Chapter 5
The Finite Volume Method

Abstract Similar to other numerical methods developed for the simulation of fluid flow, the finite volume method transforms the set of partial differential equations into a system of linear algebraic equations. Nevertheless, the discretization procedure used in the finite volume method is distinctive and involves two basic steps. In the first step, the partial differential equations are integrated and transformed into balance equations over an element. This involves changing the surface and volume integrals into discrete algebraic relations over elements and their surfaces using an integration quadrature of a specified order of accuracy. The result is a set of semi-discretized equations. In the second step, interpolation profiles are chosen to approximate the variation of the variables within the element and relate the surface values of the variables to their cell values and thus transform the algebraic relations into algebraic equations. The current chapter details the first discretization step and presents a broad review of numerical issues pertaining to the finite volume method. This provides a solid foundation on which to expand in the coming chapters where the focus will be on the discretization of the various parts of the general conservation equation. In both steps, the selected approximations affect the accuracy and robustness of the resulting numerics. It is therefore important to define some guiding principles for informing the selection process.

5.1 Introduction

The popularity of the Finite Volume Method (FVM) [1–3] in Computational Fluid Dynamics (CFD) stems from the high flexibility it offers as a discretization method. Though it was preceded for many years by the finite difference [4, 5] and finite element methods [6], the FVM assumed a particularly prominent role in the simulation of fluid flow problems and related transport phenomena as a result of the work done by the CFD group at Imperial College in the early 70 s under the direction of Professor Spalding [7], with such contributors as Patankar [8], Gosman [9], and Runchal [10, 11] to cite a few. The FVM owes much of its flexibility and popularity to the fact that discretization is carried out directly in the physical space with no need for any transformation between the physical and the computational coordinate system. Furthermore its adoption of a collocated arrangement [12] made it suitable for solving flows in complex geometries. These developments have expanded the applicability of the FVM to encompass a wide range of applications while retaining the simplicity of its mathematical formulation. Another important aspect of the FVM is that its numerics mirrors the physics and the conservation principles it models, such as the integral property of the governing equations, and the characteristics of the terms it discretizes. In what follows the semi-discretized form of a general scalar equation is derived. Then the properties required from the discretization method are discussed along with some guiding principles. The chapter ends with a discussion of a number of issues pertinent to the FVM. The transformation of the semi-discretized equation into algebraic equations will be the subject of a number of chapters to follow.

5.2 The Semi-Discretized Equation

In step 1 of the finite volume discretization process, the governing equations are integrated over the elements (or finite volumes) into which the domain has been subdivided, then the Gauss theorem is applied to transform the volume integrals of the convection and diffusion terms into surface integrals. Following this step, the surface and volume integrals are transformed into discrete ones and integrated numerically through the use of integration points (ip). To clarify this approach and to develop an adequate appreciation for the subtleties of the advanced discretization schemes discussed in later sections, the following example illustrates the application of the technique for a two-dimensional transport problem.

As presented in Chap. 3, the conservation equation for a general scalar variable ϕ can be expressed as

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\textit{transient term}} + \underbrace{\nabla\cdot(\rho\mathbf{v}\phi)}_{\textit{convective term}} = \underbrace{\nabla\cdot\left(\Gamma^{\phi}\nabla\phi\right)}_{\textit{diffusion term}} + \underbrace{Q^{\phi}}_{\textit{source term}} \tag{5.1}$$

The steady-state form of the above equation is obtained by dropping the transient term and is given by

$$\nabla\cdot(\rho\mathbf{v}\phi) = \nabla\cdot\left(\Gamma^{\phi}\nabla\phi\right) + Q^{\phi} \tag{5.2}$$

By integrating the above equation over the element C shown in Fig. 5.1; Eq. (5.2) is transformed to

$$\int_{V_C} \nabla\cdot(\rho\mathbf{v}\phi)dV = \int_{V_C} \nabla\cdot\left(\Gamma^{\phi}\nabla\phi\right)dV + \int_{V_C} Q^{\phi}dV \tag{5.3}$$

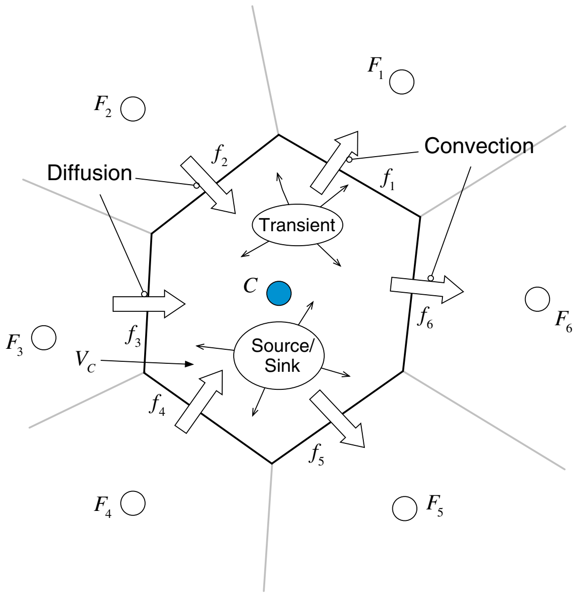


Fig. 5.1 Conservation in a discrete element

Replacing the volume integrals of the convection and diffusion terms by surface integrals through the use of the divergence theorem, the above equation becomes

$$\oint_{\partial V_C} (\rho \mathbf{v} \phi) \cdot d\mathbf{S} = \oint_{\partial V_C} \left(\Gamma^{\phi} \nabla \phi\right) \cdot d\mathbf{S} + \int_{V_C} Q^{\phi} dV \tag{5.4}$$

where **bold** letters indicate vectors, $(\cdot)$ is the dot product operator, Q^{ϕ} represents the source term, $\mathbf{S}$ the surface vector, $\mathbf{v}$ the velocity vector, ϕ the conserved quantity, and $\oint_{\partial V_C}$ the surface integral over the volume V_C.

5.2.1 Flux Integration Over Element Faces

Denoting the convection and diffusion flux terms by $\mathbf{J}^{\phi,C}$ and $\mathbf{J}^{\phi,D}$, respectively, their expressions are given by

$$\mathbf{J}^{\phi,C} = \rho \mathbf{v} \phi \tag{5.5}$$

$$\mathbf{J}^{\phi,D} = -\Gamma^{\phi} \nabla \phi \tag{5.6}$$

Further, defining the total flux $\mathbf{J}^{\phi}$ as the sum of the convection and diffusion fluxes, it can be written as

$$\mathbf{J}^{\phi} = \mathbf{J}^{\phi,C} + \mathbf{J}^{\phi,D} \tag{5.7}$$

Replacing the surface integral over cell *C* by a summation of the flux terms over the faces of element *C*, the surface integrals of the convection, diffusion, and total fluxes become

$$\oint_{\partial V_C} \mathbf{J}^{\phi,C} \cdot d\mathbf{S} = \sum_{f \sim faces(V_C)} \left(\int_f (\rho \mathbf{v} \phi) \cdot d\mathbf{S} \right) \tag{5.8}$$

$$\oint_{\partial V_C} \mathbf{J}^{\phi,D} \cdot d\mathbf{S} = \sum_{f \sim faces(V_C)} \left(\int_f \left(\Gamma^{\phi} \nabla \phi \right) \cdot d\mathbf{S} \right) \tag{5.9}$$

$$\oint_{\partial V_C} \mathbf{J}^{\phi} \cdot d\mathbf{S} = \sum_{f \sim faces(V_C)} \left(\int_f \mathbf{J}_f^{\phi} \cdot d\mathbf{S} \right) \tag{5.10}$$

In Eqs. (5.8)–(5.10) the surface fluxes are evaluated at the faces of the element rather than integrated within it. This transformation has important consequences on the properties of the FVM, one of which is that it renders the method conservative, as will be discussed later.

To proceed further with the discretization, the surface integral at each face of the element in addition to the volume integral of the source term have to be evaluated. Using a Gaussian quadrature the integral at the face *f* of the element becomes

$$\int_f \mathbf{J}^{\phi} \cdot d\mathbf{S} = \int_f \left(\mathbf{J}^{\phi} \cdot \mathbf{n} \right) dS = \sum_{ip \sim ip(f)} \left(\mathbf{J}^{\phi} \cdot \mathbf{n} \right)_{ip} \omega_{ip} S_f \tag{5.11}$$

where *ip* refers to an integration point and *ip*(*f*) the number of integration points along surface *f*. As seen in Fig. 5.2, a number of options are available with their accuracy depending on the number of integration points used and the weighing function ω_{ip}. For a simple mean value integration (Fig. 5.2a), also known as the trapezoidal rule, only one integration point located at the centroid of the face is used with a weighing function of value equal to 1 (i.e., $ip = \omega_{ip} = 1$). This approximation is second order accurate and is applicable in two and three dimensions. Another option (Fig. 5.2b) in two dimensions, which is third order accurate, involves two integration points ($ip = 2$) positioned at $\xi_1 = (3 - \sqrt{3})/6$ and $\xi_2 = (3 + \sqrt{3})/6$ where ξ is distance along the face measured from one end and normalized by the total length, with weights $\omega_1 = \omega_2 = 1/2$. A third option depicted in Fig. 5.2c uses

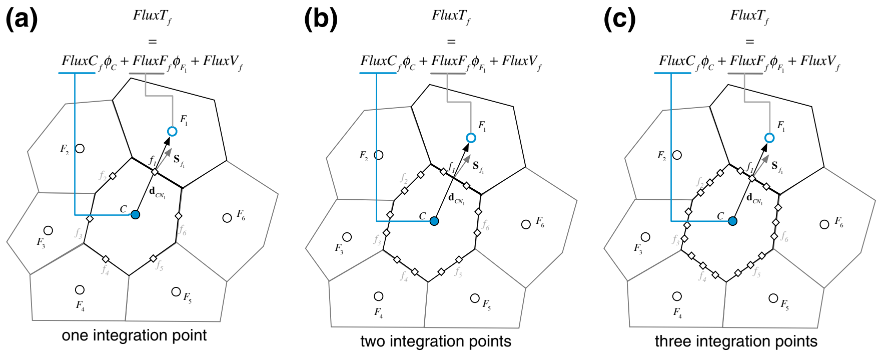


Fig. 5.2 Surface integration of fluxes using **a** one integration point, **b** two integration points, and **c** three integration points

three integration points (ip = 3) positioned at $\xi_1 = (5 - \sqrt{15})/10$, $\xi_2 = 1/2$, and $\xi_3 = (5 + \sqrt{15})/10$, with weights ω_1 = 5/18, ω_2 = 4/9, and ω_3 = 5/18. It is clear that the computational cost rises with the number of integration points used in the approximation. With $ip(f)$ denoting the number of integration points along face f, the general discretized relations for the convection and diffusion terms become

$$\oint_{\partial V_C} (\rho \mathbf{v} \phi) \cdot d\mathbf{S} = \sum_{f \sim faces(V)} \sum_{ip \sim ip(f)} \left(\omega_{ip} (\rho \mathbf{v} \phi)_{ip} \cdot \mathbf{S}_f \right) \tag{5.12}$$

$$\oint_{\partial V_C} \left(-\Gamma^{\phi} \nabla \phi\right) \cdot d\mathbf{S} = \sum_{f \sim faces(V)} \sum_{ip \sim ip(f)} \left(\omega_{ip} \left(-\Gamma^{\phi} \nabla \phi\right)_{ip} \cdot \mathbf{S}_f \right) \tag{5.13}$$

5.2.2 Source Term Volume Integration

Volume integration is used for the source term. Adopting a Gaussian quadrature integration, the volume integral of the source term is computed as

$$\int_V Q^{\phi} dV = \sum_{ip \sim ip(V)} \left(Q^{\phi}_{ip} \omega_{ip} V \right) \tag{5.14}$$

As with surface flux integration, Fig. 5.3 shows different options for volume integration with their accuracy depending on the number of integration points used (ip) and the weighing function ω_{ip}.

For one point Gauss integration (Fig. 5.3a), $ip = \omega_{ip} = 1$ with the integration point located at the centroid of the element. This approximation is second order accurate and is applicable in two and three dimensions. In two dimensions, four point gauss

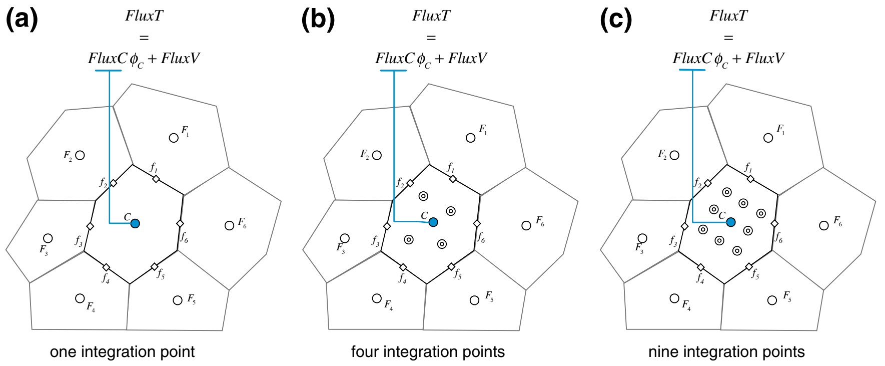

(a)
FluxT = FluxC ϕ_C + FluxV
one integration point
(b)
FluxT = FluxC ϕ_C + FluxV
four integration points
(c)
FluxT = FluxC ϕ_C + FluxV
nine integration points
F_1 F_2 F_3 F_4 F_5 F_6 C f_1 f_2 f_3 f_4 f_5 f_6

Fig. 5.3 Volume integration of source terms using **a** one integration point, **b** four integration points, and **c** nine integration points

integration (Fig. 5.3b) involves the use of four integration points. The weights are computed as the product of the one dimensional weights and the integration points (ξ, η) are obtained from the one dimensional profiles. Therefore the function is calculated at $\left[\left(3-\sqrt{3}\right)/6,\left(3+\sqrt{3}\right)/6\right]$, $\left[\left(3+\sqrt{3}\right)/6,\left(3+\sqrt{3}\right)/6\right]$, $\left[\left(3-\sqrt{3}\right)/6,\left(3-\sqrt{3}\right)/6\right]$, and $\left[\left(3+\sqrt{3}\right)/6,\left(3-\sqrt{3}\right)/6\right]$ with the weights being equal (ω_{ip} = 1/4 *for ip* = 1 *to* 4). The nine point gauss integration method (Fig. 5.3c) involves the use of nine integration points. The accuracy increases with increasing the number of integration points but so does the computational cost.

5.2.3 The Discrete Conservation Equation for One Integration Point

While the above terms can be discretized with any specified number of integration points, it is customary for the finite volume method to use one integration point, yielding second order accuracy. This was found to be a good compromise between accuracy and flexibility while keeping the method simple and relatively of low computational cost. Following the mid-point integration approximation, the semi-discrete steady state finite volume equation for element *C* shown in Fig. 5.4 can be finally simplified to

$$\sum_{f\sim nb(C)}\left(\rho\mathbf{v}\phi-\Gamma^{\phi}\nabla\phi\right)_f\cdot\mathbf{S}_f=Q_C^{\phi}V_C \tag{5.15}$$

The aim of the second stage of the discretization process is to transform Eq. (5.15) into an algebraic equation by expressing the face and volume fluxes in terms of the values of the variable at the neighboring cell centers. This **linearization of the fluxes** is at the core of the second discretization step.

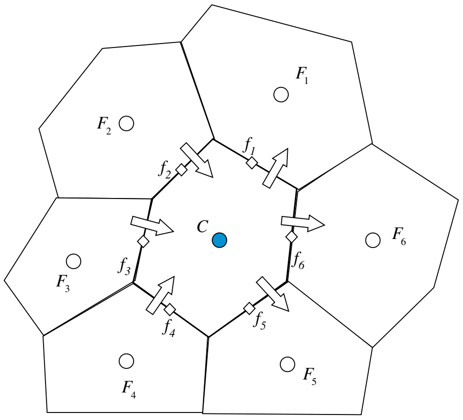


Fig. 5.4 Fluxes at element surfaces

5.2.4 Flux Linearization

As schematically depicted in Fig. 5.2a, the face flux can be split into a linear part, function of the ϕ values at the nodes straddling the face (*i.e., ϕ_C and ϕ_F*), and a non-linear part, which includes the remaining portion that cannot be expressed in terms of ϕ_C and ϕ_F. The resulting equation can be written as

$$\mathbf{J}_f^{\phi} \cdot \mathbf{S}_f = \underbrace{FluxT_f}_{\substack{\textit{total flux}\\ \textit{for face } f}} = \underbrace{FluxC_f}_{\substack{\textit{flux linearization}\\ \textit{coefficient for } C}} \phi_C + \underbrace{FluxF_f}_{\substack{\textit{flux linearization}\\ \textit{coefficient for } F}} \phi_F + \underbrace{FluxV_f}_{\textit{non-linearized part}} \tag{5.16}$$

where $FluxT_f$ represents the total flux through face f, and is decomposed into three terms. The first two terms represent the contributions of the two elements sharing the face and are written via the linearization coefficients $FluxC_f$ and $FluxF_f$. The last term describes the nonlinear contribution that cannot be expressed in terms of ϕ_C and ϕ_F and is given by the non-linear term $FluxV_f$. The values of $FluxC_f$, $FluxF_f$, and $FluxV_f$ obviously depend on the discretized term and the scheme used for its discretization.

The flux linearization is thus obtained by substituting Eq (5.16) into the left hand side of Eq. (5.15). Repeating for all cell faces yields

$$\begin{aligned}\sum_{f\sim nb(C)} \left(\mathbf{J}_f^{\phi} \cdot \mathbf{S}_f\right) &= \sum_{f\sim nb(C)} \left(FluxT_f\right)\\ &= \sum_{f\sim nb(C)} \left(FluxC_f\,\phi_C + FluxF_f\,\phi_F + FluxV_f\right)\end{aligned} \tag{5.17}$$

The linearization of the volume flux is performed, as shown in Fig. 5.3a, by expressing it as a linear function of the element node value ϕ_C and is given by

$$\begin{aligned} Q_C^{\phi} V_C &= FluxT \\ &= FluxC\,\phi_C + FluxV \end{aligned} \tag{5.18}$$

In the case of a constant source term, the volume flux, which represents the right hand side of Eq. (5.15), reduces to

$$\begin{aligned} FluxC &= 0 \\ FluxV &= Q_C^{\phi} V_C \end{aligned} \tag{5.19}$$

Substitution of Eqs. (5.17) and (5.18) in Eq. (5.15), yields the sought after algebraic relation as

$$a_C \phi_C + \sum_{F \sim NB(C)} (a_F \phi_F) = b_C \tag{5.20}$$

where the relations between equation coefficients and flux linearization coefficients are expressed as

$$\begin{aligned} a_C &= \sum_{f \sim nb(C)} FluxC_f - FluxC \\ a_F &= FluxF_f \\ b_C &= -\sum_{f \sim nb(C)} FluxV_f + FluxV \end{aligned} \tag{5.21}$$

Example 1

Find the linearization coefficients for the discretization of the convection term when the velocity field is in the positive direction using the approximation $\phi_f = \phi_C$ (this is known as the upwind scheme).

Solution

$$\mathbf{J}_f^{\phi} = (\rho \mathbf{v} \phi)_f$$

thus

$$\mathbf{J}_f^{\phi} \cdot \mathbf{S}_f = (\rho \mathbf{v} \phi)_f \cdot \mathbf{S}_f = \left(\rho_f \mathbf{v}_f \cdot \mathbf{S}_f\right) \phi_f = \dot{m}_f \phi_f$$

With $\phi_f = \phi_C$, the coefficients in the total flux equation are obtained as

$$FluxC_f = \dot{m}_f$$
$$FluxF_f = 0$$
$$FluxV_f = 0$$

and the total flux is expressed as

$$FluxT_f = \dot{m}_f \phi_C + 0\phi_F + 0$$

5.3 Boundary Conditions

The evaluation of the fluxes at the faces of a domain boundary does not require, in general, a profile assumption. Rather a direct substitution is usually performed. The type of boundary conditions are numerous. However, two of the most widely used ones for general scalars are the Dirichlet and the Neumann boundary conditions. In mathematical terms these are respectively a value specified (or a first type) and a flux specified (or a second type) boundary condition.

5.3.1 Value Specified (Dirichlet Boundary Condition)

Consider the case where some scalar ϕ is being convected through an inlet. Assuming the diffusion of ϕ to be negligible, the boundary condition can be expressed as

$$\phi_b = \phi_{b,specified} \tag{5.22}$$

For the boundary face shown in Fig. 5.5, the boundary flux is evaluated using the known value of ϕ_b given by Eq. (5.22). Therefore the value of the boundary flux is not unknown, rather it can be directly evaluated as

$$\begin{aligned} \mathbf{J}_b^{\phi} \cdot \mathbf{S}_b &= \mathbf{J}_b^{\phi,C} \cdot \mathbf{S}_b \\ &= (\rho \mathbf{v} \phi)_b \cdot \mathbf{S}_b \\ &= FluxC_b \phi_C + FluxV_b \\ &= (\rho_b \mathbf{v}_b \cdot \mathbf{S}_b)\phi_b = \dot{m}_f \phi_{b,specified} \end{aligned} \tag{5.23}$$

Thus

$$\begin{aligned} FluxC_b &= 0 \\ FluxV_b &= \dot{m}_f \phi_{b,specified} \end{aligned} \tag{5.24}$$

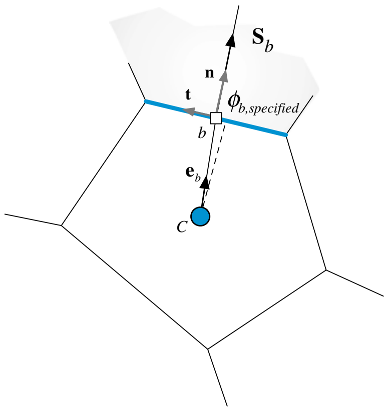


Fig. 5.5 Dirichlet boundary condition

5.3.2 Flux Specified (Neumann Boundary Condition)

Considering the case shown in Fig. 5.6 where the boundary face of the element C represents physically a wall where a flux for the quantity ϕ is specified. Mathematically this is equivalent to writing

$$\begin{aligned}\mathbf{J}_b^{\phi}\cdot\mathbf{S}_b &= \underbrace{\mathbf{J}_b^{\phi}\cdot\mathbf{n}_b}_{specified\,flux}\; S_b \\ &= q_{b,specified}S_b\end{aligned} \tag{5.25}$$

In the above equation $q_{b,specified}$ is a known quantity specified by the user, representing the flux per unit area.

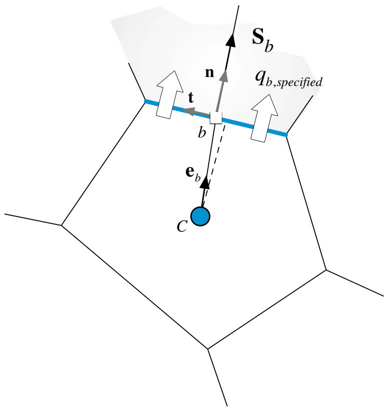


Fig. 5.6 Neumann boundary condition

Thus

$$\begin{aligned} FluxC_b &= 0 \\ FluxV_b &= q_{b,specified} S_b \end{aligned} \tag{5.26}$$

The treatment of these two boundary conditions and others will be detailed in the following chapters along with the treatment of the terms related to step two discretization.

5.4 Order of Accuracy

As discussed earlier, fluxes at the faces and sources over the element are evaluated following the mean value approach, i.e., using the value at the centroid of the surface (midpoint rule) and cell, respectively. This treatment, in addition to the assumed variation of ϕ in space around point C, i.e., $\phi = \phi(\mathbf{x})$, determine the accuracy of the discretization procedure. In the adopted method, ϕ is assumed to vary linearly in space, i.e.,

$$\phi(\mathbf{x}) = \phi_C + (\mathbf{x} - \mathbf{x}_C) \cdot (\nabla\phi)_C \text{ where } \phi_C = \phi(\mathbf{x}_C) \tag{5.27}$$

5.4.1 Spatial Variation Approximation

The spatial variation of the variable $\phi = \phi(\mathbf{x})$ within the element shown in Fig. 5.7 can be described via a Taylor series expansion around point $\mathbf{x}_C$ as

$$\begin{aligned} \phi(\mathbf{x}) = \phi_C + (\mathbf{x} - \mathbf{x}_C) \cdot (\nabla\phi)_C &+ \frac{1}{2}(\mathbf{x} - \mathbf{x}_C)^2 \colon (\nabla\nabla\phi)_C \\ &+ \frac{1}{3!}(\mathbf{x} - \mathbf{x}_C)^3 \colon\colon (\nabla\nabla\nabla\phi)_C + \ldots\ldots\ldots\ldots\ldots \\ &+ \frac{1}{n!}(\mathbf{x} - \mathbf{x}_C)^n \underbrace{\colon\colon \ldots \colon\colon}_{(n-1) times} \Bigg(\underbrace{\nabla\nabla \ldots \nabla\phi}_{n\, times}\Bigg)_C + \ldots\ldots\ldots\ldots\ldots \end{aligned} \tag{5.28}$$

The expression $(\mathbf{x} - \mathbf{x}_C)^n$ in the equation and consequent ones represents the nth tensorial product of the vector $(\mathbf{x} - \mathbf{x}_C)$ with itself, producing an nth rank tensor. The operator (:) is the inner product of two 2nd rank tensors, (::) is the inner product of two 3rd rank tensors, and more generally "$\underbrace{\colon\colon \ldots \colon\colon}_{(n-1) times}$" is the inner product of two nth rank tensors, all yielding a scalar.

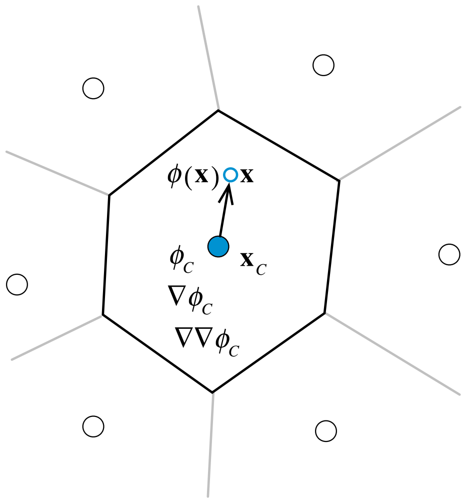


Fig. 5.7 Variation within an element

Comparison between the assumed variation given by Eq. (5.27) and the Taylor series expansion [Eq. (5.28)] indicates an error proportional to $|(\mathbf{x} - \mathbf{x}_C)^2|$ and implying a second order spatial accuracy.

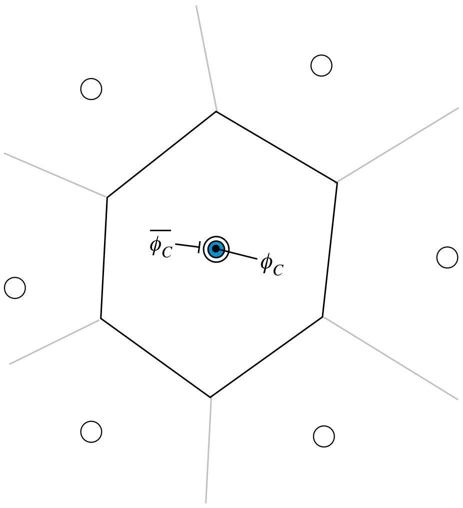
5.4.2 *Mean Value Approximation*

The accuracy of the mean value approximation can be derived by integrating the variable $\phi(\mathbf{x})$ over the cell of centroid C depicted in Fig. 5.8 and leading to

$$\begin{aligned}\overline{\phi}_C &= \frac{1}{V_C}\int_{V_C}\phi\, dV\\ &= \frac{1}{V_C}\int_{V_C}\left[\phi_C + (\mathbf{x}-\mathbf{x}_C)\cdot(\nabla\phi)_C + O\left(|\mathbf{x}-\mathbf{x}_C|^2\right)\right]dV\\ &= \frac{\phi_C}{V_C}\int_{V_C}dV + \frac{1}{V_C}\int_{V_C}(\mathbf{x}-\mathbf{x}_C)\cdot(\nabla\phi)_C\, dV + \frac{1}{V_C}\int_{V_C}O\left(|\mathbf{x}-\mathbf{x}_C|^2\right)dV\\ &= \phi_C + O\left(|\mathbf{x}-\mathbf{x}|^2\right)\end{aligned} \tag{5.29}$$

where V_C is the volume of the element. The second term is equal to zero because point C is the centroid of the element. Neglecting the last term in the equation introduces an error of order 2, demonstrating that the mean value approximation is second order accurate.

Fig. 5.8 Mean-value theorem



The convective flux at face f of element C shown in Fig. 5.9 is computed as

$$\begin{aligned}
\overline{(\rho_f \mathbf{v}_f \cdot \mathbf{S}_f)\phi_f} &= \int_f (\rho \mathbf{v}\phi) \cdot d\mathbf{S} \\
&= \int_f \rho_f \mathbf{v}_f \left[\phi_f + (\mathbf{x} - \mathbf{x}_f) \cdot (\nabla\phi)_f + O\left(|\mathbf{x} - \mathbf{x}_f|^2\right)\right] \cdot d\mathbf{S} \\
&= \rho_f \mathbf{v}_f \phi_f \cdot \int_f d\mathbf{S} + \int_f (\mathbf{x} - \mathbf{x}_f) \cdot (\nabla\phi)_f \rho_f \mathbf{v}_f \cdot d\mathbf{S} + \int_f O\left(|\mathbf{x} - \mathbf{x}_f|^2\right) \rho_f \mathbf{v}_f \cdot d\mathbf{S} \\
&= (\rho_f \mathbf{v}_f \cdot \mathbf{S}_f)\left[\phi_f + O\left(|\mathbf{x} - \mathbf{x}_f|^2\right)\right]
\end{aligned} \tag{5.30}$$

where again the second term is equal to zero as $\mathbf{x}_f$ is the centroid of the respective element surface. Here subscript f denotes the value of the variable, in this case ϕ, at the face centroid and $\mathbf{S}$ is the outward pointing face surface vector.

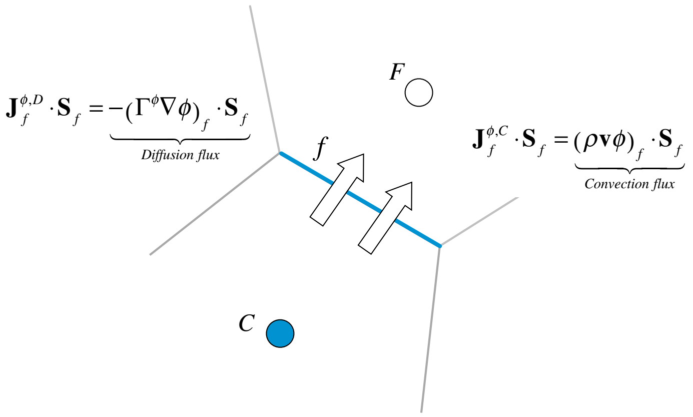

$\mathbf{J}_f^{\phi,D} \cdot \mathbf{S}_f = \underbrace{-(\Gamma^\phi \nabla\phi)_f \cdot \mathbf{S}_f}_{\text{Diffusion flux}}$
F
f
$\mathbf{J}_f^{\phi,C} \cdot \mathbf{S}_f = \underbrace{(\rho \mathbf{v}\phi)_f \cdot \mathbf{S}_f}_{\text{Convection flux}}$
C

Fig. 5.9 Convection and diffusion fluxes at element faces

For the diffusive flux, truncating second order terms and higher, the following can be written:

$$\begin{aligned}\int_f \left(\Gamma^\phi \nabla\phi\right)\cdot d\mathbf{S} &= \int_f \left[\left(\Gamma^\phi\nabla\phi\right)_f + \left(\mathbf{x}-\mathbf{x}_f\right)\cdot\left(\nabla\left(\Gamma^\phi\nabla\phi\right)\right)_f + O\left(\left|\mathbf{x}-\mathbf{x}_f\right|^2\right)\right]\cdot d\mathbf{S} \\ &= \left(\Gamma^\phi\nabla\phi\right)_f\cdot\int_f d\mathbf{S} + \left[\int_f \left(\mathbf{x}-\mathbf{x}_f\right)d\mathbf{S}\right] : \left(\nabla\left(\Gamma^\phi\nabla\phi\right)\right)_f + O\left(\left|\mathbf{x}-\mathbf{x}_f\right|^2\right) \\ &= \left(\Gamma^\phi\nabla\phi\right)_f\cdot\mathbf{S}_f + O\left(\left|\mathbf{x}-\mathbf{x}_f\right|^2\right)\end{aligned} \tag{5.31}$$

Hence for a second order method, the surface integral and the variation of ϕ within the element should be second order accurate.

Higher order accuracy can be achieved by increasing the order of accuracy of the surface integral or of the assumed profile for ϕ. One method developed by Lilek and Peric [13] in two dimensions used a fourth order accurate surface integral, evaluated via Simpson's rule as

$$\int_f \phi d\mathbf{S} = \left(\frac{\phi_{ip_1} + 4\phi_{ip_2} + \phi_{ip_3}}{6}\right)\mathbf{S}_f \tag{5.32}$$

As shown in Fig. 5.10, for face f the value of ϕ is needed at three integration points, i.e., the surface centroid ip_2, and the vertices of the face ip_1 and ip_3.

To obtain a formally fourth order accurate discretization, the assumed profile of ϕ within the element should also be fourth order accurate such that

$$\phi(\mathbf{x}) = \phi_C + (\mathbf{x}-\mathbf{x}_C)\cdot(\nabla\phi)_C + \frac{1}{2}(\mathbf{x}-\mathbf{x}_C)^2 : (\nabla\nabla\phi)_C + \cdots \tag{5.33}$$

In this case the accuracy of the gradient calculation has to be second order or higher and that of the Hermitian first order or higher.

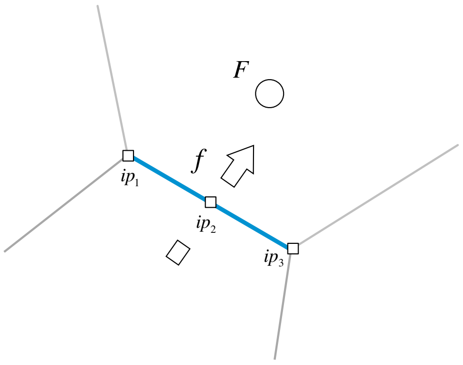


Fig. 5.10 A element face f showing the integration points where values are needed for a higher order approximation of the surface integral

5.5 Transient Semi-Discretized Equation

For unsteady state the temporal term in Eq. (5.1) is retained and in addition to integrating over the volume of the element, integration over time is also needed. In this case the integrated equation becomes

$$\begin{aligned}\int_{t}^{t+\Delta t}\int_{V_C}\frac{\partial(\rho\phi)}{\partial t}dVdt+\int_{t}^{t+\Delta t}\left[\sum_{f\sim nb(C)}\left(\int_{f}(\rho\mathbf{v}\phi)_f\cdot d\mathbf{S}\right)\right]dt\\ -\int_{t}^{t+\Delta t}\left[\sum_{f\sim nb(C)}\left(\int_{f}(\Gamma\nabla\phi)_f\cdot d\mathbf{S}\right)\right]dt\\ =\int_{t}^{t+\Delta t}\left[\int_{V_C}Q^{\phi}dV\right]dt\end{aligned}\tag{5.34}$$

Further simplification of this equation requires a choice with regard to the time integration accuracy required. This will be dealt with in Chap. 13. For fixed grids, where the volume and surface of each element are constant in time, the first term can be integrated as

$$\int_{t}^{t+\Delta t}\int_{V_C}\frac{\partial(\rho\phi)}{\partial t}dV\,dt=\int_{t}^{t+\Delta t}\frac{\partial}{\partial t}\left(\int_{V_C}\rho\phi dV\right)dt=\int_{t}^{t+\Delta t}\frac{\partial\left(\overline{\rho\phi}\right)}{\partial t}V_C\,dt\tag{5.35}$$

where

$$\overline{\rho\phi}_C=\frac{1}{V_C}\int_{V_C}\rho\phi dV=(\rho\phi)_C+O\left(\Delta^2\right)\tag{5.36}$$

Substituting back, Eq. (5.34) reduces to

$$\begin{aligned}\int_{t}^{t+\Delta t}\frac{\partial(\rho\phi)}{\partial t}V_C dt+\int_{t}^{t+\Delta t}\left[\sum_{f\sim nb(C)}\left(\int_{f}(\rho\mathbf{v}\phi)_f\cdot d\mathbf{S}\right)\right]dt\\ -\int_{t}^{t+\Delta t}\left[\sum_{f\sim nb(C)}\left(\int_{f}(\Gamma\nabla\phi)_f\cdot d\mathbf{S}\right)\right]dt\\ =\int_{t}^{t+\Delta t}\left[\int_{V_C}Q^{\phi}dV\right]dt\end{aligned}\tag{5.37}$$

and using the midpoint rule, Eq. (5.37) becomes

$$\begin{aligned}\int_{t}^{t+\Delta t} \frac{\partial(\rho\phi)}{\partial t} V_C dt + \int_{t}^{t+\Delta t} \left[\sum_{f\sim nb(C)} (\rho \mathbf{v}\phi)_f \cdot \mathbf{S}_f\right] dt \\ - \int_{t}^{t+\Delta t} \left[\sum_{f\sim nb(C)} (\Gamma\nabla\phi)_f \cdot \mathbf{S}_f\right] dt \\ = \int_{t}^{t+\Delta t} Q_C^{\phi} V_C dt\end{aligned} \tag{5.38}$$

Going forward beyond this point requires some assumptions as to how the variable is changing in time.

5.6 Properties of the Discretized Equations

As the size of the element tends to zero, the numerical solution is expected to be the exact solution of the general conservation equation [e.g., Eq. (5.2)] irrespective of the interpolation profile used to evaluate the element ϕ values. However, since finite volumes are used, it is crucial for the discretized equations to possess some properties in order to ensure a meaningful solution field. These properties are discussed next.

5.6.1 Conservation

From a physical point of view it is very important for the transported variables, which are generally conservative quantities (e.g., mass, energy, etc.), to be conserved in the discretized solution domain too, otherwise results may be unrealistic. This property is inherent to the FVM because the fluxes integrated at an element face are based on the values of the elements sharing the face [8]. Thus for any surface common to two elements, the flux leaving the face of one element will be exactly equal to the flux entering the other element through that same face. Thus these fluxes are of equal magnitudes but of opposite signs (Fig. 5.11). Any method that possesses this property is said to be conservative.

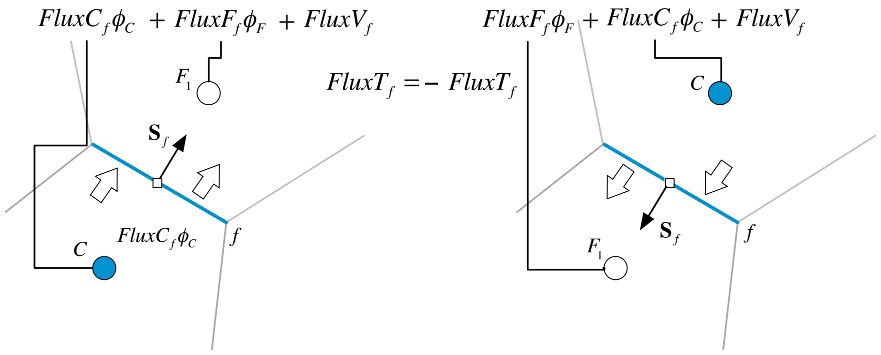


Fig. 5.11 Fluxes on neighboring elements

5.6.2 Accuracy

Accuracy refers to how close a numerical solution is to the exact solution. However, in most of the cases the exact solution for the problem to be solved is unknown. Therefore a direct comparison to check accuracy is not possible. An alternative is to consider the truncation error as a measure of accuracy. The error associated with the first step discretization of the various terms presented above was $O|(\mathbf{x} - \mathbf{x}_f)|^2$, which represents a second order accuracy. This means that if the number of grid points is doubled then the discretization error will be reduced by a factor of 4. The truncation error of a discretization scheme is the largest truncation error of each of the individual terms in the discretized equation. The discretization error does not give the value of the error on a certain grid. It does tell, however, how fast the error will decrease with grid refinement. The higher the order of the error the faster it will decrease with mesh refinement.

5.6.3 Convergence

The nonlinear nature of the conservation equations dealt with here necessitates an iterative approach. Starting with an initial guess, solutions are obtained by repeatedly applying a solution algorithm with the solution at the end of an iteration used as an initial guess for the following iteration. Ideally a solution is said to be converged when it does not change any further as iterations progress. In practice, however, a solution is established converged when changes between two consecutive iterations fall below a vanishing quantity ε. In general convergence is used to indicate the obtainment of a solution with any method. Sometimes the term "convergence" is used to indicate the attainment of a grid independent solution, i.e., a solution that does not change with any further grid refinement.

5.6.4 Consistency

A solution to an algebraic equation approximating a given partial differential equation is said to be ***consistent*** if, at each point in the solution domain, the numerical solution approaches the exact solution of the partial differential equation as the time step and grid spacing tend to zero, i.e., as the discretization error approaches zero. For this to be true, the discretization error should be some power of Δt and/or $\Delta \mathbf{x}$. If the discretization error is expressed in terms of $\Delta \mathbf{x}/\Delta t$ then, for consistency, $\Delta \mathbf{x}$ should tend to zero at a faster rate than Δt.

5.6.5 Stability

Stability refers to the behavior of the discretized equations to be resolved by an iterative solver. It indicates whether the resulting system of algebraic equations can be solved under a variety of initial and boundary conditions. In this sense stability is not really a property of the discretization process but rather a property of the resulting system of equations. As mentioned in a previous chapter, a sufficient condition for a system of linear equations to be stable and converge to a solution is for it to satisfy the Scarborough criterion, i.e., for its matrix of coefficients to be diagonally dominant.

For transient problems, a stable numerical scheme keeps the error in the solution bounded as time marching proceeds. The use of explicit or implicit transient schemes has direct impact on the stability of the numerical method. The stability of explicit methods is ensured by limiting the size of the time step. On the other hand, the stability of implicit methods can be enhanced by under-relaxing the discretized set of algebraic equations either through the use of under relaxation factors or by applying the false-transient approach, as described in a later chapter.

5.6.6 Economy

Economy is an important consideration in the development and application of CFD codes. It is understood to mean that the time required to compute realistic flow problems should not be prohibitive.

5.6.7 Transportiveness

The directional properties exhibited by fluid transport are well known and are signaled by the change in the type of the transport scalar equation [Eq. (5.2)], which

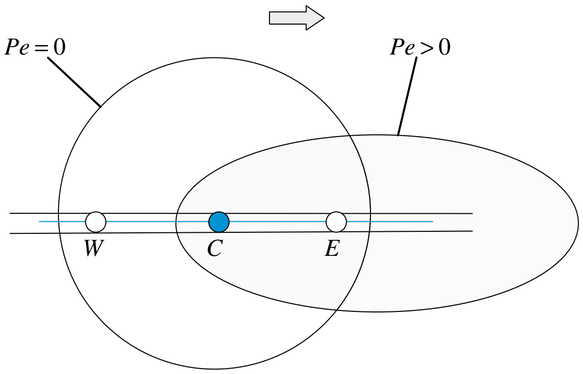


Fig. 5.12 Illustration of the transportive property of fluid flow

may become hyperbolic under certain conditions. The implications on the finite volume equation, visualized in Fig. 5.12, can be explained as follows. If there is a constant source of ϕ within an element C in a flow field with uniform velocity and diffusivity, then the shapes of the contours of constant scalar ϕ will be influenced by the ratio of convection to diffusion strengths, i.e., the Péclet number (Pe) defined as

$$Pe = \frac{Convection\ strength}{Diffusion\ strength} = \frac{\rho u}{\Gamma / \Delta x} \tag{5.39}$$

Based on Eq. (5.39), the case $Pe = 0$ indicates that the transport of ϕ is governed by diffusion, which has an elliptic behavior [8]. In this case, isolines of ϕ are circular and the value of ϕ at C is influenced by the surrounding nodes W and E (Fig. 5.12). Increasing convection effects (i.e., increasing Pe), the circular contours become elliptic in shape and the region influencing the value of ϕ at C shifts in the direction of the flow. Therefore for high Pe flows, events at node C will have a weak or no influence on upstream nodes, while downstream nodes will be strongly affected.

Failure to observe this requirement in the selected discretization schemes can give rise to unstable solutions (i..e., unphysical oscillations).

5.6.8 Boundedness of the Interpolation Profile

Ensuring conservation does not guarantee that other important properties of the original partial differential equation will be maintained by the discretized equation. For example physical considerations lead to the conclusion that in the absence of sources, the value of the conserved variable ϕ within the domain should be bounded by the values at the domain boundaries [14]. The discretized equation

$$a_C\phi_C + \sum_{F\sim NB(C)} a_F\phi_F = b_C \tag{5.40}$$

would fulfill this requirement when the nodal ϕ_C values satisfy the following constraint:

$$\min_{i=1}^{N}\left(\phi_{F_i}\right) \le \phi_C \le \max_{i=1}^{N}\left(\phi_{F_i}\right) \tag{5.41}$$

where F_i represents the ith neighbor of C and N their number. This can be achieved by a judicial control of the discretization schemes and their linearization, as will be detailed in later chapters.

5.7 Variable Arrangement

While the cell-centered variable arrangement is the one selected in OpenFOAM® and generally preferred with the FVM (Fig. 5.13a), vertex-centered [15] (Fig. 5.13b) variable arrangement methods have also been used. Two vertex-centered arrangements have been adopted resulting in either overlapping elements or a dual mesh, respectively. Because the dual mesh method is more popular, it is described next as a representative of vertex-centered methods.

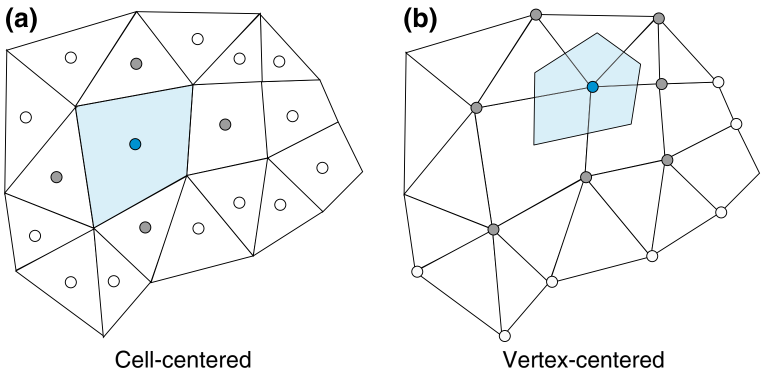


Fig. 5.13 **a** Cell-centered arrangement; **b** vertex-centered arrangement

5.7.1 Vertex-Centered FVM

In a vertex-centered arrangement the flow variables are stored at the vertices (or grid points), with elements constructed around the variable locations by using the concept of a dual mesh and dual cells [16] (Fig. 5.14). Adopting this concept, a cell can be created around a grid point in several ways. In two dimensions, one approach (Fig. 5.14a) is by connecting the centroids of the cells having the grid point in common. As displayed in Fig. 5.14b a second possibility is to join the centroids of the surrounding elements to the centroids of their faces. The same construction procedure can be used in three dimensional situations where an element around a grid point is created by properly connecting surrounding cells' centroids, faces' centroids, and edges' centroids resulting in non-overlapping elements.

The use of a vertex-centered arrangement of variables allows for an explicit profile to be defined over the elements in terms of the vertex variables. In this case the variables represent point values and variation through the element can be computed using shape functions or interpolation profiles. This approach permits an accurate resolution of face fluxes for all mesh topologies, but yields a lower order accuracy of element-based integrations since the vertex is not necessarily at the element centroid. Moreover, it increases the storage requirements due to the creation of larger matrix. Furthermore handling of boundary conditions in cell-vertex schemes, as shown in Fig. 5.14b, requires additional treatment in order to ensure a consistent solution at boundary points shared by multiple grid blocks. Still a major disadvantage is the need to base the mesh on a set of element types for which a shape function can be defined.

Another shortcoming of the vertex-centered scheme with dual elements appears at solid boundaries where only a portion of an element is formed and the node where values are stored is at the wall (Fig. 5.14b). In a regular cell, the integration of face fluxes results in a residual located at the main point inside the element where values are stored. In this case however, the residuals will be stored at the wall

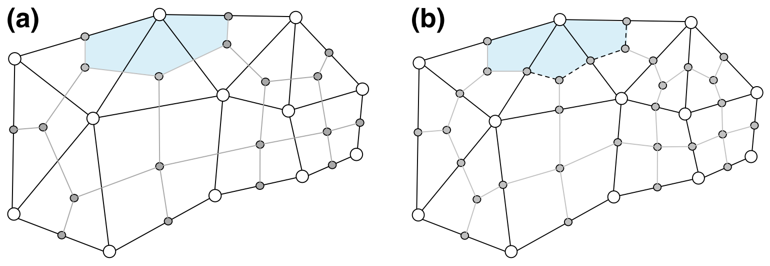


Fig. 5.14 Vertex-centered arrangement: **a** dual cells connecting centroids of cells, and **b** dual cells connecting centroids of cells to centroids of faces

boundary creating a discrepancy and causing an increase in the discretization error there in comparison with cell-centered schemes. Additional complications may also arise at sharp edges and branch cuts.

5.7.2 Cell-Centered FVM

The cell-centered variable arrangement is currently the most popular type of variable arrangement used with the FVM. With this practice, the variables and their related quantities are stored at the centroids of grid cells or elements. Thus, the elements are identical to the discretization elements and, in general, the method is second order accurate since all quantities are computed at element and face centroids, where the difference between the value of the variable and its average is $O(\Delta x^2)$. Variations within the cell can be re-constructed using a Taylor series expansion. Another advantage of the cell-centered formulation is that it allows for the use of general polygonal elements with no need for pre-defined shape functions. This permits a straightforward implementation of a full multigrid strategy.

However two important disadvantages of the method are its treatment of non-conjunctional elements and the manner the diffusion term is discretized on non-orthogonal cells. The first issue influences the accuracy of the method, the second its robustness, while both being affected by the quality of the mesh.

Consider the two-cell arrangement shown in Fig. 5.15. It is clear that any average of a value defined at C and F will be defined at f' rather than f, the centroid of the face. Thus any discretization procedure using this interpolated value will not have an $O(\Delta x^2)$ accuracy.

For a cell-centered scheme the discretization error depends strongly on the smoothness of the grid. Results for such a situation are displayed in Fig. 5.16. The physical situation displayed in Fig. 5.16a represents an annulus between two horizontal cylinders of rhombic cross-sections. The inner cylinder is maintained at the uniform hot temperature T_h while the outer cylinder is kept at the cold temperature T_c. The difference in temperature creates density variation within the enclosure and gives rise to buoyancy forces that establish a flow field. Using the grid system

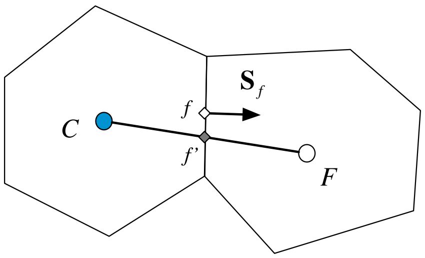


Fig. 5.15 Two non-junctional cell-centered elements

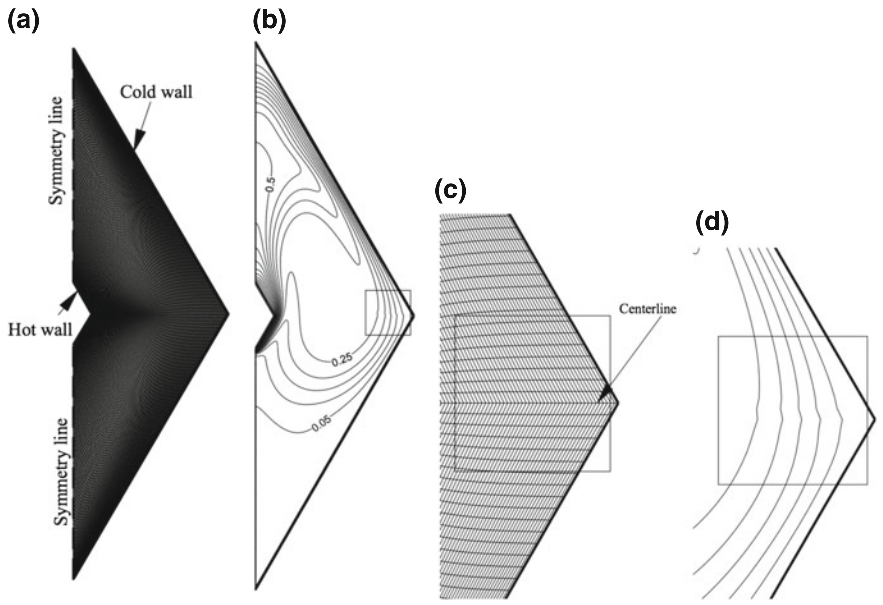


Fig. 5.16 **a** Grid used in solving for natural convection in the annulus between horizontal cylinders of rhombic cross sections; **b** isotherms over the domain; **c** a magnified region showing the grid around the horizontal centerline of the domain; **d** a magnified region around the horizontal centerline showing the kinks in the generated isotherms

displayed in Fig. 5.16a, the velocity and temperature distributions within the enclosure are obtained numerically using a finite volume method. Generated isotherms are displayed in Fig. 5.16b. By carefully inspecting Fig. 5.16b small kinks can be seen in the region around the horizontal centerline of the domain. The region around the kinks is magnified and the grid and isotherms in that region are displayed in Fig. 5.16c, d, respectively. The kinks in the isotherms are clear in the magnified plot and are due to the use of the grid with slope discontinuity causing a discretization error that cannot be reduced independent of how much the grid is refined. This zero order error does not arise with a cell-vertex scheme. Despite this fact, for a sufficiently smooth grid, the cell-centered arrangement can attain accuracy of order two or higher.

The other issue that affects cell centered FVM is the treatment of non-orthogonality in the discretization of the diffusion term. This will be covered in detail in Chap. 8, however a brief explanation is useful at this stage.

In the discretization of the diffusion term (Fig. 15.17) the value of the angle θ extending between the unit vector **e** (which is in the direction of the line joining the centroids of the elements C and F) and the unit vector **n** (which is normal to the face shared by the elements C and F) affects the degree of implicitness and hence the robustness of the method applied in the discretization of the diffusion term.

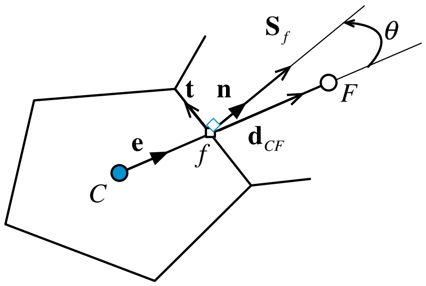


Fig. 15.17 A non-orthogonal element

For discretization, the diffusion term is generally written as

$$\nabla\phi \cdot \mathbf{S} = \underbrace{\nabla\phi \cdot \mathbf{E}}_{\textit{Implicit orthogonal-like contribution}} + \underbrace{\nabla\phi \cdot \mathbf{T}}_{\textit{Explicit non-orthogonal like contribution}} \tag{5.42}$$

The larger the angle θ is, the larger the explicit term will be, and the less robust the discretization method becomes.

To summarize, the vertex-centered scheme with dual elements and the cell-centered scheme are numerically very similar in the interior of a domain for steady state calculations. The only situation where the performance of the vertex-centered scheme is superior to that of the cell-centered scheme is over a distorted grid. In all other situations, it is more advantageous to use the cell-centered arrangement as it leads to a more straightforward implementation in a computer code.

5.8 Implicit Versus Explicit Numerical Methods

As described in Sect. 5.1, once the number of integration points and the type of linearization are defined, the cell-centered finite volume discretization method results in a set of equations with the values of the dependent variables at the cell centers as unknowns. The way these unknowns are organized and solved, classifies the adopted computational approach. Numerical solution schemes are often referred to as being either explicit or implicit.

An explicit numerical method is one in which the dependent variables are computed directly via already known values. In this case any discretization operator can be directly evaluated based on the actual variable values.

On the other hand, a numerical method is said to be implicit when the dependent variables are treated as unknowns and assembled to form a coupled set of equations which are then solved via special numerical tools using either a direct or an iterative solution algorithm.

The conservation equations dealt with in computational fluid dynamics are nonlinear and the implicit approach is most often preferred over the explicit method for solving them. Once the proper discretization and linearization are performed, the last step is to solve the set or system of algebraic equations to get the solution. Solving the corresponding system with a direct equation solver, as discussed in a previous chapter, is not feasible. The iterative approach being more economical, is the most widely, if not the only one, used. In difference with a direct approach, starting from an initial guess an iterative method progresses toward solution by using results obtained at the end of an iteration as the initial guess for the following iteration. The sequence of events is said to be converging if the intermediate computed solution is approaching the final solution, otherwise it is said to be diverging. The process is general and is used whether solving for one time step in a transient problem or for a final solution in a steady state problem. In fact adopting an iterative approach to obtain a steady state solution is computationally cheaper than marching in time until steady state is reached. Additional details about matrix iterative solvers will be covered in Chap. 10.

5.9 The Mesh Support

Now that the basis of the numerics of the FVM and its properties have been presented, it is worth outlining the needed mesh support. In all preceding derivations it has been implicitly assumed that certain geometric and topological information is readily available. Even though the description of the finite volume mesh will be the subject of the next two chapters, based on the covered material so far, it has become possible to draw a simplified list of its characteristics.

The application of the FVM as a numerical discretization technique necessitates a mesh support that provides a range of information both geometric and topological, as described next.

For an element, the needed information includes: its index, its centroid, a list of bounding faces, and a list of neighboring elements. For a face, information is needed about its index, its centroid, its surface vector, a list of neighboring elements (2 for an interior face and one for a boundary face), in addition to a list of vertices that defines it. Also needed is information about the boundaries of the computational domain, i.e., the boundary faces that define each boundary patch.

Another issue to be resolved is the orientation of the normal vector to an element face. In the above derivations it was assumed that the normal to all faces of the element were pointing outward. The normal vectors to faces in a computational domain cannot be all pointing outward. Generally any element will have some of its faces with their normal vectors pointing outward while for the remaining faces they will be pointing inward. This indicates that a proper account should be made of the face sign.

Why and how the above is defined and used will be the focus of the next chapter.

5.10 Computational Pointers

Throughout this book uFVM, a Matlab®-based finite volume CFD educational code, and openFOAM® [17], an open source finite volume code capable of solving industrial type problems, will be used to illustrate implementation details and concretize various numerical procedures. Generally, comments about implementation techniques, adopted methods, and data structure in these two codes will be added, unless otherwise stated, in the Computational Pointers's section of every chapter to follow. Moreover, a number of uFVM test cases (testDiffusion, testAdvection, testFlow, testSource, etc.) are available for the reader and can be downloaded from the book website at the URL address mentioned earlier.

5.10.1 uFVM

The computer program uFVM is an unstructured three dimensional finite volume code that was developed for academic purposes. It is written in Matlab® and as such lends itself substantially to any type of dissection by the user. Moreover, the Matlab® environment allows users to unroll the various data structures adopted in the code at any time during the running of a case. Furthermore because it is intended for academic and teaching use, clarity of coding was high on the priority list while devising the implementation details of the various algorithms and schemes. Still all of its numerics and algorithms are similar in many ways to those used in industrial oriented CFD codes and thus uFVM is quite useful as a guide for anyone interested in developing a CFD code.

The discretized system of equations in uFVM is stored in a matrix format such that each row represents the discretized equation in one element, with the off-diagonal coefficients representing the mutual influence of neighboring elements. The computational representation of this matrix usually takes into account the fact that each row will contain only a small number of non-zero elements, specifically in each row there will be as many non-zero elements as neighbors to the element associated with the row.

For the example shown in Fig. 5.18, the mesh consists of 7 elements, and element 3 has four neighbors. Thus the discretized equation for element 3 will include 4 coefficients in addition to the diagonal coefficient. It is clear why for large meshes many of the non-diagonal elements are zero.

In both uFVM and as is shown later in OpenFOAM®, and indeed in all CFD codes, a sparse matrix is used to store the various coefficients of the resulting system of equations. In uFVM, the matrix **A** is stored in an array of arrays, where the first element of each row represents the diagonal element, while the remaining elements of the array store the non-diagonal coefficients. The right hand side of the system is stored in a separate array **B**. This is illustrated in Fig. 5.18.

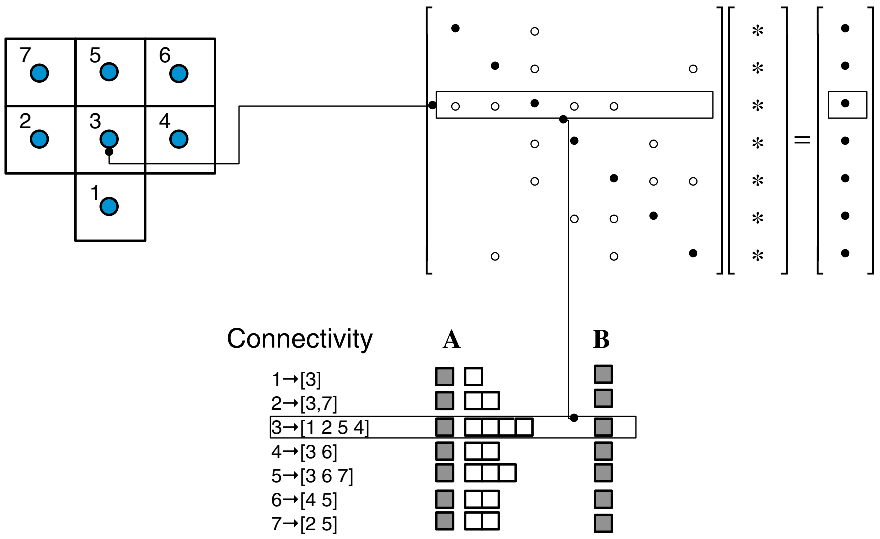


Fig. 5.18 Coefficients for an element equation with element connectivity

5.10.2 OpenFOAM®

OpenFOAM® (Open source Field Operation And Manipulation) is an object-oriented C++ framework that can be used to build a variety of computational solvers for problems in continuum mechanics with a focus on finite volume discretization. OpenFOAM® also includes several ready solvers, utilities, and applications that can be directly used. At the core of these libraries are a set of object classes that allow the programmer to manipulate meshes, geometries, and discretization techniques at a high level of coding. Tables 5.1, 5.2, 5.3 and 5.4 present a list of the main OpenFOAM® classes and their functions. These classes represent the basic bricks for the development of OpenFOAM® based applications and utilities. They enable programmers constructing a variety of algorithms while allowing for extensive code re-use.

Another characteristic of OpenFOAM® is its use of operator overloading that allows algorithms to be expressed in a natural way. For example, the discretization of the transport equation for a generic scalar ϕ given by

Table 5.1 Numerics and discretization

Objects	Type of data	OpenFOAM® Class
Interpolation	Differencing schemes	surfaceInterpolation<template>
Explicit discretization: differential operator	ddt, div, grad, curl	fvc::
Implicit discretization: differential operator	ddt, d2dt2, div, laplacian	fvm::

Table 5.2 Computational domain

Objects	Type of data	OpenFOAM® Class
Variables	Primitive variables	scalar, vector, tensor
Mesh components	Point, face, cell	point, face, cell
Finite volume mesh	Computational mesh	fvMesh, polyMesh
Time	Time database	Time

Table 5.3 Field operation

Objects	Type of data	OpenFOAM® Class
Field	List of values	Field<template>
Dimensions	Dimension set up	dimensionSet
Variable field	Field + mesh + boundaries + dimension	GeometricField<template>
Algebra	+, −, pow, = , sin, cos…	field operators

Table 5.4 Linear equation systems and linear solvers

Objects	Type of data	OpenFOAM® Class
Sparse matrix	Matrix coefficients and manipulation	lduMatrix, fvMatrix
Iterative solver	Iterative matrix solvers	lduMatrix::solver
Preconditioner	Matrix preconditioner	lduMatrix::preconditioner

$$\underbrace{\frac{\partial}{\partial t}(\phi)}_{unsteady\,term} + \underbrace{\nabla\cdot(\mathbf{v}\phi)}_{convection\,term} = \underbrace{\nabla\cdot\left(D^{\phi}\nabla\phi\right)}_{diffusion\,term} + \underbrace{P\phi - C}_{source\;and\;sink\,term} \tag{5.43}$$

is basically written in OpenFOAM® as (Listing 5.1)

```
(
    fvm::ddt(phi)
  + fvm::div(mDot, phi)
  - fvm::laplacian(Dphi, phi)
 ==
    fvm::Sp(P,phi)
   -fvc::(C)
);
```

Listing 5.1 Script for solving a simple transport equation using OpenFOAM®

The fvm::div operator for example, takes the convective flux as a coefficient field defined over the faces of the control elements and phi as the variable field defined over the cell centroids, and returns a system of equations including a LHS matrix and a RHS source that represent the discretization of the convection operator. These LHS matrices and RHS vectors are generated for each of the operators and then added or subtracted as needed to yield the final system of equations that

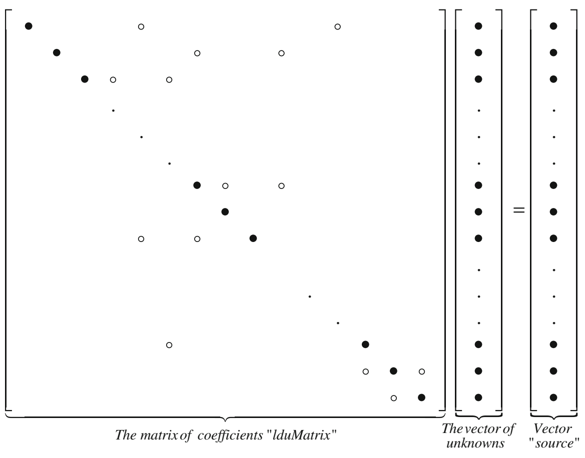


Fig. 5.19 The matrix “lduMatrix” and the vector “source” in which the coefficients and sources are stored

represent the discretized set of algebraic equations defined over each element of the computational domain.

The namespaces fvm:: and fvc:: allow for the evaluation of a variety of operators implicitly and explicitly, respectively.

The explicit operator fvc, named “finite volume calculus”, returns an equivalent field based on the actual field values. For example the operator fvc::div(ϕ) returns an equivalent geometricField in which each cell contains the value of the divergence of the variable (ϕ).

The fvm implicit operator, instead, defines the implicit finite volume discretization in terms of matrices of coefficients. For example fvm::laplacian(ϕ) returns an fvMatrix in which all the coefficients are based on the finite volume discretization of the laplacian.

The role of the fvm:: and fvc:: operators is to construct the LHS and RHS of a system of equations representing the discretized form of Eq. (5.43) over each element in the mesh. The discretization process yields a system of equations that can be represented in the matrix form shown in Fig. 5.19.

Each row of the system represents the discretized equation in one element, with the off-diagonal coefficients representing the mutual influence of the neighboring elements.

In OpenFOAM® the matrix of coefficients is stored in a class named lduMatrix and the specialized fvMatrix. The chosen storage format of the coefficients is based on the ldu sparse format, in which all coefficients are stored in three main arrays: diagonal, upper, and lower. The diagonal coefficients are thus stored in one array accessed by diag() function with its dimension being equal to the number of elements in the computational domain. The upper and lower coefficients are stored each in an array as shown in Fig. 5.20. The source or the right hand side is stored under a specific vector called source with its dimension being also equal to the number of cells over the computational domain.

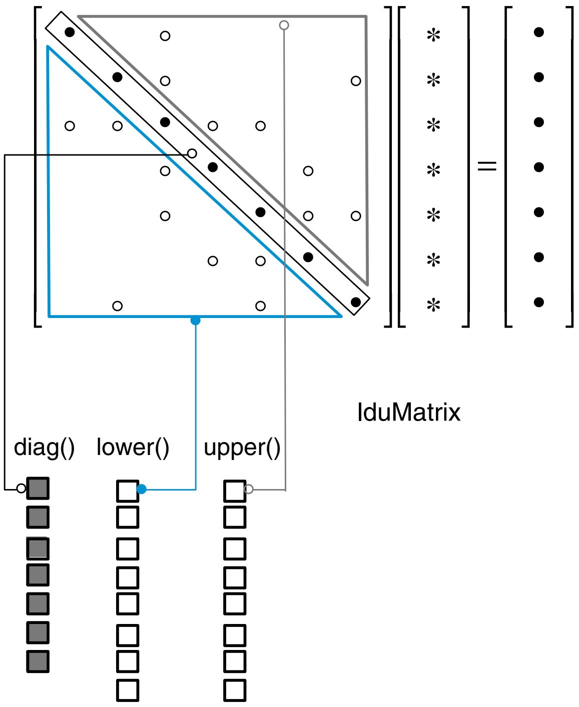


Fig. 5.20 The ldu sparse matrix storage (lduMatrix)

It is worth noting that despite defining the fvMatrix class as a "template" the vector specialization of the class does not imply a block coupled form of the matrix itself but just a vector of scalar equations that are solved in a standard segregated approach. This is confirmed by the code defined inside the fvMatrix class shown in Listing 5.2 with its member function, which returns the diagonal vector coefficients of the matrix, only returning as object a scalar field independently of the template type.

```
template<class Type>
class fvMatrix
:
    public refCount,
    public lduMatrix
{
…

//- Return the matrix scalar diagonal
  tmp<scalarField> D() const;
…

//- Return the central coefficient
tmp<volScalarField> A() const;
```

Listing 5.2 Excerpt of the code defined inside the fvMatrix class returning as object a scalar field

More details on the coefficients' storage techniques used in uFVM and OpenFOAM® will be presented in Chap. 7.

5.11 Closure

In this chapter a general overview of the FVM was presented. The discretization process was shown to involve two distinct steps, with step 1 resulting in a set of semi-discretized equations. The chapter also discussed some guiding principles for the discretization process, which guarantee that the resulting discretized equation will possess some desirable attributes. Before embarking onto a detailed description of the numerical techniques that will be used in step 2 of the discretization process, the focus of the next chapter will be on the discretization of the computational domain and some related issues.

5.12 Exercises

Exercise 1
A critical tool used with OpenFOAM® is Doxygen [18]. It is an open source software that generates automatic documentation from annotated C++ sources. Doxygen uses the UML (http://www.uml-diagrams.org/class-reference.html) graph formalism and is capable of visually displaying the relations between the various classes, inheritance diagrams, and collaboration diagrams, which are all generated automatically as easily browsed HTML documents. Two options are available to access Doxygen. The first is to generate the documents by direct compilation on the local machine (the main files are placed in $WM_PROJECT_DIR/doc/Doxygen). The second option is to access Doxygen on the web at the address http://www.openfoam.org/docs/cpp/. This latest option yields always up to date documentation. To start, either open the browser at the address http://www.openfoam.org/docs/cpp/ or open the file $WM_PROJECT_DIR/doc/Doxygen/html/index.html. Once opened, click on the *Classes* button that appears in the menu in the upper part of the screen. Then click on the *Class Index* button. Once clicked, a list ordered alphabetically of all classes defined in OpenFOAM® is displayed. The inheritance diagram of any class along its public member functions and derived classes are obtained by simply clicking on the class name appearing in the list. Doxygen is used to quickly browse all source files as well as checking all the properties of the classes and namespaces. Basically the most common information to look for are class inheritances and public member functions as well as public class constructors.

(a) Using the Doxygen documentation find the class definition of the following type of data:

Foam::face
Foam::cell
Foam::fvMesh
Foam::fvMatrix

(b) Find the parent classes for each of the classes of (a).
(c) Find the public constructors for each of the classes defined in (a).
(d) Find all the public member functions defined in each of the classes of (a).
(e) Find all the public member functions defined in the parent class of each of the classes of (a).

Exercise 2
Run testDiffusion (which is available as a test case with uFVM) for one iteration, then

(a) Use 'cfdGetCoefficients' to get the global matrix data structure and compare it to that shown in Fig. 5.18.
(b) Use cfdGetMesh to get a reference to the Mesh object and find the definition of a face, a cell and a vertex.

References

1. Blazek J (2005) Computational fluid dynamics: principles and applications. Elsevier, Amsterdam
2. Ferziger JH, Peric M (2002) Computational methods for fluid dynamics, 3rd edn. Springer, Berlin
3. Versteeg HK, Malalasekera W (2007) An introduction to computational fluid dynamics: the finite volume method. Prentice Hall, New Jersey
4. Courant R, Friedrichs KO, Lewy H (1928) Über die partiellen Differenzengleichungen der Mathematischen Physik. Math Ann 100:32–74 (English translation, with commentaries by Lax, P.B., Widlund, O.B., Parter, S.V., in IBM J. Res. Develop. 11 (1967))
5. Crank J, Nicolson P (1947) A practical method for numerical integration of solution of partial differential equations of heat-conduction type. Proc Cambridge Philos Soc 43:50–67
6. Clough RW (1960) The finite element method in plane stress analysis. Proceedings of second ASCE conference on electronic computation, Pittsburg, Pennsylvania, 8:345–378
7. Launder BE, Spalding DB (1972) Mathematical models of turbulence. Academic Press, Massachusetts
8. Patankar SV (1967) Heat and mass transfer in turbulent boundary layers, Ph.D. Thesis, Imperial College, London University, UK
9. Gosman AD, Pun WM, Runchal AK, Spalding DB, Wolfshtein M (1969) Heat and mass transfer in recirculating flows. Academic Press, London
10. Runchal AK (1969) Transport processes in steady two-dimensional separated flows. Ph.D. Thesis, Imperial College of Science and Technology, London, UK

11. Runchal AK, Wolfshtein M (1969) Numerical integration procedure for the steady-state Navier-Stokes equations. Mech Eng Sci II 5:445–453
12. Rhie CM, Chow WL (1983) A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation. AIAA J 21:1525–1532
13. Lilek Z, Peric M (1995) A fourth-order finite volume method with collocated variable arrangement. Comput Fluids 24(3):239–252
14. Leonard BP (1988) Universal limiter for transient interpolation modeling of the advective transport equations: the ultimate conservative difference scheme. NASA Technical Memorandum 100916, ICOMP-88-11
15. Baliga BR, Patankar SVA (1983) Control volume finite-element method for two-dimensional fluid flow and heat transfer. Numer Heat Transf 6:245–261
16. Vaughan RV (2009) Basic control volume finite element methods for fluids and solids. IISc research monographs series, IISc Press
17. OpenFOAM (2015) Version 2.3.x. http://www.openfoam.org
18. OpenFOAM Doxygen (2015) Version 2.3.x. http://www.openfoam.org/docs/cpp/