Chapter 16
Fluid Flow Computation: Compressible Flows

Abstract The previous chapter presented the methodology for solving incompressible flow problem using pressure based algorithms. In this chapter these algorithms are extended to allow for the simulation of compressible flows in the various Mach number regimes, i.e., over the entire spectrum from subsonic to hypersonic speeds. While incompressible flow solutions do not generally require solving the energy equation, compressibility effects couple hydrodynamics and thermodynamics necessitating the simultaneous solution of the continuity, momentum, and energy equations. The dependence of density on pressure and temperature, a relation expressed via an equation of state, further complicates the velocity-pressure coupling present in incompressible flows. The derivation of the pressure correction equation now involves a density correction that introduces to the equation a convection-like term, in addition to the diffusion-like term introduced by the velocity correction. Another difficulty is introduced by the complex boundary conditions that arise in compressible flow problems. Details on resolving all these issues are presented throughout this chapter.

16.1 Historical

Computational Fluid Dynamics methods have been traditionally classified into two families denoted by density-based and pressure-based. Specifically density-based methods have historically dominated the simulation of transonic and supersonic flows usually encountered in the aeronautics industry, and were well-established when the SIMPLE algorithm was first instigated. SIMPLE, a pressure-based method, was initially developed to address this shortcoming and was quite efficient in resolving incompressible and low Mach number flows.

Early on, efforts were directed towards extending the operation of each of these approaches to flow regimes customarily dominated by the other. Harlow and Amsden [1, 2] were amongst the earliest to simulate fluid flow at all speeds. In their work, the use of pressure as a main variable in preference to density was presented an

F. Moukalled et al., *The Finite Volume Method in Computational Fluid Dynamics*,
Fluid Mechanics and Its Applications 113, DOI 10.1007/978-3-319-16874-6_16

advantage since its variations remained finite irrespective of the Mach number value. Nonetheless it was the work of Patankar [3] that provided a clear resolution to this problem, and allowed for SIMPLE-based methods to genuinely develop into methods capable of resolving fluid flow at all speeds [4–13]. The critical development was the reformulation of the pressure equation to include density and velocity correction such that the type of the equation changed from purely elliptic for incompressible flows to hyperbolic in transonic and supersonic compressible flows [14, 15]. This allowed the SIMPLE-family of methods to seamlessly solve flow problems across the entire Mach number spectrum, with pressure playing the dual role of affecting density in the limit of high Mach compressible flow and velocity in the limit of incompressible flow [16], in order to enforce mass conservation.

16.2 Introduction

An important advantage of the pressure-based approach is its ability to resolve fluid flows in the various Mach number regimes without any artificial treatment to promote convergence and stabilize computations. This ability of the pressure based method is due to the dual role the pressure plays in compressible flows [17], which can best be described by considering the following two extreme cases:

1. At very low Mach numbers, the pressure gradient needed to establish the flow field through momentum conservation is so small that it does not significantly influence the density, and the flow can be considered to be incompressible. Hence, density and pressure in addition to density and velocity are very weakly related indicating that variations in density are not sensitive to variations in velocity. In this case the continuity equation can no longer be considered as an equation for density, rather, it acts as a constraint on the velocity field.
2. At hypersonic speeds, changes in velocity become relatively small as compared to the magnitude of the velocity indicating that variations in pressure do significantly affect density. Consequently, the pressure now acts on density alone through the equation of state to satisfy mass conservation [16, 18] and the continuity equation can be viewed as the equation for density.

The above limiting cases highlight the dual role played by pressure in compressible flow situations. It clearly shows that pressure acts on both the density field through the equation of state and the velocity field via the gradient in the momentum equation to enforce mass conservation. This dual role explains the success of the pressure-based approach to predict fluid flow at all speeds. This fact however did not deter workers in the density based track from using the artificial compressibility technique [19] to develop methods capable of solving fluid flow at all speeds. To overcome degradation in performance due to the stiff matrices encountered in these methods, preconditioning of the resulting stiff matrices was introduced and several methods [20, 21] using this technique have recently appeared in the literature.

Similarly several pressure-based methods [22] for predicting fluid flow at all speeds following either a staggered grid approach [23] or a collocated variable formulation have been developed. While in some methods primitive variables were used, others employed the momentum components as dependent variables. Some workers adopted the stream-wise directed density-retardation concept, which is controlled by Mach-number-dependent monitor functions [24, 25], to account for the hyperbolic character of the conservation laws in the transonic and supersonic regimes. Other techniques used the first order upwind scheme for evaluating the density at the control volume faces at high Mach number values and the central difference scheme at low values [26].

This chapter extends the collocated pressure-based technique developed in the previous chapter allowing the simulation of fluid flows at all Mach number values. The adopted method is easy to implement, highly accurate, and does not require any explicit addition of damping terms to improve robustness or to properly resolve shock waves.

16.3 The Conservation Equations

The conservation equations for solving compressible flow problems include the continuity, momentum, and energy equations. For a Newtonian fluid behaving as an ideal gas, these equations can be expressed as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{16.1}$$

$$\frac{\partial}{\partial t}[\rho \mathbf{v}] + \nabla \cdot \{\rho \mathbf{v}\mathbf{v}\} = \nabla \cdot \{\mu \nabla \mathbf{v}\} - \nabla p + \nabla \cdot \left\{\mu (\nabla \mathbf{v})^{\mathrm{T}}\right\} - \frac{2}{3}\nabla(\mu \nabla \cdot \mathbf{v}) + \mathbf{f}_b \tag{16.2}$$

$$\frac{\partial}{\partial t}\left(\rho c_p T\right) + \nabla \cdot \left[\rho c_p \mathbf{v} T\right] = \nabla \cdot [k \nabla T] + \rho T \frac{D c_p}{D t} + \frac{D p}{D t} - \frac{2}{3}\mu \Psi + \mu \Phi + \dot{q}_V \tag{16.3}$$

where the form of the energy equation adopted here is the one expressed in terms of temperature. The above set of equations should be appended by an equation of state relating density to pressure and temperature, i.e., $\rho = \rho(p, T)$, which for an ideal gas is given by

$$\rho = \frac{p}{RT} \tag{16.4}$$

where R is the gas constant.

In the derivations to follow, superscript n refers to values used at the beginning of an iteration, superscript * indicates values updated once during an iteration, and superscript ** refers to values updated twice during the same iteration.

16.4 Discretization of the Momentum Equation

The discretized momentum equation, Eq. (16.2), over the control volume C shown in Fig. 16.1 is similar to its incompressible form given in Chap. 15. The only two differences are related to the interpolation of density to the interface and the additional term $-(2/3)\nabla(\mu\nabla\cdot\mathbf{v})$ involving the bulk viscosity. Starting with the first difference, the density in compressible flows is no longer constant and since it is stored at the control volume centroids it has to be interpolated to find its value at the control volume faces where it is needed for computing the mass flow rate. The use of a linear interpolation profile (central difference) causes oscillation at high speeds. Thus a bounded upwind biased scheme should be used. Any of the bounded convective schemes presented in Chaps. 11 and 12 can be adopted for that purpose.

The second difference is the additional term involving $\nabla(\mu\nabla\cdot\mathbf{v})$. This term has not been discretized so far and its discretized form is obtained by making use of Eq. (2.85) based on which the volume integral of the gradient of a scalar quantity is

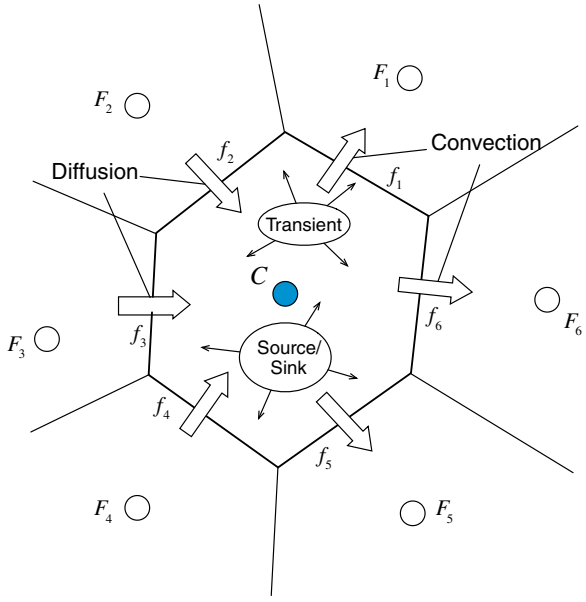


Fig. 16.1 A schematic of a control volume C with its neighbors

transformed into a surface integral and then into a summation of fluxes over the faces of the control volume according to

$$\int_{V_C} [\nabla(\mu\nabla\cdot\mathbf{v})]dV = \int_{\partial V_C} (\mu\nabla\cdot\mathbf{v})d\mathbf{S} = \sum_{f\sim nb(C)} (\mu\nabla\cdot\mathbf{v})_f\mathbf{S}_f \tag{16.5}$$

The divergence of the velocity vector at the faces is computed as

$$(\mu\nabla\cdot\mathbf{v})_f = \mu_f^{(n)}\left[\left(\frac{\partial u}{\partial x}\right)_f^{(n)} + \left(\frac{\partial v}{\partial y}\right)_f^{(n)} + \left(\frac{\partial w}{\partial z}\right)_f^{(n)}\right] \tag{16.6}$$

where the gradient of $\phi = u, v$ or w is interpolated linearly to the face

$$\left(\frac{\partial\phi}{\partial x}\right)_f^{(n)} = g_C\left(\frac{\partial\phi}{\partial x}\right)_C^{(n)} + g_F\left(\frac{\partial\phi}{\partial x}\right)_F^{(n)} \tag{16.7}$$

The final discretized form of the momentum equation is given by Eq. (15.70) with its coefficients given by Eq. (15.71) with the term $-(2/3)\sum_{f\sim nb(C)}(\mu\nabla\cdot\mathbf{v})_f\mathbf{S}_f$ added to the source term in that equation.

As for incompressible flow problems, the algebraic equations are under relaxed and written in the form of Eq. (15.78), which is suitable for the derivation of the pressure correction equation.

16.5 The Pressure Correction Equation

The pressure correction equation for compressible flows is obtained by a simple extension of that for incompressible flows. The difference is related to variations in density which are accounted for by defining a density correction field ρ' and relating it to the pressure-correction field through a pressure-density relation. This, however, yields substantial differences in the treatment of boundary conditions, as will be explained later in the chapter.

For an ideal gas the relation between pressure and density is written as

$$\rho RT = p \tag{16.8}$$

Using this relation, an equation relating density correction to pressure correction can be derived by expanding Eq. (16.8) via a Taylor series to yield

$$\rho|_{(p^{(n)}+p')} = \rho|_{(p^{(n)})} + \frac{\partial\rho}{\partial p}\,p' = \rho^* + \rho' \Rightarrow \rho' = \frac{\partial\rho}{\partial p}p' = \frac{1}{RT}p' = C_\rho p' \tag{16.9}$$

The corrected pressure, density, velocity, and mass flow rate fields are defined as

$$\begin{aligned} p &= p^{(n)} + p' \\ \rho &= \rho^* + \rho' \\ \mathbf{v} &= \mathbf{v}^* + \mathbf{v}' \\ \dot{m} &= \dot{m}^* + \dot{m}' \end{aligned} \tag{16.10}$$

and the semi-discretized continuity equation can be written in terms of the correction fields as

$$\frac{\left(\rho_C^* + \rho_C' - \overset{o}{\rho_C}\right)}{\Delta t} V_C + \sum_{f \sim nb(C)} \left(\dot{m}_f^* + \dot{m}_f'\right) = 0 \tag{16.11}$$

where

$$\begin{aligned} \dot{m}_f &= \left(\rho_f^* + \rho_f'\right)\left(\mathbf{v}_f^* + \mathbf{v}_f'\right) \cdot \mathbf{S}_f \\ &= \underbrace{\rho_f^* \mathbf{v}_f^* \cdot \mathbf{S}_f}_{\dot{m}_f^*} + \underbrace{\rho_f^* \mathbf{v}_f' \cdot \mathbf{S}_f + \rho_f' \mathbf{v}_f^* \cdot \mathbf{S}_f + \rho_f' \mathbf{v}_f' \cdot \mathbf{S}_f}_{\dot{m}_f'} \end{aligned} \tag{16.12}$$

The second order correction term $\rho_f' \mathbf{v}_f' \cdot \mathbf{S}_f$ is usually neglected because it is considerably smaller than other terms. This approximation does not influence the convergence rate except during the first few iterations of the solution process. In addition, the final solution is not affected, since at the state of convergence, the correction fields vanish.

Using the Rhie-Chow interpolation for $\mathbf{v}_f^*$ and $\mathbf{v}_f', \dot{m}_f^*$ and $\dot{m}_f'$ are respectively expressed as

$$\dot{m}_f^* = \underbrace{\rho_f^* \overline{\mathbf{v}_f^*} \cdot \mathbf{S}_f - \rho_f^* \overline{\mathbf{D}_f^{\mathbf{v}}} \left(\nabla p_f^{(n)} - \overline{\nabla p_f^{(n)}}\right) \cdot \mathbf{S}_f}_{\rho_f^* \mathbf{v}_f^* \cdot \mathbf{S}_f} \tag{16.13}$$

and

$$\begin{aligned} \dot{m}_f' &= \underbrace{\rho_f^* \overline{\mathbf{v}_f'} \cdot \mathbf{S}_f - \rho_f^* \overline{\mathbf{D}_f^{\mathbf{v}}} \left(\nabla p_f' - \overline{\nabla p_f'}\right) \cdot \mathbf{S}_f}_{\rho_f^* \mathbf{v}_f' \cdot \mathbf{S}_f} + \underbrace{\left(\frac{\dot{m}_f^*}{\rho_f^*} \cdot \mathbf{S}_f\right) C_{\rho,f} p_f'}_{\rho_f' \mathbf{v}_f \cdot \mathbf{S}_f} \\ &= -\rho_f^* \overline{\mathbf{D}_f^{\mathbf{v}}} \nabla p_f' \cdot \mathbf{S}_f + \left(\frac{\dot{m}_f^*}{\rho_f^*} \cdot \mathbf{S}_f\right) C_{\rho,f} p_f' + \left(\rho_f^* \overline{\mathbf{v}_f'} \cdot \mathbf{S}_f + \rho_f^* \overline{\mathbf{D}_f^{\mathbf{v}}}\, \overline{\nabla p_f'} \cdot \mathbf{S}_f\right) \\ &= -\rho_f^* \overline{\mathbf{D}_f^{\mathbf{v}}} \nabla p_f' \cdot \mathbf{S}_f + \left(\frac{\dot{m}_f^*}{\rho_f^*} \cdot \mathbf{S}_f\right) C_{\rho,f} p_f' - \underline{\rho_f^* \overline{\mathbf{H}_f}[\mathbf{v}'] \cdot \mathbf{S}_f} \end{aligned} \tag{16.14}$$

where the second order term is neglected. Note the substitution of ρ_f' with $C_{\rho,f} p_f'$.

The underlined term in Eq. (16.14) presents the same difficulties as for the incompressible algorithm and is usually dropped from the equation. Neglecting this term, the correction to the mass flow rate becomes

$$\dot{m}'_f = -\rho^*_f \overline{\mathbf{D}^{\mathbf{v}}_f} \nabla p'_f \cdot \mathbf{S}_f + \left(\frac{\dot{m}^*_f}{\rho^*_f} \cdot \mathbf{S}_f \right) C_{p,f} p'_f \tag{16.15}$$

where the first term on the right hand side of Eq. (16.15) is similar to that arising in incompressible flow while the second term is the new density correction contribution. This second term is important as it transforms the pressure correction equation from an elliptic equation to a hyperbolic one capable of resolving shock waves that may arise at supersonic and hypersonic speeds. This allows the compressible SIMPLE algorithm to be used for predicting fluid flow at all speeds without the need for any special preconditioning.

More insight can be gained through a simple normalization procedure whereby Eq. (16.15) is divided by $\left(\dot{m}^*_f \cdot \mathbf{S}_f\right) C_{p,f} \Big/ \rho^*_f$ yielding a weighting factor of 1 for the p'_f term, and a weighting factor proportional to $1/(M^2)$ (where M is the Mach number of the flow) for the $\nabla p'_f$ term, i.e.,

$$\dot{m}'_f = -\frac{RT\left(\rho^*_f\right)^2 \overline{\mathbf{D}^{\mathbf{v}}_f}}{\left(\dot{m}^*_f \cdot \mathbf{S}_f\right)} \nabla p'_f \cdot \mathbf{S}_f + p'_f \tag{16.16}$$

For flows at low Mach number values, the $\nabla p'$ correction term dominates returning the equation to an elliptic form as in the incompressible case. On the other hand, for flows at very high Mach number values the p'_f correction term can no longer be neglected giving a hyperbolic character to the correction equation. This combined behavior allows the prediction of fluid flow at all speeds.

Substitution of Eq. (16.14) in the continuity equation, Eq. (16.11), yields the compressible form of the pressure correction equation and is written as

$$\begin{aligned} \frac{V_C}{\Delta t} C_\rho p'_C + \sum_{f \sim nb(C)} \left\{ -\rho^*_f \overline{\mathbf{D}^{\mathbf{v}}_f} \nabla p'_f \cdot \mathbf{S}_f + \left(\frac{\dot{m}^*_f}{\rho^*_f} \right) C_\rho p'_f \right\} \\ = -\left(\frac{\rho^*_C - \overset{\circ}{\rho}_C}{\Delta t} V_C + \sum_{f \sim nb(C)} \dot{m}^*_f \right) + \underline{\sum_{f \sim nb(C)} \overline{\mathbf{H}_f}[\mathbf{v}'] \cdot \mathbf{S}_f} \end{aligned} \tag{16.17}$$

Again the treatment of the underlined term yields the different variants of the SIMPLE family of algorithms. Dropping the underlined term, the pressure correction equation for the SIMPLE algorithm is obtained as

$$\underbrace{\frac{V_C C_\rho}{\Delta t} p'_C}_{transient-like\ term} + \underbrace{\sum_{f\sim nb(C)} C_\rho \left(\frac{\dot{m}^*_f}{\rho^*_f}\right) p'_f}_{convection-like\ term} - \underbrace{\sum_{f\sim nb(C)} \rho^*_f \mathbf{D}^{\mathbf{v}}_f (\nabla p')_f \cdot \mathbf{S}_f}_{diffusion-like\ term}$$
$$= \underbrace{-\frac{\left(\rho^*_C - \overset{\circ}{\rho}_C\right)}{\Delta t} V_C - \sum_{f\sim nb(C)} \dot{m}^*_f}_{source-like\ term} \tag{16.18}$$

Equation (16.18) can be obtained directly by substituting Eq. (16.15) in Eq. (16.11).

It is worth stressing that the convection-like term came about naturally during the derivation of the pressure correction equation and its presence is critical for the ability of the algorithm to resolve flows at all speeds. Moreover, for flows at high Mach number values, density correction is convected (i.e., exhibiting a hyperbolic behavior) and the mathematical operator describing this phenomenon is the first order divergence operator. Thus, contrary to incompressible flows where only the diffusion-like term is present implying that the pressure correction equation exhibits an elliptic behavior, pressure correction solutions of the form $p' + C$ can no longer satisfy the equation. This indicates that while for incompressible flows any pressure value can be set as a boundary condition without affecting the solution, for compressible flows it is important to define the exact value of pressure at the boundaries because the chosen value will affect the final solution.

It should also be noted that because at convergence the correction field is zero, the order of the scheme used to discretize the convection-like term is of no consequence on the accuracy of the final results. However, this is not the case for $\dot{m}^*_f$ where the use of high order schemes in its evaluation does improve the shock capturing properties of the algorithm. Thus, to enhance robustness, it is helpful to use an upwind scheme for the discretization of the convection like term. Further, neglecting the non-orthogonal contribution of the diffusion-like term as explained in Chap. 15, the pressure correction equation and its coefficients become

$$\begin{aligned}
&a^{p'}_C p'_C + \sum_{F\sim NB(C)} a^{p'}_F p'_F = b^{p'}_C \\
&a^{p'}_F = -\rho^*_F \mathcal{D}_f - \left\| -\dot{m}^*_f, 0 \right\| \frac{C_{\rho,f}}{\rho^*_f} \\
&a^{p'}_C = \frac{V_C C_\rho}{\Delta t} + \sum_{f\sim nb(C)} \left(\frac{C_{\rho,f}}{\rho^*_f} \left\| \dot{m}^*_f, 0 \right\| \right) + \sum_{f\sim nb(C)} \rho^*_f \mathcal{D}_f \\
&b^{p'}_C = -\left(\frac{\left(\rho^*_C - \overset{\circ}{\rho}_C\right)}{\Delta t} V_C + \sum_{f\sim nb(C)} \dot{m}^*_f \right) + \underbrace{\sum_{f\sim nb(C)} \rho^*_f \left(\mathbf{D}^{\mathbf{v}}_f \nabla p'_f \right) \cdot \mathbf{T}_f}_{non-orthogonal\ term\ usually\ neglected}
\end{aligned} \tag{16.19}$$

Following the calculation of the pressure-correction field p' by solving Eq. (16.19) the velocity, pressure, density, and mass flow rate fields are corrected using the following equations:

$$\mathbf{v}_C^{**} = \mathbf{v}_C^{*} + \mathbf{v}_C' \quad \mathbf{v}_C' = -\mathbf{D}_C^{\mathbf{v}}(\nabla p')_C \tag{16.20}$$

$$p_C^{*} = p_C^{(n)} + \lambda^p p_C' \tag{16.21}$$

$$\rho_C^{**} = \rho_C^{*} + \lambda^\rho C_\rho p_C' \tag{16.22}$$

$$\dot{m}_f^{**} = \dot{m}_f^{*} + \dot{m}_f' \quad \dot{m}_f' = -\rho_f^{**}\overline{\mathbf{D}_f^{\mathbf{v}}}\nabla p_f' \cdot \mathbf{S}_f + \left(\frac{\dot{m}_f^{*}}{\rho_f^{**}} \cdot \mathbf{S}_f\right) C_{\rho,f} p_f' \tag{16.23}$$

where λ^ρ is the under relaxation factor for density.

16.6 Discretization of The Energy Equation

The discretization of the unsteady, convection, and diffusion terms of the energy equation, Eq. (16.3), follows the general procedures described in previous chapters and will not be repeated.

16.6.1 Discretization of the Extra Terms

The focus here will be on the discretization over the control volume C shown in Fig. (16.1), of the new terms appearing on the right hand side of Eq. (16.3). These are specific to the energy equation and have not been handled during the discretization of the general scalar equation. Many of the terms are treated as source terms and evaluated at the centroid of the element during their integration to ensure a second order accurate discretization.

16.6.1.1 The Specific Heat Term

The discretization of the term involving the specific heat proceeds as follows:

$$\int_{V_C} \rho T \frac{Dc_p}{Dt} dV = \rho_C^{**} T_C^{(n)} \left(\frac{Dc_p}{Dt}\right)_C V_C$$
$$= \rho_C^{**} T_C^{(n)} \left[\frac{c_p^{(n)} - c_p^{\circ}}{\Delta t} + u_C^{**} \left(\frac{\partial c_p}{\partial x}\right)_C^{(n)} + v_C^{**} \left(\frac{\partial c_p}{\partial y}\right)_C^{(n)} + w_C^{**} \left(\frac{\partial c_p}{\partial z}\right)_C^{(n)}\right] V_C \tag{16.24}$$

16.6.1.2 The Substantial Derivative Term

The discretization of the substantial derivative of the pressure term is performed as

$$\int_{V_c} \frac{Dp}{Dt} dV = \left(\frac{Dp}{Dt}\right)_C^{*} V_C = \left[\frac{p_C^{*} - p_C^{\circ}}{\Delta t} + u_C^{**} \left(\frac{\partial p}{\partial x}\right)_C^{*} + v_C^{**} \left(\frac{\partial p}{\partial y}\right)_C^{*} + w_C^{**} \left(\frac{\partial p}{\partial z}\right)_C^{*}\right] V_C \tag{16.25}$$

16.6.1.3 The Dissipation Term

The discretized form of the dissipation term involving the bulk viscosity is obtained as

$$\int_{V_C} \mu \Psi dV = \mu_C^{(n)} \Psi_C^{**} V_C = \mu_C^{(n)} \left[\left(\frac{\partial u}{\partial x}\right)_C^{**} + \left(\frac{\partial v}{\partial y}\right)_C^{**} + \left(\frac{\partial w}{\partial z}\right)_C^{**}\right]^2 V_C \tag{16.26}$$

16.6.1.4 The Viscous Dissipation Term

The discretization of the viscous dissipation term is performed in a way similar to the term involving the bulk viscosity and is given by

$$\int_{V_C} \mu \Phi dV = \mu_C^{(n)} \Phi_C^{**} V_C$$
$$= \mu_C^{(n)} \left\{ \begin{array}{l} 2\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial z}\right)^2\right]_C^{**} + \left[\left(\frac{\partial u}{\partial y}\right)_C^{**} + \left(\frac{\partial v}{\partial x}\right)_C^{**}\right]^2 + \\ \left[\left(\frac{\partial u}{\partial z}\right)_C^{**} + \left(\frac{\partial w}{\partial x}\right)_C^{**}\right]^2 + \left[\left(\frac{\partial v}{\partial z}\right)_C^{**} + \left(\frac{\partial w}{\partial y}\right)_C^{**}\right]^2 \end{array} \right\} V_C \tag{16.27}$$

16.6.1.5 The Source/Sink Term

The term involving the heat source/sink per unit volume is discretized as

$$\int_{V_C} \dot{q}_V \, dV = (\dot{q}_V)_C V_C \tag{16.28}$$

The discrete forms of the above terms are substituted into the energy equation to yield the algebraic form of the energy equation as described next.

16.6.2 The Algebraic Form of the Energy Equation

Assuming a first order Euler scheme for the discretization of the unsteady term and a high resolution scheme for the discretization of the convection term applied in the context of a deferred correction approach, the final algebraic form of the energy equation can be written as

$$a_C^T T_C + \sum_{F \sim NB(C)} a_F^T T_F = b_C^T \tag{16.29}$$

where the coefficients are given by

$$\begin{aligned}
a_F^T &= -k_f \frac{E_f}{d_{CF}} - \left\| -\dot{m}_f, 0 \right\| (c_p)_f \\
a_C^T &= \dot{a_C} - \sum_{F \sim NB(C)} a_F^T + \sum_{f \sim nb(C)} \dot{m}_f (c_p)_f + \left\| -a_C^{c_p}, 0 \right\| \\
\dot{a_C} &= \frac{\rho_C (c_p)_C V_C}{\Delta t} \quad \overset{\circ}{a_C} = \frac{\overset{\circ}{\rho_C} \left(\overset{\circ}{c_p} \right)_C V_C}{\Delta t} \quad a_C^{c_p} = \rho_C V_C \left(\frac{Dc_p}{Dt} \right)_C \\
b_C^T &= \sum_{f \sim nb(C)} \left(k_f (\nabla T)_f \cdot \mathbf{T}_f \right) - \sum_{f \sim nb(C)} \dot{m}_f (c_p)_f \left(T_f^{HR} - T_f^U \right) + \overset{\circ}{a_C} \overset{\circ}{T_C} \\
&\quad + T_C \left\| a_C^{c_p}, 0 \right\| + \left[\left(\frac{Dp}{Dt} \right)_C + \mu_C \left(-\frac{2}{3} \Psi_C + \Phi_C \right) + (\dot{q}_V)_C \right] V_C
\end{aligned} \tag{16.30}$$

Similar to other variables, under relaxation of the energy equation is usually required.

16.7 The Compressible SIMPLE Algorithm

The various elements of the collocated compressible SIMPLE algorithm are displayed in Fig. 16.2 and can be summarized as follows:

1. To compute the solution at time $t + \Delta t$, start with the solution at time t for pressure, velocity, density, temperature, and mass flow rate fields $p^{(n)}$, $\mathbf{v}^{(n)}$, $\rho^{(n)}$, $T^{(n)}$, and $\dot{m}^{(n)}$, respectively, as the initial guess.

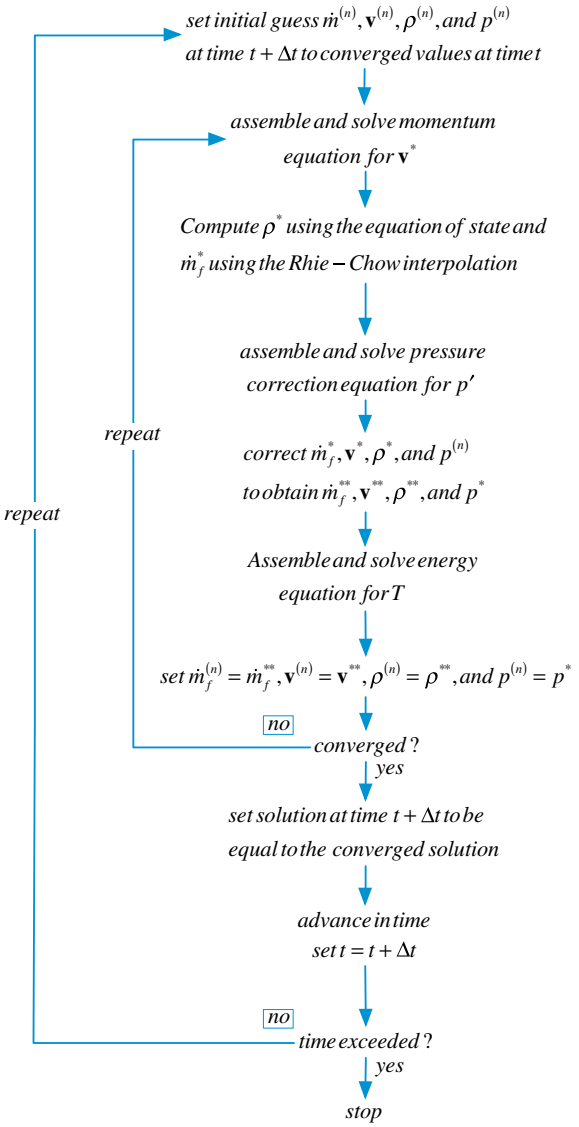


Fig. 16.2 A flow chart of the SIMPLE algorithm for compressible fluid flow

2. Solve the momentum equation given by Eq. (16.2) to obtain a new velocity field $\mathbf{v}^*$.
3. Use the equation of state to calculate a new density field ρ^*.
4. Update the mass flow rate at the control volume faces using the Rhie-Chow interpolation technique (Eq. 16.13) to obtain a momentum satisfying mass flow rate field $\dot{m}^*$.
5. Using the new mass flow rates calculate the coefficients of the pressure correction equation and solve it (Eq. 16.19) to obtain a pressure correction field p'.
6. Update the pressure, density, and velocity fields at the control volume centroids and the mass flow rate at the control volume faces to obtain continuity-satisfying fields using Eqs. (16.20)–(16.23).
7. Solve the energy equation to obtain a new temperature field T^*.
8. Set $\mathbf{v}^{**}, \dot{m}^{**}, \rho^{**}, T^*$, and p^* as the initial guess for velocity, mass flow rate, density, temperature, and pressure.
9. Go back to step 2 and repeat until convergence.
10. Set the solution at time $t + \Delta t$ to be equal to the converged solution.
11. Advance to the next time step by setting the current time t to $t + \Delta t$.
12. Go to step 1 and repeat until the last time step is reached.

16.8 Boundary Conditions

Generally, there is no difference in the implementation of boundary conditions for the momentum equation between incompressible and compressible flow. Therefore the required modifications in the momentum equation are those discussed in Chap. 15 and will not be repeated here. However substantial differences do arise with the pressure correction equation and will form the main subject of this section. The boundary conditions for the energy equation follow the ones described for a general scalar variable ϕ and also will not be repeated here (inlet, outlet, Dirichlet, Van-Neumann, and symmetry conditions).

For a boundary cell, such as the one shown in Fig. 16.3, the continuity equation is written as

$$\frac{\left(\rho_C^* + \rho_C' - \overset{\circ}{\rho}_C\right)}{\Delta t} V_C + \sum_{f \sim nb(C)} \left(\dot{m}_f^* + \dot{m}_f'\right) + \underbrace{\left(\dot{m}_b^* + \dot{m}_b'\right)}_{boundary\,face} = 0 \tag{16.31}$$

where the contribution of the boundary face is separately displayed with $\dot{m}_b^*$ representing the boundary mass flux and $\dot{m}_b'$ its correction. Moreover, expressing the Rhie-Chow interpolation at the boundary faces by adopting the same approach that was used for incompressible flow, the velocity, mass flow rate, and mass flow rate correction at a boundary face are expressed as

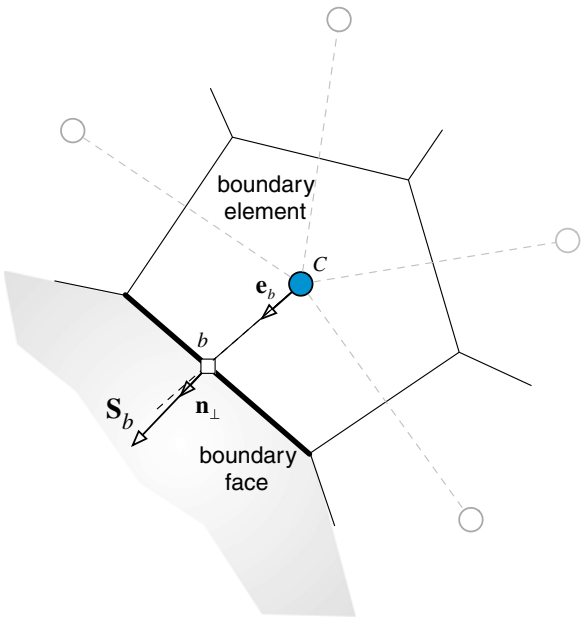


Fig. 16.3 A boundary control volume

$$\underbrace{\mathbf{v}_b^*}_{boundary\,face} = \underbrace{\mathbf{v}_C^* - \mathbf{D}_C\left(\nabla p_b^{(n)} - \nabla p_C^{(n)}\right)}_{boundary\,Rhie-Chow} \tag{16.32}$$

$$\dot{m}_b^* = \rho_b^{(n)}\mathbf{v}_C^* \cdot \mathbf{S}_b - \rho_b^* \mathbf{D}_C^{\mathbf{v}}\left(\nabla p_b^{(n)} - \nabla p_C^{(n)}\right) \cdot \mathbf{S}_b \tag{16.33}$$

$$\dot{m}_b' = -\rho_b^* \mathcal{D}_C\left(p_b' - p_C'\right) + \left(\frac{\dot{m}_b^*}{\rho_b^*}\right) C_{\rho,b} p_b' \tag{16.34}$$

The only difference between these expressions and those presented in Chap. 15 is in the correction equation of the mass flow rate, which has an additional term related to density correction. Moreover, there is no difference in the implementation of boundary conditions at a wall and a symmetry plane between incompressible and compressible flows. Therefore the modifications to the boundary elements presented in the previous chapter for wall and symmetry boundary conditions in the pressure correction equation are applicable here with no need to be repeated.

The boundary conditions left to be discussed here are those applicable at the inlet and outlet. For compressible flow, the conditions to be imposed are dictated by the Mach number values. For an inviscid flow, the mathematical type of the equations changes from elliptic to hyperbolic as the flow changes from subsonic to supersonic. Details regarding their implementation are given next.

16.8.1 Inlet Boundary Conditions

At the inlet to a domain the flow may be subsonic or supersonic necessitating different treatment since the flow equations may be either of the elliptic or the hyperbolic type.

16.8.1.1 Subsonic Flow at Inlet

At subsonic speeds several conditions at inlet to a domain can be imposed. This include specified velocity, specified static pressure and velocity direction, or specified stagnation pressure and velocity direction. The last type should be used when transition to supersonic speed occurs within the domain.

$$\text{Specified Velocity}(p_b = ?;\ \dot{m}_b = ?;\ \mathbf{v}_b \textit{ specified})$$

Unlike incompressible flow, since for compressible flow the density depends on pressure, the mass flux remains unknown even with a specified velocity at inlet $\left(\text{i.e, } \dot{m}'_b = \rho'_b \mathbf{v}^*_b \cdot \mathbf{S}_b \neq 0\right)$. At an inlet boundary the coefficient multiplying the pressure correction p'_b is given by

$$a_b^{p'} = C_{\rho,b} \frac{\dot{m}^*_b}{\rho^*_b} \tag{16.35}$$

For implementation in the pressure correction equation, p'_b is expressed in terms of internal nodes and the coefficients at these nodes modified accordingly. For the constant profile case (i.e., $p_b = p_C$), the a_C coefficient is obtained as

$$a_C^{p'} = \frac{V_C C_\rho}{\Delta t} + \underbrace{\sum_{f\sim nb(C)} \left(\frac{C_{\rho,f}}{\rho^*_f} \left\| \dot{m}^*_f, 0 \right\| \right) + \sum_{f\sim nb(C)} \rho^*_f \mathcal{D}_f}_{\textit{interior faces contribution}} + \underbrace{C_{\rho,b} \frac{\dot{m}^*_b}{\rho^*_b}}_{\textit{boundary face contribution}} \tag{16.36}$$

The value of the pressure p_b is again obtained by extrapolation from the interior as explained in the incompressible flow chapter.

$$\text{Specified Static Pressure and Velocity Direction } \left(p_b = p_{specified};\ \mathbf{e}_\mathbf{v} \textit{ specified};\ \dot{m}_b = ?;\ \mathbf{v}_b = ?\right)$$

In the case of a specified static pressure at inlet, p_b is known and thus p'_b is set to zero and consequently ρ'_b is also zero. Therefore the implementation is similar to the incompressible case with the inlet treated as a Dirichlet boundary condition. Knowing the velocity direction, its magnitude is computed as in the incompressible

case using Eq. (16.33) leading to an equation similar to Eq. (15.137). The coefficient of the pressure-correction equation becomes

$$a_C^{p'} = \frac{V_C C_\rho}{\Delta t} + \underbrace{\sum_{f\sim nb(C)} \left(\frac{C_{\rho,f}}{\rho_f^*} \left\| \dot{m}_f^*, 0 \right\| \right) + \sum_{f\sim nb(C)} \rho_f^* \mathcal{D}_f}_{\text{interior faces contribution}} + \underbrace{\rho_b^* \mathcal{D}_C}_{\text{boundary face contribution}} \tag{16.37}$$

Specified Total Pressure and Velocity Direction ($p_{o,b} = p_{o,\,specified}$; $\mathbf{e_v}$ *specified*; $\dot{m}_b = ?$; $\mathbf{v}_b = ?$)

For this case, the magnitude of the velocity and the pressure at the boundary are unknown while related through the total pressure equation given by

$$p_{o,b} = p_b \left(1 + \frac{\gamma - 1}{2} M_b^2 \right)^{\gamma/(\gamma-1)} \tag{16.38}$$

where b refers to the boundary, $p_{o,b}$ is the total pressure, p_b the static pressure, γ the ratio of specific heats, and M_b the Mach number which is equivalent to

$$M_b = \sqrt{\frac{\mathbf{v}_b \cdot \mathbf{v}_b}{\gamma R T_b}} \tag{16.39}$$

Equation (16.37) can be rearranged to give the static pressure in terms of the total pressure as

$$p_b = p_{o,b} \left(1 + \frac{\gamma - 1}{2} \frac{\left(\dot{m}_b^*\right)^2}{\left(\rho_b^*\right)^2 (\mathbf{e_v} \cdot \mathbf{n} S_b)^2 \gamma R T_b} \right)^{-\gamma/(\gamma-1)} \tag{16.40}$$

where $\mathbf{e_v}$ is the unit vector in the direction of the velocity vector. Differentiating Eq. (16.40) with respect to $\dot{m}_b^*$ gives

$$\frac{dp_b}{d\dot{m}_b^*} = -\frac{\gamma \dot{m}_b^* p_{o,b}}{\left(\rho_b^*\right)^2 (\mathbf{e_v} \cdot \mathbf{n} S_b)^2 \gamma R T_b} \left(1 + \frac{\gamma - 1}{2} \frac{\left(\dot{m}_b^*\right)^2}{\left(\rho_b^*\right)^2 (\mathbf{e_v} \cdot \mathbf{n} S_b)^2 \gamma R T_b} \right)^{-\frac{(2\gamma-1)}{(\gamma-1)}} \tag{16.41}$$

Substituting Eq. (16.41) into Eq. (15.163) an equation for pressure correction function of the mass flux correction is obtained as

$$\begin{aligned} p'_b &= -\frac{\gamma \dot{m}^*_b p_{o,b}}{\left(\rho^*_b\right)^2 (\mathbf{e_v} \cdot \mathbf{n} S_b)^2 \gamma R T_b} \left(1 + \frac{\gamma - 1}{2} \frac{\left(\dot{m}^*_b\right)^2}{\left(\rho^*_b\right)^2 (\mathbf{e_v} \cdot \mathbf{n} S_b)^2 \gamma R T_b}\right)^{-\frac{(2\gamma-1)}{(\gamma-1)}} \dot{m}'_b \\ &= c_b \dot{m}'_b \end{aligned} \tag{16.42}$$

Replacing p'_b in Eq. (16.34) by its equivalent expression given by Eq. (16.42), the mass flux correction becomes

$$\dot{m}'_b = \frac{\rho^*_b \mathcal{D}_b}{1 + \rho^*_b \mathcal{D}_b c_b - \left(\frac{\dot{m}^*_b}{\rho^*_b}\right) C_{\rho,b} c_b} p'_c \tag{16.43}$$

The modified boundary cell coefficient is obtained by substituting $\dot{m}'_b$ from Eq. (16.43) in the expanded continuity equation and is given by

$$a_C^{p'} = \frac{V_C C_\rho}{\Delta t} + \underbrace{\sum_{f \sim nb(C)} \left(\frac{C_{\rho,f}}{\rho^*_f} \left\| \dot{m}^*_f, 0 \right\|\right) + \sum_{f \sim nb(C)} \rho^*_f \mathcal{D}_f}_{\text{interior faces contribution}} + \underbrace{\frac{\rho^*_b \mathcal{D}_C}{1 + \rho^*_b \mathcal{D}_C c_b - \left(\frac{\dot{m}^*_b}{\rho^*_b}\right) C_{\rho,b} c_b}}_{\text{boundary face contribution}} \tag{16.44}$$

It should be mentioned that the boundary condition for the energy equation at a subsonic inlet is usually either a specified static temperature T_b or a specified stagnation temperature $T_{o,b}$. If the static temperature is specified, then this is similar to a Dirichlet condition. If the stagnation temperature is specified then at each iteration the value of the static temperature is extracted from the stagnation temperature equation using

$$T_{o,b} = T_b + \frac{\mathbf{v}_b \cdot \mathbf{v}_b}{2 c_p} \tag{16.45}$$

and the obtained value treated as known. Thus a Dirichlet-type boundary condition is also applied.

16.8.1.2 Supersonic Flow at Inlet

Specified static pressure, velocity, and temperature

$$\left(p_b = p_{specified};\ \mathbf{v}_b = \mathbf{v}_{specified};\ T = T_{specified}\right)$$

At a supersonic inlet, values for all variables must be specified (pressure, velocity, and temperature). This is equivalent to a Dirichlet-type condition implying that $\dot{m}'_b = p'_b = 0$. Therefore the $a_C^{p'}$ coefficient of the boundary cell is found to be

$$a_C^{p'} = \frac{V_C C_\rho}{\Delta t} + \underbrace{\sum_{f\sim nb(C)} \left(\frac{C_{\rho,f}}{\rho_f^*} \left\| \dot{m}_f^*, 0 \right\| \right) + \sum_{f\sim nb(C)} \rho_f^* \mathcal{D}_f}_{\text{interior faces contribution}} \tag{16.46}$$

16.8.2 Outlet Boundary Conditions

16.8.2.1 Subsonic Flow at Outlet

$$\text{Specified Pressure} \left(p_b = p_{specified};\ \dot{m}_b\,?;\ \mathbf{v}_b\ =\ ? \right)$$

At a subsonic outlet, the pressure is usually prescribed. Therefore the pressure correction p'_b is set to zero while the mass flow rate correction $\dot{m}'_b$ is computed as

$$\dot{m}'_b = -\rho_b^* \mathcal{D}_C \left(p'_b - p'_C \right) + \left(\frac{\dot{m}_b^*}{\rho_b^*} \right) C_{\rho,b} p'_b = \rho_b^* \mathcal{D}_C p'_C \tag{16.47}$$

Since the velocity $\mathbf{v}_b^*$ is not known, it is customary to assume its direction to be that of the upwind velocity $\mathbf{v}_C^*$. The expression of the a_c coefficient in the pressure-correction equation may be written as

$$a_C^{p'} = \frac{V_C C_\rho}{\Delta t} + \underbrace{\sum_{f\sim nb(C)} \left(\frac{C_{\rho,f}}{\rho_f^*} \left\| \dot{m}_f^*, 0 \right\| \right) + \sum_{f\sim nb(C)} \rho_f^* \mathcal{D}_f}_{\text{interior faces contribution}} + \underbrace{\rho_b^* \mathcal{D}_c}_{\text{boundary face contribution}} \tag{16.48}$$

For the energy equation a zero flux Neumann-type boundary condition is applied.

$$\text{Specified Mass Flow Rate } \left(\dot{m}_b = \dot{m}_{specified};\ p_b ?\ \mathbf{v}_b ? \right)$$

For a specified mass flow rate at outlet, $\dot{m}'_b$ is zero and is simply dropped from the pressure correction equation with no modifications required for the coefficients of the boundary elements. By setting $\dot{m}'_b$ to zero in Eq. (16.34), an expression for the pressure correction at the boundary as a function of the pressure correction at the boundary cell centroid is obtained as

$$p'_b = \frac{\rho^*_b \mathcal{D}_C}{\rho^*_b \mathcal{D}_C - \left(\frac{\dot{m}^*_b}{\rho^*_b}\right) C_{\rho,b}} p'_C \tag{16.49}$$

allowing the boundary pressure and density to be computed. For the energy equation a zero flux Neumann-type boundary condition is applied.

16.8.2.2 Supersonic Flow at Outlet

At a supersonic outlet none of the variables should be specified and the values of pressure, velocity, density, and temperature are extrapolated from the interior of the domain. Thus both $\dot{m}_b$ and p_b are extrapolated from interior cells. This is equivalent to applying a Neumann boundary condition on pressure-correction, leading to the following modified a_C coefficient

$$a^{p'}_C = \frac{V_C C_\rho}{\Delta t} + \underbrace{\sum_{f \sim nb(C)} \left(\frac{C_{\rho,f}}{\rho^*_f} \left\| \dot{m}^*_f, 0 \right\| \right) + \sum_{f \sim nb(C)} \rho^*_f \mathcal{D}_f}_{\text{interior faces contribution}} + \underbrace{\left(\frac{\dot{m}^*_b}{\rho^*_b}\right) C_{\rho,b}}_{\text{boundary face contribution}} \tag{16.50}$$

16.9 Computational Pointers

16.9.1 uFVM

For compressible flows a major modification to the algorithm arises in the assembly of the pressure equation through the inclusion of the convection like term. This is added to the cfdAssembleMdotTerm shown in Listing 16.1.

```
    % assemble terms X (for compressible flow)
    %     (mdot_f/density_f)*(∂rho/∂p) P'
    % where mdot_f is the newly computed mdot_f
    %
    local_mdot_f = local_FLUXCf_1*(pressure[iElement1]+ Pref) +
local_FLUXCf_2*(pressure[iElement2]+ Pref) + local_FLUXVf;
    %
    local_FLUXCf1 = local_FLUXCf1 +  max(local_mdot_f/
density_f(iFace),0.0)*drhodp_f(iFace);
    local_FLUXCf2 = local_FLUXCf2 - max(-local_mdot_f/
density_f(iFace),0.0)*drhodp_f(iFace);
    local_FLUXVf  = local_FLUXVf -(max(local_mdot_f/density_f(iFace),
0.0)*drhodp_f(iFace)*(pressure(iElement1)+ Pref)- max(-local_mdot_f/
density_f(iFace),0.0)*drhodp_f(iFace)*(pressure(iElement2)+ Pref));
    local_FLUXVf   += -local_mdot_f;
```

Listing 16.1 Script used to assemble the additional terms for the compressible pressure correction equation

The other important change is in the treatment of boundary conditions that now necessitates accounting for a variable density field. For example the supersonic inlet condition is implemented for the pressure correction equation as shown in Listing 16.2.

```
    % assemble terms X (for compressible flow)
    %     (mdot_f/density_f)*(?rho/?p) P'
    % where mdot_f is the newly computed mdot_f
    %
    local_mdot_f = local_FLUXCf_1*(pressure[iElement1]+ Pref) +
local_FLUXCf_2*(pressure[iElement2]+ Pref) + local_FLUXVf;
    %
    local_FLUXCf1 = local_FLUXCf1 +  max(local_mdot_f/
density_f(iFace),0.0)*drhodp_f(iFace);
    local_FLUXCf2 = local_FLUXCf2 - max(-local_mdot_f/
density_f(iFace),0.0)*drhodp_f(iFace);
    local_FLUXVf  = local_FLUXVf -(max(local_mdot_f/density_f(iFace),
0.0)*drhodp_f(iFace)*(pressure(iElement1)+ Pref)- max(-local_mdot_f/
density_f(iFace),0.0)*drhodp_f(iFace)*(pressure(iElement2)+ Pref));
    local_FLUXVf   += -local_mdot_f;
```

Listing 16.2 Implementation of a supersonic inlet condition

16.9.2 OpenFOAM®

In this section simpleFoam is extended to handle compressible fluid flow at all speeds. This entails the following modifications to simpleFoam: (i) the addition of the energy equation to be solved simultaneously with the continuity and momentum equations, (ii) the use of an equation of state relating density to temperature and pressure, (iii) and the introduction of the necessary modifications to the pressure correction equation and to a number of boundary conditions. The resulting code is denoted simpleFoamCompressible with many of the extensions added in the form of supplemental include files as shown in Listing 16.3.

The *upwind.H* and *gaussConvectionScheme.H* are used to force an upwind discretization on the convective term of the pressure correction equation. The *bound.H* class is used to bound variables within certain limits.

```
#include "fvCFD.H"
#include "psiThermo.H"
#include "RASModel.H"
#include "upwind.H"
#include "gaussConvectionScheme.H"
#include "bound.H"
#include "simpleControl.H"
#include "totalPressureCompFvPatchScalarField.H"
#include "totalPressureCorrectorCompFvPatchScalarField.H"
#include "totalVelocityFvPatchVectorField.H"
#include "orthogonalSnGrad.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * //

int main(int argc, char *argv[])
{

#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
    simpleControl simple(mesh);

#   include "createFields.H"
```

Listing 16.3 The include files used in simpleFoamCompressible

The *createFields.H* now includes the definition of the density field and other variables and constants related to compressible flow physics. The *psiTermo* class, depicted in Listing 16.4, provides access to the thermophysical relations that are part of the OpenFOAM® library [27], such as the perfect gas law described in Eq. (16.4).

```
Info<< "Reading thermophysical properties\n" << endl;

autoPtr<psiThermo> thermo
(
    psiThermo::New(mesh)
);

Info<< "Calculating field rho\n" << endl;

volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    thermo->rho()
);

volScalarField& p = thermo->p();
volScalarField& h = thermo->he();

thermo->correct();

volScalarField gammaGas
(
    IOobject
    (
        "gammaGas",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    thermo->Cp() /  thermo->Cv()
);

volScalarField RGas
(
    IOobject
    (
        "RGas",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    thermo->Cp() -  thermo->Cv()
);
```

Listing 16.4 An excerpt of the *psiTermo* class

The pressure and enthalpy fields are defined in the *thermo* class, displayed in Listing 16.5, and accessed as references in *createFields.H*.

```
volScalarField& p = thermo->p();
volScalarField& h = thermo->he();

const volScalarField::GeometricBoundaryField& pbf=p.boundaryField();
wordList pbt = pbf.types();
volScalarField pp
(
    IOobject
    (
        "pp",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedScalar("zero", p.dimensions(), 0.0),
    pbt
);
```

Listing 16.5 Defining the pressure and enthalpy field in *thermo* class

The velocity is defined as in the incompressible version but the mass flux now includes density in its definition (Listing 16.6).

```
Info << "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
surfaceScalarField mDot
(
    IOobject
    (
        "mDot",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    linearInterpolate(rho*U) & mesh.Sf()
);
```

Listing 16.6 Defining the velocity and mass flux fields

In compressible flows, setting physical limits on some variables such as density and pressure can enhance robustness, especially during the first few iterations. This prevents variables from assuming non-physical values (like negative densities or pressures). Therefore bounds can be set as part of the case definition, as shown in Listing 16.7, and read in *createFields.H*.

```
dimensionedScalar rhoMax(simple.dict().lookup("rhoMax"));
dimensionedScalar rhoMin(simple.dict().lookup("rhoMin"));

dimensionedScalar pMax(simple.dict().lookup("pMax"));
dimensionedScalar pMin(simple.dict().lookup("pMin"));
```

Listing 16.7 Setting lower and upper bounds for the density and pressure fields

The momentum equation is defined with a slightly modified syntax that accounts for density and thermophysical property relations. The syntax of the linearized formula is given in Listing 16.8.

```
// Solve the Momentum equation
fvVectorMatrix UEqn
(
        fvm::ddt(rho,U)
      + fvm::div(mDot, U)
      - fvm::laplacian(thermo->mu, U)
);

UEqn.relax();

solve
(
    UEqn == -fvc::grad(p)
);
```

Listing 16.8 Syntax used to solve the momentum equation

The first instruction defines the finite volume discretization of the momentum equation in a vector form (the three components of the velocity are solved in a segregated manner despite the vectorial implementation). The system is implicitly relaxed and then solved with an iterative solver.

Once the momentum equation is solved, a new guess of the velocity field is obtained. This velocity field does not necessarily satisfy the continuity equation.

To enforce mass conservation, assembly of the pressure correction equation is now required to correct the velocity. Following Eq. (16.19) the syntax used for that purpose is shown in Listing 16.9.

```
pp = scalar(0.0)*pp;
pp.correctBoundaryConditions();


surfaceScalarField phid
(
      IOobject
      (
       "phid",
       runTime.timeName(),
       mesh,
       IOobject::NO_READ,
       IOobject::NO_WRITE
      ),
    mDot*drhodp/(rhofd)
);

fvMatrix<scalar> ppCompEqn
(
    fvm::ddt(thermo->psi(),pp) +
    fv::gaussConvectionScheme<scalar>
     (
         mesh,
         phid,
         tmp<surfaceInterpolationScheme<scalar> >
         (
             new upwind<scalar>(mesh,phid)
         )
     ).fvmDiv(phid, pp)
  - fvm::laplacian(pDiff, pp)
  + fvc::div(mDot) + fvc::ddt(rho)
)
```

Listing 16.9 Syntax used to assemble the pressure correction equation

As for the incompressible case, in order to avoid checker boarding, the *mDot mass flux field is evaluated* using the Rhie-Chow interpolation but now taking into account also the density field, evaluated based on the *thermo* model as shown in Listing 16.10.

```
…
rho = thermo->rho();

Foam::fv::orthogonalSnGrad<scalar> faceGradient(mesh);

surfaceVectorField gradp_avg_f = linearInterpolate(fvc::grad(p));
surfaceVectorField gradp_f = gradp_avg_f - (gradp_avg_f & ed)*ed +
(faceGradient.snGrad(p))*ed;

surfaceVectorField U_avg_prevIter_f = linearInterpolate(U.prevIter());
surfaceVectorField U_avg_f = linearInterpolate(U);

surfaceScalarField rhofd = upwind<scalar>(mesh,mDot).interpolate(rho);
surfaceScalarField rhof("srho",fvc::interpolate(rho));
surfaceScalarField DUf("srUA",fvc::interpolate(DU,"interpolate((1|
A(U)))"));

volScalarField dt
(
      IOobject
      (
       "dt",
       runTime.timeName(),
       mesh,
       IOobject::NO_READ,
       IOobject::NO_WRITE
      ),
      mesh ,
      dimensionedScalar("dt" ,runTime.deltaT().dimensions(),
runTime.deltaT().value()),
      zeroGradientFvPatchScalarField::typeName
);
surfaceScalarField dt_f = linearInterpolate(dt);
surfaceScalarField drhodp = linearInterpolate(thermo->psi());

scalar UURF = mesh.equationRelaxationFactor("U");

// Rhie-Chow interplation
mDot = rhof*(
              (U_avg_f & mesh.Sf()) - ( (DUf*( gradp_f - gradp_avg_f))
& mesh.Sf() )
            );
           + (scalar(1) - UURF)*(mDot.prevIter() -
( (rhof*U_avg_prevIter_f) & mesh.Sf() ) )
           +    rhof*(DUf/dt_f)*(mDot.prevIter() -
( (rhof*U_avg_prevIter_f) & mesh.Sf() ) );
```

Listing 16.10 Calculation of mass fluxes at cell faces using the Rhie-Chow interpolation

It is worth mentioning that density is interpolated to faces using an upwind scheme in order to mimic the hyperbolic behavior of compressible flows.

The pressure correction equation is fully set and is solved using the syntax displayed in Listing 16.11.

```
ppEqn.solve();
```

Listing 16.11 Syntax for solving the pressure correction equation

After solving the pressure correction equation, variables that depend on pressure correction are updated. For the mass flux field this is performed using the syntax in Listing 16.12, this is similar to the incompressible flux correction.

```
mDot += ppEqn.flux();
```

Listing 16.12 Syntax to update the mass flux field

Where again the flux() function in Listing 16.12 updates the fluxes using directly the matrix coefficients and cell values. A simplified version of the flux() function is shown in Listing 16.13.

```
for (label face=0; face<lowerAddr.size(); face++)
{
    mDotPrime[face] =
        upperCoeffs[face]*pp[upperAddr[face]]
      - lowerCoeffs[face]*pp[lowerAddr[face]];
}

return mDotPrime;
```

Listing 16.13 A simplified version of the *flux( )* function where the flux correction *mDotPrime* is computed

In Listing 16.13 the correction flux *mDotPrime* is basically evaluated by performing a loop over the faces using the upper and lower coefficients of the matrix and multiplying these coefficients with the corresponding cell values.

Finally the velocity, density and pressure at cell centroids are updated using Eqs. (16.20), (16.21), and (16.22), as shown in Listing 16.14, where the variable alphaP is the explicit relaxation factor for pressure and density updates λ^p, necessary for a stable SIMPLE solver.

```
scalar alphaP = mesh.equationRelaxationFactor("pp");

mDot += ppCompEqn.flux();

p += alphaP*pp;
p.correctBoundaryConditions();

rho += alphaP*pp*thermo->psi();

boundMinMax(p, pMin, pMax);
boundMinMax(rho, rhoMin, rhoMax);

U -= fvc::grad(pp)*DU;
U.correctBoundaryConditions();
```

Listing 16.14 Update of the velocity and pressure fields at cell centroids

In order to account for compressibility effects, the energy equation is introduced and the related temperature is calculated. In OpenFOAM®, the energy equation expressed in terms of specific static enthalpy $\left(h = C_p T\right)$, given by Eq. (3.61), is solved as depicted in Listing 16.15.

```
fvScalarMatrix hEqn
(
    fvm::ddt(rho,h)
  + fvm::div(mDot, h)
  - fvm::laplacian(turbulence->alphaEff(), h)
  + fvm::SuSp(-fvc::div(mDot),h)
 ==

    fvc::div(mDot/fvc::interpolate(rho)*fvc::interpolate(p))
  - p*fvc::div(mDot/fvc::interpolate(rho))
);


hEqn.relax();
hEqn.solve();

h.correctBoundaryConditions();

thermo->correct();

gammaGas =  thermo->Cp()/ thermo->Cv();
gammaGas.correctBoundaryConditions();

RGas =  thermo->Cp() -  thermo->Cv();
RGas.correctBoundaryConditions();
```

Listing 16.15 Solving the energy equation

Once the energy equation is solved, the new enthalpy is used to update the temperature and gas properties (e.g., specific heats).

In addition to the main solver, new *total pressure* and *total temperature* boundary conditions are implemented for *subsonic inlet* patches, these are often used boundary conditions for the simulation of compressible flows. The boundary conditions are defined in the directory "derivedFvPatchFields" and are next presented. For a better understanding, it may be beneficial to read Chap. 18 prior to going over the implementation process presented below.

- **totalPressureComp**: This implements the total pressure condition at subsonic inlet. For that purpose the updateCoeffs() function is modified as shown in Listing 16.16, which indicates that after gathering the necessary data from the solver, the boundary static pressure is computed using Eq. (16.40) and the obtained values are stored in the defined *newp* variable.

```
    const fvPatchScalarField& TB =
        patch().lookupPatchField<volScalarField, scalar>("T");

    const fvPatchField<scalar>& RB =
        patch().lookupPatchField<volScalarField, scalar>("RGas");

    const fvPatchField<scalar>& gammaB =
        patch().lookupPatchField<volScalarField, scalar>("gammaGas");

      const fvsPatchField<scalar>&  sphi =
        patch().lookupPatchField<surfaceScalarField, scalar>("mDot");

      const fvPatchField<scalar>&  srho =
    patch().lookupPatchField<volScalarField, scalar>("rho");

      const fvPatchField<vector>& UF =
patch().lookupPatchField<volVectorField, vector>("U");

    scalarField newp = max(min(p0_/pow((scalar(1.0) + (gammaB -
scalar(1.0) )*(sqr(sphi)
                                /(sqr(srho) * sqr(patch().magSf()))))/
(scalar(2.0) * gammaB * RB * TB  ))
               , gammaB/(gammaB - scalar(1.0))), p0_),SMALL);

    operator==
    (
        newp
    );
```

Listing 16.16 Modified updateCoeffs() function for implementing the total pressure boundary condition at inlet

- **totalPressureCorrectorComp**: This is also needed with a total inlet pressure boundary condition. Its role is to deal with the mass flow rate correction at inlet affecting the diagonal coefficient of the boundary element, as expressed by

Eqs. (16.43) and (16.44). In the implementation, the contributions to the coefficients are reset to zero, as can be inferred from Listing 16.17, except for the gradientInternalCoeffs() that is required to return the correct values. This ensures that for both the divergence and laplacian operator one boundary condition is applied.

```
tmp<Field<scalar> >
totalPressureCorrectorCompFvPatchScalarField::valueInternalCoeffs
(
    const tmp<scalarField>&
) const
{
    return tmp<Field<scalar> >
    (
        new Field<scalar>(this->size(), pTraits<scalar>::zero)
    );
}

tmp<Field<scalar> >
totalPressureCorrectorCompFvPatchScalarField::gradientBoundaryCoeffs()
const
{
    return tmp<Field<scalar> >
    (
        new Field<scalar>(this->size(), pTraits<scalar>::zero)
    );
}

tmp<Field<scalar> >
totalPressureCorrectorCompFvPatchScalarField::valueBoundaryCoeffs
(
    const tmp<scalarField>&
) const
{
    return tmp<Field<scalar> >
    (
        new Field<scalar>(this->size(), pTraits<scalar>::zero)
    );
}

tmp<Field<scalar> >
totalPressureCorrectorCompFvPatchScalarField::gradientInternalCoeffs()
const
{
      const fvsPatchField<scalar>&  srUA =
    patch().lookupPatchField<surfaceScalarField, scalar>("srUA");

      const fvPatchField<scalar>&  srho =
    patch().lookupPatchField<volScalarField, scalar>("rho");

    return( deltaM()/(-srUA*patch().magSf()*srho) ); //to remove the
laplacian operator (see gaussLaplacianScheme)
}
```

Listing 16.17 Script used to reset the diagonal coefficients and to return only the diffusion contribution or a total pressure boundary condition at inlet

The modified diagonal coefficient of the pressure correction equation for a boundary element is computed in Listing 16.18. Here the function deltaM() implements Eq. (16.44) as is, while the diffusion term "(-srUA*patch().magSf() *srho)" is removed from the coefficient of the pressure correction equation, as defined by the laplacian operator (see Chap. 8, computational pointers).

```
Field<scalar> totalPressureCorrectorCompFvPatchScalarField::deltaM()
const
{

      const fvPatchField<scalar>&  T =
    patch().lookupPatchField<volScalarField, scalar>("T");

      const fvPatchField<scalar>&  srho =
    patch().lookupPatchField<volScalarField, scalar>("rho");

      const fvPatchField<scalar>&  RB =
    patch().lookupPatchField<volScalarField, scalar>("RGas");

      const fvsPatchField<scalar>&  srUA =
    patch().lookupPatchField<surfaceScalarField, scalar>("srUA");

      const fvsPatchField<scalar>&  sphi =
    patch().lookupPatchField<surfaceScalarField, scalar>("mDot");

    scalarField Dp = patch().magSf()*patch().deltaCoeffs()*srUA;

    scalarField coeff = srho*Dp/(scalar(1.0) + srho*Dp*Cu() - (sphi/
srho)*Cu()/(RB*T));


      return (coeff);
}
```

Listing 16.18 Script used to modify the diagonal coefficient of the boundary element for a total pressure boundary condition at inlet

- **totalTemp**: This function implements for the energy equation the total temperature boundary condition at a subsonic inlet. The idea is to impose a static temperature as a Dirichlet boundary condition using Eq. (16.45). For that purpose the updateCoeffs() function is modified as in Listing 16.19.

```
const fvPatchField<vector>& Up =
    patch().lookupPatchField<volVectorField, vector>("U");

const fvPatchField<scalar>& gammaB =
    patch().lookupPatchField<volScalarField, scalar>("gammaGas");

scalarField gM1ByG = (gammaB - 1.0)/gammaB;

const fvPatchScalarField& TB =
    patch().lookupPatchField<volScalarField, scalar>("T");

const fvPatchField<scalar>& RB =
    patch().lookupPatchField<volScalarField, scalar>("RGas");

scalarField psip = scalar(1.0)/(RB * TB);

operator==
(
    T0_/(1.0 + 0.5*psip*gM1ByG*magSqr(Up))
);
```

Listing 16.19 Modified updateCoeffs() function for implementing the total temperature boundary condition at a subsonic inlet

At each iteration the temperature value is updated based on the inlet total temperature and the boundary velocity.

- **totalVelocity**: This function, described in Listing 16.20, implements the updates to the velocity field required with total conditions applied at a subsonic inlet. For that purpose a Dirichlet boundary condition is used, with velocity values iteratively computed based on the calculated fluxes "mDot" at the boundary itself. The algorithm is based on grabbing the flux field "mDot" (updated with new values after solving the pressure correction equation by invoking the "flux()" function) and dividing the flux by the face area and the corresponding density.

```
const fvsPatchField<scalar>&  sphi =
    patch().lookupPatchField<surfaceScalarField, scalar>("mDot");

const fvPatchField<scalar>& rhop =
    patch().lookupPatchField<volScalarField, scalar>("rho");

vectorField n = patch().nf();
scalarField ndmagS = (n & inletDir())*patch().magSf();

scalarField clip = neg(sphi);

scalarField newvel = (sphi*clip)/(rhop*ndmagS);

operator==(inletDir()*newvel);
);
```

Listing 16.20 Updating the velocity at a subsonic inlet for a total pressure boundary condition

Additionally, a clipping variable is introduced in order to prevent any "back-flow" at the inlet. Here the "clip" variable may assume either a value of zero or one. In fact the "neg" function returns 1 and 0 for negative and positive values, respectively, preventing outward velocities to be accepted. On the other hand, the "inletDir" variable gives the velocity direction at inlet, as defined by the user.

16.10 Closure

In this chapter the incompressible segregated pressure based approach developed in the previous chapter was extended to handle compressible fluid flow at all speeds. This involved modifying the pressure correction equation to include a convection-like term that changes its type from elliptic to hyperbolic. It also required alterations to the momentum equations, the solution of the energy equation, as well as the addition of an equation of state. Just as critical, are the special boundary conditions needed in the simulation of compressible flows. A number of boundary conditions were presented as well as some implementation details.

The needed modifications to the base incompressible code represent a relatively small change to the bulk of any code and yet allow a drastic extension of its capabilities. The next chapter will present the additional techniques needed for dealing with the time averaged Navier-Stokes equations required for solving turbulent flow problems.

16.11 Exercises

Exercise 1 A portion of a gas-supply system is shown in Fig. 16.4. The mass flow rate $\dot{m}$ in a pipe section is given by

$$\dot{m} = \rho C \Delta p$$

where Δp is the pressure drop over the length of the pipe section, ρ is the gas density, and C is the gas conductance. The following data is known:

$$p_1 = 400, p_2 = 350$$
$$\dot{m}_F = 25$$
$$C_A = C_C = 1.2, C_B = 1.4, C_D = 1.6, C_E = 1.8$$

with the density related to the pressure via $p = \rho R'$ with $R' = 2000$.

If the direction of the flow is as shown in the figure, find $p_3, p_4, p_5, \dot{m}_A, \dot{m}_B, \dot{m}_C, \dot{m}_D$ and $\dot{m}_E$ using the following procedure:

- Start with a guess for p_3, p_4, and p_5.
- Compute $\dot{m}^*$ values based on the guessed pressures and densities.
- Construct the pressure-correction equations and solve for p'_3, p'_4 and p'_5.
- Update the pressures and the $\dot{m}^*$ values

Do you need to iterate? Why?

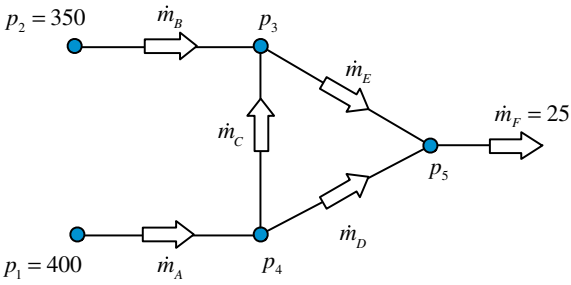


Fig. 16.4 A portion of a gas supply system

Exercise 2 Consider the flow of an ideal gas in a converging nozzle shown in Fig. 16.5, where each of the control volumes has $\Delta x = 0.5$. The area of the various surfaces are $A_{b_i} = 3$; $A_w = 2.3$; $A_e = 1.6$; $A_{b_o} = 0.9$ with $R = 2078$ and $\gamma = 1.4$.

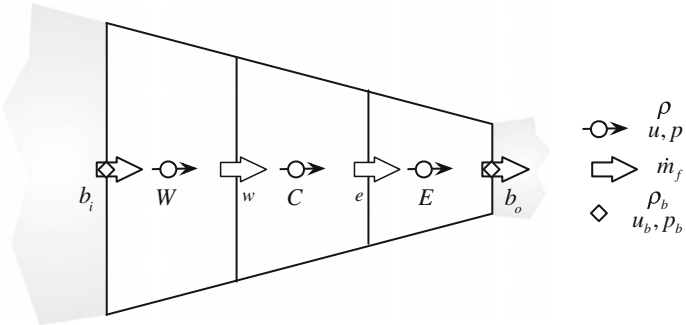
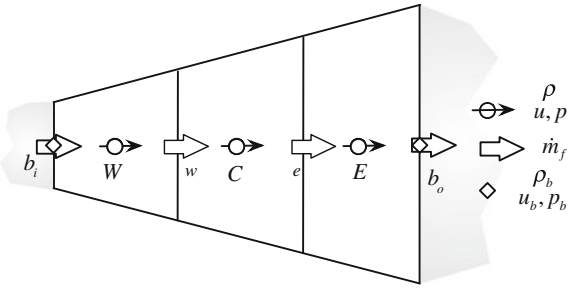


Fig. 16.5 Converging nozzle

The variable area flow is assumed to be one-dimensional and isentropic. At inlet the total pressure is $p_{o,i} = 100701.8\,\text{Pa}$ and the stagnation temperature is $T_{o,i} = 303\,\text{K}$. At exit the static pressure is $p_e = 10^5\,\text{kPa}$. Setup the momentum and pressure correction equations for the three control volumes and obtain the values of the velocity, pressure, density, and temperature starting with uniform fields of values $M = 0.1$ (M is the Mach number), $T = 290\,\text{K}$, and $p = 10^5\,\text{Pa}$. Perform three iterations. Note that there is no need to solve the energy equation as the temperature field can be extracted from the constant stagnation temperature condition.

Exercise 3 Consider the flow of an ideal gas in a diverging nozzle shown in Fig. 16.6, where each of the control volumes has $\Delta x = 0.5$. The area of the various surfaces are $A_{b_i} = 0.9$; $A_w = 1.6$; $A_e = 2.3$; $A_{b_o} = 3$ with $R = 2078$ and $\gamma = 1.4$.

Fig. 16.6 Diverging nozzle



The variable area flow is assumed to be one-dimensional and isentropic. At inlet the total pressure is $p_i = 1bar$, the Mach number is $M_i = 1.2$, and the temperature is $T_i = 303\,\mathrm{K}$. At exit the flow remains supersonic. Setup the momentum and pressure correction equations for the three control volumes and obtain the values of the velocity, pressure, density, and temperature starting with uniform fields of values $M = 1.2$, $T = 303\,\mathrm{K}$, and $p = 1bar$. Perform three iterations. Note that there is no need to solve the energy equation as the temperature field can be extracted from the constant stagnation temperature condition.

Exercise 4 (OpenFOAM®)
Define in the simpleFoamCompressible solver a new variable for the local Mach number to be visualized during simulation.

Exercise 5 (OpenFOAM®)
Modify in the totalPressureCorrectorComp boundary condition, the way Eq. (16.44) is imposed, using now the valueInternalCoeffs() function while resetting the gradientInternalCoeffs(). (Hint: consult Chaps. 11 and 19)

Exercise 6 (OpenFOAM®)
Check the rhoSimpleFoam solver that can be found in $FOAM_SRC/../applications/solvers/compressible/rhoSimpleFoam/pEqn.C and compare it with the algorithm described in this chapter.

Exercise 7 (OpenFOAM®)
Develop a compressible PISO algorithm and implement it starting with the simpleFoamCompressible code described in this chapter.

References

1. Harlow FH, Amsden AA (1968) Numerical calculation of almost incompressible flow. J Comput Phys 3:80–93
2. Harlow FH, Amsden AA (1971) A numerical fluid dynamics calculation method for all flow speeds. J Comput Phys 8(2):197–213

3. Patankar SV (1971) Calculation of unsteady compressible flows involving shocks. Mechanical Engineering Department, Imperial College, Report UF/TN/A/4
4. Issa RI, Lockwood FC (1977) On the prediction of two-dimensional supersonic viscous interactions near walls. AIAA J 15:182–188
5. Hah C (1984) A Navier-Stokes analysis of three-dimensional turbulent flows inside turbine blade rows at design and off-design conditions. ASME J Eng Gas Turbines Power 106:421–429
6. Hah C (1986) Navier-Stokes calculation of three-dimensional compressible flow across a cascade of airfoils with an implicit relaxation method. AIAA Paper 86-0555
7. Issa RI, Gosman D, Watkins A (1986) The computation of compressible and incompressible recirculating flows by a non-iterative implicit scheme. J Comput Phys 62:66–82
8. Karki KC (1986) A calculation procedure for viscous flows at all speeds in complex geometries. Ph.D. thesis, Department of Mechanical Engineering, University of Minnesota
9. Van Doormaal JP, Turan A, Raithby GD (1987) Evaluation of new techniques for the calculations of internal recirculating flows. AIAA Paper 87-0057
10. Rhie CM, Stowers ST (1987) Navier-Stokes analysis for high speed flows using a pressure correction algorithm. AIAA Paper 87-1980
11. Van Doormaal JP (1985) Numerical methods for the solution of incompressible and compressible fluid flows. Ph.D. Thesis, University of Waterloo, Ontario, Canada
12. Van Doormaal JP, Raithby GF, McDonald BD (1987) The segregated approach to predicting viscous compressible fluid flows. ASME J Turbomach 109:268–277
13. Karki KC, Patankar SV (1989) Pressure based calculation procedure for viscous flows at all speeds in arbitrary configurations. AIAA J 27:1167–1174
14. Demirdzic I, Issa RI, Lilek Z (1990) Solution method for viscous flows at all speeds in complex domains. In Wesseling P (ed) Notes on numerical fluid mechanics, vol 29, Vieweg, Braunschweig
15. Demirdzic I, Lilek Z, Peric M (1993) A collocated finite volume method for predicting flows at all speeds. Int J Numer Meth Fluids 16:1029–1050
16. Moukalled F, Darwish M (2000) A unified formulation of the segregated class of algorithms for fluid flow at all speeds. Numer Heat Transf Part B: Fundam 37:103–139
17. Rhie CM (1986) A pressure based Navier-Stokes solver using the multigrid method. AIAA paper 86-0207
18. Moukalled F, Darwish M (2001) A high resolution pressure-based algorithm for fluid flow at all-speeds. J Comput Phys 169(1):101–133
19. Turkel IE (1987) Preconditioning methods for solving the incompressible and low speed compressible equations. J Comput Phys 72:277–298
20. Turkel IE, Vatsa VN, Radespiel R (1996) Preconditioning methods for low speed flows. AIAA Paper 96-2460, Washington
21. Merkle CL, Sullivan JY, Buelow PEO, Venkateswaran S (1998) Computation of flows with arbitrary equations of state. AIAA J 36(4):515–521
22. Moukalled F, Darwish M (2006) Pressure based algorithms for single-fluid and multifluid flows. In: Minkowycz WJ, Sparrow EM, Murthy JY (eds) Handbook of numerical heat transfer, 2nd edn. Wiley, pp 325–367
23. Patankar SV (1980) Numerical heat transfer and fluid flow. Hemisphere, NewYork
24. Nerinckx K, Vierendeels J, Dick E (2005) A pressure-correction algorithm with mach-uniform efficiency and accuracy. Int J Numer Meth Fluids 47:1205–1211
25. Nerinckx K, Vierendeels J, Dick E (2006) A mach-uniform pressure-correction algorithm with AUSM + Flux definitions. Int J Numer Meth Heat Fluid Flow 16(6):718–739
26. Karimian SMH, Schneider GE (1995) Pressure-based control-volume finite element method for flow at all speeds. AIAA J 33(9):1611–1618
27. OpenFOAM, 2015 Version 2.3.x. http://www.openfoam.org