Fourier* modes and k is known as the wavenumber. Figure 8.6 shows these modes over the domain for a few values of k. Note that for low values of k we get "smooth" profiles while for higher wavenumbers the profiles are very oscillatory.

Starting with these Fourier modes, we apply the Gauss Seidel method for 50 iterations on a grid with $N = 64$. To judge how the solution is converging we plot the maximum ϕ_i (which is also the maximum error). The results are shown in Fig. 8.7(a). We see that when we start with an initial guess corresponding to $k = 1$, the maximum error has reduced by less than 20% but with a guess of $k = 16$ Fourier mode, the error reduces by over 99% even after 10 iterations.

In general cases, our initial guess will of course contain more than one Fourier mode. To see what the scheme does in such cases we start with an initial guess consist-

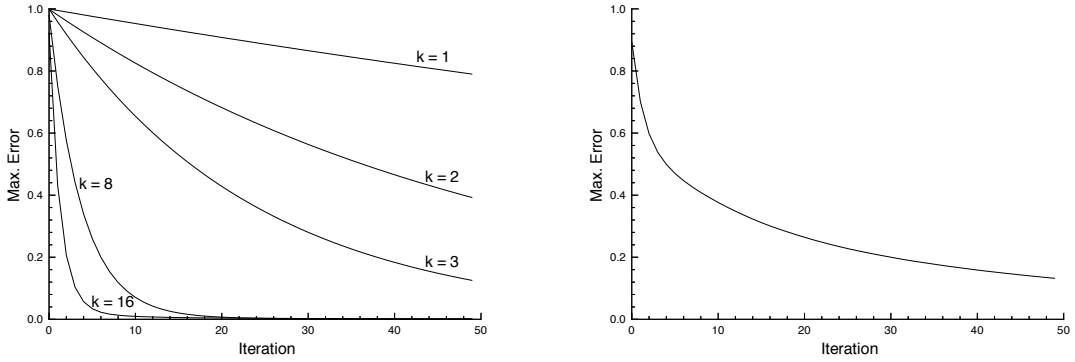


Figure 8.7: Convergence of Gauss-Seidel method on $N = 64$ grid for (a) initial guesses consisting of single wavenumbers (b) initial guess consisting of multiple modes

ing of $k = 2,8$ and 16 modes i.e.

$$\phi_i = \frac{1}{3}\left[\sin\left(\frac{2\pi x_i}{L}\right) + \sin\left(\frac{8\pi x_i}{L}\right) + \sin\left(\frac{16\pi x_i}{L}\right)\right] \tag{8.22}$$

In this case we see from Fig. 8.7(b) that the error drops rapidly at first but then stalls.

Another way of looking at the effect of the iterative scheme is to plot the solution after 10 iterations as shown in Fig. 8.8. We see that the amplitude is not significantly reduced when the initial guess is of low wave number modes but it is greatly reduced for the high wave number modes. Interesting results are obtained for the mixed mode initial guess given by Eq. 8.22. We see that the oscillatory component has vanished leaving a smooth mode error profile.

These numerical experiments begin to tell us the reasons behind the typical behavior of the Gauss-Seidel scheme. It is very effective at reducing high wavenumber errors. This accounts for the rapid drop in residuals at the beginning when one starts with an arbitrary initial guess. Once these oscillatory components have been removed we are left with smooth error profiles on which the scheme is not very effective and thus convergence stalls.

Using our sample problem we can also verify another commonly observed shortcoming of the Gauss-Seidel iterative scheme viz. that the convergence deteriorates as the grid is refined. Retaining the same form of initial guess and using $k = 2$, we solve the problem on a grid that is twice as fine, ie., $N = 128$. The resulting convergence plot shown in Fig. 8.9 indicate that the convergence becomes even worse. On the finer grid we can resolve more modes and again the higher ones among those converge quickly but the lower modes appear more "smooth" on the finer grid and hence converge slower. We also note from Fig. 8.9 that the converse is also true, ie., on a coarse grid with $N = 32$, the convergence is quicker for the same mode. It appears that the same error profile behaves as a less smooth profile when solved on a coarser grid.

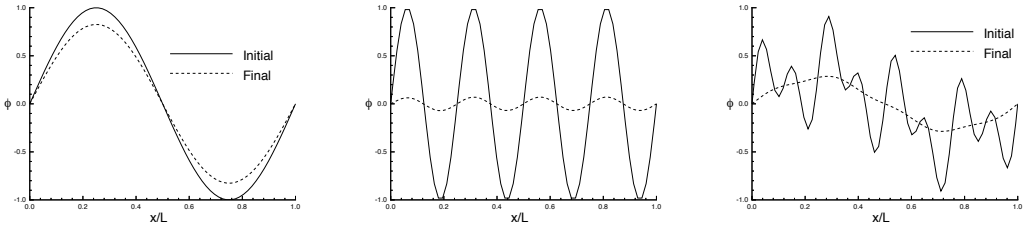


Figure 8.8: Initial and final solutin after 10 Gauss-Seidel iterations on $N = 64$ grid for (a) initial guesses consisting of $k = 2$ (b) initial guesses consisting of $k = 8$ (c) initial guess consisting of multiple modes

Some of the methods for accelerating convergence of iterative solvers are based upon this property.

8.8 Analysis Of Iterative Methods

For simple linear systems like the one we used in the examples above and for simple iterative schemes like Jacobi, it is possible to understand the reasons for the convergence behaviour analyically. We will not go into details here but briefly describe some of the important results. Using the iterative scheme expressed in the form Eq. 8.16, we can show that the error at any iteration n is related to the initial error by the following expression

$$\mathbf{e}^n = \mathbf{P}^n \mathbf{e}^0 \tag{8.23}$$

In order for the error to reduce with iterations, the *spectral radius* of the iteration matrix (which is the largest absolute eigenvalue of the matrix) must be less than one and the rate of convergence depends on how small this spectral radius is.

The eigen values of the Jacobi iteration matrix are closely related to the eigen values of matrix $\mathbf{A}$ and the two matrices have the same eigen vectors. Now, if we choose the matrix linear system to be [2]

$$\frac{1}{h}\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{N-1} \\ \phi_N \end{bmatrix} = \begin{bmatrix} hs_1 + \frac{\phi_0}{h} \\ hs_2 \\ \vdots \\ hs_{N-1} \\ hs_N + \frac{\phi_L}{h} \end{bmatrix} \tag{8.24}$$

[2] This is the system one obtains using a finite difference discretization of Eq. 8.19 with an equispaced mesh consisting of *N interior* nodes and is only slightly different from the system we used in the previous section. Both the systems have similar convergence characteristics; the reason for choosing this form instead of Eq. 8.20 is that it is much easier to study analytically.

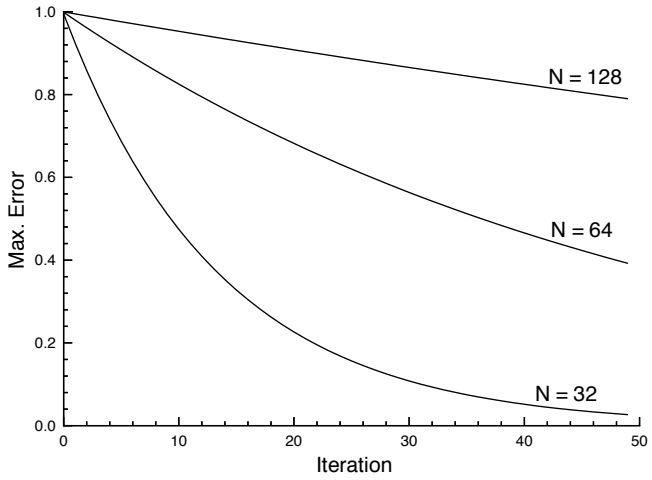


Figure 8.9: Convergence of Gauss-Seidel method on different sized grids for initial guess corresponding to $k = 2$ mode

we can find analytical expression for the eigenvalues of the corresponding Jacobi iteration matrix. They are given by

$$\lambda_k = 1 - \sin^2\left(\frac{k\pi}{2N}\right) \quad \text{k} = 1,2,...\text{N} \tag{8.25}$$

The eigenvectors of the Jacobi iteration matrix, $\mathbf{w}_k$ turn out to be the same as the Fourier modes we used in the previous section as starting guesses. The j^{th} component of the eigen vector corresponding to the eigenvalue λ_k is given by

$$w_{k,j} = \sin\left(\frac{jk\pi}{N}\right) \quad \text{j} = 1,2,...\text{N} \tag{8.26}$$

Now, if our initial error is decomposed into Fourier modes, we can write it in terms of these eigenvectors as

$$\mathbf{e}^0 = \sum \alpha_k \mathbf{w}_k \tag{8.27}$$

Substituing this expression in Eq. 8.16 and using the definition of eigenvalue, we obtain

$$\mathbf{e}^n = \sum \alpha_k \lambda_k^n \mathbf{w}_k \tag{8.28}$$

We see from this expression that the k^{th} mode of the initial error is reduced by a factor λ_k^n. From Eq. 8.25 we note that the largest eigenvalue occurs for $k = 1$ and therefore it is easy to see why the lower modes are the slowest to converge. Also note that the magnitude of the largest eigenvalue increases as N increases; this indicates the reason

behind our other observation that convergence on coarser grids is better than that on finer grids for the same mode.

Although we have analyzed the convergence behaviour of a very simple scheme on a simple matrix, these conclusions hold true in general. The eigenvalues of the matrix **A** as well as that of the iteration matrix play an important role in determining the convergence characteristics. Our insistence on maintaining diagonal dominance and positive coefficients is motivated by the requirement of keeping the spectral radius below unity. Practices such as linearizing source terms and underrelaxation that we need in order to handle non-linearities also help in reducing the spectral radius. However in many cases, e.g., the pressure correction equation, we must handle *stiff* linear systems, i.e., those with spectral radius close to 1.

Many iterative schemes have been devised to handle stiff systems. They usually involve some form of *preconditioning* to improve the eigenvalues of the iteration matrix. In general, these methods are a lot more complicated to implement compared to the simple Jacobi and Gauss-Seidel methods we have studied so far. We will not discuss any of them here but rather look at another strategy, which is based on the improved convergence characteristics of the simple iterative schemes on coarser meshes.

8.9 Multigrid Methods

We saw in the previous section that the reason for slow convergence of Gauss-Seidel method is that it is only effective at removing high frequency errors. We also observed that low frequency modes appear more oscillatory on coarser grids and then the Gauss-Seidel iterations are more effective. These observations suggest that we could accelerate the convergence of these iterative linear solvers if we could somehow involve coarser grids.

The two main questions we need to answer are (1) what problem should be solved on the coarse grid and (2) how should we make use of the coarse grid information in the fine grid solution. Certain constraints can be easily identified. We know that in general the accuracy of the solution depends on the discretization; therefore we would require that our final solution be determined only by the finest grid that we are employing. This means that the coarse grids can only provide us with corrections or guesses to the fine grid solution and as the fine grid residuals approach zero (i.e., the fine grid solution approaches the exact answer) the influence of any coarse levels should also approach zero. One consequence of this requirement is that it is enough to solve only an approximate problem at the coarse levels since its solution will never govern the final accuracy we achieve.

One strategy for involving coarse levels might be to solve the original differential equation on a coarse grid. Once we have a converged solution on this coarse grid, we could interpolate it to a finer grid. Of course, the interpolated solution will not in general satisfy the discrete equations at the fine level but it would probably be a better approximation than an arbitrary initial guess. We can repeat the process recursively on even finer grids till we reach the desired grid.

The disadvantage with this strategy, known as *nested iteration* is that it solves the problem fully on all coarse grids even though we are only interested in the finest grid

solution. It also does not make use of any guess we might have for the finest level grid from previous outer iterations. More importantly however, the physical problem may not be well resolved on the coarse grid and thus the solution on coarse grid may not always be a good guess for the fine grid solution. Smooth error modes may arise only on the finer grids and then the convergence will still be limited by them.

8.9.1 Coarse Grid Correction

A more useful strategy, known as *coarse grid correction* is based on the error equation (Eq. 8.14). Recall that the error $\mathbf{e}$ satisfies the same set of of equations as our solution if we replace the source vector by the residual:

$$\mathbf{Ae} = \mathbf{r} \tag{8.29}$$

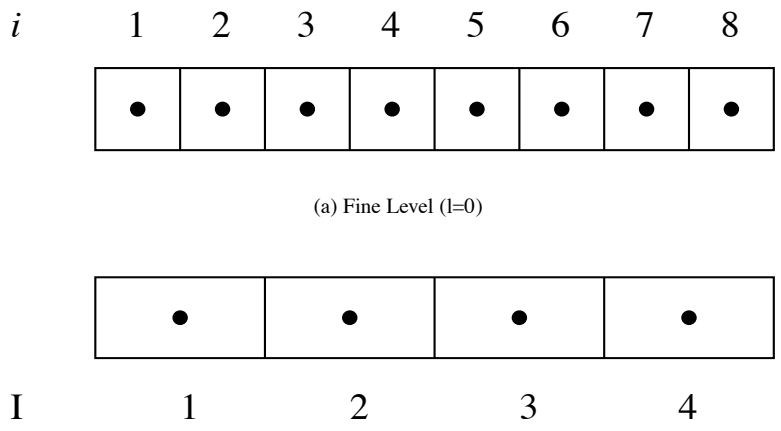
Note also that solving Eq. 8.1 with an initial guess $\mathbf{x}^0$ is identical to solving Eq. 8.29 with the residual $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$ and an initial guess of zero error. Now suppose that after some iterations on the finest grid we have a solution $\mathbf{x}$. Although we don't know the error we do know that the error at this stage is likely to be smooth and that further iterations will not reduce it quickly. Instead we could try to *estimate* it by solving Eq. 8.29 on a coarser grid. We expect that on the coarser grid the smooth error will be more oscillatory and therefore convergence will be better. When we have obtained a satisfactory solution for $\mathbf{e}$ we can use it to correct our fine grid solution.

Of course, even on the coarse level the error (which of course now is the error in Eq. 8.29, i.e., the error in the estimation of the fine grid error) will have smooth components. We can now view Eq. 8.29 as a linear problem in its own right. We can thus apply the same strategy for its solution that we used for the finest grid, i.e., solve for its error on an even coarser mesh. This can be continued recursively on successively coarser meshes till we reach one with a few number (2-4) of cells. At this stage we can simply solve the linear analytically, although using Gauss-Seidel iteration usually suffices as well.

We still have to specify exactly what we mean by solving Eq. 8.29 on a coarse grid and how exactly we intend to use the errors estimated from the coarse levels but we can already see that the strategy outlined above has the desired properties. First of all, note that if the fine grid solution is exact, the residual will be zero and thus the solution of the coarse level equation will also be zero. Thus we are guaranteed that the final solution is only determined by the finest level discretization. In addition, since we start with a zero initial guess for the coarse level error, we will achieve convergence right away and not waste any time on further coarse level iterations. Another useful characteristic of this approach is that we use coarse level to only estimate fine level errors. Thus any approximations we make in the coarse level problem only effect the convergence rate and not the final finest grid solution.

8.9.2 Geometric Multigrid

Having developed a general idea of how coarse grids might be used to accelerate convergence, let us now look at some details. To distinguish between the matrices and



(a) Fine Level (l=0)

(b) Coarse Level (l=1)

Figure 8.10: Coarsening for 1D grid

vectors at different grid levels, we shall employ a parenthicated superscript, starting with (0) for the finest grid and increasing it as the grid is coarsened ; a second superscript, if present, will denote the iteration number. Thus the problem to be solved at the level (l) is given by

$$\mathbf{A}^{(l)}\mathbf{x}^{(l)} = \mathbf{b}^{(l)} \tag{8.30}$$

and the residual at level l after k iterations is

$$\mathbf{r}^{(l),k} = \mathbf{b}^{(l)} - \mathbf{A}^{(l)}\mathbf{x}^{(l),k} \tag{8.31}$$

We have already seen how to discretize and iterate on the finest level ($l = 0$) grid of N cells and compute the residual $\mathbf{r}^{(0)}$. We also know that for coarse levels ($l > 0$), the unknown $\mathbf{x}^{(l)}$ represents the estimate of the error at the $(l-1)$ level and that the source vector $\mathbf{b}^{(l)}$ is somehow to be based on the residual $\mathbf{r}^{(l-1)}$. After doing some iterations on the coarse grid (and perhaps recursively repeating the process at further coarse levels) we would like to make use of the errors calculated at level l to correct the current guess for solution at the finer level. The next step is to define the exact means of doing these coarse grid operations and the intergrid transfers.

The simplest way of obtaining the coarse grid for our sample 1D problem is to merge cells in pairs to obtain a grid of $N/2$ cells, as shown in Fig. 8.10. The resulting grid is similar to the original grid, with a cell width of $2h$. We can now apply our usual discretization procedure on the differential equation (remembering that the unknown is a correction to the fine level unknown) and obtain the linear system of the same form

as 8.20. The coarse level matrix is given by

$$\mathbf{A}^{(1)} = \frac{1}{2h}\begin{bmatrix} 3 & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 3 \end{bmatrix} \tag{8.32}$$

We also note from Fig. 8.10 that the cell centroid of a coarse level cell lies midway between the centroids of its parent fine level cells. The source term for the coarse level cell can thus be obtained by averaging the residuals at the parent cells:

$$b_i^{(l+1)} = \frac{1}{2}\left(r_{2i-1}^{(l)} + r_{2i}^{(l)}\right) \tag{8.33}$$

This operation of transfering the residual from a fine level to the coarse level is known as *restriction* and is denoted by the operator I_l^{l+1} defined as

$$\mathbf{b}^{(l+1)} = I_l^{l+1}\mathbf{r}^{(l)} \tag{8.34}$$

For our equispaced grid we used averaging as the restriction operator; in the general case we will need to use some form of *interpolation* operator.

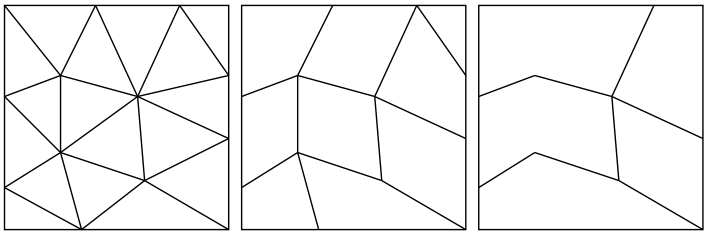
We already know that the starting guess for the coarse level unknowns $x^{(l+1),0}$ is zero. Thus we now have all the information to iterate the coarse level system and obtain an esitmate for the error. The process of transfering this correction back to the finer level is known as *prolongation* and is denoted by the operator I_{l+1}^{l}. The correction of the solution at the fine level using the coarse level solution is written as

$$\mathbf{x}^{(l)} = \mathbf{x}^{(l)} + I_{l+1}^{l}\mathbf{x}^{(l+1)} \tag{8.35}$$

The simplest prolongation operator that we can use on our 1D grid (Fig. 8.10) is to apply the correction from a coarse level cell to both the parent fine level cells, i.e. use a zeroth order interpolation. A more sophisticated approach is to use linear interpolation; for example, for cell 2 at the fine level we use

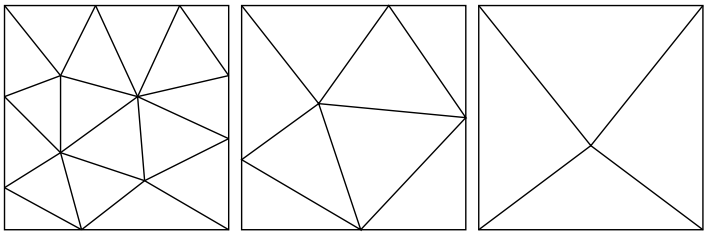
$$x_2^{(0)} = x_2^{(0)} + \left(\frac{3}{4}x_1^{(1)} + \frac{1}{4}x_2^{(1)}\right) \tag{8.36}$$

The strategy outlined in this section is known as *geometric* multigrid because we made use of the grid geometry and the differential equation at the coarse levels in order to arrive at the linear system to be solved. In the one dimensional case, of course, the coarse level cells had the same type of geometry as those at the fine level and this made the discretization process straightforward. For multidimensional cases, however, the cell shapes obtained by agglomerating fine level cells can be very different. As shown in Fig. 8.11 the coarse level cells may not even be convex. With such *nested* grid hierarchies it may not be feasible to discretize the original differential equation on the coarse level cells.



(a) l=0 (b) l=1 (c) l=2

Figure 8.11: Nested Coarsening For 2D grid



(a) l=0 (b) l=1 (c) l=2

Figure 8.12: Independent Coarsening For 2D grid

One approach to get around this problem is to use a sequence of non-nested, independent grids, as shown in Fig. 8.12. In this case there are no common faces between any two grids. The main problem with this approach is that the prolongation and restriction operators become very complicated since they involve multidimensional interpolation. Also, in some instances, for example, the pressure correction equation that we derived algebraically, we may not have a formal differential equation that can be discretized to obtain the coarse level system. For these reasons, it is useful to devise methods of obtaining coarse grid linear systems that do not depend on the geometry or the differential equation. We will look at such *algebraic* multigrid methods in the next section.

8.9.3 Algebraic Multigrid

The general principles behind algebraic multigrid methods are the same as those for the geometric multigrid method we saw in the last section. The main difference is that the coarse level system is derived purely from the fine level system without reference to the underlying grid geometry or physical principle that led to the fine level system. Instead of thinking in terms of agglomerating two or more fine level cells to obtain the geometry of a coarse level cell, we speak of agglomerating the *equations* at those cells to directly obtain the linear equation corresponding to that coarse cell.

We will see shortly how to select the equations to be agglomerated in the general case. For the moment let us just consider the 1D grid and the coarse grid levels, shown in Fig. 8.10, that we used in the geometric multigrid section above. Let i and $i+1$ be the indices of the fine level equations that we will agglomerate to produce the coarse level equation with index I. For instance, cells 1 and 2 at level 0 are combined to obtain the equation for cell 1 at level 1. Writing out the error equation (Eq. 8.14) for indices i and $i+1$ we have

$$A^{(0)}_{i,i-1}e^{(0)}_{i-1} + A^{(0)}_{i,i}e^{(0)}_{i} + A^{(0)}_{i,i+1}e^{(0)}_{i+1} = r^{(0)}_{i} \tag{8.37}$$

$$A^{(0)}_{i+1,i}e^{(0)}_{i} + A^{(0)}_{i+1,i+1}e^{(0)}_{i+1} + A^{(0)}_{i+1,i+2}e^{(0)}_{i+2} = r^{(0)}_{i+1} \tag{8.38}$$

Now, the multigrid principle is that the errors at level l are to be estimated from the unknowns at level $l+1$. We assume that the error for both the parent cells i and $i+1$ is the same and is obtained from $x^{(1)}_{I}$. Likewise, $e^{(0)}_{i-1} = x^{(1)}_{I-1}$ and $e^{(0)}_{i+2} = x^{(1)}_{I+1}$. Substituting these relations and adding Eqs. 8.37 and 8.38 gives us the required coarse level equation for index I:

$$A^{(0)}_{i,i-1}x^{(1)}_{I-1} + (A^{(0)}_{i,i} + A^{(0)}_{i,i+1} + A^{(0)}_{i+1,i} + A^{(0)}_{i+1,i+1})x^{(1)}_{I} + A^{(0)}_{i+1,i+2}x^{(1)}_{I+1} = r^{(0)}_{i} + r^{(0)}_{i+1} \tag{8.39}$$

Thus we see that the coefficients for the coarse level matrix have been obtained by summing up coefficients of the fine level matrix and without any use of the geometry or differential equation. The source term for the coarse level cells turn out be the sum of the residuals at the constituent fine levels cells. This is equivalent to the use of *addition* as the restriction operator. Our derivation above implies zeroth order interpolation as the prolongation operator.

8.9.4 Agglomeration Strategies

In the 1D case that we have seen so far, agglomeration was a simple matter of combining equations at cells $2i$ and $2i+1$ to obtain the coarse level system of equations. For linear systems resulting from two or three dimensional structured grids, the same idea can be applied in one or more directions simultaneously. Besides simplifying the book-keeping this practice has the additional advantage of maintaining the penta- or equations at a time to obtain septa-diagonal form of the linear system. This permits the use of relaxation methods such as line-by-line TDMA on the coarse level systems as well as the fine level systems.

For unstructured grids, however, we need to devise more general agglomeration criteria. One useful practice is to try to combine cells that have the largest mutual coefficients. This creates coarse level grids [3] that allow the optimal transfer of boundary information to interior regions and thus accelerates convergence.

To implement such a coarsening procedure, we associate a coarse index with each fine level cell and initialize it to zero. We also initialize a coarse level cell counter $C = 1$ We then visit the cells (or equations) in sequence, and if has not been grouped (i.e., its coarse index is 0), group it and n of its neighbours for which the coefficient $A_{i,j}$ is the largest (i.e, assign them the coarse level index C) and increment C by 1.

We have already seen that the coefficients of the coarse level matrix are obtained by summing up appropriate coefficients of the fine level matrix. Proceeding in the same manner that we used to derive Eq. 8.39 we can show that

$$A_{I,J}^{(l+1)} = \sum_{i \in G_I} \sum_{j \in G_J} A_{i,j}^{l} \tag{8.40}$$

where G_I denotes the set of fine level cells that belong to the coarse level cell I. Also, as we have seen before, the source vector for the coarse level equation is obtained by summing up the residuals of the constituent fine level cells

$$b_I^{(l+1)} = \sum_{i \in G_I} r_i^{(l)} \tag{8.41}$$

The coarse level matrices can be stored using the same storage strategies outlined in Sec. 8.2 for the finest level. Best multigrid performance is usually observed for $n = 2$, i.e., a coarse level grid that consists of roughly half the number of cells as the finest level. If such a division is continued till we have just 2 or 3 cells at the coarsest level, the total memory required for storing all the coarse levels is roughly equal to that required for the finest level.

The linear systems encountered in CFD applications are frequently stiff. This stiffness is a result of a number of factors: large aspect-ratio geometries, disparate grid sizes typical of unstructured meshes, large conductivity ratios in conjugate heat transfer problems, and others. The agglomeration strategy outlined above is very effective in accelerating the convergence rate of the linear solver.

[3] Even though we are not concerned with the actual geometry of the coarse level grids in algebraic multigrid, it is nevertheless quite useful to visualize the effective grids resulting from the coarsening in order to understand the behaviour of the method.

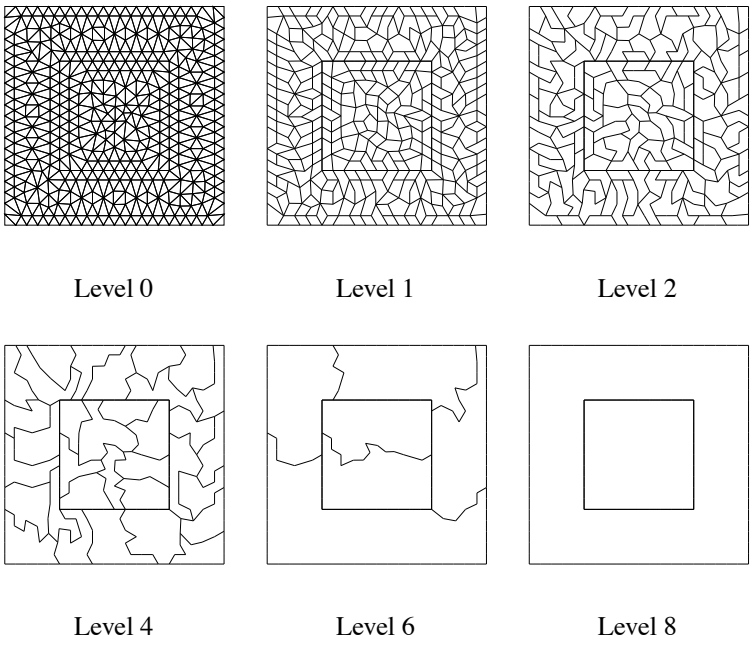


Figure 8.13: Conduction in composite domain: multigrid coarsening

Consider the situation depicted in Fig. 8.13. A composite domain consists of a low-conductivity outer region surrounding a highly conducting inner square domain. The ratio of conductivities is 1000; a ratio of this order would occur for a copper block in air. The temperature is specified on the four external walls of the domain. Convergence of typical linear solvers is inhibited by the large anisotropy of coefficients for cells bordering the interface of the two regions. Coefficients resulting from the diffusion term scale as $kA/\Delta x$, where A is a typical face area and Δx is a typical cell length scale. For interface cells in the highly conducting region, coefficients to interior cells are approximately three orders of magnitude bigger than coefficients to cells in the low-conducting region. However, Dirichlet boundary conditions, which set the level of the temperature field, are only available at the outer boundaries of the domain, adjacent to the low-conducting region. Information transfer from the outer boundary to the interior region is inhibited because the large-coefficient terms overwhelm the boundary information transferred through the small-coefficient terms. An agglomeration strategy which clusters cell neighbors with the largest coefficients results in the coarse levels shown in Fig. 8.13. At the coarsest level, the domain consists of a single cell in the high-conducting region, and another in the low-conducting region. The associated coefficient matrix has coefficients of the same order. The temperature level of the inner region is set primarily by the multigrid corrections at this level, and results in very fast convergence.

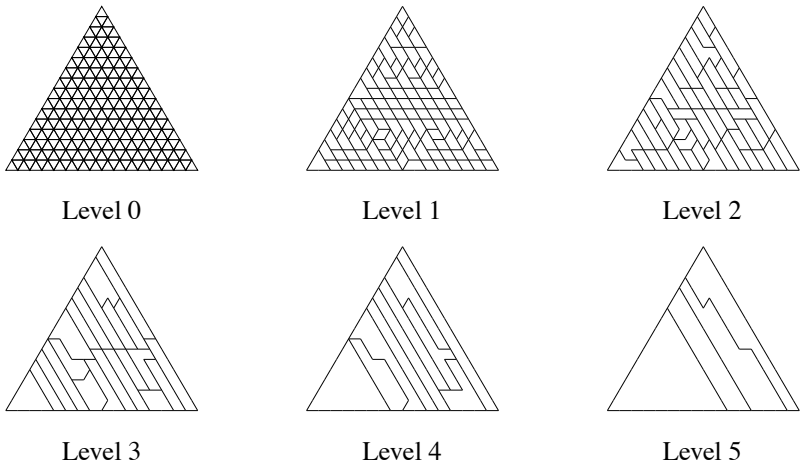


Figure 8.14: Orthotropic conduction: multigrid coarsening

Another example is shown in Fig. 8.14. The problem involves orthotropic conduction in a triangular domain with temperature distributions given on all boundaries [17]. The material has a conductivity $k_{\eta\eta} > 0$ in the η direction, aligned at $\pi/3$ radians from the horizontal; the conductivity $k_{\xi\xi}$ in the direction perpendicular to η is zero. Mesh agglomeration based on coefficient size results in coarse-level meshes aligned with η as shown. Since all faces with normals in the ξ direction have zero coefficients, the primary direction of information transfer is in the η direction. Thus, the coarse level mesh correctly captures the direction of information transfer.

We should note here that the coefficient based coarsening strategy are dersirable even on structured grids. Athough coarse levels created by agglomerating complete grid lines in each grid direction have the advantage of preserving the grid structure and permitting the use of the same line-by-line relaxation schemes as used on the finest level, they do not always result in optimal multigrid acceleration in general situations since coefficient anisotropies are not always aligned along lines.

Algebraic multigrid methods used with sequential solution procedures have the advantage that the agglomeration strategy can be equation-specific; the discrete coefficients for the specific governing equation can be used to create coarse mesh levels. Since the coarsening is based on the coefficients of the linearized equations it also changes appropriately as the solution evolves. This is specially useful for non-linear and/or transient problems. In some applications, however, the mutual coupling between the governing equations is the main cause of convergence degradation. Geometric multigrid methods that solve the coupled problem on a sequence of coarser meshes may offer better performance in such cases.

8.9.5 Cycling Strategies and Implementation Issues

The attractiveness of multigrid methods lies in the fact that significant convergence acceleration can be achieved just by using simple relaxation sweeps on a sequence of coarse meshes. Various strategies can be devised for the manner in which the coarse levels are visited. These *cycling* strategies can be expressed rather compactly using recursion and this makes it very easy to implement them, specially using a computer language such as C that allows recursion (i.e., a function is allowed call itself).

Mutigrid cycles can broadly be classified into two categories – (1) fixed cycles that repeat a set pattern of coarse grid visits and (2) flexible cycles that involve coarse grid relaxations as and when they are needed. We will look at both of these ideas next. The general principles of these cycling strategies are applicable for both geometric and algebraic multigrid methods but we shall concentrate on the latter.

Fixed Cycles

We have seen that the coarse level source vector is computed from the residuals at the previous fine level and thus it changes every time we visit a coarse level. However, the coarse level matrix is only a function of coefficients of the fine level matrix and thus remains constant. The starting point in all fixed grid methods therefore is to compute all the coarse level coefficients. With algebraic multigrid it is usually desirable to keep coarsening the grid till there are only two or three cells left; for geometric multigrid the coarsest possible grid size might be dictated by the minimum number of cells required to reasonably discretize the governing equation.

The simplest fixed cycle is known as the *V cycle* and consists of two legs. In the first leg we start with the finest level and perform a fixed number of relaxation sweeps, then transfer the residuals to the next coarse level and relax on that level, continuing till we reach the coarsest level. After finishing sweeps on the coarsest level we start the upward leg, using the solution from the current level to correct the the solution at the next finer level, then perfoming some relaxation sweeps at that finer level and continuing the process till we reach the finest level. The two parameters defining the V-cycle are the number of sweeps performed on the down and up legs, ν_1 and ν_2 respectively. The two need not be equal; in many applications it is most efficient to have $\nu_1 = 0$, i.e., to not do any sweeps on the down leg but to simply keep injecting the residuals till the coarsest level. The coarsest level then establishes an average solution over the entire domain which is then refined by relaxation sweeps on the upward leg. We should note that since the coarse grids only provide an *estimate* of the error, it is generally a good idea to always have non-zero ν_2 in order to ensure that the solution satisfies the discrete equation at the current level. This cycle is graphically illustrated in Fig. 8.15(a) where each circle represents relaxation sweeps and the up and down arrows represent prolongation and restriction operators, respectively.

For very stiff systems, the V-cycle may not be sufficient and more coarse level iterations are called for. This can be achieved by using a μ cycle. It is best understood through a recursive definition. One can think of the V-cycle as a fixed grid cycle where the cycle is recursively applied at each coarse grid if we haven't reach the coarsest grid. The μ-cycle can then be thought of as a cycle which is recursively applied μ

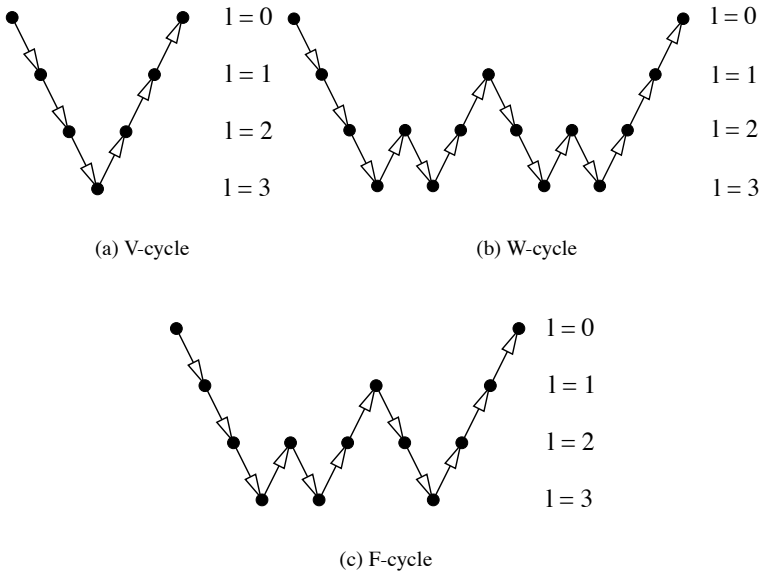


Figure 8.15: Relaxation and Grid Transfer Sequences for Some Fixed Cycles

times at each successive level. The commonly used version is the one corresponding to $\mu = 2$, which is also known as the W-cycle. It is illustrated in Fig. 8.15(b). Because of the recursiveness, a W-cycle involves a lot of coarse level relaxation sweeps, specially when the number of levels is large. A slight variant of the W-cycle, known as the F-cycle, involves somewhat less coarse level sweeps but still more than the V-cycle. It can be thought of as a fixed cycle where one recursively applies one fixed cycle followed by a V-cycle at each successive level. It is illustrated in Fig. 8.15(c).

All the fixed grid cycles we have seen so far can be expressed very compactly in the recursive pseudocode shown in Fig. 8.16 The entire linear solver can then be expressed using the code shown in Fig. 8.17.

Here α is the *termination criterion* which determines how accurately the system is to be solved and nmax is the maximum number of fixed multigrid cycles allowed. $||\mathbf{x}||$ represents some suitable norm of the vector $\mathbf{x}$. Usually the L-2 norm (i.e., the RMS value) or the L-∞ norm (i.e., the largest value) is employed.

Flexible Cycles

For linear systems that are not very stiff, it is not always economical to use all multigrid levels all the time in a regular pattern. For such cases the use of *flexible* cycles is preferred. Here, we monitor the residuals after every sweep on a given grid level and if the ratio is above a specified rate β, we transfer the problem to the next coarse level and continue sweeps at that level. If the ratio is below β we continue sweeps at the current level till the termination criterion is met. When we meet the termination criterion at

```
Fixed_Cycle(l, cycle_type)
{
   Perform ν_1 relaxation sweeps on A^(l) x^(l) = b^(l);
   if (l ≠ lmax)
      {
         r^(l) = b^(l) − A^(l) x^(l);
         b^(l+1) = I_l^(l+1) r^(l);
         x^(l+1) = 0;
         Fixed_Cycle(l+1, cycle_type);
         if (cycle_type = μ_CYCLE)
            Fixed_Cycle(l+1, W_CYCLE) (μ − 1) times;
         else if (cycle_type = F_CYCLE)
            Fixed_Cycle(l+1, V_CYCLE);
         x^(l) = x^(l) + I_(l+1)^l x^(l+1);
      }
   Perform ν_2 relaxation sweeps on A^(l) x^(l) = b^(l);
}
```

Figure 8.16: Fixed Cycle Algorithm

```
Solve(A^(0), x^(0), x^(0), α, cycle_type)
{
   Compute r_0 = ||b^(0) − A^(0) x^(0)||;
   Compute all coarse level matrices;
   for n = 1 to nmax
      {
         Fixed_Cycle(0,cycle_type);
         Compute r_n = ||b^(0) − A^(0) x^(0)||;
         if (r_n/r_0 < α)
            return;
      }
}
```

Figure 8.17: Driver Algorithm for Fixed Cycle

```
Flexible_Cycle(l)
{
   Compute r_0 = ||b^(l) − A^(l) x^(l)||;
   Set r_old = r_0;
   if total number of sweeps on level l not exhausted
      {
         Perform ν_1 relaxation sweeps on A^(l) x^(l) = b^(l);
         Compute r = ||b^(l) − A^(l) x^(l)||;
         if (r/r_0) < α
            return;
         else if ((r/r_old) > β) and (l ≠ lmax)
            {
               Compute A^(l+1) if first visit to level l+1;
               r^(l) = b^(l) − A^(l) x^(l);
               b^(l+1) = I_l^(l+1) r^(l);
               x^(l+1) = 0;
               Flexible_Cycle(l+1);
               x^(l) = x^(l) + I_(l+1)^l x^(l+1);
            }
         else
            {
               r_old = r;
            }
      }
}
```

Figure 8.18: Flexible Cycle Algorithm

any level, and we are not already at the finest level, the solution at that level is used to correct the solution at the next finer level and the process continues. In practical implementation, a limit is imposed on the number of relaxation sweeps allowed at any level. The flexible cycle can also be described compactly in a recursive form, as shown in Fig. 8.18

8.10 Closure

In this chapter, we examined different approaches to solving the linear equation sets that result from discretization. We saw that the only viable approaches for most fluid flow problems were iterative methods. The line-by-line TDMA algorithm may be used for structured meshes, but is not suitable for unstructured meshes. However, methods

like Gauss-Seidel or Jacobi iteration do not have adequate rates of convergence. We saw that these schemes are good at reducing high frequency errors, but cannot reduce low-frequency errors. By the same token, they are also inadequate on fine meshes. To accelerate these schemes, we examined geometric and algebraic multigrid schemes, which use coarse mesh solutions for the error to correct the fine mesh solution. These schemes have been shown in the literature to substantially accelerate linear solver convergence, and are very efficient way to solve unstructured linear systems.

Bibliography

[1] R. Courant, K.O. Friedrichs, and H. Lewy. Uber die partiellen differenzgleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74, 1928.

[2] R. Courant, K.O. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM J. Res. Dev.*, 11:215–234, 1967.

[3] B.P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, 19:59–98, 1979.

[4] T. J. Barth and D. C. Jespersen. The design and application of upwind schemes on unstructured meshes. AIAA 89-0366, 1989.

[5] D.A. Anderson, J.C. Tannehill, and R. H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. Hemisphere Publishing Corporation, 1984.

[6] A.J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, 1967.

[7] K.C. Karki and S.V. Patankar. Pressure based calculation procedure for viscous flows at all speeds in arbitrary configurations. *AIAA Journal*, 27(9):1167–1174, 1989.

[8] S.P. Vanka. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *Journal of Computational Physics*, 65:138–158, 1986.

[9] S.V. Patankar and D.B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15:1787–1805, 1972.

[10] S. V. Patankar. *Numerical Heat Transfer and Fluid Flow*. McGraw-Hill New York, 1980.

[11] J.P. Van Doormal and G.D. Raithby. Enhancements of the SIMPLE method for predicting incompressible fluid flows. *Numerical Heat Transfer*, 7:147–163, 1984.

[12] B. R. Baliga and S. V. Patankar. A control-volume finite element method for two-dimensional fluid flow and heat transfer. *Numerical Heat Transfer*, 6:245–261, 1983.

[13] C. Hsu. *A Curvilinear-Coordinate Method for Momentum, Heat and Mass Transfer in Domains of Irregular Geometry*. PhD thesis, University of Minnesota, 1981.

[14] C.M. Rhie and W.L. Chow. A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation. *AIAA Journal*, 21:1525–1532, 1983.

[15] M. Peric. *A Finite Volume Method for the Prediction of Three Dimensional Fluid Flow in Complex Ducts*. PhD thesis, University of London, August 1985.

[16] S. Majumdar. Role of underrelaxation in momentum interpolation for calculation of flow with nonstaggered grids. *Numerical Heat Transfer*, 13:125–132, 1988.

[17] J.Y. Murthy and S.R. Mathur. Computation of anisotropic conduction using unstructured meshes. *Journal of Heat Transfer*, 120:583–591, August 1998.