An Introduction to Computational Fluid Dynamics

Chapter 20 in Fluid Flow Handbook

By

Nasser Ashgriz &
Javad Mostaghimi
Department of Mechanical & Industrial Eng.
University of Toronto
Toronto, Ontario

1 Introduction:

This chapter is intended as an introductory guide for Computational Fluid Dynamics CFD. Due to its introductory nature, only the basic principals of CFD are introduced here. For more detailed description, readers are referred to other textbooks, which are devoted to this topic.[1,2,3,4,5] CFD provides numerical approximation to the equations that govern fluid motion. Application of the CFD to analyze a fluid problem requires the following steps. First, the mathematical equations describing the fluid flow are written. These are usually a set of partial differential equations. These equations are then discretized to produce a numerical analogue of the equations. The domain is then divided into small grids or elements. Finally, the initial conditions and the boundary conditions of the specific problem are used to solve these equations. The solution method can be direct or iterative. In addition, certain control parameters are used to control the convergence, stability, and accuracy of the method.

All CFD codes contain three main elements: (1) A pre-processor, which is used to input the problem geometry, generate the grid, define the flow parameter and the boundary conditions to the code. (2) A flow solver, which is used to solve the governing equations of the flow subject to the conditions provided. There are four different methods used as a flow solver: (i) finite difference method; (ii) finite element method, (iii) finite volume method, and (iv) spectral method. (3) A post-processor, which is used to massage the data and show the results in graphical and easy to read format.

In this chapter we are mainly concerned with the flow solver part of CFD. This chapter is divided into five sections. In section two of this chapter we review the general governing equations of the flow. In section three we discuss three standard numerical solutions to the partial differential equations describing the flow. In section four we introduce the methods for solving the discrete equations, however, this section is mainly on the finite difference method. And in section five we discuss various grid generation methods and mesh structures. Special problems arising due to the numerical approximation of the flow equations are also discussed and methods to resolve them are introduced in the following sections.

2 Mathematical Formulation

2.1 *Governing equations*

The equations governing the fluid motion are the three fundamental principles of mass, momentum, and energy conservation.

Continuity $$\frac{\partial \rho}{\partial t} + \nabla.(\rho \mathbf{V}) = 0 \tag{1}$$

Momentum $$\rho \frac{D\mathbf{V}}{Dt} = \nabla.\boldsymbol{\tau}_{ij} - \nabla p + \rho \mathbf{F} \tag{2}$$

Energy $$\rho \frac{De}{Dt} + p(\nabla.\mathbf{V}) = \frac{\partial Q}{\partial t} - \nabla.\mathbf{q} + \boldsymbol{\Phi} \tag{3}$$

where ρ is the fluid density, $\boldsymbol{V}$ is the fluid velocity vector, $\boldsymbol{\tau}_{ij}$ is the viscous stress tensor, p is pressure, $\mathbf{F}$ is the body forces, e is the internal energy, Q is the heat source term, t is time, $\boldsymbol{\Phi}$ is the dissipation term**,** and $\nabla.\mathbf{q}$ is the heat loss by conduction. Fourier's law for heat transfer by conduction can be used to describe $\mathbf{q}$ as:

$$\mathbf{q} = -k\nabla T \tag{4}$$

where k is the coefficient of thermal conductivity, and T is the temperature. Depending on the nature of physics governing the fluid motion one or more terms might be negligible. For example, if the fluid is incompressible and the coefficient of viscosity of the fluid, μ, as well as, coefficient of thermal conductivity are constant, the continuity, momentum, and energy equations reduce to the following equations:

$$\nabla.\mathbf{V} = 0 \tag{5}$$

$$\rho \frac{D\mathbf{V}}{Dt} = \mu \nabla^2 \mathbf{V} - \nabla p + \rho \mathbf{F} \tag{6}$$

$$\rho \frac{De}{Dt} = \frac{\partial Q}{\partial t} + k\nabla^2 T + \boldsymbol{\Phi} \tag{7}$$

Presence of each term and their combinations determines the appropriate solution algorithm and the numerical procedure. There are three classifications of partial differential equations[6]; elliptic, parabolic and hyperbolic. Equations belonging to each of

these classifications behave in different ways both physically and numerically. In particular, the direction along which any changes are transmitted is different for the three types. Here we describe each class of partial differential equations through simple examples:

Elliptic:

Laplace equation is a familiar example of an elliptic type equation.

$$\nabla^2 u = 0 \tag{8}$$

where u is the fluid velocity. The incompressible irrotational flow (potential flow) of a fluid is represented by this type of equation. Another practical example of this equation is the steady state pure heat conduction in a solid, i.e., $\nabla^2 T=0$, as can be readily obtained from equation (7).

Parabolic:

The unsteady motion of the fluid due to an impulsive acceleration of an infinite flat plate in a viscous incompressible fluid exemplifies a parabolic equation:

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u \tag{9}$$

where ν is the kinematic viscosity. Transient diffusion equation is represented with a similar equation. In this type of equations, events propagate into the future, and a monotone convergence to steady state is expected.

Hyperbolic:

Qualitative properties of hyperbolic equations can be explained by a wave equation.

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{10}$$

where c is the wave speed. In this case, values of solution depend locally on the initial conditions. The propagation signal speed is finite. Continuous boundary and initial values can give rise to discontinuity. Solution is no more continuous and therefore shocks can be observed and captured in this class of equations.

Depending on the flow, the governing equations of fluid motion can exhibit all three classifications.

2.2 *Boundary Conditions*

The governing equation of fluid motion may result in a solution when the boundary conditions and the initial conditions are specified. The form of the boundary conditions that is required by any partial differential equation depends on the equation itself and the way that it has been discretized. Common boundary conditions are classified either in terms of the numerical values that have to be set or in terms of the physical type of the boundary condition. For steady state problems there are three types of spatial boundary conditions that can be specified:

I. Dirichlet boundary condition: $$\phi = f_1(x, y, z) \tag{11}$$

Here the values of the variable ϕ on the boundary are known constants f_1. This allows a simple substitution to be made to fix the boundary value. For example, if u is the flow velocity, its value may be fixed at the boundary of the domain. For instance, for no-slip and no-penetration conditions on the solid walls, the fluid velocity is the same as the velocity of the wall.

II. Neuman boundary condition: $$\frac{\partial \phi}{\partial n} = f_2(x, y, z) \tag{12}$$

Here the derivatives of the variable ϕ on the boundary are known f_2, and this gives an extra equation, which can be used to find the value at the boundary. For example, if the velocity does not change downstream of the flow, we can assume that the derivative of u is zero at that boundary.

III. Mixed type boundary condition: $$a\phi + b\frac{\partial \phi}{\partial n} = f_3(x, y, z) \tag{13}$$

The physical boundary conditions that are commonly observed in the fluid problems are as follows:

(A) Solid walls: Many boundaries within a fluid flow domain will be solid walls, and these can be either stationary or moving walls. If the flow is laminar then the velocity components can be set to be the velocity of the wall. When the flow is turbulent, however, the situation is more complex.

(B) Inlets: At an inlet, fluid enters the domain and, therefore, its fluid velocity or pressure, or the mass flow rate may be known. Also, the fluid may have certain characteristics, such as the turbulence characterizes which needs to be specified.

(C) Symmetry boundaries: When the flow is symmetrical about some plane there is no flow through the boundary and the derivatives of the variables normal to the boundary are zero.

(D) Cyclic or periodic boundaries: These boundaries come in pairs and are used to specify that the flow has the same values of the variables at equivalent positions on both of the boundaries.

(E) Pressure Boundary Conditions: The ability to specify a pressure condition at one or more boundaries of a computational region is an important and useful computational tool. Pressure boundaries represent such things as confined reservoirs of fluid, ambient laboratory conditions and applied pressures arising from mechanical devices. Generally, a pressure condition cannot be used at a boundary where velocities are also specified, because velocities are influenced by pressure gradients. The only exception is when pressures are necessary to specify the fluid properties, e.g., density crossing a boundary through an equation of state.

There are typically two types of pressure boundary conditions, referred to as static or stagnation pressure conditions. In a static condition the pressure is more or less continuous across the boundary and the velocity at the boundary is assigned a value based on a zero normal-derivative condition across the boundary.

In contrast, a stagnation pressure condition assumes stagnation conditions outside the boundary so that the velocity at the boundary is zero. This assumption requires a pressure drop across the boundary for flow to enter the computational region. Since the static pressure condition says nothing about fluid velocities outside the boundary (i.e., other than it is supposed to be the same as the velocity inside the boundary) it is less specific than the stagnation pressure condition. In this sense the stagnation pressure condition is generally more physical and is recommended for most applications.

As an example, consider the problem of flow in a section of pipe. If the upstream end of the computational region coincides with the physical entrance to the pipe then a stagnation condition should be used to represent the external ambient conditions as a large reservoir of stationary fluid. On the other hand, if the upstream boundary of the computing region is inside the pipe, and many diameters away from the entrance, then the static pressure condition would be a more reasonable approximation to flow conditions at that location.

(F) Outflow Boundary Conditions: In many simulations there is a need to have fluid flow out of one or more boundaries of the computational region. At such "outflow" boundaries there arises the question of what constitutes a good boundary condition.

In compressible flows, when the flow speed at the outflow boundary is supersonic, it makes little difference how the boundary conditions are specified since flow disturbances cannot propagate upstream. In low speed and incompressible flows, however, disturbances introduced at an outflow boundary can have an affect on the entire computational region. It is this possibility that is discussed in this article.

The simplest and most commonly used outflow condition is that of a “continuative" boundary. Continuative boundary conditions consist of zero normal derivatives at the

boundary for all quantities. The zero-derivative condition is intended to represent a smooth continuation of the flow through the boundary.

It must be stressed that the continuative boundary condition has no physical basis, rather it is a mathematical statement that may or may not provide the desired flow behavior. In particular, if flow is observed to enter the computational region across such a boundary, then the computations may be wrong because nothing has been specified about flow conditions existing outside the boundary.

As a general rule, a physically meaningful boundary condition, such as a specified pressure condition, should be used at out flow boundaries whenever possible. When a continuative condition is used it should be placed as far from the main flow region as is practical so that any adverse influence on the main flow will be minimal.

(G) Opening Boundary Conditions: If the fluid flow crosses the boundary surface in either directions an opening boundary condition needs to be utilized. All of the fluid might flow out of the domain, or into the domain, or a combination of the two might happen.

(H) Free Surfaces and Interfaces: If the fluid has a free surface, then the surface tension forces need to be considered. This requires utilization of the Laplace's equation which specifies the surface tension-induced jump in the normal stress p_s across the interface:

$$p_s = \sigma \kappa \tag{14}$$

where σ represents the liquid-air surface tension and κ the total curvature of the interface[7,8,9,10]. A boundary condition is required at the contact line, the line at which the solid, liquid and gas phases meet. It is this boundary condition which introduces into the model information regarding the wettability of the solid surface.

2.2.1 *Example*

In this example a converging-diverging nozzle with a distributed inlet ports is considered. Inlet mass flow rate is known and flow exits to the ambient air with atmospheric pressure.

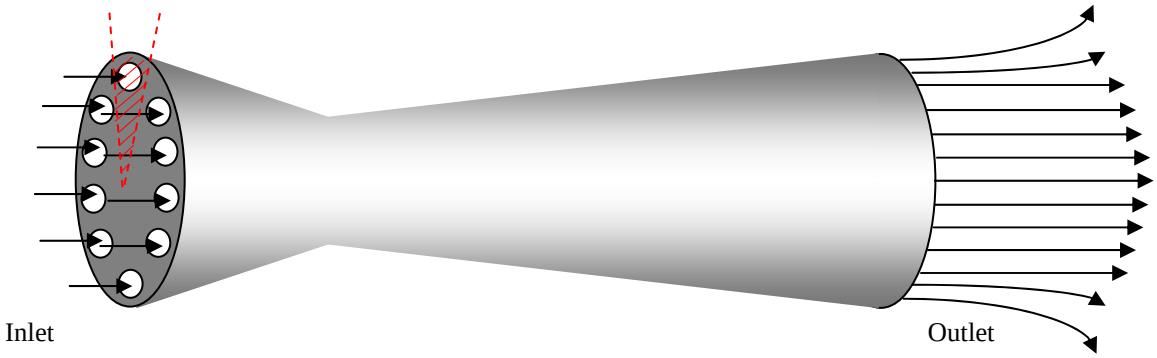


Figure 1. Schematic of the flow inside and outside of a converging-diverging nozzle

Choosing the appropriate boundary conditions can reduce the computer effort. In this example the slice shown in Figure 1 is repeated to produce the whole physical domain. Using the periodic boundary condition at the imaginary planes shown in Figure 2 can reduce the computational domain to a much smaller area. Figure 3 shows the other boundary conditions applied to the problem.

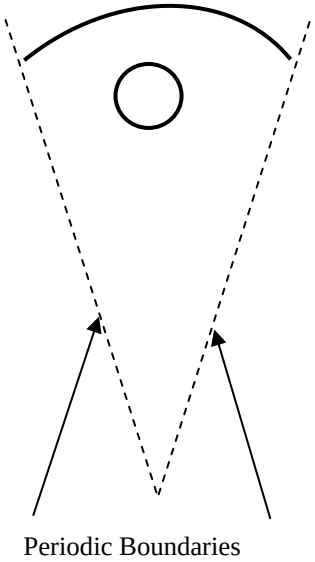


Figure 2. Minimizing the computational domain using periodic boundary condition

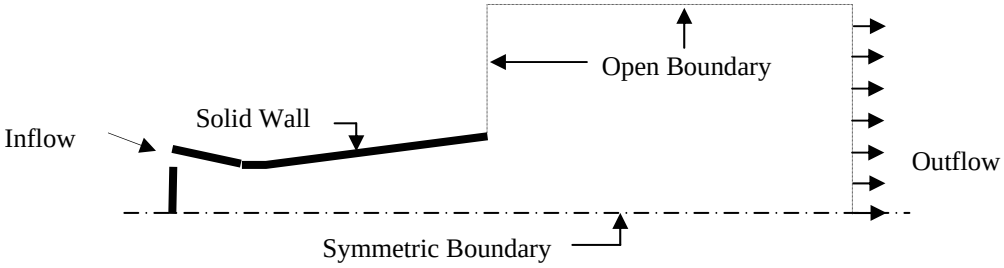


Figure 3. Various Boundary Conditions

3 Techniques for Numerical Discretization

In order to solve the governing equations of the fluid motion, first their numerical analogue must be generated. This is done by a process referred to as discretization. In the discretization process, each term within the partial differential equation describing the flow is written in such a manner that the computer can be programmed to calculate. There are various techniques for numerical discretization. Here we will introduce three of the most commonly used techniques, namely: (1) the finite difference method, (2) the finite element method and (3) the finite volume method. Spectral methods are also used in CFD, which will be briefly discussed.

3.1 *The Finite Difference Method*

Finite difference method utilizes the Taylor series expansion to write the derivatives of a variable as the differences between values of the variable at various points in space or time. Utilization of the Taylor series to discretize the derivative of dependent variable, e.g., velocity u, with respect to the independent variable, e.g., special coordinated x, is shown in Figure 4. Consider the curve in Fig. 4 which represent the variation of u with x, *i.e., u(x)*. After discretization, the curve *u(x)* can be represented by a set of discrete points, u_i's. These discrete points can be related to each other using a Taylor series expansion. Consider two points, ($i+1$) and ($i-1$), a small distance Δx from the central point, (i). Thus velocity u_i can be expressed in terms of Taylor series expansion about point (i) as:

$$u_{i+1} = u_i + \left(\frac{\partial u}{\partial x}\right)\Delta x + \left(\frac{\partial^2 u}{\partial x^2}\right)\frac{(\Delta x)^2}{2} + \left(\frac{\partial^3 u}{\partial x^3}\right)_i \frac{(\Delta x^3)}{6} + \ldots \tag{15}$$

and

$$u_{i-1} = u_i - \left(\frac{\partial u}{\partial x}\right)\Delta x + \left(\frac{\partial^2 u}{\partial x^2}\right)\frac{(\Delta x)^2}{2} - \left(\frac{\partial^3 u}{\partial x^3}\right)_i \frac{(\Delta x^3)}{6} + \ldots \tag{16}$$

These equations are mathematically exact if number of terms are infinite and Δx is small. Note that ignoring these terms leads to a source of error in the numerical calculations as the equation for the derivatives is truncated. This error is referred to as the truncation error. For the second order accurate expression, the truncation error is:

$$\sum_{n=3}^{\infty}\left(\frac{\partial^n u}{\partial x^n}\right)_i \frac{(\Delta x)^{n-1}}{n!}$$

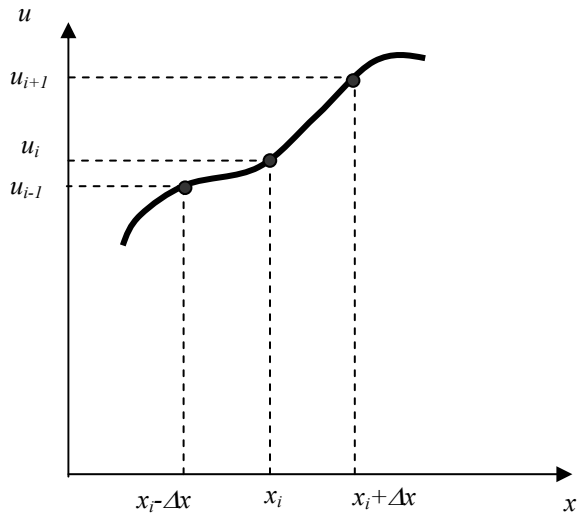


Figure 4. Location of points for Taylor series

By subtracting or adding these two equations, new equations can be found for the first and second derivatives at the central position *i*. These derivatives are

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{u_{i+1} - u_{i-1}}{2\Delta x} - \left(\frac{\partial^3 u}{\partial x^3}\right)_i \frac{(\Delta x)^2}{6} \tag{17}$$

and

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_i = \frac{u_{i-1} - 2u_i + u_{i+1}}{(\Delta x)^2} + O(\Delta x)^2 \tag{18}$$

Equations (17) and (18) are referred to as the central difference equations for the first and the second derivatives, respectively. Further derivatives can also be formed by considering equations (15) and (16) in isolation. Looking at equation (15), the first-order derivative can be formed as

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{u_{i+1} - u_i}{\Delta x} - \left(\frac{\partial^2 u}{\partial x^2}\right)_i \frac{(\Delta x)}{2} \tag{19}$$

This is referred to as the Forward difference. Similarly, from equation (16) another first-order derivative can be formed, i.e.,

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{u_i - u_{i-1}}{\Delta x} - \left(\frac{\partial^2 u}{\partial x^2}\right)_i \frac{(\Delta x)}{2} \tag{20}$$

This is referred to as the Backward difference. As noted by the expressions, difference formulae are classified in two ways: (1) by the geometrical relationship of the points, namely, central, forward, and backward differencing; or (2) by the accuracy of the expressions, for instance, central difference is second-order accurate, whereas, both forward and backward differences are first-order accurate, as the higher order terms are neglected.

We can obtain higher order approximations by applying the Taylor series expansion for more points. For instance, a 3-point cluster would result in a second order approximation for the forward and backward differencing, rather than a first order approximation:

Forward difference:

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{1}{2\Delta x}\left(-3u_i + 4u_{i+1} + u_{i+2}\right) + O\left(\Delta x\right)^2 \tag{21}$$

Backward difference:

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{1}{2\Delta x}\left(u_{i-2} - 4u_{i-1} + 3u_i\right) + O\left(\Delta x\right)^2 \tag{22}$$

Similarly a 4-point cluster results in a third order approximation for the forward and backward differencing:

Forward difference:

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{1}{6\Delta x}\left(-2u_{i-1} - 3u_i - 6u_{i+1} - u_{i+2}\right) + O\left(\Delta x\right)^3 \tag{23}$$

Backward difference:

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{1}{6\Delta x}\left(u_{i-2} + 6u_{i-1} + 3u_i + 2u_{i+1}\right) + O\left(\Delta x\right)^3 \tag{24}$$

The above difference equations are used to produce the numerical analogue of the partial differential equations describing the flow. In order to apply this discretization method to the whole flow field, many points are placed in the domain to be simulated. Then, at each of these points the derivatives of the flow variables are written in the difference form, relating the values of the variable at each point to its neighboring points. Once this process is applied to all the points in the domain, a set of equations are obtained which are solved numerically. For more discussion on this topic refer to text books on numerical analysis such as Hildebrand[11], and Chapra and Canale[12].

3.2 *The Finite Element Method*

In the finite element method, the fluid domain under consideration is divided into finite number of sub-domains, known as elements. A simple function is assumed for the variation of each variable inside each element. The summation of variation of the

variable in each element is used to describe the whole flow field. Consider the two-nodded element shown in Figure 5, in which variable u varies linearly inside the element. The end points of the element are called the nodes of the element. For a linear variation of u, the first derivative of u with respect to x is simply a constant. If u is assumed to vary linearly inside an element, we cannot define a second derivative for it. Since most fluid problems include second derivative, the following technique is designed to overcome this problem. First, the partial differential equation is multiplied by an unknown function, and then the whole equation can be integrated over the domain in which it applies. Finally the terms that need to have the order of their derivatives reduced are integrated by parts. This is known as producing a variational formulation.

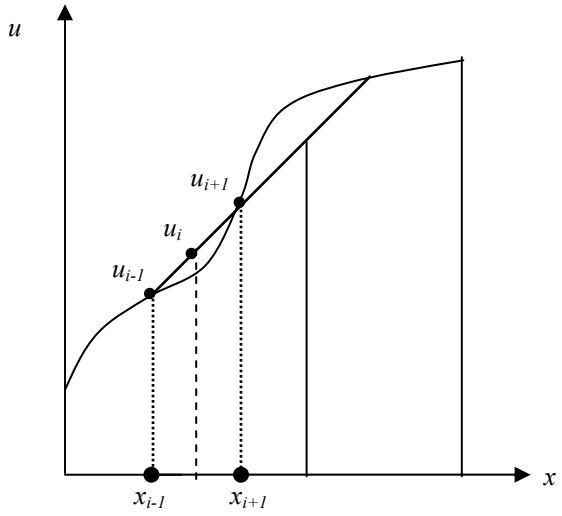


Figure 5. A two-noded linear element

As an example, we will develop the finite element formulation of the Laplace's Equation in one dimensions:

$$\frac{d^2u}{dx^2} = 0 \tag{25}$$

where velocity u is a function of the spatial coordinates x. We multiply equation (25) by some function W and integrate it over the domain of interest denoted by Ω:

$$\int W\left[\frac{d^2u}{dx^2}\right] d\Omega = 0 \tag{26}$$

Equation (26) can be integrated by parts to result in:

$$\int \left[-\frac{dW}{dx}\frac{du}{dx} \right] d\Omega + \int \left[W \frac{du}{dx} n_x \right] d\Gamma = 0 \tag{27}$$

where Γ denotes the boundary of the domain Ω and n_x is the unit outward normal vector to the boundary Γ. The second order derivative in equation (26) is now transformed into products of first order derivatives. Equation (27) is used to produce the discrete form of the partial differential equation for the elements in the domain. Equation (27) is known as the variational form of the partial differential equation (25). Although this technique reduces the order of the derivatives, it introduces the terms corresponding to the boundary of the domain into the governing equation (27).

We will now divide the domain into several elements and assume a function for the variation of the variable *u* in each element. If a two-noded linear element is assumes (see Fig. 5), the variation of *u* in each element can be represented by

$$u_i = u_{i-1} + (u_{i+1} - u_{i-1}) \left[\frac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}} \right] \tag{28}$$

or

$$u_i = u_{i-1} \left[\frac{x_{i+1} - x_i}{x_{i+1} - x_{i-1}} \right] + u_{i+1} \left[\frac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}} \right] \tag{29}$$

The terms in the brackets are called the shape functions and are denoted as N_i's. u_{i-1} and u_{i+1} are the nodal values of the variable *u* and are denoted as u_i's. Therefore, the variable *u* can be written in the following form

$$u_i = N_{i-1} u_{i-1} + N_{i+1} u_{i+1} \tag{30}$$

Thus, the shape functions corresponding to the two-nodal linear element, represented by equation (28) are

$$N_{i-1} = \frac{x_{i+1} - x_i}{x_{i+1} - x_{i-1}} \tag{31}$$

and

$$N_{i+1} = \frac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}} \tag{32}$$

We can now determine the derivatives of the variable *u*, using equation (30):

$$\frac{du}{dx} = \sum_{i=1}^{m} \frac{dN_i}{dx} u_i \tag{33}$$

where m is the number of nodes on the element. Note that u_i's are nodal values of u and they are not variables, therefore, they are not differentiated.

In order to solve equation (27) we still need to describe the function W. There are several methods, which are used for the specification of the variable W. However, the most common method is the Galerkin method in which W is assumed to be the same as the shape function for each element. Therefore, equation (27) is discretized by using equations similar to equation (33) for the derivatives of the variable, and equations similar to equations (31) and (32) for W. For every element there can be several equations depending on the number of the nodes in that element. The set of equations generated in this form are then solved together to find the solution.

The above formulation was based on a linear variation of the variable in each element. If higher order variations are used, e.g., quadratic or cubic, second derivatives will appear which require more points to describer them. This makes the computation more cumbersome. References [3] and [4] are recommended for more detailed discussion of FEM.

3.3 *The Finite Volume Method*

The finite volume method is currently the most popular method in CFD. The main reason is that it can resolve some of the difficulties that the other two methods have. Generally, the finite volume method is a special case of finite element, when the function W is equal to 1 everywhere in the domain. This technique is discussed in detail by Patankar.[34]

A typical finite volume, or cell, is shown in Fig. 6. In this figure the centroid of the volume, point P, is the reference point at which we want to discretize the partial differential equation.

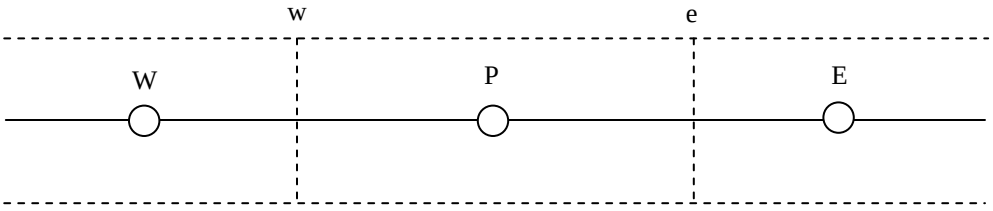


Figure 6. A finite volume in one dimension.

The neighboring volumes are denoted as, W, volume to the west side, and E, the volume to the east side of the volume P. For the one-dimensional finite volume shown in Fig. 6, the volume with centroid P, has two boundary faces at w and e.

The second derivative of a variable at P can be written as the difference between the 1st derivatives of the variable evaluated at the volume faces:

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_p = \frac{\left[\left(\frac{\partial u}{\partial x}\right)_e - \left(\frac{\partial u}{\partial x}\right)_w\right]}{x_e - x_w} \tag{34}$$

The first derivatives at the volume faces can be written as to be the differences in the values of the variable at the neighboring volume centroids:

$$\left[\frac{\partial u}{\partial x}\right]_e = \frac{u_E - u_P}{x_E - x_P} \tag{35}$$

and

$$\left[\frac{\partial u}{\partial x}\right]_w = \frac{u_P - u_W}{x_P - x_W} \tag{36}$$

We can apply this technique to equation (25) to obtain its finite volume formulation. The above method is also referred to as the Cell Centered (CC) Method, where the flow variables are allocated at the center of the computational cell. The CC variable arrangement is the most popular, since it leads to considerably simpler implementations than other arrangements. On the other hand, the CC arrangement is more susceptible to truncation errors, when the mesh departs from uniform rectangles.

Traditionally the finite volume methods have used regular grids for the efficiency of the computations. However, recently, irregular grids have become more popular for simulating flows in complex geometries. Obviously, the computational effort is more when irregular grids are used, since the algorithm should use a table to lookup the geometrical relationships between the volumes or element faces. This involves finding data from a disk store of the computer, which increases the computational time.

3.4 *Spectral Methods*

Another method of generating a numerical analog of a differential equation is by using Fourier series or series of Chebyshev polynomials to approximate the unknown functions. Such methods are called the Spectral method. Fourier series or series of Chebyshev polynomials are valid throughout the entire computational domain. This is the main difference between the spectral method and the FDM and FEM, in which the approximations are local. Once the unknowns are replaced with the truncated series, certain constraints are used to generate algebraic equations for the coefficients of the Fourier or Chebyshev series. Either weighted residual technique or a technique based on forcing the approximate function to coincide with the exact solution at several grid points

is used as the constraint. For a detailed discussion of this technique refer to Gottlieb and Orzag.[13]

3.5 *Comparison of the Discretization Techniques*

The main differences between the above three techniques include the followings. The finite difference method and the finite volume method both produce the numerical equations at a given point based on the values at neighboring points, whereas the finite element method produces equations for each element independently of all the other elements. It is only when the finite element equations are collected together and assembled into the global matrices that the interaction between elements is taken into account.

Both FDM and FVM can apply the fixed-value boundary conditions by inserting the values into the solution, but must modify the equations to take account of any derivative boundary conditions. However, the finite element method takes care of derivative boundary conditions when the element equations are formed and then the fixed values of variables must be applied to the global matrices.

One advantage that the finite element method has is that the programs are written to create matrices for each element, which are then assembled to form the global equations before the whole problem is solved. Finite volume and finite difference programs, on the other hand, are written to combine the setting up of the equations and their solution. The decoupling of these two phases, in finite element programs, allows the programmer to keep the organization of the program very clear and the addition of new element types is not a major problem. Adding new cell types to a finite volume program can, however, be a major task involving a rewrite of the program and so some finite volume programs can exhibit problems if they have multiple cell types. The differences between the three techniques become more pronounced once they are applied to two- and three-dimensional problems.

4 Solving The Fluid Dynamic Equations

As it was stated in section two, CFD provides the solution to the governing equations of the flow subject to a particular initial and boundary conditions. Equations (1) to (3) plus the equations of the state or the property relations are the general form of the governing equations. These equations are highly nonlinear and very difficult to solve even numerically. In applying these equations to a particular problem, some of the terms may disappear or be negligible which makes the solution much simpler. Various numerical techniques are developed for each of the particular application of the general flow equations and their simplified forms. In order to introduce various computational techniques we will first consider a simple form of the momentum equation, and then discretize various forms of that equation. The momentum equation (2) for a 1-dimensional, incompressible flow with no body force, and constant properties reduces to

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial^2 x} - \frac{1}{\rho}\frac{\partial p}{\partial x} \tag{37}$$

The first term in equation (37) is the transient term, the second is the convective term, the third is the diffusive term, and the fourth is the pressure term. We will consider various combinations of the terms in this equation and discuss the methods to solve them.

4.1 *Transient-Diffusive Terms*

Consider only the 1st and the 3rd terms in the above equation and, to further simplify, assume $\nu=1$:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{38}$$

This is the transient diffusion equation which consists of a first derivative in the time direction t and a second derivative in the space direction x. This is a parabolic partial differential equation that can be used to model the temporal changes in the diffusion of some quantity through a medium. For instance, the transient diffusion of heat (conduction) in a solid. We will solve this equation using both a finite difference and a finite element approach.

4.1.1 *Finite Difference Approach*

First we will describe the domain of the problem. Lets assume the diffusion occurs along a zone with thickness L. The time is usually started from $t=0$ and it is extended in the positive direction. Once we have identified the range of this domain, we place points throughout this domain. Figure 7 shows a simple method of placing points in the

domain. The spacings in the x and t directions can be the same or they may be different. Each point is labeled using i for special discretization and n for temporal discretization.

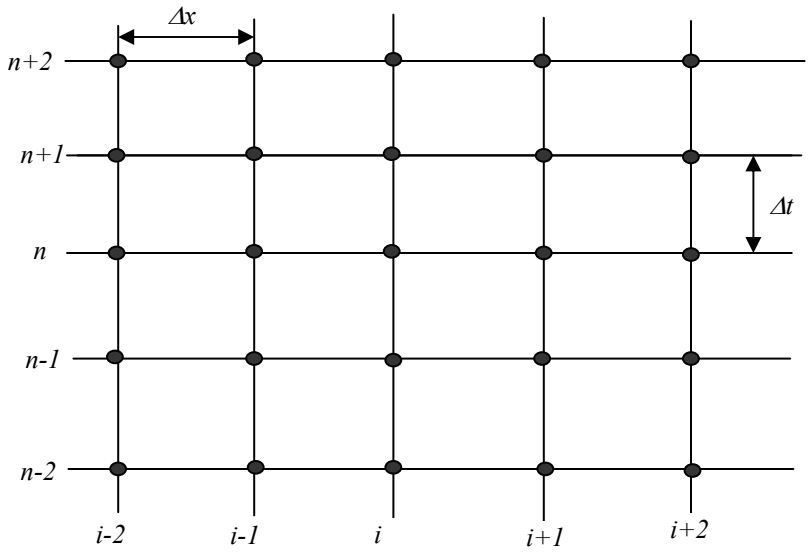


Figure 7. The discretized domain.

This procedure is referred to as the grid generation. Once the grid is generated one of the differencing scheme can be used to discretize the governing equation of the problem, equation (38). The type of differencing scheme used depends on the particular problem. It is mainly through testing that one may find the accuracy and efficiency of one scheme over another. One simple method to discretized the diffusion equation is to use a forward difference formula for the time derivative and a central difference formula for the spatial derivative. The discretized equation will then be

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} \tag{39}$$

This can be written in the following form:

$$u_i^{n+1} = \frac{\Delta t}{\Delta x^2} u_{i-1}^n + \left[1 - 2\frac{\Delta t}{\Delta x^2}\right] u_i^n + \frac{\Delta t}{\Delta x^2} u_{i+1}^n \tag{40}$$

Note that the velocity at position i and time $n+1$ depends on the three values at the time level n. Thus by knowing the values of u at time level n, its value at the next time level $n+1$ can be calculated. Therefore, to start the calculation, values of u in all the domain, e.g. all the x locations, should be known. These known values at $t=0$ are known as the initial conditions.

We can generate other differencing equations. For instance, the left hand side of equation (38) can be discretized based on the next time level $n+1$:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \left[\frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2} \right] \tag{41}$$

Equations (40) and (41) define an **explicit and implicit** form of equations, respectively. In equation (40), an unknown variable is directly related to a set of known variables. When a direct computation of the dependent variables can be made in terms of known quantities, the computation is said to be explicit. Some common explicit methods for parabolic partial differential equations (e.g., equation 38) are:

(1) **The Forward Time/Central Space (FTCS) method** which is represented by equation (39) and it is stable for $\Delta t/\Delta x \leq 1/2$.

(2) **The Richardson method**[14], where central difference is used for both time and space and it is unconditionally unstable with no practical value:

$$\frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t} = +\left[\frac{u_{i-1}^{n} - 2u_i^{n} + u_{i+1}^{n}}{\Delta x^2} \right]. \tag{42}$$

(3) **The DuFort-Frankel Method**[15], which also uses central difference for both time and space, but u_i^n in the diffusion term is replace by $(u_i^{n+1} + u_i^{n-1})/2$. This modification makes the difference equations unconditionally stable.

$$\frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t} = \left[\frac{u_{i-1}^{n} - u_i^{n+1} - u_i^{n-1} + u_{i+1}^{n}}{\Delta x^2} \right]. \tag{43}$$

The truncation error for DuFort-Frankel method is order of $O[(\Delta t)^2, (\Delta x)^2, (\Delta t/\Delta x)^2]$. In equation (43) the only unknown variable is u_i^{n+1}, therefore, it is explicit.

In equation (41), several unknown variables are related to the several known variables. This is referred to as an implicit equation. When the dependent variables are defined by coupled sets of equations, and either a matrix or iterative technique is needed to obtain the solution, the numerical method is said to be implicit. Some common implicit methods for parabolic partial differential equations are:

(1) **The Laasonen method**[16], which is the same as equation (41):

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \left[\frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2} \right].$$

This scheme has first-order accuracy with a truncation error of $O[\Delta t, (\Delta x)^2]$ and is unconditionally stable.

(2) The Crank-Nicolson method[17] , which is formed by averaging the present and the next time differences, *i.e.,* average of equations (39) and (41):

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left[\frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2}\right] + \frac{1}{2}\left[\frac{u_{i-1}^{n} - 2u_i^{n} + u_{i+1}^{n}}{\Delta x^2}\right] \tag{44}$$

(3) The General Formulation, which is obtained by a weighted average of the spatial derivatives at two time levels n and $n+1$:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha\left[\frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2}\right] + (1-\alpha)\left[\frac{u_{i-1}^{n} - 2u_i^{n} + u_{i+1}^{n}}{\Delta x^2}\right] \tag{45}$$

where α and $(1 - \alpha)$ are used to weight the derivatives.

In an explicit scheme, once we know both the initial conditions and the boundary conditions, we can calculate the values of the variables at the internal points. Using the known values at the first row of points, the values at the next row are found. Then the boundary conditions are applied to get the values at the boundary points. This gives us a second complete row of points where we know all the values of the variable. These can be used as a new set of initial conditions and so the process can be repeated to give the next row.

In an implicit scheme in order to calculate both fixed-value boundary conditions and derivative boundary conditions extra equations are added to those already generated from the partial differential equation. With these extra equations the number of equations should match the number of unknowns and so the full set of simultaneous equations can be solved.

One final comment should be made about the differencing equations mentioned above, namely, the spacings between the points are assumed to be the same. However, one can develop a set of difference equations based on variable spacings. In addition, the difference equations developed here is based on a line of points, which is a characteristics of a Cartesian Coordinates. However, other coordinates can also be used. The finite difference method requires, however, that the grid of points is topologically regular. This means that the grid must look cuboid in a topological sense. If distributions of points with a regular topology are used, then the calculation procedure carried out by a computer program is efficient and very fast.

4.1.2 *Finite Element Approach*

We will derive the finite element formulation of equation (38). It is easier to use a difference equation for the time derivative. Therefore, similar to the previous case, if a forward difference for the time derivative is used, equation (38) can be written as

$$\frac{u^{n+1}-u^{n}}{\Delta t}=\frac{\partial^{2}u}{\partial x^{2}} \tag{46}$$

Variational form of equation (46) is produced by first multiplying it by a function W and integrating it over the whole domain:

$$\int W\left[\frac{u^{n+1}-u^{n}}{\Delta t}\right]d\Omega=\int W\left[\frac{\partial^{2}u}{\partial x^{2}}\right]d\Omega \tag{47}$$

We now integrate the second derivative on the right hand side by parts to obtain:

$$\int W\left[\frac{u^{n+1}-u^{n}}{\Delta t}\right]d\Omega=\int\left[-\frac{\partial W}{\partial x}\frac{\partial u}{\partial x}\right]d\Omega+\int\left[W\frac{\partial u}{\partial x}n_{x}\right]d\Gamma \tag{48}$$

Note that the continuity requirement for u is reduced from second to first derivatives, therefore, we say it is weakened. We will now divide the domain into a series of linear elements and use the Galerkin[3] method to derive the finite element formulation. On each element the variation of u is described by:

$$u=\sum_{i=1}^{m}N_{i}u_{i} \tag{49}$$

where m is the number of nodes on the element and the N_i terms are the shape functions. After substituting for the multiplier W, for the values of u at the two time levels and for the spatial derivatives of u at the *n'th* time level, an explicit form of equation (49) is obtained:

$$\int N_{i}\left[\frac{N_{j}u_{j}^{n+1}-N_{j}u_{j}^{n}}{\Delta t}\right]d\Omega=\int\left[-\frac{\partial N_{i}}{\partial x}\frac{\partial N_{j}u_{j}^{n}}{\delta x}\right]d\Omega+\int\left[W\frac{\partial u_{j}^{n}}{\partial x}n_{x}\right]d\Gamma \tag{50}$$

where the *i,j* indices refer to the summation. In many problems the boundary term is not discretized. Usually, this so-called flux can be taken to be a known value that needs to be

added later. On the faces of most elements the flux term is ignored, as we assume that the fluxes cancel out across those faces that are internal to the domain. This is an equilibrium condition. It is only on the boundaries of the domain that the flux terms need to be added. If the fluxes are not added, they will be calculated by the method as being zero, and because of this they are known as natural boundary conditions. If we specify the value of u at a boundary then the flux term is not required, just as with the finite difference method, and this is known as an essential boundary condition.

For simple elements the shape functions N_i are simple functions of the coordinates, say x, and so equation (50) can be integrated exactly over each element, but for more complex elements this integration has to be performed numerically. If we use simple one-dimensional elements that have two nodes, then the above equation can be integrated to yield two separate equations for each element in terms of the nodal values of u at the *n+1'th* time level, if the values at time level n are known. This equation can be expressed as a matrix equation as follows:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \tag{51}$$

where the terms a_{ij} are functions of position derived from the integration of the first term on the left hand side of equation (50), and the terms f_i come from all the other terms in equation (50). This matrix equation is, in fact, part of a larger matrix equation for all the unknown values of u. Once all the equations for each element, the so-called element equations, are known then the full set of equations for the whole problem has to be produced. This is shown in Fig. 8 where two elements are shown. An expanded version of the element equations can be formed by relating the local node on an element to its global node number. For example, on element 2 the local node numbered 1 is global node number 2. Combining the two expanded equations produces a global matrix equation, and the process of combination is known as assembling the equations. This is done by adding all the element equations together as follows:

$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ 0 \end{pmatrix} \tag{52}$$

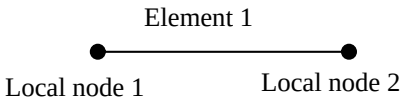
$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & b_{11} & b_{12} \\ 0 & b_{21} & b_{22} \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \end{pmatrix} = \begin{pmatrix} 0 \\ g_1 \\ g_2 \end{pmatrix} \tag{53}$$

This gives:

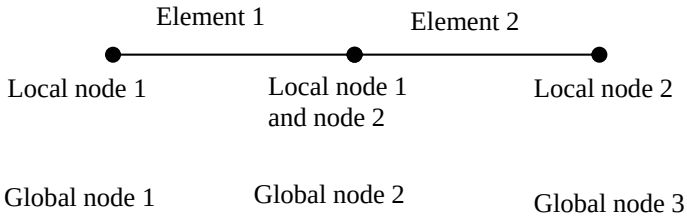
$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22}+b_{11} & b_{12} \\ 0 & b_{21} & b_{22} \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2+g_1 \\ g_2 \end{pmatrix} \tag{54}$$

where terms f_i and g_i come from the terms on the right side of equation (50). The matrix on the left hand side is called the stiffness matrix and the matrix on the right hand side is called the load vector.

Once these global matrices have been created, the fixed value boundary conditions are imposed on the matrices and the equations can be solved. Again the solution of the original partial differential equation (38) has been reduced to the solution of a set of simultaneous equations. Finite elements produce the numerical equations for each element from data at known points on the element and nowhere else. Consequently, there is no restriction on how the elements are connected so long as the faces of neighboring elements are aligned correctly. By this we mean that the faces between elements should have the same nodes for each of the adjoining elements. This flexibility of element placement allows a group of elements to model very complex geometry.



(a) Single Element



(b) Two Elements

Figure 8. Numbering of Two-Nodded Linear Elements

4.2 *Transient-Convective Terms*

By considering the 1st and the 2nd terms in equation (37) we obtain the main parts of the equation representing the inviscid flows. If we further assume that the velocity u in the convection term is a constant (we differ the discussion of the nonlinear terms to the next section), we obtain the wave equation. Thus, the first order wave equation becomes:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{55}$$

where c is the wave speed propagating in the x-direction. For an initial condition given by

$$u(x,0) = f(x) \tag{56}$$

where $f(x)$ is monotonic in x, the exact solution for a wave of constant shape is

$$u = f(x - ct) \tag{57}$$

(1) Euler Explicit Method: An explicit differencing of equation (55) results in the following formulation:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_{i+1}^n - u_i^n}{\Delta x} = 0 \tag{58}$$

This is an explicit equation since only one unknown, u_i^{n+1} , appears in the equation. This method is refereed to as Euler Explicit Method and, unfortunately, it is unconditionally unstable and will not work for solving the wave equation. This method is first-order since the lowest-order term in the truncation error is first order, i.e., Δt, and Δx.

(2) First-Order Upwind Method: The Euler method can be made stable by using a backward difference instead of a forward difference for a positive wave speed[1]:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 \tag{59}$$

This method is stable for $0 \le c\Delta t/\Delta x \le 1$, where $c\Delta t/\Delta x$ is referred to as the CFL (**Courant-Friedrichs-Lewy**) number. This method is referred to as the First-Order Upwind Method.

For discretized transport problems, the CFL number determines how many mesh cells, a fluid element passes during a timestep. For compressible flow, the definition is different. Here, the CFL number determines how many cells are passed by a propagating

[1] For a negative wave speed, forward difference must be used.

perturbation. Hence, the wave-speed, i.e., fluid speed plus the sound speed, is employed. For explicit time-stepping schemes, such as Runge-Kutta, the CFL number must be less than the stability limit for the actual scheme to converge. For implicit and semi-implicit schemes, the CFL limit does not constitute a stability limit. On the other hand, the range of parameters in which these schemes converge may often be characterized by the CFL number.

(3) Lax Method: Another method of making the Euler equation stable is by using an average value for u_i^n based on the two neighboring points:

$$\frac{u_i^{n+1} - (u_{i+1}^n + u_{i-1}^n)/2}{\Delta t} + c\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} = 0 \tag{60}$$

This is referred to the Lax Method[18] which is stable for CFL ≤ 1.

(4) Euler Implict Methods are another way of solving Euler equation:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} = 0 \tag{61}$$

These methods are unconditionally stable for all time steps, however, a system of equations must be solved for each time level.

The above methods are all first-order accurate. More accurate second-order methods are developed to solve the PDEs describing the flow. The commonly used methods are:

(5) Leap Frog Method

$$\frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t} + c\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} = 0 \tag{62}$$

(6) Lax-Wendroff Method[19]

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} = c^2\frac{\Delta t}{2(\Delta x)^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) \tag{63}$$

(7) MacCormack Method[20]

This is an explicit, predictor-corrector method which is written in the following form.

Predictor: $$(u_i^{n+1})^* = u_i^n - c\frac{\Delta t}{\Delta x}(u_{i+1}^n - u_i^n) \tag{64}$$

Corrector: $$u_i^{n+1} = \frac{1}{2}\left\{u_i^n + (u_i^{n+1})^* - c\frac{\Delta t}{\Delta x}\left[(u_i^{n+1})^* - (u_{i-1}^{n+1})^*\right]\right\} \tag{65}$$

Here, $(u_i^{n+1})^*$ is the predicted value for u at point i and time level $n+1$. The forward and backward differencing used in the above equations can be changed depending on the particular problem.

(8) Second-Order Upwind Method

This is a modification of the MacCormack method where upwind (backward) differences are used in both predictor and corrector.

Predictor: $$(u_i^{n+1})^* = u_i^n - c\frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) \tag{66}$$

Corrector: $$u_i^{n+1} = \frac{1}{2}\left\{u_i^n + (u_i^{n+1})^* - c\frac{\Delta t}{\Delta x}\left[(u_i^{n+1})^* - (u_{i-1}^{n+1})^*\right] - c\frac{\Delta t}{\Delta x}\left(u_i^n - 2u_{i-1}^n + u_{i-2}^n\right)\right\} \tag{67}$$

The fluid dynamics of inviscid flows are governed by Euler equations. These equations may have different character for various flow regimes. For time-dependent flows, the equations are hyperbolic for all Mach numbers. Therefore, a time-marching method can be used to obtain the solution. In steady inviscid flows, the Euler equations are elliptic for subsonic conditions, and hyperbolic for supersonic conditions. Several simplified form of the Euler equations are used for inviscid flows. For instance, if the flow is incompressible, by consider the flow is irrotational as well; a solution to the Laplace's equation for the velocity potential or stream function can describe the flow field. The traditional method of solving hyperbolic PDEs are by the method of characteristics. Alternatively, there are numerous FDM based solution schemes for such flows.

4.3 *Shock Capturing Methods*

For flows with shocks, several shock-capturing techniques are developed. Gudonov[21] schemes have been particularly efficient for shock problems. Godunov supposed that the initial data could be replaced by a set of piecewise constant data with discontinuities and used exact solutions of Riemann problems to advance the piecewise constant data. One of the key points in Godunov schemes is to calculate the flux at each interface of numerical cells through a Riemann problem. A major extension to the Godunov's scheme was made by Van Leer[22,23] in his MUSCL scheme (Monotone Upstream-centered Scheme for Conservation Laws) which used a Riemann solver to advance piecewise linear data. Other examples of Godunov schemes include Roe's method[24,25], the piecewise parabolic method (PPM)[26], the TVD (Total Variation Diminishing) methods[27].

Godunov schemes for hydrodynamical equations may be second-order accurate in time, but they are explicit. The time step in an explicit scheme is restricted by the largest CFL number, which may not be larger than unity for a stable calculation. The stability limit in

an explicit scheme is imposed by the local conditions in the regions, where wave speeds are high, regardless of the significance of spatial variations prevailing in the problems. The regions drastically reduce the time step possible from explicit schemes. Implicit schemes for hydrodynamical equations are favored over their explicit counterparts for some problems, in which the time-step size necessary for procuring a required temporal accuracy may be significantly larger than that dictated by the explicit stability condition. Implicit–explicit hybrid schemes are useful when a flow attains different wave speeds either in different regions or at different instants, and the time accuracy is important in some parts of simulation domains.

Hybrid implicit-explicit schemes have also been developed, which use a combination of both schemes. The difference approximation in time is either implicit or explicit, separately for each family of characteristics and for each cell in the finite difference grid, depending on whether the local CFL number for that family is greater than or less than one. To the extent possible, the hybridization is continuous at CFL number equal to one, and the scheme for the explicit modes is a second-order Godunov method of a type discussed by Colella.[28,29]

4.4 *Convective-Diffusive Terms*

Consider the convective term (2nd term) and the diffusive term (3rd term) in equation (37).

$$u\frac{\partial u}{\partial x} = v\frac{\partial^2 u}{\partial x^2} \tag{68}$$

The first term contains a nonlinearity due to the convective term. These factors increase the complexity of the solution. The nonlinearity of the equations will make the iteration procedure very complex. Therefore, the equations are linearized at each time step. Linearization is achieved by using the current value of the velocity at a point or in a volume or element as the velocity multiplier. For example, the convective term (1st term) can be written as

$$\bar{u}\left[\frac{u_{i+1} - u_{i-1}}{2\Delta x}\right] \tag{69}$$

where a central difference is used for the derivative and $\bar{u}$ is found from the current solution for u: $\bar{u} = u_i^n$. This linearization technique is conducted on all the nonlinear terms in the equations before solving the set of simultaneous equations. The solution procedure for this type of equations is shown in Figure 9. As shown in this figure, there are several levels of iteration before a solution is obtained.

One other problem that needs to be addressed is that of producing numerical forms of the convection operator. Problems occur when this operator is discretized using central differences for the first derivative of the velocity. For example, take the linearized form of the equation (68):

$$\overline{u}\frac{\partial u}{\partial x} = v\frac{\partial^2 u}{\partial x^2} \tag{70}$$

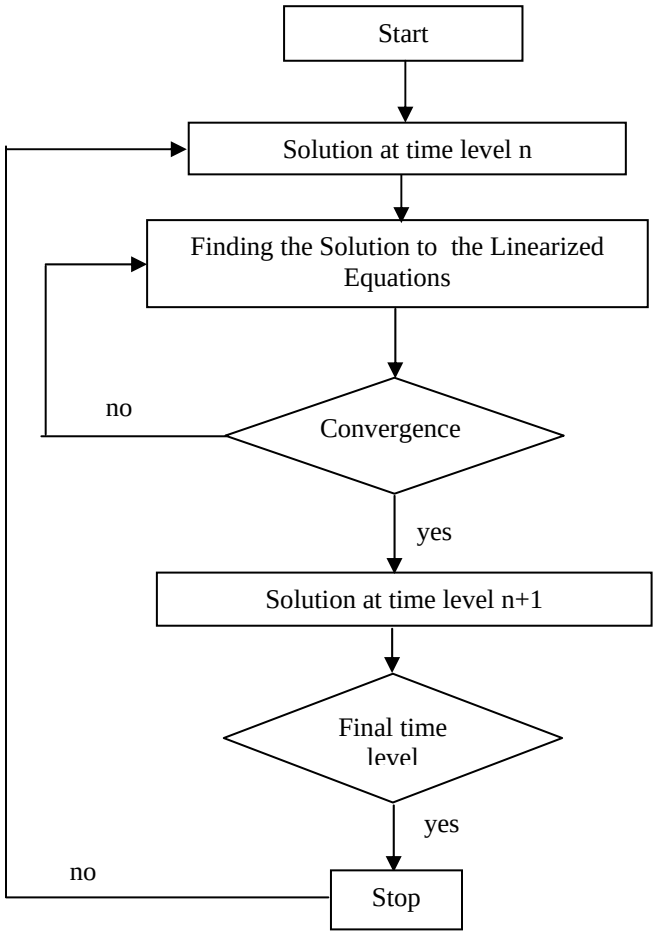


Figure 9. Solution procedure for a nonlinear set of equations.

Using central differences for both the first and second derivatives in this equation gives

$$\overline{u}\left[\frac{u_{i+1,j} - u_{i-1j}}{2\Delta x}\right] = v\left[\frac{u_{i+1j} - 2u_{ij} + u_{i-1j}}{(\Delta x)^2}\right] \tag{71}$$

which can be rearranged to give

$$u_{i,j} = \frac{1}{2} u_{i-1,j} \left[1 - \frac{\mathrm{Re}}{2} \right] + \frac{1}{2} u_{i-1,j} \left[1 + \frac{\mathrm{Re}}{2} \right] \tag{72}$$

where Re is the **Cell Reynolds number**, given by

$$\mathrm{Re} = \frac{\overline{u} \Delta x}{v} \tag{73}$$

The value of the cell Reynolds number has an important effect on the numerical equation. When the Reynolds number is less than two both terms on the right hand side have positive coefficients but when the Reynolds number is greater than two the first term on the right hand side becomes negative. This negative term result in very poor results. Therefore, a restriction is put on the cell Reynolds number. In a two-dimensional problem, each mesh cell has one cell Reynolds number for each of its directions, defined by the cell dimension and the flow speed in that direction.

One way around this limitation is to use a first-order accurate difference equation to model the first derivative in equation (70) instead of the second-order accurate difference equation used above. However, the reduction in accuracy can lead to a poor solution. Typically the use of lower-order accuracy schemes gives results, which are the results for a flow which has more viscosity than the one we are trying to model. Such schemes are in common use together with more accurate schemes. A good review of this topic is given by Abbott and Basco[30]. The following options for the discretization of the convection operator.

(1) Upwind Schemes:

In an upwind (UW) scheme the convection term is formed using a first-order accurate difference equation equating the velocity derivative to the values at the reference point and its nearest neighbor taken in the upstream direction. This can give very inaccurate solutions but they are easy to obtain as they converge readily. For compressible flows, UW is viewed in a different light. Here, instead of the primitive variables, a set of characteristic variables is often used. The governing equations for the characteristic variables are locally hyperbolic. Hence, their solutions are wavelike and upwind differences are the correct treatment. UW here appears under designations such as flux splitting, flux difference splitting, fluctuation splitting etc.

(2) Hybrid Schemes:

A hybrid scheme, where the upwind scheme is used if the Reynolds number is greater than two, and central differences are used if the Reynolds number is two or less. This is more accurate than the upwind scheme but does not converge on some grids of points.

(3) QUICK Upwind Schemes:

The quadratic upstream interpolation for convective kinetics (QUICK) scheme[31] is a quadratic upwind scheme used mainly in the finite volume formulation and is more accurate than the two schemes described above. This scheme uses a three-point upstream-weighted quadratic interpolation for cell face values. In the QUICK scheme, one adds one point in each direction and calculates the derivative using the cubic polynomial drawn through the four involved points. Local truncation error analysis shows third order accuracy. The QUICK scheme is unconditionally bounded up to cell Reynold numbers of 5. Beyond this limit, it may become unbounded. The QUICK scheme is normally applied as a correction to the donor cell scheme. In situations with unboundedness, the correction may locally be limited, thus reverting to the donor cell scheme. The QUICK scheme has a somewhat different form in finite volume contexts, since here the differences rather than the derivatives are of interest.

(4) **Power-Law Schemes:** Power-law schemes are derivatives of QUICK but are more accurate.

4.5 *Incompressible Navier-Stokes Equations*

When considering all the terms in equation (37) a special difficulty arises due to the weak coupling of the velocity and pressure fields. For the incompressible fluids, the continuity equation is only function of velocity and not a function of pressure. Only the momentum equations contain pressure terms. Since most of the terms in the momentum equations are functions of the velocity components it is natural to use these equations to produce the solutions for the velocity components. Then, the problem is how to obtain the pressure solution, since continuity does not contain pressure. A direct method is to discretize all the equations, i.e., continuity and momentum, and solve them simultaneously. This results in a very large solution vector that contains all variables and consequently very large computational effort. There are two commonly used methods to resolve this problem: (1) pressure-based methods, and (2) methods based on the concept of artificial compressibility (also known as pseudo-compressibility).

4.5.1 *Pressure-Based Methods*

In the pressure-based method (PBM),[32,33,34] (also known as pressure correction, uncoupled, or segregated methods) a Poisson equation for pressure corrections is formulated, and then it is updated for the pressure and velocity fields until a divergence-free velocity field is obtained. There are numerous variety of this method, some of the more popular ones are the marker-and-cell (MAC) method[35], SIMPLE and SIMPLER methods[34], the fractional-step method[36], and the pressure-implicit with splitting of operators (PISO) method[37,38].

Here we will only discuss the SIMPLE (Semi-Implicit Pressure Linked Equations) algorithm which is one of the most common algorithms for the incompressible flow calculations. This method is based on first guessing and then correcting the flow variables in an iterative manner to obtain the solution. The velocity components are first calculated from the momentum equations using a guessed pressure field. The pressure and velocities are then corrected in order to satisfy the continuity. The procedure is repeated until convergence is achieved. (PISO method is somewhat similar to SIMPLE method, except that it involves one predictor step and two corrector steps.)

For instance, in a two-dimensional problem, the momentum equation in the *x*-direction is solved for *u* velocity component (i.e., velocity in the *x*-direction) and the momentum equation in *y*-direction is solved for *v* velocity component using the lagged pressure terms. Therefore, the velocity components are first obtained without using the continuity equation. The velocity components determined this way will not satisfy the continuity equation initially. However, a modified form of the continuity equation is developed which is used to solve for the pressure equation and it is iterated until the velocity components converge to values which satisfy the continuity equation. In this method the velocity and pressure are written in the following form:

$$\begin{aligned} u &= u^* + u' \\ v &= v^* + v' \\ p &= p^* + p' \end{aligned} \tag{74}$$

where *u, v,* and *p* are the actual velocity components and pressure, u^*, v^*, and p^* are the guessed or the intermediate values, and *u', v'*, and *p'* are the corrections for the velocity components and pressure. After substitution in the continuity equation:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{75}$$

we obtain

$$\frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y} = -\frac{\partial u^*}{\partial x} - \frac{\partial v^*}{\partial y} \tag{76}$$

In this equation the derivatives of the correction velocity components depend on the derivatives of the velocity components that satisfy the momentum equations. An approximate form of the momentum equation (6) is used to relate the pressure correction to the velocity corrections. In order to obtain this approximate form, we start with the momentum equations:

$$Au_j = Bp_j \tag{77}$$

and

$$Cv_j = Dp_j \tag{78}$$

where ***A***, ***B, C*** and ***D*** are matrices, and u_j, v_j and p_j are vectors of the variables at grid points or nodes. These equations can be rewritten if the variables are split using equation (74), to give

$$Au_j^* + Au_j' = Bp_j^* + Bp_j' \tag{79}$$

and

$$Cv_j^* + Cv_j' = Dp_j^* + Dp_j' \tag{80}$$

Since in each step of calculation we are solving for the estimated values or the intermediate values, then

$$Au_j^* = Bp_j^*$$

and

$$Cv_j^* = Dp_j^*$$

Therefore, these terms can be subtracted from the matrix equations (79) and (80) giving

$$Au_j' = Bp_j'$$

and

$$Cv_j' = Dp_j'$$

These are the approximate forms of the momentum equation and can be written as

$$u_j' = A^{-1}Bp_j' \tag{81}$$

and

$$v_j' = C^{-1}Dp_j' \tag{82}$$

Using these two forms of the equations we can find the pressure from the continuity equation. This is done by substituting them into the modified continuity equation (76), to produce an equation for the correction pressure p_j' which has on its right hand side the imbalance in the continuity of the flow after the momentum equations have been solved. Once the correction pressure p' has been found, so u' and v' can be formed using equations (81) and (82). Finally equations (74) are used to find the corrected velocity components and pressure. At this stage in the solution the velocity components satisfy the continuity equation and a new value of pressure has been calculated, but the velocity components do not satisfy the momentum equations. To resolve both the solution of the momentum equations and the non-linearity, the momentum equations are used again to produce further simultaneous equations, which are solved, followed by the calculation of the correction pressure and the correction velocities.

In the momentum equation (6) the pressure variable appear in a first-order spatial derivative. The conversion of these derivatives to numerical form can lead to problems, as the use of central differences can produce values for the pressure variable at a given point which are not related to the pressure variables at neighbouring points. This, in turn, can lead to a pressure solution oscillating in what is known as a chequerboard pattern. Two different grid arrangements have been used to overcome this problem: (i) staggered grids[39] with different control volumes for velocities and pressure and (ii) collocated grids[40] with the same control volume for all variables. In the staggered grids, effectively, the pressure is stored at the centroid of a volume and the velocity components are stored at the volume faces[34]. However, the use of staggered grids introduces significant complexities in code development, increases the number of storage allocations, and requires intense interpolations. More recently several programs have turned to storing all the variables at volume centroids using the transformation of Rhie and Chow[40] to prevent chequerboarding. These collocated grids are becoming more popular.[41,42,43]

5 Basic Solution Techniques

In the implicit set of equations each equation has several unknowns, therefore, they should be solved simultaneously. There are many different methods for solving such a set of equations. Here, we will describe only the general procedure for solving a set of equations simultaneously.

5.1 *Direct Method*

Consider the matrix equation

$$\boldsymbol{Au=b} \tag{83}$$

where vector $\boldsymbol{u}$ represents the unknown variables, $\boldsymbol{A}$ is an operator on the vector of variables $\boldsymbol{u}$, and $\boldsymbol{b}$ is a vector of known values. The solution to the above equation is written as follows:

$$\boldsymbol{u=A^{-1}b} \tag{84}$$

where $\boldsymbol{A^{-1}}$ is the inverse of the matrix $\boldsymbol{A}$. The general method for determining the inverse of matrix $\boldsymbol{A}$ is referred to as $\boldsymbol{LU}$ decomposition[3]. In this method the matrix $\boldsymbol{A}$ is described by two other matrices as follows:

$$\boldsymbol{A=LU} \tag{85}$$

where $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{U}$ is an upper triangular matrix. The inverse can be easily found once matrix $\boldsymbol{A}$ is decomposed into $\boldsymbol{L}$ and $\boldsymbol{U}$. This is referred to as the direct method. Direct methods are commonly used in the finite element methods. However, the problem associated with the direct methods is that it requires significant amount of computational times for large matrices. Many iterative methods are developed to resolve this problem and reduce the computational effort.

5.2 *Iterative Methods*

As the name suggests, iterative methods obtain the solution by iteratively guessing the solution until the correct one is found. In addition, in computational fluid dynamics, the governing equations are nonlinear and the number of unknown variables is typically very large. Under these conditions implicitly formulated equations are almost always solved using iterative techniques. Iterations are used to advance a solution through a sequence of steps from a starting state to a final, converged state. This is true whether the solution sought is either one step in a transient problem or a final steady-state result. In either case, the iteration steps resemble a time-like process. Of course, the iteration steps

usually do not correspond to a realistic time-dependent behavior. In fact, it is this aspect of an implicit method that makes it attractive for steady-state computations, because the number of iterations required for a solution is often much smaller than the number of time steps needed for an accurate transient that asymptotically approaches steady conditions.

Various iterative schemes are designed and used in the numerical analysis.[3,4] We will introduce the more commonly used ones in CFD applications. Consider a system of three equations as:

$$\begin{aligned} a_{11}u_1 + a_{12}u_2 + a_{13}u_3 &= b_1 \\ a_{21}u_1 + a_{22}u_2 + a_{23}u_3 &= b_2 \\ a_{31}u_1 + a_{32}u_2 + a_{33}u_3 &= b_3 \end{aligned} \tag{86}$$

5.2.1 *Jacobi and Gauss-Seidel methods.*

In these two methods equations (86) are rewritten as:

$$\begin{aligned} u_1 &= \frac{1}{a_{11}}\left[b_1 - a_{12}u_2 - a_{13}u_3\right] \\ u_2 &= \frac{1}{a_{22}}\left[b_2 - a_{21}u_1 - a_{23}u_3\right] \\ u_3 &= \frac{1}{a_{33}}\left[b_3 - a_{31}u_1 - a_{32}u_2\right] \end{aligned} \tag{87}$$

Note that this method can only work if in equation (87) the diagonal terms of matrix ***A***, i.e., the terms a_{ii}, are not zero. The Jacobi method takes the right hand side of equation (87) to be the known values at the *k'th* iteration and the left hand side to be the new values at the *k+1'th* iteration:

$$\begin{aligned} u_1^{\,k+1} &= \frac{1}{a_{11}}\left[b_1 - a_{12}u_2^{\,k} - a_{13}u_3^{\,k}\right] \\ u_2^{\,k+1} &= \frac{1}{a_{22}}\left[b_2 - a_{21}u_1^{\,k} - a_{23}u_3^{\,k}\right] \\ u_3^{\,k+1} &= \frac{1}{a_{33}}\left[b_3 - a_{31}u_1^{\,k} - a_{32}u_2^{\,k}\right] \end{aligned} \tag{88}$$

The Gauss-Seidel method uses the new values at the *k+1'th* iteration on the right hand side of the equations giving:

$$
\begin{aligned}
u_1^{k+1} &= \frac{1}{a_{11}}\left[b_1 - a_{12}u_2^{k} - a_{13}u_3^{k}\right] \\
u_2^{k+1} &= \frac{1}{a_{22}}\left[b_2 - a_{21}u_1^{k+1} - a_{23}u_3^{k}\right] \\
u_3^{k+1} &= \frac{1}{a_{33}}\left[b_3 - a_{31}u_1^{k+1} - a_{32}u_2^{k+1}\right]
\end{aligned}
\tag{89}
$$

Both of these methods require that an initial guess to the solution be made which can then be used during the first iteration. Then the numerical equations are used to produce a more accurate approximation to the numerically correct solution, which is one in which all the variables satisfy the governing equations. This new approximation, the updated solution, is then used as the new starting solution and the process is repeated until the error in the solution is sufficiently small. Each repetition of the solution process is known as an iteration.

Sometimes during an iterative process the updated solution at the end of one iteration can be very different from the solution at the start of the iteration. If we consider Fig. 10 we can see a graph of velocity against time. Let us imagine that we have a numerical scheme that predicts the velocity u^{k+1} at some time Δt ahead of the current time by using values of the current acceleration a^k and the current velocity u^k in the following way:

$$\frac{u^{k+1} - u^k}{\Delta t} = a^k \tag{90}$$

or

$$u^{k+1} = u^k + a^k \Delta t \tag{91}$$

This is a first-order method in time. If we know both the acceleration, a^k, and velocity, u^k, then we can predict the new velocity, and so given the new acceleration and velocity we can march forward in time finding the velocity-time relationship. Looking at the figure we can see the actual velocity-time relationship and two approximations bases on the above equations. In both of these the initial acceleration is used to predict the velocity. It is clear from this that if the time interval is small, say Δt_1, then the error ε_1 between the predicted velocity and the actual velocity is small, but if the time interval Δt_2 is large then the error ε_2 is large. Similar errors can occur when carrying out a CFD simulation and if the error gets ever larger during the solution we will have a very inaccurate flow solution and convergence of the solution will not be achieved. In order to see whether such inaccuracies occur, we need a measure of the error of the solution.

There are several other parameter that can be used to control the convergence of the solution. These are: (i) the number of time steps to run; (ii) the number of iterations

within each time for solving the non-linearity of the problem; (iii) the number of internal iterations required in solving the simultaneous equations; and (iv) limits on the residual errors.

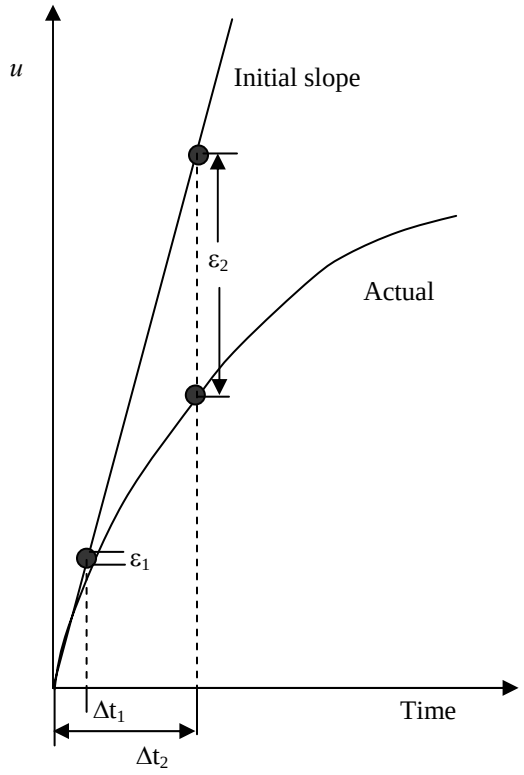


Figure 10. The influence of the time step on the solution.

5.2.2 *Relaxation methods.*

One method to accelerate the iteration process is by using a relaxation factor. At each point in the iteration process a finite error is resulted since the guess is not the exact solution. For instance, in equation (83) the so called "residual error" at each iteration step can be written as:

$$\boldsymbol{r = b - Au} \tag{92}$$

As the solution process progresses from iteration to iteration, the residual errors from each equation should reduce. If they do reduce then the solution is said to be converging. If the residuals become ever larger, then the process is said to be diverging. This process can be accelerated in various ways. If the solution scheme is time dependent or quasi-time dependent then the solution at the end of each time step needs to be converged before moving to the next time step. This can mean controlling several iteration

procedures. One iterative procedure might solve the simultaneous equations generated by linearizing the partial differential equations, the second iterative procedure finds a solution at one time step and accounts for the non-linearity of the problem and a final iteration procedure, if required, moves the solution through the different time levels. All of these iteration processes need to be controlled.

A common method is known as the successive over-relaxation (SOR) method[44]. To do this we take the equations of the Gauss-Seidel method (equation 87) and both add and then subtract the terms u_i^k to the right hand side:

$$\begin{aligned} u_1^{k+1} &= u_1^k + \left[\frac{1}{a_{11}}\left[b_1 - a_{11}u_1^k - a_{12}u_2^k - a_{13}u_3{}^k\right]\right] \\ u_2^{k+1} &= u_2^k + \left[\frac{1}{a_{22}}\left[b_2 - a_{21}u_1^{k+1} - a_{22}u_2^k - a_{23}u_3^k\right]\right] \\ u_3^{k-1} &= u_3^k + \left[\frac{1}{a_{33}}\left[b_3 - a_{31}u_1^{k+1} - a_{32}u_2^{k+1} - a_{33}u_3^k\right]\right] \end{aligned} \tag{93}$$

The terms in the brackets represent the residuals. We can multiply the residual by a factor ω in order to accelerate the iteration process:

$$\begin{aligned} u_1^{k+1} &= u_1^k + \left[\frac{\omega}{a_{11}}\left[b_1 - a_{11}u_1^k - a_{12}u_2^k - a_{13}u_3{}^k\right]\right] \\ u_2^{k+1} &= u_2^k + \left[\frac{\omega}{a_{22}}\left[b_2 - a_{21}u_1^{k+1} - a_{22}u_2^k - a_{23}u_3^k\right]\right] \\ u_3^{k-1} &= u_3^k + \left[\frac{\omega}{a_{33}}\left[b_3 - a_{31}u_1^{k+1} - a_{32}u_2^{k+1} - a_{33}u_3^k\right]\right] \end{aligned} \tag{94}$$

The factor ω is called the relaxation factor and, for most systems, it is set between one and two. If ω is unity the method becomes the original Gauss-Seidel method. The proper selection of the relaxation factor depends on the particular problem under consideration and it is generally obtained by trial-and-error.

5.2.3 *ADI Method:*

One of the problems associated with two-dimensional problems is that the matrix formed by the difference equations may not be tridiagonal. This is the case for all the implicit schemes mentioned earlier. One way to resolve this problem is by using the Alternating Direction Implicit (ADI) method.[45,46,47] In this method the operator $\boldsymbol{A}$ is split in parts. For a two-dimensional flow, one part includes only x-derivatives, and another part includes only y-derivatives: $\boldsymbol{A}_x\boldsymbol{A}_y$. The mixed derivative terms are moved to the right-hand side of the equations. In this way, both $\boldsymbol{A}_x$ and $\boldsymbol{A}_y$ are tridiagonal matrices and therefore, the split-operator system can be solved in a non-iterative, or implicit manner as a sequence of two simple systems of equations. This method will converge if $\boldsymbol{A}_x\boldsymbol{A}_y$ is approximately equal to $\boldsymbol{A}=\boldsymbol{A}_x+\boldsymbol{A}_y$.

5.3 *Convergence and Stability*

The numerical solution is said to converge if it tends to the analytical solution as the grid spacing or element size reduces to zero. However, for most problems we do not have an analytical solution. Therefore, practically, a numerical solution is said to converge if the values of the variables at the points in the domain tend to move towards some fixed value as the solution progresses. Also, the numerical solution procedure is said to be stable if the errors in the discrete solution do not increase so much that the results are not realistic anymore.

Numerical stability has to do with the behavior of the solution as the time-step Δt is increased. If the solution remains well behaved for arbitrarily large values of the time step, the method is said to be unconditionally stable. This situation never occurs with explicit methods, which are always conditionally stable. It is easy to see that this is so by dividing the u-equation by Δt and then letting Δt approach infinity. In this limit there are no $n+1$ terms remaining in the equation so no solution exists for u^{n+1}, indicating that there must be some limit on the size of the time step for there to be a solution. In an implicit formulation, a solution for the unknowns at level $n+1$ may be obtained for any size time step. Of course, the solution for very large times may not be realistic unless the implicit formulation has been carefully constructed. Numerous methods have been developed to test the stability of the numerical method. Among these, von Neumannn's method, the matrix method[5] the discrete perturbation analysis method, and Hirt's[48] method are the more common methods.

5.4 *Von Neuman Stability Analysis*

Consider the one-dimensional transient diffusion equation (Eqn. 38) in which the time derivative is descretized by forward difference scheme and the diffusion term by central difference in space and explicit in time (Eqn. 39). Let u^* signify the numerical solution to the finite difference equation; i.e.,

$$\frac{u^{*n+1}_i - u^{*n}_i}{\Delta t} = \frac{u^{*n}_{i-1} - 2u^{*n}_i + u^{*n}_{i+1}}{\Delta x^2}$$

Let us assume that the (round off) error between the numerical solution u^* and the exact solution to the finite-difference equation, u, is ε, i.e.,

$$u^* - u = \varepsilon$$

Since both u and u* satisfy Eqn. 39, so does the error:

$$\frac{\varepsilon^{n+1}_i - \varepsilon^n_i}{\Delta t} = \frac{\varepsilon^n_{i-1} - 2\varepsilon^n_i + \varepsilon^n_{i+1}}{\Delta x^2} \tag{95}$$

If ε_l decreases as the solution is progressed in time, then the solution is stable. In other words, for stability, we need to have the following conditions:

$$\left|\varepsilon^{n+1}_i / \varepsilon^n_i\right| \le 1$$

Considering that the solution to the transient diffusion is often exponential in time and its spatial dependence is in the form of a Fourier series, it is reasonable to assume that the error has a similar dependence on time and space, hence:

$$\varepsilon = e^{\alpha t} \sum_l e^{jk_l x}$$

where k_l is the wave number, $j = \sqrt{-1}$ and α can be a real or a complex number. For the stability condition to be satisfied, it is sufficient that each term in the above series satisfy the stability condition, i.e.,

$$\left|\frac{e^{\alpha(t+\Delta t)} e^{jk_l x_i}}{e^{\alpha t} e^{jk_l x_i}}\right| \le 1 \quad \Rightarrow \quad \left|e^{\alpha \Delta t}\right| \le 1$$

Substituting one term of the above series in (95) and rearranging, yields:

$$\frac{e^{\alpha \Delta t} - 1}{\Delta t} = \frac{e^{-jk_l \Delta x} - 2 + e^{jk_l \Delta x}}{(\Delta x)^2}$$

But,

$$e^{j\theta} = \cos\theta + j \sin\theta$$

Hence;

$$\frac{e^{\alpha \Delta t} - 1}{\Delta t} = 2\frac{\cos(k_l \Delta x) - 1}{(\Delta x)^2} = -\frac{4\sin^2(1/2 k_l \Delta x)}{(\Delta x)^2}$$

and,

$$\boldsymbol{e}^{\alpha\Delta t} = 1 - \frac{4\Delta \boldsymbol{t}}{(\Delta \boldsymbol{x})^2}\sin^2[1/2\boldsymbol{k}_l \Delta \boldsymbol{x}] \equiv \boldsymbol{G}$$

where *G* is referred to as the *amplification factor*. For a stable solution, the magnitude of *G* should be less than unity;

$$-1 < \boldsymbol{G} < 1$$

which results in the following stability conditions:

$$\frac{\Delta \boldsymbol{t}}{(\Delta \boldsymbol{x})^2} \le \frac{1}{2}$$

Application of the Von Neuman stability analysis to Eqn. 58 (or Eqn. 59) will result in the Courant-Friedrichs-Lewy stability criterion, i.e.,

$$CFL = \frac{c\Delta t}{\Delta x} \le 1$$

Hence for stable solution of the wave equation, the time-step should be,

$$\Delta \boldsymbol{t} \le \Delta \boldsymbol{x} / \boldsymbol{c}$$

5.5 *Convergence of Jacobi and Gauss-Seidel Methods (iterative methods):*

There are several methods to predict the convergence of the Jacobi and Gauss-Seidel methods. These are discussed below.

Let us consider Eqn. 83:

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$$

The coefficient matrix ***A*** can be split into three parts, ***B***, ***T***, and ***D,*** which contains only the diagonal elements, i.e.,

$$\boldsymbol{A} = \boldsymbol{B} + \boldsymbol{D} + \boldsymbol{T}$$

For example, for the system of Eqns. 86, ***B***, ***T***, and ***D*** are:

$$\boldsymbol{B}=\begin{pmatrix}0 & 0 & 0\\ \boldsymbol{a}_{21} & 0 & 0\\ \boldsymbol{a}_{31} & \boldsymbol{a}_{32} & 0\end{pmatrix};\quad \boldsymbol{D}=\begin{pmatrix}\boldsymbol{a}_{11} & 0 & 0\\ 0 & \boldsymbol{a}_{22} & 0\\ 0 & 0 & \boldsymbol{a}_{33}\end{pmatrix};\quad \boldsymbol{T}=\begin{pmatrix}0 & \boldsymbol{a}_{12} & \boldsymbol{a}_{13}\\ 0 & 0 & \boldsymbol{a}_{23}\\ 0 & 0 & 0\end{pmatrix}$$

The Jacobi iterative method (Eqn. 88) is then as follows:

$$\boldsymbol{D}\boldsymbol{u}^{k+1}=\boldsymbol{b}-(\boldsymbol{B}+\boldsymbol{T})\boldsymbol{u}^{k}$$

or,

$$\boldsymbol{u}^{k+1}=\boldsymbol{D}^{-1}\boldsymbol{b}-\boldsymbol{P}\boldsymbol{u}^{k}\qquad \text{where,}\quad \boldsymbol{P}=-\boldsymbol{D}^{-1}(\boldsymbol{B}+\boldsymbol{T})$$

alternatively,

$$\boldsymbol{u}_i^{k+1}=\frac{1}{\boldsymbol{a}_{ii}}(\boldsymbol{b}_i-\sum_{j\neq i}\boldsymbol{a}_{ij}\boldsymbol{u}_j^{k})$$

or:

$$\boldsymbol{u}_i^{k+1}=\boldsymbol{u}_i^{k}+\frac{1}{\boldsymbol{a}_{ii}}(\boldsymbol{b}_i-\sum_{j}\boldsymbol{a}_{ij}\boldsymbol{u}_j^{k})$$

Jacobi method is convergent if the iteration matrix $\boldsymbol{P}$ is convergent. A convergent matrix is a matrix whose *spectral radius* is less than unity. Spectral radius is the largest the eigenvalue of a square matrix, in this case $\boldsymbol{P}$.

A more practical method to study the convergence of the method is to study the norm of the solution vector change, δ, from one iteration to another. δ is given as:

$$\delta^{k}=\sum_{i}\left|\boldsymbol{u}_i^{k}-\boldsymbol{u}_i^{k-1}\right|$$

The method is convergent if $\delta^{k+1}/\delta^{k}<1$ for large values of k. In fact, for large k's, this ratio is approximately equal to the spectral radius of $\boldsymbol{P}$. It should be noted that if Jacobi method is convergent, the Gauss-Seidel method will have a faster convergence.

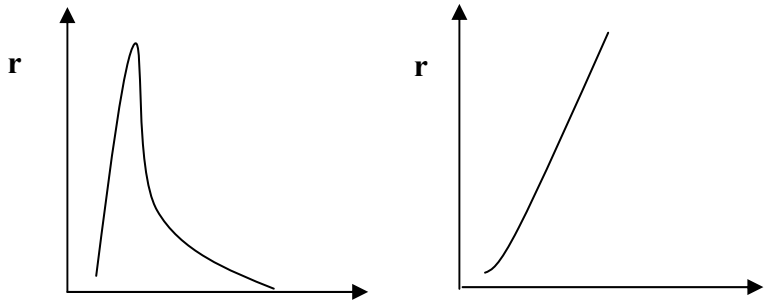
Finally, Jacobi and Gauss-Seidel methods converge if the coefficient matrix A is *strictly diagonally dominant*, i.e.,

$$\left|\boldsymbol{a}_{ii}\right|>\sum_{j\neq i}\left|\boldsymbol{a}_{ij}\right|$$

The convergence rate increases if the left hand side is much bigger than the right hand side. The above condition is indeed the simplest. As an example, when applied to the Laasonen method, this condition requires that:

$$(\Delta x)^2 / \Delta t > 0$$

that is, the time-step must be positive. For a given grid spacing, smaller time-steps will result in faster convergence. Similarly, for a fixed time-step, larger grid spacing results in a faster convergence. It should be stressed that a converged numerical solution is not necessarily an accurate solution. Some typical graphs of the residual value for one of the flow equations plotted against iteration number are shown in Fig. 11.



(a) Converging (b) Diverging

Fig. 11. Variation of Residual with the number of iteration

6 Building a Mesh

One of the most cumbersome and time consuming part of the CFD is the mesh generation. Although for very simple flows, mesh generation is easy, it becomes very complex when the problem has many cavities and passages. Mesh generation is basically the discretization of the computational domain. The mesh in finite difference methods consists of a set of points, which are called nodes. The finite volume method considers points that form a set of volumes which are called cells. The finite element methods use sub-volumes called elements which have nodes where the variables are defined. Values of the dependent variables, such as velocity, pressure, temperature, etc. will be described for each element.

6.1 *Element Form*

Various forms of elements can be used. However, the most common type in CFD programs is a hexahedron with eight nodes, one at each corner, and this is known as a brick element or volume. For two-dimensional applications the equivalent element is a four-noded quadrilateral. Some finite volume programs have now been released which have the ability to use tetrahedral in three dimensions or triangles in two dimensions. Most finite element CFD codes will allow these elements to be used together with a small range of other element types. Figure 12 shows some of the commonly used sub-domains.

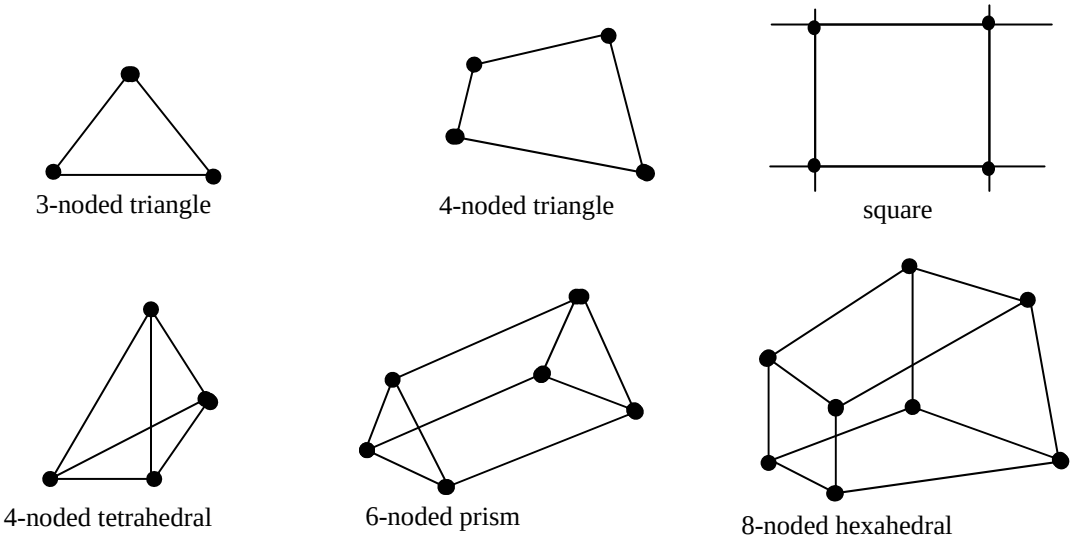


Figure 12. Typical computational elements.

Before generating the mesh, we should know something about the flow behavior. For instance, where in the flow field we have boundary layers, vortices, large gradients in pressure or velocity, etc. The mesh size and shape should be such that it can capture the proper physical conditions that occur in the flow. For regions where large gradients exist,

large number of points within the mesh is needed. This is due to using very simple variation of the parameter, usually, linear, within the each element. Thus the mesh should be small enough so that a linear approximation between two points is valid.

This is depicted in Fig. 13, were the variation of function *u* is given along the coordinate *x*. We will use a linear variation between the points of a numerical solution. If a coarse mesh (Δx_L) is used for the numerical calculation of the curve, the results would be far from the actual variation. However, a fine mesh (Δx_S) can produce results which are close to the actual points. The linear approximation results in large errors where the gradient of u along x is large.

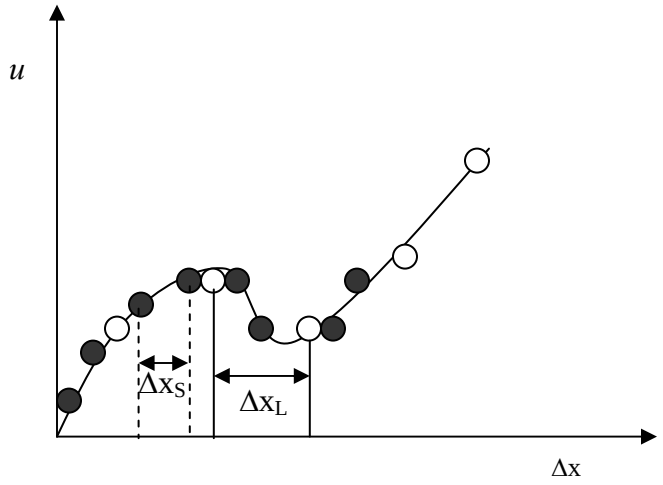


Figure 13. Coarse and fine mesh representation of function *u*.

One of the main difficulties of mesh generation is that, in many cases we do not know where the large gradients are. Usually, along the solid surfaces, where the boundary layer is developed, we need to put more points close to the surface in the direction normal to the surface. Another example is the large pressure changes close to a shock wave in compressible flows. Grid refinement is needed to resolve important flow details. Adaptive grid generation is the solution for complex physical and geometrical problems in which the location of large gradients is not predictable or varies with time, but this is beyond the scope of this text. Generally, refinement is needed near walls, stagnation points, in separation regions, and in wake regions. By increasing number of nodes better accuracy is achieved. Solution should always (if possible) be based on grid independence tests with same style and mesh arrangement.

Grid generation can be assigned to two distinct categories, structured or unstructured grids. Relating the mesh structure to the numerical method; finite difference programs require a mesh to have a regular structure and finite element programs can use a mesh

with an irregular structure. In theory finite volume programs could use a mesh with an irregular structure, but many implementations insist that the mesh has a regular structure. When a mesh with a regular structure is used there is an advantage in that the solver program should run faster than if a mesh with an irregular structure is used. This is due to the implicit relationship that exists between the number of a cell or a point and the number of its neighbors in a regular mesh, which enables data to be found easily. No such relationship occurs for meshes that have an irregular structure and so when trying to find the values of flow variables in neighboring volumes there must be a computational overhead. This often takes the form of a look-up table which relates the faces to the cells or the nodes to the elements.

6.2 *Structured Grid*

The main objective of generating a structured grid is to determine the coordinates transformation that maps the body fitted non-uniform non-orthogonal physical space (x,y,z) into the transformed orthogonal computational space (ξ,η,ζ).

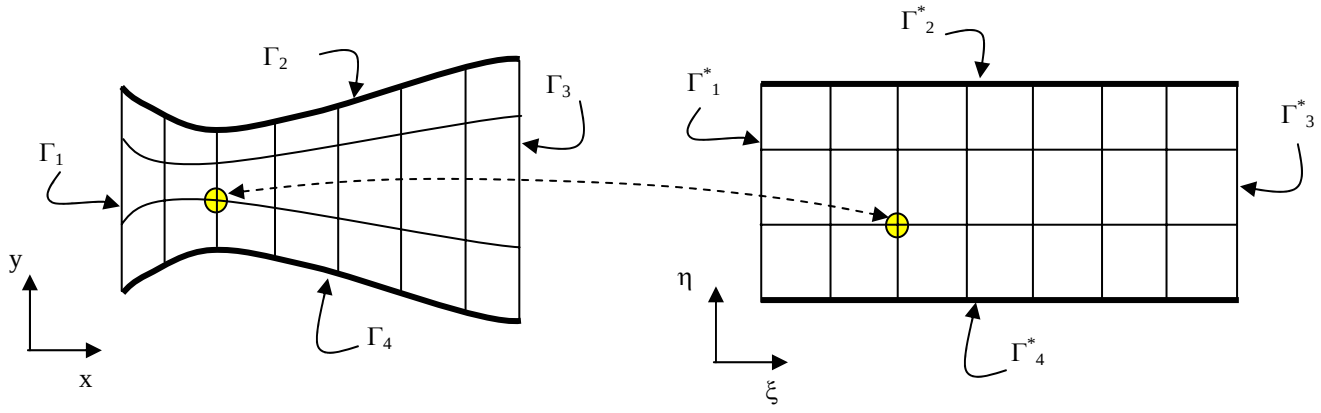


Figure 14- The Transformed Computational Domain

There are two steps in generating a structured grid: a) specification of the boundary point distribution, b) determination of the interior point distribution. The three popular methods for generating structured grids are:

6.2.1 *Conformal mapping method*

In a conformal mapping the angles between grid lines in computational and physical domains are the same. This is the most accurate method, but the application of this method is limited to two-dimensional problems with simple geometries.

6.2.2 *Algebraic method*

This is one of the most common methods used in commercial codes appropriate for several engineering applications. Clustering and stretching of grid elements using algebraic method can be done by different functions such as: polynomial, trigonometric, logarithmic, and geometric functions. Using the algebraic grid generation results in a good control over the grid structure and is relatively simple to apply.

6.2.3 *Differential equation method*

The partial differential equations used to generate a grid can be of elliptic, parabolic, or hyperbolic type. The most applied one is the elliptic type. In this case we want to have control over the followings:

a) Grid point distribution on the boundaries,
b) The angle between the boundaries and the gridlines, and
c) The spacing between the gridlines.

6.2.4 *Block-structured method*

When the geometry is complex, it is very difficult to generate a single zone grid with adequate control on the distribution of the mesh points using structured grids. There are three main types of domain decomposition. These are patched zones, overlapped zones, and overlaid zones. Patched zones have a common boundary line (see Fig. 15). The mesh lines across the boundaries may be continuous or discontinuous[49]. In the second technique, an overlap region exists between the zones. The extent of that region may be from one up to several mesh points. In the third technique, which is also known as the Chimera method. Smaller zones are defined on top of a base grid. Inter-zone data transfer is accomplished by interpolation.

The application of block-structured grid with an algebraic grid generation for each block is explained by the following example:

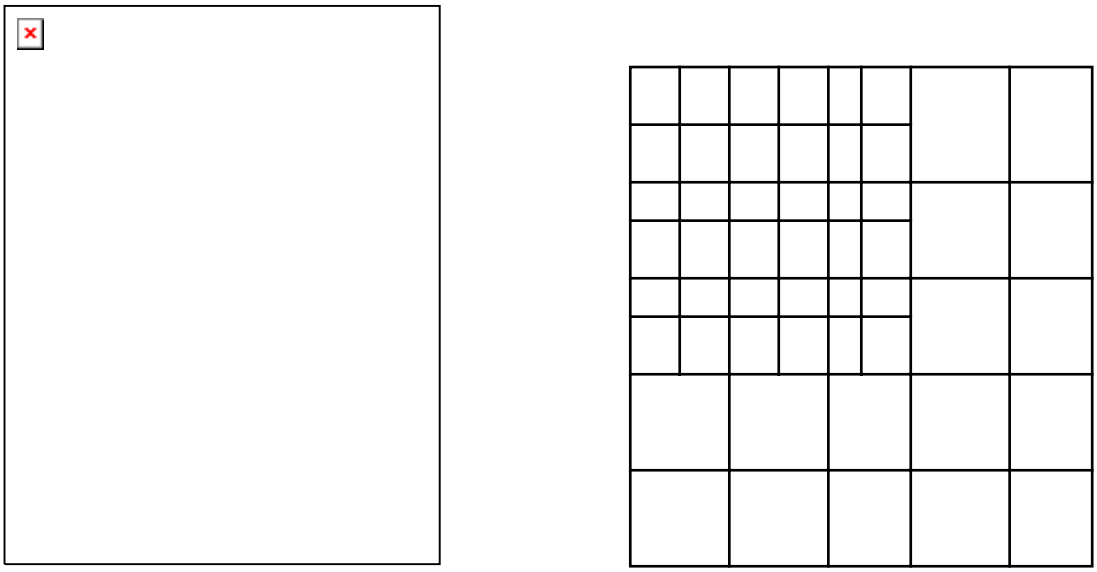
6.3 *Unstructured grid*

Unstructured grids have the advantage of generality in that they can be made to conform to nearly any desired geometry. This generality, however, comes with a price. The grid

generation process is not completely automatic and may require considerable user interaction to produce grids with acceptable degrees of local resolution while at the same time having a minimum of element distortion. Unstructured grids require more information to be stored and recovered than structured grids (e.g., the neighbor connectivity list), and changing element types and sizes can increase numerical approximation errors.

A popular type of unstructured grid consists of tetrahedral elements (Fig. 15). These grids tend to be easier to generate than those composed of hexahedral elements, but they generally have poorer numerical accuracy. For example, it is difficult to construct approximations that maintain an accurate propagation of one-dimensional flow disturbances because tetrahedral grid elements have no parallel faces.

In summary, the best choice for a grid system depends on several factors: convenience in generation, memory requirements, numerical accuracy, flexibility to conform to complex geometries and flexibility for localized regions of high or low resolution.



Polyhedra Elements **Mesh Subdivision and/or Multiblock**

Figure 15. Different Mesh Types

7 References

[1] Hirsch, C., "Numerical Computation of Internal and External Flows", John Wiley & Sons, 1992.

[2] Tannehill, J.C., Anderson, D.A., and Pletcher, R.H., *"Computatioal Fluid Mechanics and Heat Transfer,"* Taylor & Francis, 1997.

[3] Zienkiewicz, O.C. and Taylor, R.L., *"The Finite Element Method- Vol 2: Solid and Fluid Mechanics,"* McGraw-Hill, New York, 1991.

[4] Reddy, J.N., *"An Introduction to the Finite Element Method,"* McGraw-Hill, 1993.

[5] Smith, G.D., "*Numerical Solution of Partial Differential Equations: Finite Difference Methods"*, 3rd edn, Claredon Press, Oxford, 1985.

[6] Tyn Myint-U, *"Partial Differential Equations of Mathematical Physics,"* Elsevier North Holland, Inc., 1980.

[7] Poo, J.Y., and Ashgriz, N., "Curvature Calculation in Interfaces," *J. Compt. Phys.,* vol. 84, no. 2, pp.483-491, 1989.

[8] Brackbill, J.U., Kothe, D.B. and Zemach, C., "A continuum method for modeling surface tension," J. Comput. Phys. 100 , 335 (1992).

[9] Ashgriz, N., and Poo, J. Y. "FLAIR: Flux Line-segment Advection and Interface Reconstruction," *Journal of Computational Physics*, Vol. 93, No. 2, pp. 449-46, 1991.

[10] Mashayek, F., and Ashgriz, N., "A Hybrid Finite Element - Volume of Fluid Method for Simulating Free Surface Flows and Interfaces," *Int. Journal of Numerical Methods in Fluids*, Vol. 20, No. 10, pp. 1363-1380, 1995.

[11] Hildebrand, F.B., *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956.

[12] Chapra, S.C., Canale, R.P., *Numerical Methods for Engineers*, McGraw-Hill, New York, 1988.

[13] Gottlieb, D. and Orzag, S.A., "Numerical Analysis of Spectral Methods: Theory and Applications," *SIAM,* Philadelphia, 1977.

[14] Richardson, L.F., "The ApproximateArithmeticalSolution by Finite Differences of Physical Problems Involving Differenctial Equations, with an Application to the Stresses in a Masonary Dam," *Philos. Trans. R. Soc. London, Ser. A*, vol. 210, pp. 307-357, (1910).

[15] DuFort, E.C., and Frankel. S.P., "Stability Conditions in the Numerical Treatment of Parabolic Equations," *Math. Tables Other Aids Comput.,* vol. 7, pp. 135-152, (1953).

[16] Laasonen, P., Über eine Methode zur Lösung der Warmeleitungsgleichung, *Acta Math*., vol. 81, pp. 309-317, 1949.

[17] Crank, J. and Nicolson, P., " Practical Method for Numerical Evaluation of Solutions of Partial Differential Equations of the Heat-Conduction Type, *Proc. Cambridge Philos. Soc.* , vol. 43, pp. 50-67, 1947.

[18] Lax, P.D., "Weak Solution of Nonlinear Hyperbolic Equations and their Numerical Computations., *Commun. Pure Appl. Math*., vol. 7, pp. 159-193, 1954.

[19] Lax, P.D., and Wendroff, B., " Systems of Conservation Law, *Commun. Pure Appl. Math.,* vol. 13, pp. 217-237, 1960.

[20] MacCormack, R.W., "The effect of Viscosity in Hypervelocity Impact Cratering," *AIAA paper 69-354*, Cincinnati, Ohio, 1969.

[21] Godunov, S.K., "Finite Difference Method for Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics," *Mat. Sb.,* vol. 47, pp. 271-306, 1959.

[22] Van Leer, B., "Towards the Ultimate Conservative Difference Scheme, V: A Second order Sequel to Gudonov's Method," *J. Comput. Phys.,* vol. 32, pp. 101-136, 1979.

[23] Van Leer, B. "Towards the ultimate conservative difference scheme. I. The quest of monotinicity," in *Lecture Notes in Physics,* vol. 18, Springer Verlag, Berlin, p. 163, 1973.

[24] Roe, P.L., "The Use of the Reimann Problem in Finite-Diference Schemes*," Lect. Note Phys.,* vol. 141, Springer-Verlag, New York, pp 354-359, 1980.

[25] Roe, P.L., "Approximate Reimann Solvers, Parameter Vectors and Difference Schemes", J. Comput. Phys., vol 43, pp. 357-372, 1981.

[26] Woodward, P.R., and P. Colella, P.,*Lecture Notes in Physics,* Vol. **141** (Springer-Verlag, New York/Berlin, (1981), p. 434.

[27] Harten, A., "High-Resolution Schemes for Hyperbolic Conservation Laws," *J. Comput. Phys.* **49,** pp. 357-385, 1983.

[28] Colella, P., and H.M. Glaz, *J. Comp. Phys.* 59, 264, 1985.

[29] Colella, P., and P.R. Woodward, J. Comp. Phys. 54, 174, 1984.

[30] Abbott, M. B. and Basco, D. R., "Computational fluid dynamics: An introduction for engineers," Harlow, Essex, England: Longman Scientific & Technical; New York, NY: Wiley, 1989.

[31] Leonard, B.P., "A Stable and Accurate Convective Modeling Procedure Based on Quadratic Upstream Interpolation," *Comput. Methods Appl. Mech. Eng.* Vol. 19, pp. 59-98, 1979.

[32] Harlow, F. H., and J. E. Welch, *Phys. Fluids* **8,** 2182, 1965.

[33] Patankar, S.V., and D. B. Spalding, *Int. J. Heat Mass Transfer* **15,** 1787, 1972.

[34] Patankar, S.V., *Numerical Heat Transfer and Fluid Flow,* Hemisphere, Washington, DC, 1980.

[35] Harlow, F.H. and Welch, J.E., "Numerical Calculation of Time –Dependent Viscous Incompressible Flow of Fluids with Free Surface, *Phys. Fluids*, vol. 8, pp- 2182-2189, 1965.

[36] Chorin, A.J., "Numerical Solution to the Navier-Stokes Equations", *Math. Comput*., vol. 22, pp. 745-762, 1968.

[37] Issa, R.I., A. Gosman, A.D., and Watkins, A.P., "The computation of compressible and incompressible flows by a non-iterative implicit scheme, *J. Comput. Phys.* **6**2, 66 (1986).

[38] Issa, R.I., "Solution of the implicitly discretized fluid flow equations by operator-splitting," *J. Comput. Phys.* **6**2, 40 (1986).

[39] Arakawa, A., *J. Comput. Phys.* **1,** 119, 1966.

[40] Rhie, C.M. and Chow, W.L., *AIAA J.* **11,** 1525, 1983.

[41] Rhie, C.M., *Comput. Fluids* **13,** 443 (1985).

[42] Melaaen, M.C., *Int. J. Numer. Methods Fluids* **15,** 895 (1991).

[43] Coelho, P.J., and J. C. F. Pereira, *Int. J. Numer. Methods Fluids* **14,** 423 (1992).

[44] Frankel, S.P., "Convergence Rates of Iterative Treatments of Partial Differential Equations," *Math. Tables Other Aids Comput*., vol. 4, pp. 65-75, (1950).

[45] Peceman, D.W., and Rachford, H.H., "The Numerical Solution of Parabolic and Elliptic Differential Equations., *J. Soc. Ind. Appl. Math.*vol. 3, pp. 28-41, 1955.

[46] Briley, W.R. and McDonald, H., "Solution of the Three-Dimensional Compressible Navier-Stokes Equations by an Implicit Technique," *Proc. Fourth Int. Conf. Num. Methods Fluid Dyn*., Boulder Colorado, Lect. Notes Phys., vol. 35, Springer-Verlag, New York, pp. 105-110, 1974.

[47] Beam, R.M., and Warming, R.F., "An Implicit Finite Difference Algorithm for Hyperbolic Systems in Conservation Law Form," *J. Comput. Physics*., vol. 22, pp. 87-110., 1976.

[48] Hirt, C.W., "Heuristic Stability Theory for Finite Difference Equatios," *J. Comput. Phys.,* vol. 2, pp.339-355, 1968.

[49] Atta, E.H., and Vadyak, J., *AIAA J*., vol. 21, 1271, 1983.