TUDFINVOL:

Mathematical & Programmers Guide

Ivo Wenneker

version 1.1

March 12, 2001

Contents

Part I

Mathematical Guide

Chapter 1

Introduction

In this manual we describe the mathematical techniques that are used in the TUDFINVOL program. TUDFINVOL is a program that computes compressible flows at unstructured, staggered grids, using a finite volume approach.

Chapter 2

Basic equations

2.1 Introduction

In the TUDFINVOL program we restrict ourselves to 2D flows of a perfect gas.

2.2 Governing equations

Starting points are the compressible Navier-Stokes equations, describing the conservation of mass, momentum and energy, and the equation of state for a perfect gas.

Continuity equation
Mass conservation leads to the continuity equation

$$\frac{\partial \rho}{\partial t} + m^{\alpha}_{,\alpha} = 0, \tag{2.1}$$

where ρ stands for the fluid density, $m^{\alpha} = \rho u^{\alpha}$ represents the momentum vector and u^{α} the velocity vector. The Einstein summation-convention is used, so for example $m^{\alpha}_{,\alpha} = \text{div}\,\mathbf{m}$ and $m^{\alpha}m^{\alpha} = \mathbf{m} \cdot \mathbf{m}$.

Conservation of momentum
Conservation of momentum leads to the momentum equations, one for each dimension:

$$\frac{\partial m^{\alpha}}{\partial t} + \left(u^{\beta}m^{\alpha}\right)_{,\beta} = -p_{,\alpha} + \tau^{\alpha\beta}_{,\beta} + \rho f^{\alpha}, \tag{2.2}$$

where p is the pressure, and the body forces have been incorporated in the term f^{α}. This term will be neglected for the moment. The relation between the deviatoric stress tensor $\tau^{\alpha\beta}$ and the motion of a Newtonian fluid is given by the constitutive relation

$$\tau^{\alpha\beta} = 2\mu\left(e^{\alpha\beta} - \frac{1}{3}\Delta\delta^{\alpha\beta}\right), \tag{2.3}$$

where μ is called the dynamic viscosity coefficient, $\delta^{\alpha\beta}$ is the Kronecker delta

$$\delta^{\alpha\beta} = \begin{cases} 0 & \text{for } \alpha \neq \beta, \\ 1 & \text{for } \alpha = \beta, \end{cases} \tag{2.4}$$

$e^{\alpha\beta}$ represents the rate of strain tensor

$$e^{\alpha\beta} = \frac{1}{2}\left(u^{\alpha}_{,\beta} + u^{\beta}_{,\alpha}\right) \tag{2.5}$$

and

$$\Delta = u^{\alpha}_{,\alpha}. \tag{2.6}$$

With the stress-tensor $\sigma^{\alpha\beta}$ defined as:

$$\sigma^{\alpha\beta} = -p\,\delta^{\alpha\beta} + \tau^{\alpha\beta}, \tag{2.7}$$

equation (2.2) is written as:

$$\frac{\partial m^{\alpha}}{\partial t} + \left(u^{\beta}m^{\alpha}\right)_{,\beta} = \sigma^{\alpha\beta}_{,\beta} + \rho f^{\alpha}. \tag{2.8}$$

Conservation of energy
The energy equation, describing the conservation of energy, reads:

$$\frac{\partial \rho E}{\partial t} + (m^\alpha H)_{,\alpha} = (u^\alpha \tau^{\alpha\beta} + kT_{,\beta})_{,\beta}. \tag{2.9}$$

Herein H stands for the total enthalpy per unit of mass, E for the total energy per unit of mass, T for the temperature and k for the thermal conductivity. The relation between H and E is given by:

$$H = E + \frac{p}{\rho}. \tag{2.10}$$

The enthalpy h and the total enthalpy H are related:

$$H = h + \frac{1}{2} u^\alpha u^\alpha, \tag{2.11}$$

and the relation between E and the internal energy e is:

$$E = e + \frac{1}{2} u^\alpha u^\alpha. \tag{2.12}$$

In case of a calorically perfect gas (c_v is a constant), the thermodynamic quantities e and h are related to the temperature T through

$$e = c_v T, \quad h = c_p T, \tag{2.13}$$

where c_v is the specific heat at constant volume and c_p the specific heat at constant pressure. The specific heat ratio γ is defined as

$$\gamma = \frac{c_p}{c_v}, \tag{2.14}$$

being 7/5 for a diatomic perfect gas. Note that

$$h = \gamma e. \tag{2.15}$$

Equation of state
There are four equations ((2.1), (2.2) and (2.9)) in 2D, with five unknowns: m^α and the variables of state (e.g. ρ, h and p). According to thermodynamics, for simple systems there are only two independent variables of state on which the other variables of state depend. The relation that couples the variables of state is the equation of state for a perfect gas:

$$p = \frac{\gamma - 1}{\gamma} \rho h = \frac{\gamma - 1}{\gamma} \rho \left[H - \frac{1}{2} u^\alpha u^\alpha \right]. \tag{2.16}$$

As primary variables we take m^α, ρ and H; all other variables can be derived from these.

Values for air
The following values for air hold (see pages 258 and 259 of [5]):

- ρ at standard conditions equals approx. 1.22 kg/m^3.

- $R = 287$ m^2/(s^2 K) and $\gamma = 1.4$. Consequently, $c_p = 1{,}005$ m^2/(s^2 K) and $c_v = 718$ m^2/(s^2 K).

- The viscosity and thermal conductivity follow from the experimental laws of Sutherland:

$$\mu = \mu(T) = C_1 \frac{T^{3/2}}{C_2 + T} \quad k = k(T) = C_3 \frac{T^{3/2}}{C_4 + T}, \tag{2.17}$$

  where C_1-C_4 are constants for a given gas. For air at moderate temperatures, one has $C_1 = 1.458 \cdot 10^{-6}$ kg/(m s K$^{1/2}$), $C_2 = 110.4$ K, $C_3 = 2.495 \cdot 10^{-3}$ (kg m)/(s^3 K$^{3/2}$) and $C_4 = 194$ K.

- The Prandtl number

$$\mathrm{Pr} = \frac{c_p \mu}{k} \tag{2.18}$$

  is often used to determine the thermal conductivity k once μ is known. This is possible because the ratio (c_p/Pr), which appears in the expression

$$k = \frac{c_p}{\mathrm{Pr}} \mu \tag{2.19}$$

  is approximately constant for most gases. For air at standard conditions, Pr = 0.72.

- Coefficient k is called the thermal conductivity, appearing in Fourier's law $q = -k\nabla T$. The thermal diffusion coefficient k_H is defined as

$$k_H = \frac{k}{\rho c_p}. \tag{2.20}$$

Euler equations

The Euler equations are the compressible Navier-Stokes in which the viscous and diffusion terms are omitted, i.e. $\mu = k = 0$. This leads to the following set of equations:

- Continuity equation (the same as equation (2.1):

$$\frac{\partial \rho}{\partial t} + m^\alpha_{,\alpha} = 0. \tag{2.21}$$

- Momentum equation (see equation (2.2)):

$$\frac{\partial m^\alpha}{\partial t} + \left(u^\beta m^\alpha\right)_{,\beta} = -p_{,\alpha}. \tag{2.22}$$

- Energy equation (see equation (2.9):

$$\frac{\partial \rho E}{\partial t} + (m^\alpha H)_{,\alpha} = 0 \tag{2.23}$$

- Equation of state (the same as equation (2.16)):

$$p = \frac{\gamma - 1}{\gamma} \rho \left[H - \frac{1}{2} u^\alpha u^\alpha \right]. \tag{2.24}$$

Incompressible equations
The incompressible Euler equations are obtained by inserting $\rho = 1$ (constant) in the Euler equations. Due to this, the energy equation and equation of state become obsolete.

- Continuity equation:

$$u^{\alpha}_{,\alpha} = \nabla \cdot \mathbf{u} = 0. \tag{2.25}$$

- Momentum equation (see equation (2.2)):

$$\frac{\partial u^{\alpha}}{\partial t} + \left(u^{\beta} u^{\alpha}\right)_{,\beta} = -p_{,\alpha}. \tag{2.26}$$

Notice that the only variables that play a role now are the velocity (which is equal to the momentum) and the pressure. The density and energy are not present anymore

2.3 Non-dimensionalization of the governing equations

In this section the non-dimensionalization of the governing equations is given. In Section 2.3.1 the 'standard' compressible non-dimensionalization is discussed. In Section 2.3.2 the non-dimensionalization as is done for the Mach uniform formulation is discussed.

2.3.1 Standard approach

The Navier-Stokes equations are non-dimensionalized. To this aim we need to choose four independent reference values (namely, ρ_r T_r, u_r and a reference length L_r); all other reference values are readily obtained. For example, $t_r = L_r/u_r$ (time-scale), $\mu_r = \mu(\rho_r, T_r)$ (dynamic viscosity), $h_r = c_p T_r$ (enthalpy), $a_r = \sqrt{(\gamma-1)h_r}$ (speed of sound) and

$$p_r = \frac{\gamma - 1}{\gamma}\rho_r h_r. \tag{2.27}$$

Continuity equation
Non-dimensionalization leaves the continuity equation (2.1) invariant:

$$\frac{\partial \rho}{\partial t} + m^{\alpha}_{,\alpha} = 0. \tag{2.28}$$

Momentum equation
Non-dimensionalization of the momentum equation leads to:

$$\frac{\partial m^{\alpha}}{\partial t} + \left(u^{\beta} m^{\alpha}\right)_{,\beta} = -\frac{1}{\gamma M_r^2} p_{,\alpha} + \frac{1}{\mathrm{Re}}\tau^{\alpha\beta}_{,\beta}, \tag{2.29}$$

where we made use of

$$\frac{u_r^2}{h_r} = \frac{u_r^2}{a_r^2/(\gamma-1)} = (\gamma-1)M_r^2, \tag{2.30}$$

and the reference Mach number M_r is defined by:

$$M_r = \frac{u_r}{a_r}. \tag{2.31}$$

The Reynolds number is given by:

$$\mathrm{Re} = \frac{\rho_r u_r L_r}{\mu_r}. \tag{2.32}$$

Energy equation
Since $h = \gamma e$, we have

$$h_r \tilde{h} = \gamma e_r \tilde{e}. \tag{2.33}$$

We define $h_r = \gamma e_r$, resulting in $\tilde{h} = \tilde{e}$. Non-dimensionalization of the energy equation yields:

$$\frac{1}{\gamma}\frac{\partial \rho E}{\partial t} + (m^{\alpha} H)_{,\alpha} = \left(\frac{(\gamma-1)M_r^2}{\mathrm{Re}} u^{\alpha}\tau^{\alpha\beta} + \frac{1}{\mathrm{Re}\,\mathrm{Pr}} k h_{,\beta}\right)_{,\beta}, \tag{2.34}$$

where the Prandtl number is given in equation (2.18).
Non-dimensionalization of equation (2.11) is somewhat more subtle. Let quantities with a tilde denote the dimensionless values, quantities with subscript r denote the reference values and the quantities without subscript or tilde denote the original, dimensionful value. We write:

$$H = H_r\tilde{H} \qquad h = h_r\tilde{h} \qquad u = u_r\tilde{u}. \tag{2.35}$$

Inserting this in equation (2.11) leads to:

$$H = h_r\left[\tilde{h} + \frac{1}{2}\frac{u_r^2}{h_r}\tilde{u}^2\right]. \tag{2.36}$$

With (2.30) this becomes

$$H = h_r\left[\tilde{h} + \frac{1}{2}(\gamma-1)M_r^2\tilde{u}^2\right]. \tag{2.37}$$

Choosing $H_r = h_r$ gives for the dimensionless total enthalpy, omitting the tildes:

$$H = h + \frac{1}{2}(\gamma-1)M_r^2\, u^\alpha u^\alpha. \tag{2.38}$$

Similarly, writing $E_r = e_r$, yields the following relations for the dimensionless total energy in dimensionless quantities, and omitting the tildes:

$$\begin{aligned} E &= e + \frac{1}{2}\gamma(\gamma-1)M_r^2u^2 = h + \frac{1}{2}\gamma(\gamma-1)M_r^2u^2 = \\ &= h + \frac{1}{2}(\gamma-1)M_r^2u^2 + \frac{1}{2}(\gamma-1)(\gamma-1)M_r^2u^2 = H + \frac{1}{2}(\gamma-1)^2M_r^2u^2. \end{aligned} \tag{2.39}$$

In the second step we used $h = e$ (quantities are dimensionless), and in the last step we used (2.38).

Equation of state
The non-dimensional equation of state is:

$$p = \rho h = \rho\left[H - \frac{1}{2}(\gamma-1)M_r^2\, u^\alpha u^\alpha\right]. \tag{2.40}$$

Euler equations
The non-dimensionalized Euler equations are given by:

- Continuity equation:

$$\frac{\partial\rho}{\partial t} + m^\alpha_{,\alpha} = 0. \tag{2.41}$$

- Momentum equation:

$$\frac{\partial m^\alpha}{\partial t} + \left(u^\beta m^\alpha\right)_{,\beta} = -\frac{1}{\gamma M_r^2}p_{,\alpha}. \tag{2.42}$$

- Energy equation:

$$\frac{1}{\gamma}\frac{\partial \rho E}{\partial t} + (m^\alpha H)_{,\alpha} = 0. \tag{2.43}$$

- Equation of state:

$$p = \rho h = \rho \left[H - \frac{1}{2}(\gamma - 1) M_r^2 \, u^\alpha u^\alpha \right]. \tag{2.44}$$

Incompressible equations
The non-dimensionalized incompressible equations are the same as the dimensionalized incompressible equations, equations (7.1) and (7.2).

Remarks

- Note that, once the dimensionless initial and boundary conditions are available, we need two reference values (M_r and Re) to solve the dimensionless set of equations (2.28), (2.29), (2.34) and (2.40). For the Euler equations, we only need one reference value, namely M_r.
  To compute the Mach number M in each point of the flow, using only dimensionless quantities, the following relation must be used:

$$M = \frac{u}{a} = \frac{u}{\sqrt{(\gamma-1)h}} = \frac{u_r \tilde{u}}{\sqrt{(\gamma-1) h_r \tilde{h}}} = \frac{u_r}{a_r}\frac{\tilde{u}}{\sqrt{\tilde{h}}} = M_r \frac{\tilde{u}}{\sqrt{\tilde{h}}}, \tag{2.45}$$

  where q indicates the dimensionful quantity q, and $\tilde{q}$ refers to the dimensionless quantity.

- The Courant number is defined by

$$\sigma = \frac{(u+a)\Delta t}{\Delta x}. \tag{2.46}$$

  Non-dimensionalization leads to, with $\tilde{u}$ and $\tilde{h}$ as input:

$$\sigma = \frac{(\tilde{u} + \frac{1}{M_r}\sqrt{\tilde{h}})\tilde{\Delta} t}{\tilde{\Delta} x}. \tag{2.47}$$

2.3.2 Mach uniform non-dimensionalization

Start with the compressible Euler equations:

$$\frac{\partial \rho}{\partial t} + (u^\alpha \rho)_{,\alpha} = 0, \tag{2.48}$$

$$\frac{\partial m^\alpha}{\partial t} + (u^\beta m^\alpha)_{,\beta} = -p_{,\alpha}, \tag{2.49}$$

$$\frac{\partial (\rho E)}{\partial t} + (u^\alpha \rho H)_{,\alpha} = 0. \tag{2.50}$$

Meaning of the quantities: ρ is density; t is time; u^α is velocity; m^α is momentum; p is pressure; E is total energy and H is total enthalpy. The system of equations is closed by the equation of state for a perfect gas:

$$p = (\gamma - 1)\rho e. \tag{2.51}$$

Additional useful relations:

$$\begin{aligned} m^\alpha &= \rho u^\alpha && (2.52)\\ H &= h + \frac{1}{2}\mathbf{u}^2, && (2.53)\\ E &= e + \frac{1}{2}\mathbf{u}^2, && (2.54)\\ h &= c_p T, && (2.55)\\ e &= c_v T, && (2.56)\\ h &= \gamma e, && (2.57)\\ M &= u/a && (2.58)\\ a^2 &= (\gamma - 1)h, && (2.59) \end{aligned}$$

where h is the enthalpy; e is the internal energy; $\mathbf{u}^2 = \mathbf{u} \cdot \mathbf{u} = u^\alpha u^\alpha$; c_p and c_v are specific heat values; $\gamma = c_p/c_v$ is the specific heat ratio; M is the Mach number, a is the speed of sound. The equations are made dimensionless by choosing suitable reference quantities u_r, L_r (length), T_r (temperature) and ρ_r. The dimensionless quantities, indicates by $\tilde{\phi}$, follow from:

$$\tilde{\phi} = \phi/\phi_r, \tag{2.60}$$

where ϕ_r is the reference quantity, and ϕ is any quantity (for example, $\mathbf{x}$, ρ, $\mathbf{u}$, h, e, M, and so on), except p. We can derive, from the relations given above, the following expressions:

$$\begin{aligned} h_r &= \gamma e_r, && (2.61)\\ u_r^2/h_r &= (\gamma - 1)M_r^2, && (2.62)\\ \tilde{h} &= \tilde{e} && (2.63)\\ t_r &= L_r/u_r. && (2.64) \end{aligned}$$

The pressure is made dimensionless using:

$$\tilde{p} = (p - p_r)/\rho_r u_r^2, \tag{2.65}$$

where, from the equation of state:

$$p_r = (\gamma - 1)\rho_r e_r. \tag{2.66}$$

Note that if we, in the case of subsonic flow, pressure $\tilde{p} = 0$ at the outflow, we write actually: $p_r = p_{out}$. Making use of the relations given above yields the following dimensionless equation of state:

$$\tilde{h} = \frac{1}{\tilde{\rho}}(1 + \gamma M_r^2 \tilde{p}). \tag{2.67}$$

We can also write this expression as follows:

$$\tilde{\rho} = \frac{1}{\tilde{h}}(1 + \gamma M_r^2 \tilde{p}), \qquad \tilde{p} = \frac{1}{\gamma M_r^2}(\tilde{\rho}\tilde{h} - 1). \tag{2.68}$$

Note that, when $M_r \downarrow 0$, the density becomes indepent of the pressure, and that the expression for $\tilde{p}$ becomes singular (this can be avoided by considering $\gamma M_r^2 \tilde{p}$.

The next point is to arrive at a dimensionless version of the Euler equations. It is not hard to show that the continuity equation and momentum equation are not affected by the scaling. After some manipulations, using the relations given above, one arrives at the following dimensionless energy equation:

$$M_r^2 \left\{ \frac{\partial}{\partial t} \left[p + \frac{1}{2}(\gamma - 1)\rho \mathbf{u}^2 \right] + \nabla \cdot \mathbf{u} \left[\gamma p + \frac{1}{2}(\gamma - 1)\rho \mathbf{u}^2 \right] \right\} + \nabla \cdot \mathbf{u} = 0, \tag{2.69}$$

where now the tildes have been omitted. Note that, as $M_r \downarrow 0$, the solenoidality condition on the velocity field for incompressible flows is recovered.

2.3.3 Choice among primary energy variables (implicit time-integration)

For implicit time-integration with respect to a primitive variable ϕ, it is necessary that ϕ itself is present in both the time-derivative as convection term. For explicit time-integration, see Section 2.3.4, this is not necessary.
Depending on the primary energy variable we take, the dimensionless energy equation (2.43) and equation of state (2.40) takes the following forms:

- primary variable h.
  Energy equation:

$$\frac{1}{\gamma}\frac{\partial}{\partial t}\left[\rho h + \frac{1}{2}\gamma(\gamma-1)M_r^2\frac{m^2}{\rho}\right] + \nabla\cdot\mathbf{m}\left[h + \frac{1}{2}(\gamma-1)M_r^2u^2\right] = 0 \tag{2.70}$$

  Equation of state:

$$p = \rho h \tag{2.71}$$

- primary variable H.
  Energy equation:

$$\frac{1}{\gamma}\frac{\partial}{\partial t}\left[\rho H + \frac{1}{2}(\gamma-1)^2M_r^2\frac{m^2}{\rho}\right] + \nabla\cdot\mathbf{m}H = 0 \tag{2.72}$$

  Equation of state:

$$p = \rho[H - \frac{1}{2}(\gamma-1)M_r^2u^2] \tag{2.73}$$

- primary variable (ρH).
  Energy equation:

$$\frac{1}{\gamma}\frac{\partial}{\partial t}\left[(\rho H) + \frac{1}{2}(\gamma-1)^2M_r^2\frac{m^2}{\rho}\right] + \nabla\cdot\mathbf{m}\frac{(\rho H)}{\rho} = 0 \tag{2.74}$$

  Equation of state:

$$p = (\rho H) - \frac{1}{2}(\gamma-1)M_r^2\rho u^2 \tag{2.75}$$

- primary variable (ρE).
  Energy equation:

$$\frac{1}{\gamma}\frac{\partial(\rho E)}{\partial t} + \nabla\cdot\mathbf{m}\left[\frac{(\rho E)}{\rho} - \frac{1}{2}(\gamma-1)^2M_r^2u^2\right] = 0 \tag{2.76}$$

  Equation of state:

$$p = (\rho E) - \frac{1}{2}\gamma(\gamma-1)M_r^2\rho u^2 \tag{2.77}$$

The following aspect with respect to the choice of primary variables must be noted:

- The advantage of choosing H or (ρH) as primary variable, is that when considering steady problems, we have to solve

$$\nabla \cdot \mathbf{m}H = \nabla \cdot \mathbf{u}\rho H = 0 \tag{2.78}$$

  which is simpler than the other cases. In this case we don't have a convection term consisting of two terms.

- Since $\mathbf{u}$ is the convecting velocity (and not $\mathbf{m}$), terms like $\nabla \cdot \mathbf{m}\phi$ have to be discretized as:

$$\int \nabla \cdot \mathbf{m}\phi \, d\mathbf{x} = \int \nabla \cdot \mathbf{u}\rho\phi \, d\mathbf{x} = \oint (\mathbf{u} \cdot \mathbf{n})\rho\phi \, d\Gamma \approx \sum_e u_e \rho_e \phi_e \bar{l}_e, \tag{2.79}$$

  where $u_e = m_e/\rho_e$, and ρ_e follows from a weighted averaging.

- The advantage of choosing (ρE) as primary variable, is that the time-derivative is trivial.

- The advantage of choosing h as primary variable, is that the equation of state is trivial.

2.3.4 Choice among primary energy variables (explicit time-integration)

As mentioned in Section 2.3.3, for explicit time-integration it is not necessary that primitive variable ϕ is directly present in both the time-derivative as convection term. Define

$$q = \rho E \qquad\qquad \phi = \rho H, \tag{2.80}$$

then we see that (equation (2.10))

$$\phi = q + p \tag{2.81}$$

holds for the dimensionful equations.
For the dimensionless variant, the following relations are relevant. The energy equation reads:

$$\frac{1}{\gamma}\frac{\partial q}{\partial t} + \nabla \cdot \mathbf{u}\phi = 0. \tag{2.82}$$

The equation of state is given by:

$$p = q - \frac{1}{2}\gamma(\gamma - 1)M_r^2 \rho u^2 \tag{2.83}$$

The relation between the dimensionless q and ϕ is given by

$$q = \gamma\phi - (\gamma - 1)p \qquad \Longleftrightarrow \qquad \phi = \frac{q + (\gamma - 1)p}{\gamma} \tag{2.84}$$

2.4 Non-dimensionalization of the Riemann problem

When considering Riemann problems (e.g. Lax, Sod, Mach 3), one gives (dimensionless) left and right initial states for the following quantities $\{\tilde{u}, \tilde{p}, \tilde{\rho}\}$. Two problems arise:

1. for the TUDFINVOL-code, I need to give initial conditions for the quantities $\{\tilde{m}, \tilde{h}, \tilde{p}\}$. See also Section 2.5

2. I need to compute quantities like the Mach number, total enthalpy and so on.

To solve these problems, the following approach must be taken.
We choose the reference values $\rho_r = L_r = u_r = p_r = 1$. As a consequence, we have $m_r = \rho_r u_r = 1$, hence $\tilde{m} = \tilde{\rho}\tilde{u}$. Since we have defined, in the beginning of Section 2.3 the relation $p_r = \frac{\gamma-1}{\gamma}\rho_r h_r$, we get

$$h_r = \frac{\gamma}{\gamma - 1}. \tag{2.85}$$

The dimensionless enthalpy, of course, can be obtained from the dimensionless equation of state (2.40): $\tilde{h} = \tilde{p}/\tilde{\rho}$.
The total enthalpy $\tilde{H}$ can be computed using relation (2.38). The reference Mach number M_r is computed as follows:

$$M_r = \frac{u_r}{a_r} = \frac{1}{\sqrt{(\gamma-1)h_r}} = \frac{1}{\sqrt{(\gamma-1)\frac{\gamma}{\gamma-1}}} = \frac{1}{\sqrt{\gamma}}, \tag{2.86}$$

and the Mach number M then follows from (2.45)

2.5 Initial conditions

In order to solve the (dimensionless) set of Navier-Stokes or Euler equations, all variables have to be known over the whole domain at the initial stage: the initial conditions. Since there are four independent variables, we need to prescribe these four independent variables at $t = 0$. At this moment, we prescribe (in the TUDFINVOL-code) $\mathbf{m}$, p and h. Relating these to the primary variables ($\mathbf{m}$, ρ and H) is done using the equation of state to obtain ρ, while obtaining the energy variable is done using relations given in the previous section, depending on which energy variable is taken to be the primary variable. The computation of the quantity m^2 at the cell-centers is discussed in Section 6.1.
For the incompressible flow we need to prescribe $\mathbf{u}$ and p at the initial time-level.

2.6 Boundary conditions

In order to solve the (dimensionless) set of Navier-Stokes or Euler equations, we need to prescribe boundary conditions: values of quantities at the boundaries, possibly a function of time and position at the boundary. Especially for the Euler equations it is a delicate matter which conditions must be prescribed at what boundary. For a more thorough treatment, we refer to Chapter 16.4 in [4].

Convection-diffusion equation
The most occuring types of boundary conditions for a convection-diffusion equation for scalar ϕ are:

- Dirichlet condition: ϕ is given as function of time and position at the boundary;
- homogeneous Neumann condition: $\partial\phi/\partial\mathbf{n} = 0$ at the boundary.

Momentum equation
For the momentum equation, the following types of boundary conditions can occur:

- the momentum-vector $\mathbf{m}$ is given as function of position on the boundary and time;
- σ^{nt} and the momentum component normal to the boundary are given;
- σ^{nn} and the momentum component tangential to the boundary are given;
- σ^{nn} and σ^{nt} are given.

With the stress-tensor $\sigma^{\alpha\beta}$ given by equation (2.7), n^i the unit vector normal to the boundary and t^i the unit vector tangential to the boundary, we can deduce that (in dimensionfull quantities):

$$\begin{aligned}
\sigma^{nn} = n_i\sigma^{ij}n_j &= -p - \frac{2}{3}\Delta\mu + 2\mu\frac{\partial u^n}{\partial n} \qquad u^n = u^i n^i \\
\sigma^{tt} = t_i\sigma^{ij}t_j &= -p - \frac{2}{3}\Delta\mu + 2\mu\frac{\partial u^t}{\partial t} \qquad u^t = u^i t^i \\
\sigma^{nt} = n_i\sigma^{ij}t_j &= \mu\left(\frac{\partial u^n}{\partial t} + \frac{\partial u^t}{\partial n}\right).
\end{aligned}$$

Note that prescribing σ^{nn} is for the Euler equations equivalent to prescribing the pressure.

Chapter 3

Description of an unstructured grid

3.1 Definition of an unstructured grid

In general we distinguish between two types of grids, namely structured and unstructured grids. A structured grid usually consists of blocks of cells. Each block can be mapped onto a rectangular grid with a constant number of cells in each of the coordinate directions. For example, the number of cells that meet each other in an internal vertex is always 4 in a 2D grid with quadrilaterals. In unstructured grids this restriction is abandonned. Furthermore, unstructured grids in 2D usually consist of triangles, like finite element grids.
The unstructured grids that we consider, consist solely out of triangles, also called cells. The edges of the triangles are called faces.

3.2 Relations between number of cells, vertices and faces

For a 2D mesh consisting solely of triangles, the number of cells C, the number of boundary faces E_b and internal faces E_i are related, see [2]:

$$C = \frac{1}{3}(E_b + 2E_i). \tag{3.1}$$

The number of faces E satisfies: $E = E_i + E_b$. With V the number of vertices and H the number of holes one can derive that

$$C + V = E + 1 - H. \tag{3.2}$$

For fine meshes the condition $E_i \gg E_b$ is usually satisfied. Using this condition and assuming that H is negligibly small, we arrive at

$$C \approx 2V, \qquad E \approx 3V. \tag{3.3}$$

Comparing the number of equations with a vertex-centered scheme, we see that for a scalar equation the staggered approach results in approximately twice as many equations. For the momentum equation, one in each direction for a vertex-centered scheme, the staggered approach results in approximately 3/2 as many equations.

Internal and boundary cells
We distinguish between two kinds of cells:

- Internal cells: internal cells have no faces at the boundary. Consequently, in the discretized equations no measures have to be taken to include boundary conditions.
- Boundary cells: one or more of its faces are positioned at a boundary. Consequently, in the discretized equations one or more variables are prescribed or related directly to values at the boundary.

Let C be the the total number of cells and C_i the number of internal cells, then $(C - C_i)$ is the number of boundary cells. The separation into two different cell types as proposed here is made on basis of geometry alone.

Real internal, quasi internal and boundary faces
We distinguish between two kinds of faces:

- Internal faces: internal faces are not positioned at the boundary of the domain.
- Boundary faces: are positioned at a boundary.

There are E_i internal faces and E_b boundary faces, and $E = E_i + E_b$.
For a suitable implementation of the discretization of the momentum equation, see Chapter 5, we need to distinguish between two kinds of internal faces:

- Real internal faces: the two cells adjacent to this face are both internal cells.
- Quasi internal faces: at least one of the adjacent cells is a boundary cell.

Let the number of real internal faces be E_{ir}, then the number of quasi internal faces follows from $(E_i - E_{ir})$.

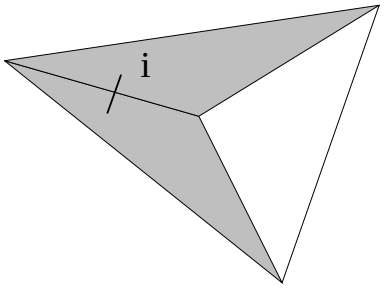

Figure 3.1: A (forbidden) situation in which only three faces meet at a vertex. The control volume for the momentum at face i is shaded.

3.3 Conditions imposed to the grid

In order to avoid problems with the discretization, or badly shaped grids, the grids must satisfy the following properties, see also Section 16.1:

- the number of faces/cells that meet at each vertex in the interior of the mesh must be at least four. The discretization is not suited to deal with situations in which only three faces meet at a vertex, see Figure 3.1. This because otherwise the treatment of the convection-term becomes too complicated.
  (Note: Until now no meshes were encountered in which this criterion was not satisfied: probably the SEPRAN mesh-generator itself does not allow such a situation.)

- the angles inside each triangle must be smaller than a certain value. Let $\mathbf{a}$ and $\mathbf{b}$ be the tangential vectors of two faces of a triangle, then the angle ϕ between these two faces follows from:

$$\cos\phi = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|}. \tag{3.4}$$

  At this moment we don't allow for angles with $\cos\phi < -0.8$, or, equivalently, $\phi > 143.13^o$.

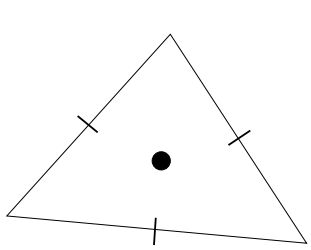

Figure 3.2: Staggered grid. At the cell-centroids the scalars are located; the normal momentum components are positioned at the midpoints of the faces.

3.4 Staggered set-up

The primitive variables in our approach are the density ρ, the momentum vector $\mathbf{m}$ and an energy variable (e.g. h, H or (ρH)). The pressure follows from the equation of state (2.40).
In Figure 3.2 the employed staggered placement of variables in the grid is shown. At the cell centroids the scalar variables like density, pressure and enthalpy are located. The projected momentum m, i.e. the component of the momentum vector parallel to the normal of the cell face, is stored at the midpoint of this face. This placement of the variables is similar to the classic staggered scheme on structured grids with quadrilateral cells ([3]), used by our group in, for example, [1], [9], [8], [10] and [7]. At every face a normal vector $\mathbf{N}$ is defined in a unique manner. The momentum equation for $m = \mathbf{m} \cdot \mathbf{N} = m^\alpha N^\alpha$ follows directly from a projection of (2.29) on $\mathbf{N} = N^\alpha$.

Chapter 4

Spatial discretization of a convection-diffusion equation

4.1 Introduction

A general form for a convection-diffusion equation for scalar ϕ is given by:

$$\frac{\partial(a\phi)}{\partial t} + \nabla \cdot (b\mathbf{u}\phi) - \nabla \cdot (c\nabla\phi) + d\phi = g, \tag{4.1}$$

where a, b, c, d and g are coefficients that may depend on space, time and previously computed solutions. The second term in (4.1) is called the convection term, and the third term is the diffusion (conduction) term. In this chapter we discuss the discretization, implementation of boundary conditions and the numerical treatment of the convection term. Since the energy and continuity equation are special forms of (4.1), they are also treated in this chapter.

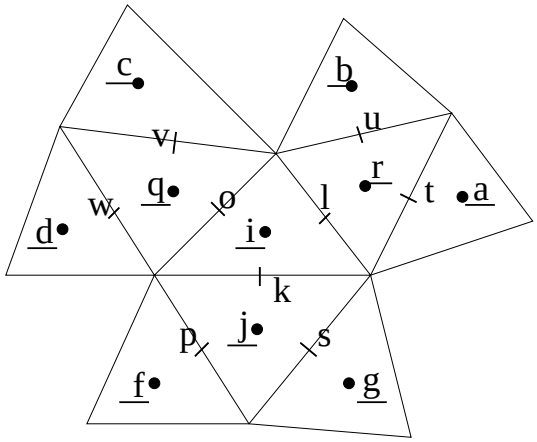

Figure 4.1: Cell-numbers are underlined, face-numbers not.

4.2 Convection-diffusion equation: discretization

The control volume (CV) for scalar ϕ_i, positioned in the cell-center of cell i, is the triangle i itself with area Ω_i, see Figure 4.1. The stencil as given in this figure, is the one that we have implemented.

Integration of (4.1) over the CV and application of Gauss' divergence theorem leads to:

$$\begin{aligned}\Omega_i \frac{a_i^{n+1}\phi_i^{n+1}}{\tau} \quad &+ \quad \sum_e b_e^{n+1} u_e^{n+1} \phi_e^{n+1} \bar{l}_e - \sum_e c_e^{n+1} (\nabla \phi^{n+1} \cdot \mathbf{N})_e \bar{l}_e + \Omega_i d_i^{n+1} \phi_i^{n+1} = \\ &= \quad \Omega_i \frac{a_i^n \phi_i^n}{\tau} + \Omega_i g_i^{n,n+1}. \end{aligned} \tag{4.2}$$

The following aspects must be noted:

- An implicit Euler time-integration scheme is used, see Chapter 9. Parameter τ represents the time-step.
- The summation over e represents the summation over the faces of the CV, i.e. $e = \{k, l, o\}$.
- With $\mathbf{N}_e$ the uniquely defined normal at face e, and $\mathbf{n}_e$ the, with respect to cell i, outwards pointing normal, the convection term is treated as follows:

$$\int_{\Omega_i} \nabla \cdot (b\mathbf{u}\phi)\, d\Omega = \oint_{\partial\Omega_i} b(\mathbf{u}\cdot\mathbf{n})\phi\, d\Gamma \approx \sum_e b_e(\mathbf{u}_e \cdot \mathbf{n}_e)\phi_e l_e = \sum_e b_e u_e \phi_e \bar{l}_e, \tag{4.3}$$

  where $u_e = \mathbf{u}_e \cdot \mathbf{N}_e$ and

$$\bar{l}_e = l_e(\mathbf{n}_e \cdot \mathbf{N}_e). \tag{4.4}$$

  This can be proven by noting that $|\mathbf{n}_e| = |\mathbf{N}_e| = 1$ and $\mathbf{n}_e = \pm\mathbf{N}_e$ leads to:

$$\mathbf{n}_e = (\mathbf{n}_e \cdot \mathbf{N}_e)\mathbf{N}_e. \tag{4.5}$$

  Note that $(\mathbf{n}_e \cdot \mathbf{N}_e) = \pm 1$ always.

- Note that u_e is computed using $u_e = m_e/\rho_e$, where ρ_e follows from linear interpolation among adjacent cell-center values.
- Computation of b_e is done by interpolating values of b located at the cell-centers.
- The computation of ϕ_e^{n+1} is treated in Section 4.4.
- The computation of $(\nabla\phi^{n+1} \cdot \mathbf{N})_e$ can be found in Section 5.7.
- Note that (4.2) forms a linear system of the form: $A\mathbf{x} = \mathbf{b}$, where A represents the $C \times C$-matrix following from the left-hand side, vector $\mathbf{x} = (\phi_1^{n+1}, \ldots, \phi_C^{n+1})$, with C the number of cells, and $\mathbf{b}$ stands for the right-hand side of (4.2). For more information concerning the linear solver, see Chapter 10.
- For explicit time-integration, see Section 9.3.

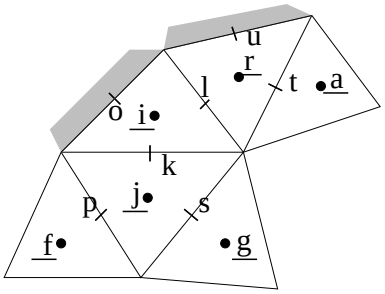

Figure 4.2: Boundary cell i with boundary face o.

4.3 Convection-diffusion equation: boundary conditions

At this moment we have restricted ourselves to Dirichlet and homogeneous Neumann boundary conditions. Note that for the Euler equations ($c = 0$), no homogeneous Neumann conditions may be given. However, boundary faces on which no Dirichlet condition for ϕ is given, are treated numerically (using one-sided differences) as if a homogeneous Neumann condition were given.
Consider a boundary cell i with boundary face o, see Figure 4.2. Note that the problem is to find expressions for ϕ_o^{n+1} and $(\nabla\phi^{n+1} \cdot \mathbf{N})_o$ in the convection and diffusion term in equation (4.2). Note that this equation is not altered in the presence of boundaries; only the numerical treatment is different. At this moment, we restrict ourselves to the Euler case, leading to the following consequences of the boundary conditions:

- Dirichlet boundary condition: ϕ_o^{n+1} is given. The term $-b_o^{n+1}u_o^{n+1}\bar{l}_o\phi_o^{n+1}$ is moved to the right-hand side. The coefficients $\alpha_{o,i}$ and $\alpha_{o,q}$, equation (4.6), are put to zero.

- Homogeneous Neumann boundary condition: $(\nabla\phi^{n+1} \cdot \mathbf{N})_o = 0$. This condition is treated using one-sided differences: $\phi_o^{n+1} = \phi_i^{n+1}$. This leads to a contribution of size $b_o^{n+1}u_o^{n+1}\bar{l}_o$ to the diagonal. The coefficients $\alpha_{o,i}$ and $\alpha_{o,q}$, equation (4.6), are put to zero.

4.4 Convection-diffusion equation: convection term

The term ϕ_e^{n+1}, appearing in the convection term of equation (4.2), is obtained from interpolations from neighbouring scalar values using central or upwind methods. We have implemented a first-order upwind and a central interpolation scheme. This means that ϕ_e^{n+1} is determined by the scalar values in the two cells adjacent to the face. Consequently, we write, with reference to Figure 4.1, and omitting the superscript $n+1$:

$$\begin{aligned} \phi_k &= \alpha_{k,i}\phi_i + \alpha_{k,j}\phi_j, \\ \phi_l &= \alpha_{l,i}\phi_i + \alpha_{l,r}\phi_r \\ \phi_o &= \alpha_{o,i}\phi_i + \alpha_{o,q}\phi_q. \end{aligned} \tag{4.6}$$

The coefficients α depend on the choice of upwind method. Note that consistency demands that $\alpha_{k,i} + \alpha_{k,j} = \alpha_{l,i} + \alpha_{l,r} = \alpha_{o,i} + \alpha_{o,q} = 1$.

4.4.1 Convection-diffusion equation: first order upwind scheme

The first order upwind scheme is based on taking upstream values. For ϕ_o, see Figure 4.1, this results in:

$$\phi_o = \begin{cases} \phi_q & \text{if the flow is from cell } q \text{ to } i; \\ \phi_i & \text{if the flow is from cell } i \text{ to } q. \end{cases} \tag{4.7}$$

This statement can be rewritten in the following form:

$$\phi_o = \begin{cases} \phi_q & \text{if } m_o^{n+1}\bar{l}_o < 0; \\ \phi_i & \text{if } m_o^{n+1}\bar{l}_o > 0. \end{cases} \tag{4.8}$$

Using $\phi_o = \alpha_i\phi_i + \alpha_q\phi_q$, yet another form of this equation is

$$\alpha_i = \begin{cases} 0 & \text{if } m_o^{n+1}\bar{l}_o < 0; \\ 1 & \text{if } m_o^{n+1}\bar{l}_o > 0. \end{cases} \tag{4.9}$$

and $\alpha_q = 1 - \alpha_i$.

4.4.2 Convection-diffusion equation: central scheme

With $\phi_o = \alpha_i\phi_i + \alpha_q\phi_q$, we find that, see below:

$$\alpha_i = \frac{\Omega_q}{\Omega_i + \Omega_q} \qquad \alpha_q = \frac{\Omega_i}{\Omega_i + \Omega_q}. \tag{4.10}$$

Using arguments as given in [6], it may be better to use

$$\alpha_i = \frac{1}{2} \qquad \alpha_q = \frac{1}{2} \tag{4.11}$$

to reduce the global error. This is discussed here below.

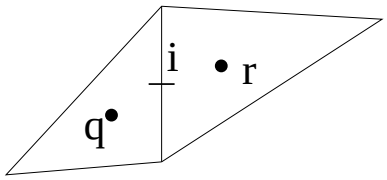

Figure 4.3: The cell-centers are q and r, and the face is indicated with i.

Spectral properties of the central scheme
The discretized convection equation may be written as:

$$D\frac{d\boldsymbol{\phi}}{dt} + C(\mathbf{u})\boldsymbol{\phi} = \mathbf{r}, \tag{4.12}$$

where D is a diagonal positive definite matrix containing the areas of the control volumes, C is the discretized convection operator, $\boldsymbol{\phi}$ is the solution vector and $\mathbf{r}$ are the other terms. Veldman et al. [6] suggest that the discrete operators should mimick physical properties, hence C should be anti-symmetric:

$$C_{ij} = -C_{ji} \qquad \forall\, i,j \qquad\qquad C_{ii} = 0. \tag{4.13}$$

Using central interpolations of the type: $\phi_o = \frac{1}{2}(\phi_i + \phi_q)$, leads to the following matrix elements:

$$C_{ii} = \frac{1}{2}(u_o\bar{l}_o + u_k\bar{l}_k + u_l\bar{l}_l) \qquad\qquad C_{iq} = \frac{1}{2}u_o\bar{l}_o. \tag{4.14}$$

Notice that $C_{ii} = 0$ for incompressible flows. Doing the same, but now for cell q, leads to:

$$C_{qi} = \frac{1}{2}u_o\bar{l}_o, \tag{4.15}$$

where $\bar{l}_o$ now has the opposite sign as in C_{iq}. Hence, $C_{iq} = -C_{qi}$.

Weighted interpolation
In several parts of the computations, a weighted interpolation is used to interpolate scalars, known at the cell-centers, to the cell faces.

Knowing ϕ_q and ϕ_r, we want to compute ϕ_i, see Figure 4.3. Weighted interpolation of ϕ along line qr yields, at the intersection between this line and face i:

$$\frac{\Omega_q}{\Omega_q + \Omega_r}\phi_r + \frac{\Omega_r}{\Omega_q + \Omega_r}\phi_q, \tag{4.16}$$

with Ω indicating the area of the cells.
This can be proven by noting that, with h_q (h_r) the shortest distance between cell q (r) and face i, weighted interpolation leads to

$$\frac{h_q}{h_q + h_r}\phi_r + \frac{h_r}{h_q + h_r}\phi_q \tag{4.17}$$

at the mentioned intersection. With $\Omega_i = \frac{1}{2}(3h_i)l_o$ and $\Omega_q = \frac{1}{2}(3h_q)l_o$ this results in equation (4.16).
We use (4.16) as an approximation for ϕ_i, even when the mentioned intersection is not positioned at the midpoint of face i.

4.5 Energy equation

The energy equation is of the form (4.1). For the moment, we restrict ourselves to the dimensionless Euler form, i.e. equation (2.43).

4.5.1 Evaluation of time derivative of kinetic energy

The term

$$\int_{\Omega_i} \left(\frac{(\mathbf{m}\cdot\mathbf{m})^{n+1}}{\tau\rho^{n+1}} - \frac{(\mathbf{m}\cdot\mathbf{m})^{n}}{\tau\rho^{n}} \right) d\Omega \tag{4.18}$$

must be evaluated. Omitting the superscripts, we write:

$$\int_{\Omega_i} \frac{(\mathbf{m}\cdot\mathbf{m})}{\rho}\, d\Omega \approx \Omega_i \frac{(\mathbf{m}\cdot\mathbf{m})_i}{\rho_i}. \tag{4.19}$$

The computation of $(\mathbf{m}\cdot\mathbf{m})_i$ will be discussed in Chapter 6.

4.5.2 Energy equation: enthalpy as primary variable

With h as primary variable the energy equation is given by (2.70). Discretization and comparison with (4.2 leads to the following relations for the coefficients:

$$\begin{aligned}
a_i^{n+1} &= \frac{\rho_i^{n+1}}{\gamma} \qquad a_i^n = \frac{\rho_i^n}{\gamma} \qquad b_e^{n+1} = \rho_e^{n+1} \qquad c_e^{n+1} = d_i^{n+1} = 0 \\
\Omega_i g_i^{n,n+1} &= -\frac{1}{2}(\gamma-1)M_r^2\Omega_i \left(\frac{(\mathbf{m}\cdot\mathbf{m})_i^{n+1}}{\tau\rho_i^{n+1}} - \frac{(\mathbf{m}\cdot\mathbf{m})_i^{n}}{\tau\rho_i^{n}} \right) - \\
&- \frac{1}{2}(\gamma-1)M_r^2 \sum_e m_e^{n+1} \frac{(\mathbf{m}\cdot\mathbf{m})_e^{n+1}}{(\rho_e^{n+1})^2} \bar{l}_e.
\end{aligned} \tag{4.20}$$

The implementation of several terms in coefficient $\Omega_i g_i^{n,n+1}$ is discussed in Sections 4.5.1 and 4.5.7.

4.5.3 Energy equation: total enthalpy as primary variable

With H as primary energy variable, we have (2.72) as energy equation. Discretization and comparison with (4.2) leads to the following relations for the coefficients:

$$\begin{aligned}
a_i^{n+1} &= \frac{\rho_i^{n+1}}{\gamma} \qquad a_i^n = \frac{\rho_i^n}{\gamma} \qquad b_e^{n+1} = \rho_e^{n+1} \qquad c_e^{n+1} = d_i^{n+1} = 0 \\
\Omega_i g_i^{n,n+1} &= -\frac{1}{2}\frac{(\gamma-1)^2}{\gamma} M_r^2 \Omega_i \left(\frac{(\mathbf{m}\cdot\mathbf{m})_i^{n+1}}{\tau\rho_i^{n+1}} - \frac{(\mathbf{m}\cdot\mathbf{m})_i^{n}}{\tau\rho_i^{n}} \right) d\Omega.
\end{aligned} \tag{4.21}$$

The implementation of the terms in coefficient $\Omega_i g_i^{n,n+1}$ is discussed in Section 4.5.1.
When H is the primary energy variable, we have a Dirichlet boundary condition for H only when both h and $\mathbf{m}$ (or $|\mathbf{m}|$) are given at the boundary. The total enthalpy at the boundary is then computed using

$$H = h + \frac{1}{2}(\gamma-1)M_r^2 \frac{(\mathbf{m}\cdot\mathbf{m})}{\rho^2}, \tag{4.22}$$

where for the density ρ we take the value in the adjacent boundary cell.

4.5.4 Energy equation: density times total enthalpy as primary variable

With (ρH) as primary energy variable, we have (2.74) as energy equation. Discretization and comparison with (4.2) leads to the following relations for the coefficients:

$$\begin{aligned} a_i^{n+1} &= \frac{1}{\gamma} \qquad a_i^n = \frac{1}{\gamma} \qquad b_e^{n+1} = 1 \qquad c_e^{n+1} = d_i^{n+1} = 0 \\ \Omega_i g_i^{n,n+1} &= -\frac{1}{2}\frac{(\gamma-1)^2}{\gamma} M_r^2 \Omega_i \left(\frac{(\mathbf{m}\cdot\mathbf{m})_i^{n+1}}{\tau \rho_i^{n+1}} - \frac{(\mathbf{m}\cdot\mathbf{m})_i^{n}}{\tau \rho_i^{n}} \right) d\Omega. \end{aligned} \tag{4.23}$$

The implementation of the terms in coefficient $\Omega_i g_i^{n,n+1}$ is discussed in Section 4.5.1.
When (ρH) is the primary energy variable, we have a Dirichlet boundary condition for (ρH) only when both h and $\mathbf{m}$ (or $|\mathbf{m}|$) are given at the boundary. The quantity (ρH) at the boundary is then computed using

$$\rho H = \rho h + \frac{1}{2}(\gamma-1) M_r^2 \frac{(\mathbf{m}\cdot\mathbf{m})}{\rho}, \tag{4.24}$$

where for the density ρ we take the value in the adjacent boundary cell.

4.5.5 Energy equation: density times total energy as primary variable

With (ρE) as primary variable the energy equation is given by (2.76). Discretization and comparison with (4.2 leads to the following relations for the coefficients:

$$\begin{aligned} a_i^{n+1} &= \frac{1}{\gamma} \qquad a_i^n = \frac{1}{\gamma} \qquad b_e^{n+1} = 1 \qquad c_e^{n+1} = d_i^{n+1} = 0 \\ \Omega_i g_i^{n,n+1} &= \frac{1}{2}(\gamma-1)^2 M_r^2 \sum_e m_e^{n+1} \frac{(\mathbf{m}\cdot\mathbf{m})_e^{n+1}}{(\rho_e^{n+1})^2} \bar{l}_e. \end{aligned} \tag{4.25}$$

The implementation of several terms in coefficient $\Omega_i g_i^{n,n+1}$ is discussed in Sections 4.5.1 and 4.5.7.
When (ρE) is the primary energy variable, we have a Dirichlet boundary condition for (ρE) only when both h and $\mathbf{m}$ (or $|\mathbf{m}|$) are given at the boundary. The quantity (ρE) at the boundary is then computed using

$$\rho E = \rho h + \frac{1}{2}\gamma(\gamma-1) M_r^2 \frac{(\mathbf{m}\cdot\mathbf{m})}{\rho}, \tag{4.26}$$

where for the density ρ we take the value in the adjacent boundary cell.

4.5.6 Energy equation: explicit time-integration

The dimensionless energy equation, see Section 2.3.4, is discretized and implemented as well. We have implemented it only for primary energy variable $\phi = \rho H$.
A minor problem is formed by the fact that we do not store variable q. Noting that we do store primary variable ϕ and pressure p, the time-derivative is discretized as (use equation (2.84)):

$$\frac{\partial q}{\partial t} \approx \frac{q^{n+1} - q^n}{\Delta t} = \frac{q^{n+1} - \gamma\phi^n + (\gamma-1)p^n}{\Delta t}. \tag{4.27}$$

The explicit variant of equation (4.2) for the energy equation discussed here, is:

$$\begin{aligned}\Omega_i \frac{a_i^{n+1} q_i^{n+1}}{\tau} &+ \sum_e b_e^n u_e^n \phi_e^n \bar{l}_e - \sum_e c_e^n (\nabla\phi^n \cdot \mathbf{N})_e \bar{l}_e + \Omega_i d_i^n \phi_i^n = \\ &= \Omega_i \frac{a_i^n \rho_i^n \phi_i^n}{\tau} + \Omega_i g_i^{n,n+1}, \end{aligned} \tag{4.28}$$

with coefficients given by:

$$\begin{aligned} a_i^{n+1} &= \frac{1}{\gamma} \qquad a_i^n = 1 \qquad b_e^n = 1 \qquad c_e^n = d_i^n = 0 \\ \Omega_i g_i^{n,n+1} &= -(\gamma-1)\Omega_i a_i^{n+1} \frac{p_i^n}{\tau}. \end{aligned} \tag{4.29}$$

4.5.7 Convection of kinetic energy

In several convection terms in the preceding sections the following term, representing convection of the kinetic energy, appears in the right-hand side:

$$-\frac{1}{2}(\gamma-1)M_r^2 \sum_e u_e^{n+1} \frac{(\mathbf{m}\cdot\mathbf{m})_e^{n+1}}{\rho_e^{n+1}} \bar{l}_e, \tag{4.30}$$

with the summation over the three faces of the control volume. At each face e the quantity u_e^{n+1} is known (from $u_e = m_e/\rho_e$, where ρ_e follows from weighted averaging); the problem is to determine $(\mathbf{m}\cdot\mathbf{m})_e^{n+1}/\rho_e^{n+1}$. Considering face o in Figure 4.1, we will discuss the implementation of the first order upwind and central scheme.

First order upwind scheme
Assume that the flow on face o in Figure 4.1 is from cell q to i, i.e. $m_o^{n+1}\bar{l}_o < 0$. We then use: $\rho_o^{n+1} = \rho_q^{n+1}$. The term $(\mathbf{m}\cdot\mathbf{m})$ at face q, obtained using first order upwind, with a flow from q to i, will be denoted as $(\mathbf{m}\cdot\mathbf{m})_{o,q}^{n+1}$. Note that

$$(\mathbf{m}\cdot\mathbf{m})_{o,q}^{n+1} = (m_{o,q}^{n+1})^2 + (\tilde{m}_{o,q}^{n+1})^2, \tag{4.31}$$

where both $\tilde{m}_{o,q}^{n+1}$ and $m_{o,q}^{n+1}$ (note that we do not use the known value m_o^{n+1} !) are obtained using the reconstruction algorithm (also discussed, in more detail, in Section 5.4). The formulas from which $\tilde{m}_{o,q}^{n+1}$ and $m_{o,q}^{n+1}$ follow, are repeated here shortly. Determine coefficients α and β from:

$$\begin{aligned} \mathbf{N}_o &= \alpha_v \mathbf{N}_v + \alpha_w \mathbf{N}_w \\ \mathbf{t}_o &= \beta_v \mathbf{N}_v + \beta_w \mathbf{N}_w. \end{aligned}$$

Then

$$\begin{aligned} m_{o,q}^{n+1} &= \alpha_v m_v^{n+1} + \alpha_w m_w^{n+1} \\ \tilde{m}_{o,q}^{n+1} &= \beta_v m_v^{n+1} + \beta_w m_w^{n+1}. \end{aligned}$$

When face o is a boundary face, see Figure 4.2, then one-sided differences are applied for the density, normal momentum component and tangential momentum component.

Central scheme

Again considering face o in Figure 4.1, we determine ρ_o^{n+1} using averaging:

$$\rho_o^{n+1} = \frac{1}{2}(\rho_q^{n+1} + \rho_i^{n+1}). \tag{4.32}$$

With the computation of $(\mathbf{m}\cdot\mathbf{m})_{o,q}^{n+1}$ and $(\mathbf{m}\cdot\mathbf{m})_{o,i}^{n+1}$ similar to the computation of $(\mathbf{m}\cdot\mathbf{m})_{o,q}^{n+1}$, as discussed above in this section (the part on the first order upwind scheme), we compute:

$$(\mathbf{m}\cdot\mathbf{m})_o^{n+1} = \frac{1}{2}[(\mathbf{m}\cdot\mathbf{m})_{o,q}^{n+1} + (\mathbf{m}\cdot\mathbf{m})_{o,i}^{n+1}]. \tag{4.33}$$

On boundary faces, one-sided differences are taken. Hence, the treatment of boundary faces is identical for the first order upwind and the central scheme.

4.6 Continuity equation

4.6.1 Discretization of the continuity equation

The density is computed from the continuity equation, by considering this equation as a convection equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{m} = \frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \frac{\mathbf{m}}{\rho} \right) = 0. \tag{4.34}$$

With n and $(n+1)$ representing the time-levels, the discretization of this equation, with i denoting the control volume and e the three faces of it, is done as follows:

$$\Omega_i \frac{\rho_i^{n+1}}{\tau} + \sum_e \frac{1}{\rho_{e,c}^n} m_e^{n+1} \rho_{e,up}^{n+1} \bar{l}_e = \Omega_i \frac{\rho_i^n}{\tau} \tag{4.35}$$

In this equation $\rho_{e,c}^n$ represents the density at face e, obtained using weighted interpolation (denoted by the $_c$), at the old time-level; see Section 4.4.2. Quantity $\rho_{e,up}^{n+1}$ is the density at face e, obtained using density biased upwind; see Section 4.6.2. Equation (4.35) is considered as a special form of the general convection-diffusion equation (4.1).
Repeat equation (4.2), where we have to recall that ρ^{n+1} is the unknown ϕ^{n+1}:

$$\begin{aligned} \Omega_i \frac{a_i^{n+1} \phi_i^{n+1}}{\tau} + \sum_e b_e^n u_e^{n+1} \phi_e^{n+1} \bar{l}_e - \sum_e c_e^{n+1} (\nabla \phi^{n+1} \cdot \mathbf{N})_e \bar{l}_e + \\ + \Omega_i d_i^{n+1} \phi_i^n = \Omega_i \frac{a_i^n \rho_i^n \phi_i^n}{\tau} + \Omega_i g_i^{n,n+1} \end{aligned} \tag{4.36}$$

Note that in the time-derivative at the left-hand side we write ρ^n instead of ρ^{n+1}, and furthermore, in the convection term we write b_e^n instead of b_e^{n+1}. This because ρ^{n+1} is still unknown. The following coefficients result from comparing equation (4.35) with equation (4.36):

$$a_i^{n+1} = 1 \qquad a_i^n = 1 \qquad b_e^{n+1} = 1 \qquad c_e^{n+1} = d_i^{n+1} = g_i^{n+1} = 0. \tag{4.37}$$

In order to discretize correctly at the boundary (no Dirichlet boundary conditions for the density are given explicitly), we do the following. If the momentum vector $\mathbf{m}$ is given at the boundary (e.g. inflow boundary), then we assume the following Dirichlet boundary condition for the density: $\rho = m/u$, where m the given normal momentum component, and u the latest obtained given normal velocity. In all other situations, a homogeneous Neumann condition for the density is prescribed.
For explicit time-integration, in equation (4.35) the terms m_e^{n+1} and ρ_e^{n+1} are replaced by m_e^n and ρ_e^n. This consequently leads to trivial changes in equation (4.36).

4.6.2 Density biased upwind: first order & central scheme

See Section 4.4. Note that the first order scheme is also called unconditional upwind scheme.

Chapter 5

Spatial discretisation of the momentum equation

5.1 Introduction

The dimensionless momentum equation including viscous terms is given by (2.29). For the moment, we restrict ourselves to the Euler approximation, i.e. equation (2.42). In this chapter we discuss the discretisation, implementation of boundary conditions and the numerical treatment of the convection term. Furthermore, the computation of gradients located at cell-faces is discussed. The momentum equation for incompressible flows is exactly the same as for compressible flows. The only differences are: (1) the primary variable is the velocity for the incompressible case. The relation between velocity and momentum is simple, since $\mathbf{m} = \mathbf{u}$; (2) The factor $1/\gamma M_r^2$ for the pressure gradient is not present in the incompressible case.

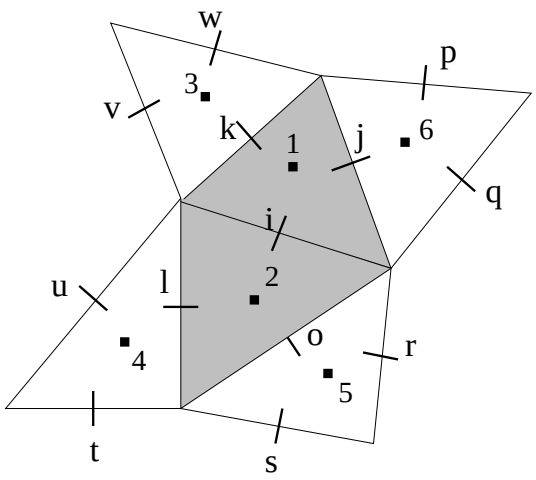

Figure 5.1: The control volume for the momentum component normal to face i is shaded. The number denote the cells, while the faces are indicated by letters.

5.2 Momentum equation: discretisation

The Euler momentum equation (2.42) is projected on a normal vector N^α:

$$\frac{\partial m^\alpha N^\alpha}{\partial t} + \left(u^\beta m^\alpha N^\alpha\right)_{,\beta} = -\frac{1}{\gamma M_r^2} p_{,\alpha} N^\alpha. \tag{5.1}$$

Defining $m = m^\alpha N^\alpha = \mathbf{m} \cdot \mathbf{N}$, being the component of the momentum in the direction of the normal vector, shortly called normal momentum component, this equation is rewritten as

$$\frac{\partial m}{\partial t} + \nabla \cdot \mathbf{u} m = -\frac{1}{\gamma M_r^2} \nabla p \cdot \mathbf{N}. \tag{5.2}$$

As mentioned in Section 3.4, at every face midpoint the normal momentum component is positioned.

5.2.1 Discretisation: control volume consisting of whole triangles

The control volume for the momentum at face i, see Figure 5.1, is chosen to consist of the two triangles adjacent to this face. For the situation in which i is a boundary face, we refer to Section 5.3. Integration of (5.2) over the control volume and numerical quadrature yields an equation for $m_i = \mathbf{m}_i \cdot \mathbf{N}_i$, with $\mathbf{N}_i$ the uniquely defined normal on face i:

$$\frac{\Omega_1 + \Omega_2}{\tau} m_i^{n+1} + \sum_e (\mathbf{m}_e^{n+1} \cdot \mathbf{N}_i) u_e^n \bar{l}_e = \frac{\Omega_1 + \Omega_2}{\tau} m_i^n - \frac{1}{\gamma M_r^2} (\Omega_1 + \Omega_2)(\nabla p^n \cdot \mathbf{N})_i. \qquad (5.3)$$

The following aspects must be noted:

- An implicit Euler time-integration scheme is used. See Chapter 9 for more information. Parameter τ represents the time-step.
- The summation over e represents the summation over the faces of the CV, i.e. $e = \{k, l, o, j\}$.
- Picard linearization of the convection term is applied:

$$\nabla \cdot (\mathbf{u}^{n+1} m^{n+1}) \approx \nabla \cdot (\mathbf{u}^n m^{n+1}). \qquad (5.4)$$

- With $\mathbf{N}_e$ the uniquely defined normal at face e, and $\mathbf{n}_e$ the, with respect to control volume Ω_i, outwards pointing normal, the convection term is treated as follows:

$$\begin{aligned} \int_{\Omega_i} \nabla \cdot (\mathbf{u} m)\, d\Omega &= \int_{\Omega_i} \nabla \cdot (\mathbf{u}(\mathbf{m} \cdot \mathbf{N}_i))\, d\Omega = \oint_{\partial\Omega_i} (\mathbf{u} \cdot \mathbf{n})(\mathbf{m} \cdot \mathbf{N}_i)\, d\Gamma \approx \\ &\approx \sum_e (\mathbf{u}_e \cdot \mathbf{n}_e)(\mathbf{m}_e \cdot \mathbf{N}_i) l_e = \sum_e (\mathbf{u}_e \cdot \mathbf{N}_e)(\mathbf{m}_e \cdot \mathbf{N}_i) \bar{l}_e, \qquad (5.5) \end{aligned}$$

  where we used (4.5) and

$$\bar{l}_e = l_e (\mathbf{n}_e \cdot \mathbf{N}_e). \qquad (5.6)$$

  The computation of $(\mathbf{m}_e \cdot \mathbf{N}_i)$ is treated in Section 5.4. The term $(\mathbf{u}_e \cdot \mathbf{N}_e)$ can be obtained using a 'conservative' approach or using a 'nonconservative' approach, see Section 5.6.

- Note that (5.3), in the case of implicit time-integration, forms a linear system of the form: $A\mathbf{x} = \mathbf{b}$, where A represents the $E \times E$-matrix following from the left-hand side, vector $\mathbf{x} = (m_1^{n+1}, \ldots, m_E^{n+1})$, with E the number of faces, and $\mathbf{b}$ stands for the right-hand side of (5.3). For more information concerning the linear solver, see Chapter 10.
- See Section 13.1, part *Face-based data structure*, concerning the definition of the unique normal $\mathbf{N}_i$.

5.2.2 Discretisation: control volume consisting of half triangles

Considering Figure 5.1, the control volume for face i consists then of half of triangle 1 and half of triangle 2. With $\mathbf{N}_i$ pointing from cell 1 to cell 2, the discretized momentum equation for face i becomes:

$$\Omega_i \frac{m_i^{n+1} - m_i^n}{\tau} + l_i(F_2 - F_1) = -\Omega_i \frac{1}{\gamma M_r^2}(\nabla p^n \cdot \mathbf{N})_i, \tag{5.7}$$

where $\Omega_i = \frac{1}{2}(\Omega_1 + \Omega_2)$ and

$$F_1 = (\mathbf{u}_1 \cdot \mathbf{N}_i)(\mathbf{m}_1 \cdot \mathbf{N}_i), \tag{5.8}$$

where $\mathbf{u}_1$ and $\mathbf{m}_1$ stand for the velocity vector respectively the momentum vector in cell 1. These have to be interpolated using a central or some sort of upwind scheme. For explicit time-integration, $\mathbf{u}$ and $\mathbf{m}$ in equation (5.8) are taken at time-level n. For implicit time-integration, a Picard linearization is used, which means that $\mathbf{u}$ is taken at time-level n and $\mathbf{m}$ at time-level $n+1$. The computation of fluxes F is the subject of Section 5.5.

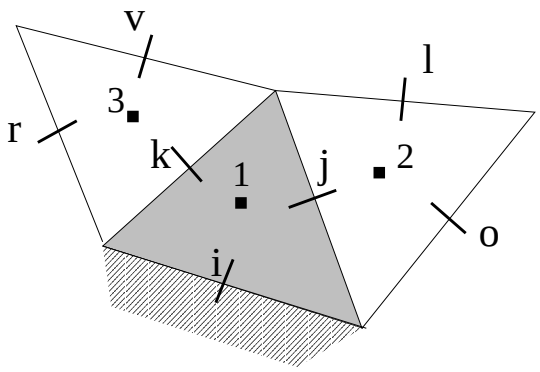

Figure 5.2: The control volume for the momentum component normal to boundary face i is shaded. The number denote the cells, while the faces are indicated by letters.

5.3 Momentum equation: boundary faces

5.3.1 Momentum given at boundary

At boundary faces several sets of boundary conditions for the momentum equation can be given, see Section 2.6. They can be divided into two groups: either the normal momentum component is prescribed, or this quantity is not prescribed.
Incorporating the first possibility is trivial: the ith row, with i indicating a boundary face, of the matrix is given by $A_{ij} = \delta^{ij}$, with δ^{ij} the Kronecker delta, and b_i, the ith element of the right-hand side vector $\mathbf{b}$, is put equal to m_i^{n+1}.

5.3.2 Boundary conditions: control volume consisting of whole triangles

When m_i is not given, we need to write down an equation similar to (5.3).

Subsonic outflow
Note that, within the Euler approximation, in this situation always $\sigma^{nn} = -p$, i.e. the pressure, is given at the boundary. For the control volume for boundary face i, we choose the corresponding boundary cell, see Figure 5.2. Integrating (5.2) over the control volume Ω_1 yields an equation for m_i^{n+1}:

$$\frac{\Omega_1}{\tau} m_i^{n+1} + \sum_e (\mathbf{m}_e^{n+1} \cdot \mathbf{N}_i)(\mathbf{u}_e \cdot \mathbf{N}_e)^n \bar{l}_e = \frac{\Omega_1}{\tau} m_i^n - \frac{1}{\gamma M_r^2} \Omega_1 (\nabla p \cdot \mathbf{N})_i. \tag{5.9}$$

The summation over e stands for the summation over the CV-faces: $e = \{i, j, k\}$. For the discretisation of the pressure gradient, see Section 5.7.7. The treatment of the convection term at the outflow boundary is discussed in Section 5.4.7.

5.3.3 Boundary conditions: control volume consisting of half triangles

Boundary faces
Let i be a boundary face. When m_i is not given (at the outflow boundary, for example, where

the pressure is given), the discretized equation reads (cf. equation (5.7) and Figure 5.2):

$$\Omega_i \frac{m_i^{n+1} - m_i^n}{\tau} + l_i(u_i m_i - F_1) = -\Omega_i \frac{1}{\gamma M_r^2}(\nabla p \cdot \mathbf{N})_i, \tag{5.10}$$

where $\Omega_i = \frac{1}{2}\Omega_1$ and F_1 is defined in (5.8). For the computation of the pressure gradient, see Section 5.7.7. For the computation of F_1, see Section 5.5.3.

Quasi internal faces

Note that, when the full momentum vector is given at a boundary face, we have only used information with respect to the normal component at the boundary face. The given tangential momentum component must also be used in some manner. Let at face i in Figure 5.2 the momentum-vector $\mathbf{m}_i$ be given. At quasi internal face j we then prescribe the normal momentum component by inserting $m_j = \mathbf{m}_i \cdot \mathbf{N}_j$. When face j is a quasi internal face for more than one boundary face at which the momentum vector is given, we do the following. Let at faces i and o the momentum vector be given. Then we prescribe the normal momentum component at face j by inserting $m_j = \frac{1}{2}(\mathbf{m}_i \cdot \mathbf{N}_j + \mathbf{m}_o \cdot \mathbf{N}_j)$. Completely similar, when there are three nearby boundary faces at which the momentum vector is given.

5.4 Convection term: integration over whole triangles

For a discussion of the treatment of the convection term when dealing with a control volume consisting of half triangles, we refer to Section 5.5.

5.4.1 Discussion

The discretized convection term is given by

$$C = \sum_e C_e = \sum_e (\mathbf{m}_e^{n+1} \cdot \mathbf{N}_i)(\mathbf{u}_e^n \cdot \mathbf{N}_e)\bar{l}_e. \tag{5.11}$$

A relevant aspect is the computation of $(\mathbf{u}_e \cdot \mathbf{N}_e)$ in equation (5.11), to be discussed in Section 5.6.
There are two ways to compute the term $(\mathbf{m}_e \cdot \mathbf{N}_i)$:

1. The momentum vector at face e can be decomposed in a component normal to face e and a component parallel to this face:

   $$\mathbf{m}_e = m_e \mathbf{N}_e + \tilde{m}_e \mathbf{t}_e, \tag{5.12}$$

   leading to

   $$\mathbf{m}_e \cdot \mathbf{N}_i = m_e(\mathbf{N}_e \cdot \mathbf{N}_i) + \tilde{m}_e(\mathbf{t}_e \cdot \mathbf{N}_i). \tag{5.13}$$

   The difficulty lies in determining the tangential momentum component $\tilde{m}_e$, since only the normal momentum components are stored at the faces.

2. The term $(\mathbf{m}_e \cdot \mathbf{N}_i)$ itself is upwinded.

Both schemes will be discussed below.

5.4.2 Reconstruction procedure

We have, see Figure 5.1,

$$\mathbf{t}_j = \xi_q \mathbf{N}_q + \xi_p \mathbf{N}_p, \tag{5.14}$$

where ξ_q and ξ_p can be solved from:

$$\begin{bmatrix} N_{q,x} & N_{p,x} \\ N_{q,y} & N_{p,y} \end{bmatrix} \begin{bmatrix} \xi_q \\ \xi_p \end{bmatrix} = \begin{bmatrix} t_{j,x} \\ t_{j,y} \end{bmatrix}, \tag{5.15}$$

With $\mathbf{N} = (N_x, N_y)$, we define $\mathbf{t} = (t_y, -t_x)$. The inverse of a 2×2 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{5.16}$$

is given by:

$$A^{-1} = \frac{1}{\det A} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}. \tag{5.17}$$

Another way of solving for ξ is by using the orthogonality of the tangential and normal vectors:

$$\mathbf{t}_q \cdot \mathbf{N}_q = \mathbf{t}_p \cdot \mathbf{N}_p = 0. \tag{5.18}$$

Taking the inner product with $\mathbf{t}_q$ and $\mathbf{t}_p$ leads to

$$\begin{aligned} (\mathbf{t}_j \cdot \mathbf{t}_q) &= \xi_p (\mathbf{N}_p \cdot \mathbf{t}_q) \\ (\mathbf{t}_j \cdot \mathbf{t}_p) &= \xi_q (\mathbf{N}_q \cdot \mathbf{t}_p) \end{aligned}$$

from which the coefficients ξ can be found easily.
If $\mathbf{m}$ is constant, we have exactly

$$\tilde{m}_j = \mathbf{m}_j \cdot \mathbf{t}_j = \mathbf{m}_j \cdot (\xi_q \mathbf{N}_q + \xi_p \mathbf{N}_p) = \xi_q m_q + \xi_p m_p. \tag{5.19}$$

This equation will be used as approximation when $\mathbf{m}$ is variable. Of course, cell 1 can be used just as well, leading to

$$\tilde{m}_j = \xi_i m_i + \xi_k m_k. \tag{5.20}$$

An alternative way to reconstruct the momentum has also been implemented. We start from the divergence theorem:

$$\int_\Omega f^\alpha_{,\alpha} d\Omega = \oint_{\partial\Omega} f^\alpha n_\alpha d\Gamma \tag{5.21}$$

Now choose $f = (\mathbf{a} \cdot \mathbf{r})$ with $\mathbf{a}$ and $\mathbf{b}$ constant and $\mathbf{r} = \mathbf{x} - \mathbf{x}_0$ the position vector, where $\mathbf{x}_0$ is suitably chosen point. Then

$$f^\beta_{,\beta} = (a_\alpha r_\alpha b_\beta) = a_\alpha b_\beta \delta^{\alpha\beta} = a_\alpha b_\alpha, \tag{5.22}$$

and hence

$$\int_\Omega \text{div } (\mathbf{a} \cdot \mathbf{r})\, \mathbf{b} d\Omega = (\mathbf{a} \cdot \mathbf{b}) \left|\Omega\right|. \tag{5.23}$$

On the other hand,

$$\begin{aligned} &\oint_{\partial\Omega} \mathbf{f} \cdot \mathbf{n} d\Gamma \approx \sum_e \oint_{\Gamma_e} (\mathbf{a} \cdot \mathbf{r})\, (\mathbf{b} \cdot \mathbf{n})\, d\Gamma = \\ &\sum_e (\mathbf{b} \cdot \mathbf{n})\, \mathbf{a} \cdot \oint_{\Gamma_e} \mathbf{r} d\Gamma = \sum_e (\mathbf{b} \cdot \mathbf{n})\, \mathbf{a} \cdot (\mathbf{x}_e - \mathbf{x}_0)\, l_e, \end{aligned} \tag{5.24}$$

with l_e the length of the edge e. So,

$$(\mathbf{a} \cdot \mathbf{b}) \left|\Omega\right| = \mathbf{a} \cdot \sum_e (\mathbf{x}_e - \mathbf{x}_0)\, (\mathbf{b} \cdot \mathbf{n})\, l_e. \tag{5.25}$$

Since this equation holds for any arbitrary constant $\mathbf{a}$, we get

$$\mathbf{b} = \frac{1}{\left|\Omega\right|} \sum_e (\mathbf{x}_e - \mathbf{x}_0)\, (\mathbf{b} \cdot \mathbf{n})\, l_e. \tag{5.26}$$

Substitution $\mathbf{b} = \mathbf{m}$ gives

$$\mathbf{m} = \frac{1}{|\Omega|} \sum_e (\mathbf{x}_e - \mathbf{x}_0)\,(\mathbf{m} \cdot \mathbf{n})\, l_e. \tag{5.27}$$

Multiplying by $\mathbf{t}$ and chosing $\mathbf{x}_0 = mathbfx_j$ leads to the reconstruction formula

$$\tilde{m}_j = \frac{1}{|\Omega_1| + |\Omega_2|} \sum_{e=i,k,p,q} \mathbf{t} \cdot (\mathbf{x}_e - \mathbf{x}_j)\, m_e l_e. \tag{5.28}$$

in the case of the internal or quasi internal face. In general we can write

$$\tilde{m}_j = \chi_q m_q + \chi_p m_p + \chi_k m_k + \chi_i m_i, \tag{5.29}$$

where the coefficients χ depend on geometry and type of interpolation applied.

Another, related form of reconstruction, is the following. We can write

$$\mathbf{N}_j = \eta_q \mathbf{N}_q + \eta_p \mathbf{N}_p, \tag{5.30}$$

where η_q and η_p can be solved from:

$$\begin{bmatrix} N_{q,x} & N_{p,x} \\ N_{q,y} & N_{p,y} \end{bmatrix} \begin{bmatrix} \eta_q \\ \eta_p \end{bmatrix} = \begin{bmatrix} N_{j,x} \\ N_{j,y} \end{bmatrix}. \tag{5.31}$$

Of course, also in this case we can make use of the orthogonality between the tangential and normal vectors, resulting in

$$\begin{aligned} (\mathbf{N}_j \cdot \mathbf{t}_q) &= \eta_p (\mathbf{N}_p \cdot \mathbf{t}_q) \\ (\mathbf{N}_j \cdot \mathbf{t}_p) &= \eta_q (\mathbf{N}_q \cdot \mathbf{t}_p) \end{aligned}$$

5.4.3 Momentum equation: old first order upwind scheme

Has keyword `OLD_FIRST_O`, and belongs to method 1 (see Section 5.4.1).
Reference is made to Figure 5.1. For the first order upwind method, with a flow from cell 6 to 1 ($u_j \bar{l} < 0$), we take in (5.13):

$$\tilde{m}_j = \xi_q m_q + \xi_p m_p, \tag{5.32}$$

and with a flow from cell 1 to 6 ($u_j \bar{l} > 0$) we have in (5.13):

$$\tilde{m}_j = \xi_k m_k + \xi_i m_i. \tag{5.33}$$

5.4.4 Momentum equation: central scheme

Two possible schemes are implemented.

- Keyword `NONE`.
  This method is based on an averaging of the two upwind interpolations with keyword `FIRST_ORDER` as given in Section 5.4.5, i.e.

$$(\mathbf{m}_j \cdot \mathbf{N}_i) \approx \frac{1}{2}(\eta_p m_p + \eta_q m_q + m_i) \tag{5.34}$$

  For the spectral properties of this scheme, see Section 5.4.8.

- Keyword `OLD_NONE`.
  Belongs to method 1 (see Section 5.4.1).
  Reference is made to Figure 5.1 and equation (5.29). Central interpolation leads to:

$$\chi_q = \frac{\Omega_1}{\Omega_1+\Omega_6}\xi_q \qquad \chi_p = \frac{\Omega_1}{\Omega_1+\Omega_6}\xi_p \qquad \chi_k = \frac{\Omega_6}{\Omega_1+\Omega_6}\xi_k \qquad \chi_i = \frac{\Omega_6}{\Omega_1+\Omega_6}\xi_i \tag{5.35}$$

  or, see the discussion in [6],

$$\chi_q = \frac{1}{2}\xi_q \qquad \chi_p = \frac{1}{2}\xi_p \qquad \chi_k = \frac{1}{2}\xi_k \qquad \chi_i = \frac{1}{2}\xi_i \tag{5.36}$$

  At this moment we have implemented (5.36) for keyword `OLD_NONE`.

5.4.5 Momentum equation: first order upwind scheme

- Keyword `NFIRST_ORDER`.
  Belongs to method 1 (see Section 5.4.1). Reference is made to Figure 5.1. When the flow is from cell 1 to 6, we compute the desired convection term without using the expansion on normal and tangential vectors. Hence, instead of (5.33), we do:

$$(\mathbf{m}_j \cdot \mathbf{N}_i) \approx (\mathbf{m}_i \cdot \mathbf{N}_i) = m_i. \tag{5.37}$$

  When the flow is from cell 6 to 1, we use (5.32).

- Keyword `FIRST_ORDER`.
  Belongs to method 2 (see Section 5.4.1). We can solve, see Section 5.4.2, for coefficients η:

$$\mathbf{N}_i = \eta_p \mathbf{N}_p + \eta_q \mathbf{N}_q \qquad\qquad \mathbf{N}_i = \eta_i \mathbf{N}_i + \eta_k \mathbf{N}_k. \tag{5.38}$$

  Note that, clearly, we have $\eta_i = 1$ and $\eta_k = 0$. Let the flow in Figure 5.1 be from cell 6 to 1, then we write:

$$\mathbf{m}_j \cdot \mathbf{N}_i = \mathbf{m}_j \cdot (\eta_p \mathbf{N}_p + \eta_q \mathbf{N}_q) \approx \eta_p m_p + \eta_q m_q. \tag{5.39}$$

  Let the flow in Figure 5.1 be from cell 1 to 6, then we write:

$$\mathbf{m}_j \cdot \mathbf{N}_i = \mathbf{m}_j \cdot (\eta_i \mathbf{N}_i + \eta_k \mathbf{N}_k) \approx m_i. \tag{5.40}$$

  Note that this is the same relation as that we use when the flow is from cell 1 to 6 for keyword `NFIRST_ORDER`.
  Numerical experiments show that this method, with keyword `FIRST_ORDER`, is consistent.

5.4.6 Momentum equation: quasi internal faces

The four CV-faces e, $e = \{k, l, o, j\}$, Figure 5.1, are divided into two groups: internal (real and quasi) faces q and boundary faces b: $e = q \cup b$.
We divide the boundary faces b into several groups, depending on the boundary conditions given: faces $b4$ on which the full momentum vector is given, faces $b5$ on which only the normal momentum component is given, faces $b6$ on which the tangential momentum component and the pressure is given, and faces $b7$ on which only the pressure is given. We have $b = b4 \cup b5 \cup b6 \cup b7$, and $e = q \cup b4 \cup b5 \cup b6 \cup b7$.
The two adjacent faces of boundary face b are indicated with i and x. The contribution to the convection-term resulting from face b equals:

$$(\mathbf{m}_b \cdot \mathbf{N}_i) u_b \bar{l}_b = u_b \bar{l}_b [m_b (\mathbf{N}_b \cdot \mathbf{N}_i) + \tilde{m}_b (\mathbf{t}_b \cdot \mathbf{N}_i)] \tag{5.41}$$

The boundary conditions at face b are implemented using one-sided differences. Note that there is always a contribution of size $u_b \bar{l}_b (\mathbf{N}_b \cdot \mathbf{N}_i)$ to matrix-element $A_{i,b}$.
The following approach is used, for method 1 as discussed in Section 5.4.1:

- When $\tilde{m}_b$ is prescribed; $b \in \{b4, b6\}$:
  Subtract $u_b \bar{l}_b \tilde{m}_b (\mathbf{t}_b \cdot \mathbf{N}_i)$ from the right-hand side.

- When $\tilde{m}_b$ is not prescribed; $b \in \{b5, b7\}$:
  With

  $$\tilde{m}_b = \xi_i m_i + \xi_x m_x, \tag{5.42}$$

  where coefficients ξ follow from a reconstruction procedure, put numbers $u_b \bar{l}_b (\mathbf{t}_b \cdot \mathbf{N}_i) \xi_i$ and $u_b \bar{l}_b (\mathbf{t}_b \cdot \mathbf{N}_i) \xi_j$ in matrix-elements A_{ii} and A_{ij} of the matrix.

For method 2, see Section 5.4.1, we use for boundary conditions $b4$, $b5$ and $b6$ the methods as given a few lines above. The reason for switching to method 1 in this case is clearly the fact that the boundary conditions are also given in terms of tangential and/or normal momentum components. For boundary condition $b7$ (this must be at an outflow boundary) we make use of $(\mathbf{m}_b \cdot \mathbf{N}_i) \approx m_i$, which is in accordance with method 2 and which enlarges the diagonal.

5.4.7 Momentum equation: boundary faces

At boundary face i, see Figure 5.2, we need to evaluate the term $C_i = (\mathbf{m}_i \cdot \mathbf{N}_i) u_i \bar{l}_i$ in the convection term. We evaluate this term by $C_i = m_i u_i \bar{l}_i$, independent whether method 1 or 2 (section 5.4.1) is used. Note that, with $\mathbf{N}_i = \eta_j \mathbf{N}_j + \eta_k \mathbf{N}_k$, it would be more consistent with method 2 to write $C_i = (\eta_j m_j + \eta_k m_k) u_i \bar{l}_i$, but we prefer to use $C_i = m_i u_i \bar{l}_i$ since this enlarges the diagonal.

5.4.8 Spectral properties of central scheme

Here the properties of the central scheme, equation (5.34), in the light of the arguments formulated by Veldman, [6], are gathered.
The discretized momentum equation can be written as:

$$D \frac{d\mathbf{m}}{dt} + C(\mathbf{u})\mathbf{m} = \mathbf{r}, \tag{5.43}$$

where D a positive definite diagonal matrix, C the convection matrix, $\mathbf{m}$ the solution vector and $\mathbf{r}$ the other terms. Veldman [6] states that the convection operator should be skew-symmetric:

$$C_{ij} = -C_{ji} \qquad \forall\, i,j \qquad\qquad C_{ii} = 0. \tag{5.44}$$

We will show whether and when our central scheme satisfies this property.
The following relations (reconstruction and closed contour integral) holds:

$$\begin{aligned} \mathbf{N}_i &= \eta_p^i \mathbf{N}_p + \eta_q^i \mathbf{N}_q && (5.45)\\ \mathbf{N}_p &= \eta_i^p \mathbf{N}_i + \eta_k^p \mathbf{N}_k && (5.46)\\ \bar{l}_q \mathbf{N}_q + \bar{l}_p \mathbf{N}_p + \bar{l}_k \mathbf{N}_k + \bar{l}_i \mathbf{N}_i &= \mathbf{0}. && (5.47) \end{aligned}$$

Writing the convection term out in the equation for m_i, leads to, among others, the following matrix elements:

$$C_{iq} = \frac{1}{2}\eta_q^i u_j \bar{l}_j \qquad C_{ii} = \frac{1}{2}(u_j \bar{l}_j + u_k \bar{l}_k + u_l \bar{l}_l + u_o \bar{l}_o). \tag{5.48}$$

Notice that $C_{ii} = 0$ when the flow is incompressible. Writing out the equation for m_q leads to the following matrix elements:

$$C_{qi} = \frac{1}{2}\eta_i^q u_j \bar{l}_j, \tag{5.49}$$

where $\bar{l}_j$ has a sign opposite to the sign in C_{iq}. Hence, $C_{iq} = -C_{qi}$ if and only if

$$\eta_q^i = \eta_i^q. \tag{5.50}$$

We will derive a relation between η_q^i and η_i^q, using the reconstruction and closed contour integral relations given above. First, we find that

$$\mathbf{N}_q = \frac{1}{\bar{l}_q}(-\bar{l}_i \mathbf{N}_i - \bar{l}_k \mathbf{N}_k - \bar{l}_p \mathbf{N}_p) \tag{5.51}$$

Inserting this leads to:

$$\left(1 + \eta_q^i \frac{\bar{l}_i}{\bar{l}_q}\right) \mathbf{N}_i = \left(\eta_p^i - \eta_q^i \frac{\bar{l}_p}{\bar{l}_q}\right) \mathbf{N}_p - \eta_q^i \frac{\bar{l}_k}{\bar{l}_q} \mathbf{N}_k. \tag{5.52}$$

Notice that, since $\mathbf{N}_i$ and $\mathbf{N}_k$ can never be parallel, the term in front of $\mathbf{N}_p$ can never be zero. Hence, we get:

$$\eta_i^p = \frac{1 + \eta_q^i \frac{\bar{l}_i}{\bar{l}_q}}{\eta_p^i - \eta_q^i \frac{\bar{l}_p}{\bar{l}_q}} \qquad\qquad \eta_k^p = \frac{\eta_q^i \frac{\bar{l}_k}{\bar{l}_q}}{\eta_p^i - \eta_q^i \frac{\bar{l}_p}{\bar{l}_q}}. \tag{5.53}$$

One special case occurs when condition $\eta_q^i = \eta_i^q$ is satisfied, namely when $\mathbf{N}_i$ and $\mathbf{N}_p$ are parallel and $\mathbf{N}_k$ and $\mathbf{N}_q$ are parallel. This is for example the case in grids consisting of triangles with angles of 60^o and Courant grids. More generally speaking, this is the case for unstructured grids where there is an underlying structured grid that is not 'skewed'.

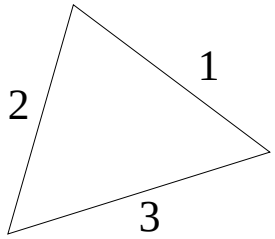

Figure 5.3: Triangle. Face 3 is the one for which the momentum equation is solved.

5.5 Convection term: integration over half triangles

The convection term in the discretized momentum equation consists of computing fluxes F_1 and F_2, equation (5.8). This problem is considered in more detail in this Section. For the sake of discussion, consider Figure 5.3. Face 3 corresponds to face i in Figure 5.1. One needs to find an approximation for $F = (\mathbf{u} \cdot \mathbf{N}_3)(\mathbf{m} \cdot \mathbf{N}_3)$, where $\mathbf{u}$ and $\mathbf{m}$ stand for approximations for the velocity and momentum vector in the triangle.

5.5.1 First order upwind: convection term with half triangles as cv

One can write, see Section 5.4.2:

$$\mathbf{N}_3 = \eta_1 \mathbf{N}_1 + \eta_2 \mathbf{N}_2, \tag{5.54}$$

where η_1 and η_2 can be computed easily. When the flow at face 3 is moving outwards with respect to triangle, a first order upwind interpolation for F leads to:

$$F = [\mathbf{u} \cdot \mathbf{N}_3][\mathbf{m} \cdot \mathbf{N}_3] = [\mathbf{u} \cdot (\eta_1 \mathbf{N}_1 + \eta_2 \mathbf{N}_2)][\mathbf{m} \cdot (\eta_1 \mathbf{N}_1 + \eta_2 \mathbf{N}_2)] \approx [\eta_1 u_1 + \eta_2 u_2][\eta_1 m_1 + \eta_2 m_2]. \tag{5.55}$$

When the flow is in the other direction, then a first order upwind interpolation for F leads to:

$$F = [\mathbf{u} \cdot \mathbf{N}_3][\mathbf{m} \cdot \mathbf{N}_3] \approx u_3 m_3. \tag{5.56}$$

It is still not clear what this means for conservativity of the numerical scheme. It seems plausible that, since also for faces 1 and 2 a momentum equation is written, the reconstructed velocity and momentum vector in the cell should be equal for all three faces. The first two items are, supposed that the mentioned notion is true, conservative, while the third one is not.

5.5.2 Central interpolation: convection term with half triangles as cv

The flux computed from a central interpolation can be seen as the average from the two possible first order upwind interpolations: $F_c = \frac{1}{2}(F_r + F_l)$, where subscripts c, r and l stand for 'central', 'right' and 'left'. With the situation as sketched in the previous section, this results in

$$F = \frac{1}{2}[(\eta_1 u_1 + \eta_2 u_2)(\eta_1 m_1 + \eta_2 m_2)] + \frac{1}{2} u_3 m_3. \tag{5.57}$$

For the situation in Section 5.2.2, Figure 5.1, one gets:

$$F_1 = \frac{1}{2}[(\eta_j u_j + \eta_k u_k)(\eta_j m_j + \eta_k m_k) + u_i m_i]; \qquad F_2 = \frac{1}{2}[(\eta_l u_l + \eta_o u_o)(\eta_l m_l + \eta_o m_o) + u_i m_i], \tag{5.58}$$

where coefficients η_j, η_k, η_l and η_o follow from:

$$\mathbf{N}_i = \eta_j \mathbf{N}_j + \eta_k \mathbf{N}_k; \qquad \mathbf{N}_i = \eta_l \mathbf{N}_l + \eta_o \mathbf{N}_o. \tag{5.59}$$

The convection term in (5.7) equals:

$$F_2 - F_1 = \frac{1}{2}[(\eta_l u_l + \eta_o u_o)(\eta_l m_l + \eta_o m_o) - (\eta_j u_j + \eta_k u_k)(\eta_j m_j + \eta_k m_k)]. \tag{5.60}$$

Note that the term corresponding to the face under consideration, cancels in the convection term, as is usual for central interpolations.

5.5.3 Boundary conditions: convection term with half triangles as cv

Computation of flux F_1 in equation (5.10) is done as follows (regardless whether a first order upwind or central scheme is applied):

$$F_1 = (\eta_j u_j + \eta_k u_k)(\eta_j m_j + \eta_k m_k). \tag{5.61}$$

5.6 Conservative and non-conservative: normal velocity

The term $(\mathbf{u}_e \cdot \mathbf{N}_e)$, in the convection term (5.5) can be evaluated in two ways: conservative and non-conservative. Before continuing, it must be noted that the terminology has nothing to do with conservation of quantities. The reason to use the words 'conservative' and 'non-conservative' nevertheless, is due to the fact that in ISNaS these words are used.

The non-conservative approach can be considered as a central interpolation approach. This approach is explained in Section 5.9.

The conservative approach is discussed below. The aim is to compute $u_j = \mathbf{u}_j \cdot \mathbf{N}_j$, see Figure 5.1. Assume the flow is from cell 6 to cell 1. Similar to the discussion in Section 5.4.2, we can write

$$\mathbf{N}_j = \eta_p \mathbf{N}_p + \eta_q \mathbf{N}_q, \tag{5.62}$$

where we can solve for coefficients η_p and η_q. The normal momentum $m_j = (\mathbf{m}_j \cdot \mathbf{N}_j)$, in this 'conservative' approach, then follows from $m_j = \eta_p m_p + \eta_q m_q$, and the normal velocity equals

$$u_j = \frac{m_j}{\rho_j} \approx \frac{\eta_p m_p + \eta_q m_q}{\rho_6}. \tag{5.63}$$

When the flow is from cell 6 to 1, a similar approach is followed.
If face i is a boundary face, then, for example, in Figure 5.2, we use

$$u_j = \frac{m_j}{\rho_j} \approx \frac{\eta_j m_j + \eta_k m_k}{\rho_1}. \tag{5.64}$$

5.7 Computation of pressure gradient

For the numerical evaluation of the term $(\nabla p \cdot \mathbf{N})_i$, with i an internal face, several methods can be utilized. In general, we will arrive at an expression of the form

$$(\nabla p \cdot \mathbf{N})_i = \sum_{j(i)} \gamma_{i,j(i)} p_{j(i)}, \tag{5.65}$$

where the summation is over the stencil $j(i)$ belonging to face i, and $\gamma_{i,j(i)}$ is a weight coefficient, depending solely on geometrical quantities. Note that consistency requires that

$$\sum_{j(i)} \gamma_{i,j(i)} = 0. \tag{5.66}$$

We compute all coefficients γ at all faces and store them. This leads to a gradient matrix with elements $G_{ij} = \gamma_{ij}$. At boundary faces i where the normal momentum component is given, the ith row of the gradient matrix consists of zeroes.
In Section 5.7.1 the path-integral method using a six-point stencil is discussed. In Sections 5.7.2 and 5.7.3 the path-integral method using a three-point respectively four-point stencil is discussed. In Section 5.7.4 a method using auxiliary points between four designated stencil-points is discussed. In Section 5.7.5 the sign criterion is discussed.

Second order derivatives
Also second order derivatives can be computed once we know the quantities $\gamma_{i,j(i)}$. The diffusion term in (4.2) reads:

$$\sum_e c_e^{n+1} (\nabla \phi^{n+1} \cdot \mathbf{N})_e \bar{l}_e. \tag{5.67}$$

Using (5.65), and replacing p by ϕ, the diffusion term is computed from:

$$\sum_e c_e^{n+1} \bar{l}_e \sum_{j(e)} \gamma_{e,j(e)} \phi_{j(e)} \cdot \tag{5.68}$$

5.7.1 Path-integral formulation for a six-point stencil

Keyword: `path_six`.
For the numerical evaluation of the term $(\nabla p \cdot \mathbf{N})_i$, with i an internal face, the so-called path-integral formulation can be employed. This formulation is based on the idea that first an approximation of the vector $(\nabla p)_i$ is made, after which taking the inner product with $\mathbf{N}_i$ is a trivial task. The heart of the path-integral formulation is formed by the integral identity

$$p_b - p_a = \int_a^b \nabla p \cdot d\mathbf{x}. \tag{5.69}$$

Discretisation of this expression leads to

$$p_b - p_a \approx (\nabla p)_{ab} \cdot (\mathbf{x}_b - \mathbf{x}_a), \tag{5.70}$$

where the gradient term is assumed to be evaluated somewhere between $\mathbf{x}_a$ and $\mathbf{x}_b$.
We will show how the path-integral methods leads to an evaluation of $(\nabla pi \cdot \mathbf{N})_i$. Application of (5.70) on the path from the center of cell 1 to the center of cell 2, see Figure 5.1, gives

$$p_2 - p_1 \approx (\nabla p)_i \cdot (\mathbf{x}_2 - \mathbf{x}_1), \tag{5.71}$$

where we note that, at least in regular grids, the midpoint of face i is located in the vicinity of the line between $\mathbf{x}_1$ and $\mathbf{x}_2$. Equation (5.71) alone is not sufficient to obtain $(\nabla p)_i$; to this aim we need one additional relation. Using (5.70) on the path from 5 to 3 leads to

$$p_3 - p_5 \approx (\nabla p)_i \cdot (\mathbf{x}_3 - \mathbf{x}_5),$$

and, similarly,

$$p_4 - p_6 \approx (\nabla p)_i \cdot (\mathbf{x}_4 - \mathbf{x}_6).$$

Combination of these two expressions leads to

$$p_3 - p_6 + p_4 - p_5 \approx \nabla p_i \cdot (\mathbf{x}_3 - \mathbf{x}_6 + \mathbf{x}_4 - \mathbf{x}_5). \tag{5.72}$$

Solution of the 2×2-system (5.71)-(5.72) results in a discretisation of $(\nabla p)_i$. Then the inner product with $\mathbf{N}_i$ is easily taken, leading to an equation of the form (5.65). In the vicinity of boundary, when one or more cells in the stencil of Figure 5.1, the path appearing in (5.72) is modified. For explicit formulas, see Chapter 21.2.
It must be noted that the path-integral method using a six-point stencil is, unlike the methods using a three-point or four-point stencil, not equivalent to the contour-integral formulation (see (5.75)). This is easily seen that, when vector $(\mathbf{x}_2 - \mathbf{x}_1)$ is parallel to $\mathbf{N}_i$, then the path-integral method using a six-point stencil results in:

$$\nabla p \cdot \mathbf{N}_i = \frac{p_2 - p_1}{|\mathbf{x}_2 - \mathbf{x}_1|}. \tag{5.73}$$

The contour-integral formulation leads to a different result.

5.7.2 Path-integral formulation for a three-point stencil

Keyword: `path_three`.
The same idea as in Section 5.7.1 is used, but now we restrict ourselves to a three-point stencil. This stencil consists of the points 1 and 2, Figure 5.1, and the third point k is the one amongst points 3, 4, 5 and 6 which lies closest to the midpoint of face i. The two equations that yield an approximation for $(\nabla p)_i$ are:

$$\begin{aligned} p_2 - p_1 &\approx (\nabla p)_i \cdot (\mathbf{x}_2 - \mathbf{x}_1) \\ p_k - p_1 &\approx (\nabla p)_i \cdot (\mathbf{x}_k - \mathbf{x}_1) \end{aligned} \tag{5.74}$$

Note that changing the second equation by taking a path from 2 to k leads to identical results. It must be noted that this method is exact for a pressure field of the form $p = a + bx + cy$, and leads to identical results as doing a contour-integral formulation along contour 1-2-k-1.

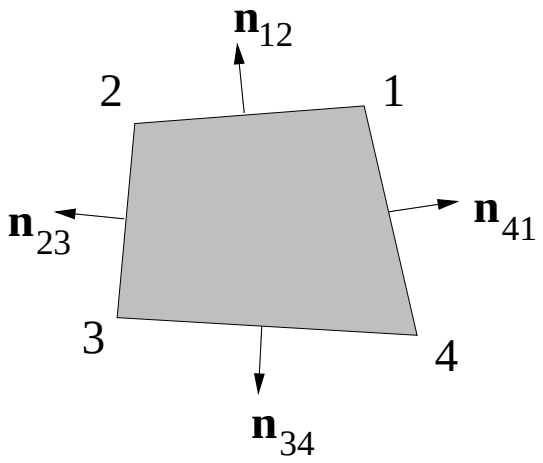

Figure 5.4: A four point stencil. The grey shaded region has area Ω.

5.7.3 Path-integral formulation for a four-point stencil

Keyword: `fourpoint`.
The path-integral method can also be applied using a four-point stencil. In Section 5.7.4 the method used to find these four points among the six points surrounding face i is given. We always make use of points 1 and 2, and we choose one point out of each set $\{3, 6\}$ and $\{4, 5\}$. The two equations that yield an approximation for $(\nabla p)_i$ are:

$$\begin{aligned} p_k - p_1 &\approx (\nabla p)_i \cdot (\mathbf{x}_k - \mathbf{x}_1) \\ p_j - p_2 &\approx (\nabla p)_i \cdot (\mathbf{x}_j - \mathbf{x}_2) \end{aligned}$$

where k is 4 or 5, and j is 3 or 6. It is interesting to note that the path-integral method for a four-point stencil is identical to computation of the pressure gradient using

$$\nabla p = \frac{1}{\Omega} \oint p\mathbf{n}\, d\Gamma, \tag{5.75}$$

where the closed contour is the one connecting points 1-j-k-2-1, and Ω is the area of this contour. This is proven below.
The stencil is sketched in Figure 5.4. The path-integral equations that yield an approximation for ∇p are of the form

$$A\nabla p = \mathbf{b}, \tag{5.76}$$

where

$$A = \begin{bmatrix} x_3 - x_1 & y_3 - y_1 \\ x_4 - x_2 & y_4 - y_2 \end{bmatrix} \qquad \nabla p = \begin{bmatrix} \partial p/\partial x \\ \partial p/\partial y \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} p_3 - p_1 \\ p_4 - p_2 \end{bmatrix}$$

Note that

$$\det A = [(\mathbf{x}_3 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_2)] = 2\Omega. \tag{5.77}$$

The pressure gradient follows from

$$\nabla p = \frac{1}{2\Omega} \begin{bmatrix} (y_4 - y_2)(p_3 - p_1) - (y_3 - y_1)(p_4 - p_2) \\ -(x_4 - x_2)(p_3 - p_1) + (x_3 - x_1)(p_4 - p_2) \end{bmatrix}$$

Consider now the contour-integral formulation. The vector $l_{ab}\mathbf{n}_{ab}$ is obtained after rotating vector $(\mathbf{x}_b - \mathbf{x}_a)$ with 90^o in the clockwise direction, hence $l_{ab}\mathbf{n}_{ab} = (y_b - y_a, x_a - x_b)$. Using the midpoint rule yields for the pressure gradient:

$$\begin{aligned}
\nabla p &= \frac{1}{\Omega}\oint p\mathbf{n}\, d\Gamma \approx \\
&\approx \frac{1}{\Omega}\left[\frac{1}{2}(p_1+p_2)l_{12}\mathbf{n}_{12} + \frac{1}{2}(p_2+p_3)l_{23}\mathbf{n}_{23} + \frac{1}{2}(p_3+p_4)l_{34}\mathbf{n}_{34} + \frac{1}{2}(p_4+p_1)l_{41}\mathbf{n}_{41}\right] = \\
&= \frac{1}{2\Omega}\begin{bmatrix} (y_4-y_2)(p_3-p_1) - (y_3-y_1)(p_4-p_2) \\ -(x_4-x_2)(p_3-p_1) + (x_3-x_1)(p_4-p_2) \end{bmatrix},
\end{aligned}$$

which is identical to the result obtained from the path-integral method.

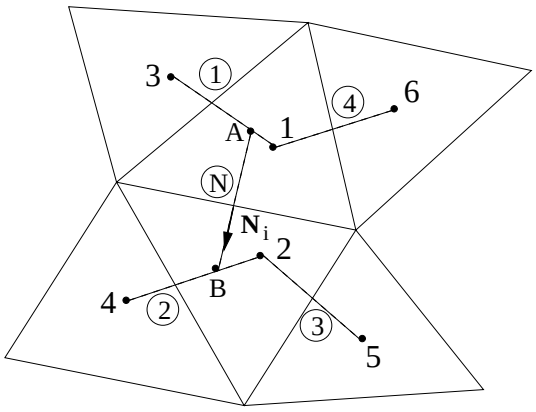

Figure 5.5: Auxiliary point method. N represents the line through the midpoint of face i and parallel to $\mathbf{N}_i$. Points A and B are the intersections between line N and the lines, indicated by the numbers in the circles, connecting the centers of the triangles.

5.7.4 Auxiliary point method

Keyword: `auxline`.
The auxiliary point method uses, as the method discussed in Section 5.7.3, a four point stencil. The auxiliary point method is different from the path-integral/contour-integral method.
Take a look at Figure 5.5. The line N through $\mathbf{x}_i$, the midpoint of face i, and parallel to $\mathbf{N}_i$ is given by

$$\mathbf{l}_N = \mathbf{x}_i + \lambda \mathbf{N}_i \qquad \lambda \in I\!R. \tag{5.78}$$

Lines 1, 2, 3 and 4 are given by

$$\mathbf{l}_{p,q} = \mathbf{x}_p + \eta(\mathbf{x}_q - \mathbf{x}_p), \tag{5.79}$$

where for line 1 we have $\{p,q\} = \{1,3\}$, for line 2 $\{p,q\} = \{2,4\}$, for line 3 $\{p,q\} = \{2,5\}$ and for line 4 $\{p,q\} = \{1,6\}$. The intersections between N, and lines 1-4, can be computed by solving $\mathbf{l}_N = \mathbf{l}_{p,q}$, which results in four values $\eta_1, \ldots, \eta_4$ and $\lambda_1, \ldots, \lambda_4$. With the situation as depicted in Figure 5.5, we will find that $0 < \eta_1 < 1$, $0 < \eta_2 < 1$, $\eta_3 < 0$ and $\eta_4 < 0$. Let A and B be the intersections between line N and 1 respectively 2, and define (linear interpolation)

$$p_A = p_1 + \eta_1(p_3 - p_1) \qquad p_B = p_2 + \eta_2(p_4 - p_2) \tag{5.80}$$

then we compute the projected pressure gradient using

$$(\nabla p \cdot \mathbf{N})_i = \frac{p_B - p_A}{|\mathbf{x}_B - \mathbf{x}_A|}. \tag{5.81}$$

Implementation of this, with taking into account the possibility that for the intersections the values for η can be smaller than 0, is done in a robust manner. With the definition of the following function

$$f(\eta) = \begin{cases} 0 & \eta < 0 \\ \eta & \eta \geq 0 \end{cases} \tag{5.82}$$

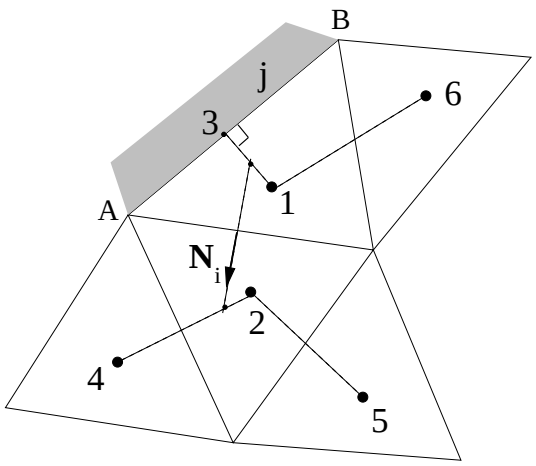

Figure 5.6: Treatment of boundaries in the auxiliary point method.

the following linear interpolations are readily obtained:

$$\begin{aligned} p_A &= p_1 + f(\eta_1)[p_3 - p_1] + f(\eta_4)[p_6 - p_1] \\ p_B &= p_2 + f(\eta_2)[p_4 - p_2] + f(\eta_3)[p_5 - p_2] \\ \mathbf{x}_A &= \mathbf{x}_1 + f(\eta_1)[\mathbf{x}_3 - \mathbf{x}_1] + f(\eta_4)[\mathbf{x}_6 - \mathbf{x}_1] \\ \mathbf{x}_B &= \mathbf{x}_2 + f(\eta_2)[\mathbf{x}_4 - \mathbf{x}_2] + f(\eta_3)[\mathbf{x}_5 - \mathbf{x}_2] \end{aligned}$$

The pressure gradient then follows from (5.81). Writing this equation out, cf. equation (5.65), yields

$$\begin{aligned} (\nabla p \cdot \mathbf{N})_i &= \left[-\frac{1 - f(\eta_1) - f(\eta_4)}{|\mathbf{x}_B - \mathbf{x}_A|}\right] p_1 + \left[\frac{1 - f(\eta_2) - f(\eta_3)}{|\mathbf{x}_B - \mathbf{x}_A|}\right] p_2 + \\ &+ \left[\frac{-f(\eta_1)}{|\mathbf{x}_B - \mathbf{x}_A|}\right] p_3 + \left[\frac{f(\eta_2)}{|\mathbf{x}_B - \mathbf{x}_A|}\right] p_4 + \left[\frac{f(\eta_3)}{|\mathbf{x}_B - \mathbf{x}_A|}\right] p_5 + \left[\frac{-f(\eta_4)}{|\mathbf{x}_B - \mathbf{x}_A|}\right] p_6 \end{aligned}$$

Consider the case in which cell 3 is not present, hence cell 1 is a boundary cell, and the boundary face is denoted with j, see Figure 5.6. For $\mathbf{x}_3$ we put the coordinates of the intersection between the boundary face j and the line which goes through $\mathbf{x}_1$ and which is parallel to the normal $\mathbf{N}_j$ at the boundary face. Let A and B be the vertices that define face j, then $\mathbf{x}_3$ is computed from solving: $\mathbf{x}_A + \eta(\mathbf{x}_B - \mathbf{x}_A) = \mathbf{x}_1 + \lambda \mathbf{N}_j$. We put in addition $p_3 = p_1$. Other situations in the vicinity of boundaries are treated similarly. Note that when the triangles are skewed, the line parallel to $\mathbf{N}_i$ may intersect the boundary face before it meets the line connecting the cell-centers.

It is not clear what has to be done in the case of skewed triangles, i.e. line $\mathbf{l}_N$ intersects the lines connecting the cell-centers for values of η larger than one. Especially in the vicinity of a boundary this can occur relatively easy.

5.7.5 Sign criterion

In the vicinity of steep gradients, the methods for computing the pressure gradient as given in the previous sections may fail since the sign of $(\nabla p \cdot \mathbf{N})_i$ is wrong, i.e. the flow is forced in the wrong direction. We will show this for the path-integral method with a six-point stencil.

The pressure gradient at face i, Figure 5.1, is computed using:

$$(\nabla p \cdot \mathbf{N})_i = \sum_{j=1}^{6} \gamma_j p_j, \tag{5.83}$$

where $\sum_j \gamma_j = 0$ (consistency), and $\gamma_1 = -\gamma_2$ and $\gamma_3 = \gamma_4 = -\gamma_5 = -\gamma_6$, see Chapter 21. Suppose there is a steep pressure gradient between cell 4 and the other cells, such that $p_1 = p_2 = p_3 = p_5 = p_6 = p_a$, $p_4 = p_b$ and $p_a < p_b$. In this situation, of course, $(\nabla p \cdot \mathbf{N})_i = 0$, but since we have included point 4 in the stencil, this will in the case that $\gamma_4 \neq 0$ not be obtained. Due to the direction of the pressure gradient, we allow for numerically computed pressure gradients satisfying $(\nabla p \cdot \mathbf{N})_i \geq 0$ (vector $\mathbf{N}_i$ points from cell 1 to cell 2). This leads to:

$$(\nabla p \cdot \mathbf{N})_i = \sum_{j=1}^{6} \gamma_j p_j = p_b \gamma_4 + p_a \sum_{j \neq 4} \gamma_j = \gamma_4 (p_b - p_a), \tag{5.84}$$

hence $\gamma_4 \geq 0$ is necessary to meet the criterion. A completely similar derivation leads to the criteria $\gamma_2 \geq 0$, $\gamma_5 \geq 0$, and $\gamma_1 \leq 0$, $\gamma_3 \leq 0$ and $\gamma_6 \leq 0$. Note that this cannot be reconciled with $\gamma_3 = \gamma_4 = -\gamma_5 = -\gamma_6$, except for the case that these coefficients are equal to zero.

More generally spoken, the weightcoefficient in the pressure gradient stencil of a point a, with coordinates $\mathbf{x}_a$, must be larger (or equal) to zero when $(\mathbf{x}_a - \mathbf{x}_i) \cdot \mathbf{N}_i > 0$, and vice versa. This will be called the *sign-criterion.*
For the three point stencil, Section 5.7.2 it can also be shown that it doesn't always meet the criterion. The auxiliary point method meets the mentioned requirement as long as the values for η are not larger than one. This can especially be a problem in the vicinity of boundaries.

In this section we will discuss a method using a stencil consisting of four points, which always (only may be in the vicinity of the boundaries) meets the sign-criterion. (*This method however sometimes lead to a stencil that was not consistent with the stencil as obtained using the procedure described in Section 16.3. Therefore we have restricted ourselves in the code to the stencil as discussed in in Section 5.7.3.* For the computation of the pressure gradient the path-integral formulation, Section 5.7.3, is used.
At face i a local $(\mathbf{N}_i, \mathbf{t}_i)$-coordinate system is defined, consisting of four quadrants, see Figure 5.7. With $\mathbf{x}$ the coordinates of a point, chosen relative to $\mathbf{x}_i$, then its quadrant is determined as follows:

- first quadrant when $\mathbf{x} \cdot \mathbf{t}_i > 0$ and $\mathbf{x} \cdot \mathbf{N}_i \leq 0$;
- second quadrant when $\mathbf{x} \cdot \mathbf{t}_i \leq 0$ and $\mathbf{x} \cdot \mathbf{N}_i < 0$;
- third quadrant when $\mathbf{x} \cdot \mathbf{t}_i < 0$ and $\mathbf{x} \cdot \mathbf{N}_i \geq 0$;

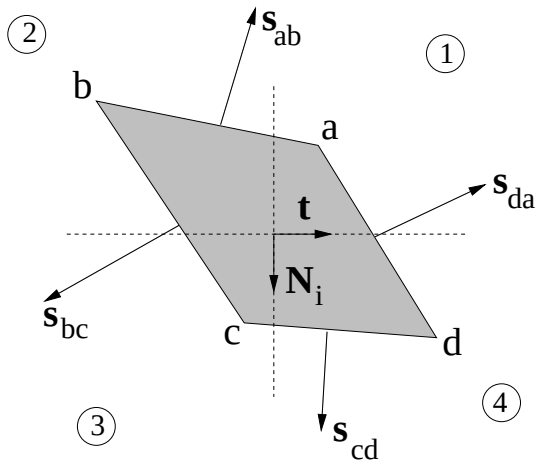

Figure 5.7: Four quadrant method. Points a, b, c and d are in the respectively first, second, third and fourth quadrant.

- fourth quadrant when $\mathbf{x} \cdot \mathbf{t}_i \geq 0$ and $\mathbf{x} \cdot \mathbf{N}_i > 0$.

When cell a lies on the boundary between quadrants 1 and 2, and cell c on the boundary between quadrants 3 and 4, then they are in quadrant 2 and 4 respectively, and the pressure gradient then follows simply from

$$(\nabla p \cdot \mathbf{N})_i = \frac{p_c - p_a}{|\mathbf{x}_c - \mathbf{x}_a|}. \tag{5.85}$$

Vector $\mathbf{s}_{jo}$, with $jo = \{ab, bc, cd, da\}$, is computed by rotating vector $(\mathbf{x}_o - \mathbf{x}_j)$ with 90^o in the clockwise direction. Hence, $\mathbf{s}_{jo} = (y_o - y_j, x_j - x_o)$. Furthermore, it is trivial to deduce that $\mathbf{s}_{jo} + \mathbf{s}_{oq} = \mathbf{s}_{jq}$. As discussed in Section 5.7.3, the path-integral formulation for a four-point stencil is equivalent to the contour-integral formulation. The pressure gradient in the direction of the normal is computed from, with Ω the area of the shaded region in Figure 5.7:

$$\begin{aligned}
\nabla p \cdot \mathbf{N}_i &= \frac{1}{\Omega}\mathbf{N}_i \cdot \oint p\mathbf{n}\, d\Gamma = \\
&= \frac{1}{\Omega}[\frac{1}{2}(p_a + p_b)\mathbf{s}_{ab} + \frac{1}{2}(p_b + p_c)\mathbf{s}_{bc} + \frac{1}{2}(p_c + p_d)\mathbf{s}_{cd} + \frac{1}{2}(p_d + p_a)\mathbf{s}_{da}] \cdot \mathbf{N}_i = \\
&= \frac{1}{2\Omega}[p_a(\mathbf{s}_{db} \cdot \mathbf{N}_i) + p_b(\mathbf{s}_{ac} \cdot \mathbf{N}_i) + p_c(\mathbf{s}_{bd} \cdot \mathbf{N}_i) + p_d(\mathbf{s}_{ca} \cdot \mathbf{N}_i)]
\end{aligned}$$

It is trivial to see that the weighting coefficients all have the correct sign (for p_a and p_b negative, and for p_c and p_d positive), and that this is due to their location in different quadrants. Furthermore, the coefficient for p_a and p_b is, apart from the sign, the same as the coefficient for, respectively p_c and p_d.
The next issue to be discussed, is how do we choose our stencil. We omit here the restriction to the six-point stencil as sketched in 5.1, and we choose our stencil among the cells that are connected to the two vertices of face i, see Figure 5.8. Nevertheless, in most of the occurring cases the four points will all be part of the six-point stencil. The criterion is that we choose in each quadrant the point that is located closest to face i. In Figure 5.8, this leads to $a = 1$, $b = 3$, $c = 2$ and $d = 5$.

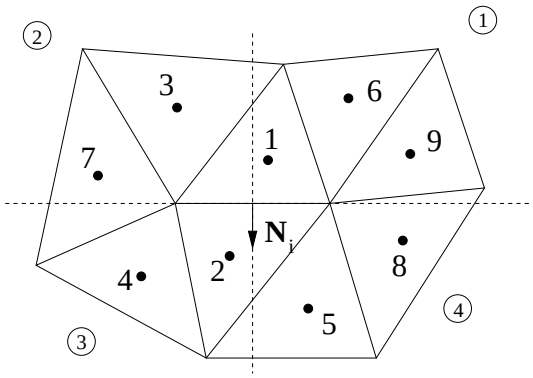

Figure 5.8: Stencil from which the four points in the four quadrant method are chosen.

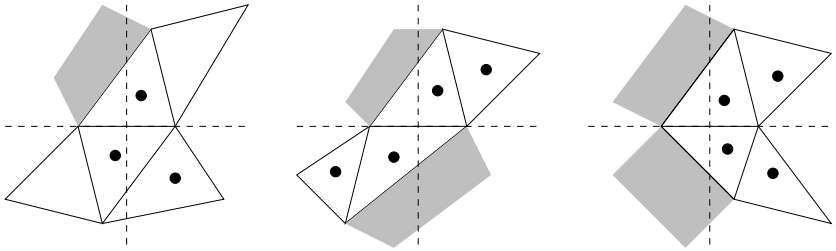

Figure 5.9: Stencil for the four quadrant method in the vicinity of boundaries. The used points are indicated with a •.

In the vicinity of boundaries, in the situation that one quadrant is empty, we use a three-point stencil. In the situation that two quadrants are empty, we use in each of the non-empty quadrant one additional point: the one second closest to face i. This is summarized in Figure 5.9.

5.7.6 Contour integral formulation

The contour integral formulation to compute the pressure gradient is based on the identity:

$$\int_A \nabla p \cdot \mathbf{N}_i \, d\mathbf{x} = \mathbf{N}_i \cdot \int_A \nabla p \, d\mathbf{x} = \mathbf{N}_i \cdot \oint_{\partial A} p\mathbf{n} \, d\Gamma, \tag{5.86}$$

where $\mathbf{n}$ is the outwards pointing normal on the boundary ∂A of area A. With A being the two triangles adjacent to face i, the pressure gradient at face i is computed from:

$$(\nabla p \cdot \mathbf{N})_i = \frac{1}{A_i} \mathbf{N}_i \cdot \oint_{\partial A_i} p\mathbf{n} \, d\Gamma \approx \frac{1}{A_i} \sum_e p_e \bar{l}_e (\mathbf{N}_e \cdot \mathbf{N}_i). \tag{5.87}$$

In this expression, $A_i = \Omega_1 + \Omega_2$ is the area of the two triangles, the loop is over the four faces $e \in \{k, l, o, j\}$ and $\bar{l}_e = l_e(\mathbf{n}_e \cdot \mathbf{N}_e)$. The pressure at face e follows from a weighted averaging of the two neighboring pressure values. The stencil consists of the six cells. Notice that for the situation in which $(\mathbf{x}_2 - \mathbf{x}_1)$ is parallel to $\mathbf{N}_i$, the contour integral formulation does not, unlike the path-integral formulation, reduce to (5.73).

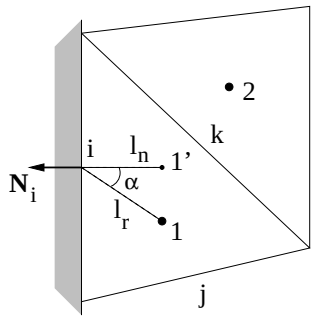

Figure 5.10: Boundary face i with boundary cell 1.

5.7.7 Pressure gradient at boundary faces

When the normal momentum component is given at a boundary face, then there is no need to discretize the pressure gradient term. When the normal momentum component is not given, then the pressure gradient must be evaluated numerically.

Subsonic outflow
At a subsonic outflow boundary the pressure is given. We have implemented, depending on the pressure gradient method, two ways to compute the pressure gradient at a boundary where the pressure is given:

- **Path-integral method, auxiliary point method, four quadrant method.**
  With Figure 5.10 in mind, we write for the pressure gradient at boundary face i:

$$p_i - p_1 = (\nabla p)_i \cdot (\mathbf{x}_i - \mathbf{x}_1) = (\nabla p)_i \cdot l_r \mathbf{e}_r, \tag{5.88}$$

  where p_i is the given pressure at the boundary face. Here we introduced $(\mathbf{x}_i - \mathbf{x}_1) = |\mathbf{x}_i - \mathbf{x}_1|\mathbf{e}_r = l_r \mathbf{e}_r$. In addition, we write

$$p_i - p_{1'} = (\nabla p)_i \cdot (\mathbf{x}_i - \mathbf{x}_{1'}) = (\nabla p \cdot \mathbf{N})_i l_n \tag{5.89}$$

  Assuming that the pressure gradient in the tangential direction, i.e. parallel to the boundary face, is negligible, we can write $p_1 = p_{1'}$. Simple geometrical considerations lead to

$$\cos\alpha = \frac{l_n}{l_r} = \frac{(\mathbf{x}_i - \mathbf{x}_1)\cdot \mathbf{N}_i}{|\mathbf{x}_i - \mathbf{x}_1||\mathbf{N}_i|} = \frac{(\mathbf{x}_i - \mathbf{x}_1)\cdot \mathbf{N}_i}{l_r}, \tag{5.90}$$

  resulting in the following pressure gradient

$$\mathbf{N}_i \cdot \int_{\Omega_i} \nabla p_i^n \, d\Omega = \Omega_i (\nabla p \cdot \mathbf{N})_i \approx \Omega_i \frac{p_i - p_1}{(\mathbf{x}_i - \mathbf{x}_1)\cdot \mathbf{N}_i}. \tag{5.91}$$

- **Contour integral method**
  The CV for face i is cell 1, hence application of the contour integral method gives:

$$\int_{\Omega_i} \nabla p \cdot \mathbf{N}_i \, d\mathbf{x} = \oint_{\partial\Omega_i} p(\mathbf{n}\cdot\mathbf{N}_i)\, d\Gamma \approx \sum_e p_e \bar{l}_e (\mathbf{N}_e \cdot \mathbf{N}_i), \tag{5.92}$$

where the summation is over faces $e \in \{i, j, k\}$. Obviously, p_i is given. With k an internal face, the pressure at face k follows from area weighted averaging as discussed in Section 4.4.2. If j is an internal face, then of course a similar averaging is done. If j is a boundary face, at this moment we have restricted us to the situation in which the pressure at face j is not given, and a one-sided differencing is used: $p_j = p_1$. In the future the situation in which also p_j is given, has to be inserted in the software.

Supersonic outflow boundary

At a supersonic outflow boundary neither the momentum nor the pressure is given. We put the pressure gradient to zero at the outflow boundary.

5.8 Computation of momentum vector at faces

Once we have obtained the normal momentum components at all faces of the grid, we want to compute, using interpolations, the full momentum vector at all faces. Recall, see Section 5.4.2, that we have developed methods to obtain the tangential momentum component at the faces. The tangential momentum component at face j, equation (5.29), is computed using weighted interpolations, i.e. with

$$\chi_q = \frac{\Omega_1}{\Omega_1+\Omega_6}\xi_q \qquad \chi_p = \frac{\Omega_1}{\Omega_1+\Omega_6}\xi_p \qquad \chi_k = \frac{\Omega_6}{\Omega_1+\Omega_6}\xi_k \qquad \chi_i = \frac{\Omega_6}{\Omega_1+\Omega_6}\xi_i \tag{5.93}$$

Note that equation is the same as (5.35). On boundary faces, one-sided interpolations are used.
Now that we have the momentum vector in terms of normal and tangential components, it is trivial to find the $x-$ and $y-$component using

$$\begin{aligned} m_x &= \mathbf{m}\cdot\mathbf{e}_x = mN_x + \tilde{m}t_x; \\ m_y &= \mathbf{m}\cdot\mathbf{e}_y = mN_y + \tilde{m}t_y. \end{aligned}$$

5.9 Computation of velocity at faces

Once we have obtained the normal momentum component m_i at face i and the density in the adjacent cell-centers q and r, the normal velocity component at face i is computed using

$$u_i = \frac{m_i}{\rho_i}, \tag{5.94}$$

where ρ_i follows from weighted interpolation (4.16). At boundary faces, a one sided approximation is used.

Chapter 6

Discretization of the equation of state

6.1 Equation of state: discretization

The relation between three thermodynamic quantities is called the equation of state. Several forms are gathered in Section 2.3.3. This equation of state is needed in the time-stepping procedure, see Section 9.4 , and for the initial conditions, see Section 2.5. Note that in many occasions the quantity $(\mathbf{m} \cdot \mathbf{m})$ or $(\mathbf{u} \cdot \mathbf{u})$ is required at the cell-centers.

To examine this problem closer, consider Figure 6.1. Let H_a, ρ_a and $\mathbf{m}_e$, $e = \{i, j, k\}$ be given. Then p_a follows from the equation of state (in dimensionless quantities):

$$p_a = \rho_a \left[H_a - \frac{1}{2}(\gamma - 1) M_r^2 (\mathbf{u} \cdot \mathbf{u})_a \right]. \tag{6.1}$$

The problem lies, because of the staggered placement of the variables, in determining $(\mathbf{u} \cdot \mathbf{u})_a$. The first step is to note that:

$$(\mathbf{u} \cdot \mathbf{u})_a = \frac{(\mathbf{m} \cdot \mathbf{m})_a}{\rho_a^2}. \tag{6.2}$$

Hence the problem is now to determine $(\mathbf{m} \cdot \mathbf{m})_a$.
To this aim we have developed methods based on two different starting points. The first one is based on using the momentum vector at the three faces of the triangle, and the second one on a least squares approach where only the normal momentum components at the three faces

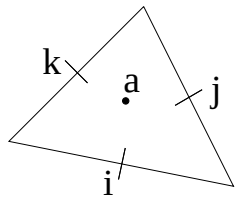

Figure 6.1: Cell a with faces i, j and k.

of the triangle are used.

Supersonic inflow

At a supersonic inflow face, both the momentum vector and the pressure are given. In order to take the given pressure into account, we put the pressure in the boundary cells adjacent to supersonic inflow faces equal to the prescribed pressure at the boundary faces.

6.2 Computation of square of momentum vector in cell-center

Consider Figure 6.1. We have developped two different methods to evaluate $\phi_{cell} = (\mathbf{m} \cdot \mathbf{m})$. The first method is based on the reconstruction procedure, and the second one on a least squares approach.

Method 0: compute $(\mathbf{m} \cdot \mathbf{m})_a$ using the least squares approach
The problem is, as said before, to determine $\mathbf{m}_a = (m_{x,a}, m_{y,a})$. Of course, since the triangles are 'small', we have $m_i \approx (\mathbf{m}_a \cdot \mathbf{n}_i)$, $m_j \approx (\mathbf{m}_a \cdot \mathbf{n}_j)$ and $m_k \approx (\mathbf{m}_a \cdot \mathbf{n}_k)$. We have implemented the following least squares approaches. Choose $\mathbf{m}_a$ such that the momentum functional

$$\mathcal{F}(m_x, m_y) = \sum_e [m_e - (\mathbf{m} \cdot \mathbf{n}_e)]^2 \tag{6.3}$$

is minimal. The summation is over the three faces of the cell.
The minimum of $\mathcal{F}(m_x, m_y)$ is found there where

$$\begin{aligned} \frac{\partial \mathcal{F}}{\partial m_x} &= \sum_e -2n_{x,e}(m_e - m_x n_{x,e} - m_y n_{y,e}) &= 0 \\ \frac{\partial \mathcal{F}}{\partial m_y} &= \sum_e -2n_{y,e}(m_e - m_x n_{x,e} - m_y n_{y,e}) &= 0. \end{aligned}$$

This is equivalent to solving the system:

$$\begin{bmatrix} \sum_e n_{x,e}^2 & \sum_e n_{x,e} n_{y,e} \\ \sum_e n_{x,e} n_{y,e} & \sum_e n_{y,e}^2 \end{bmatrix} \begin{bmatrix} m_x \\ m_y \end{bmatrix} = \begin{bmatrix} \sum_e m_e n_{x,e} \\ \sum_e m_e n_{y,e} \end{bmatrix}.$$

The routine that does this computation, is `fvmomcelc00.f`, and is the default.

Method 1: compute $(\mathbf{m} \cdot \mathbf{m})_a$ using the reconstruction procedure
For more information concerning the reconstruction procedure, we refer to Section 5.4.2.
Using the reconstruction procedure, we can compute the tangential momentum component $\tilde{m}$ at all faces. Hence, we can compute $\mathbf{m} = m\mathbf{N} + \tilde{m}\mathbf{t}$ at all faces. Let ϕ_e be a Cartesian component of the momentum vector (i.e. m_x or m_y) at face e, and let (x_e, y_e) be the coordinates of the midpoints of the faces $e = i, j, k$, then we can approximate $\phi = \phi(x, y)$ over triangle a by a linear function of the form:

$$\phi(x, y) = c_1 + c_2 x + c_3 y. \tag{6.4}$$

Coefficients c_1, c_2 and c_3 then follow from the system:

$$\begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \phi_i \\ \phi_j \\ \phi_k \end{bmatrix}.$$

The inverse A^{-1} of the 3×3 matrix is given by:

$$\frac{1}{||A||} \begin{bmatrix} x_j y_k - y_j x_k & -x_i y_k + y_i x_k & x_i y_j - y_i x_j \\ -y_k + y_j & y_k - y_i & -y_j + y_i \\ x_k - x_j & -x_k + x_i & x_j - x_i \end{bmatrix}$$

with $||A||$ the determinant of the matrix:

$$||A|| = x_j y_k - y_j x_k - x_i y_k + y_i x_k + x_i y_j - y_i x_j. \tag{6.5}$$

The coefficients c_1, c_2 and c_3 then are readily obtained for $\phi = m_x$ and for $\phi = m_y$. With (x_a, y_a) the coordinates of the cell center of cell a, the sought vector of $\mathbf{m}$ at the cell center follows from:

$$\phi(x_a, y_a) = c_1 + c_2 x_a + c_3 y_a \tag{6.6}$$

The routine that does this computation, is `fvmomcelc01.f`.

Chapter 7

Pressure-correction

7.1 Introduction

The incompressible dimensionless Euler equations are repeated here:

- Continuity equation:

$$u^{\alpha}_{,\alpha} = \text{div}\,\mathbf{u} = 0. \tag{7.1}$$

- Momentum equation (see equation (2.2)):

$$\frac{\partial u^{\alpha}}{\partial t} + \left(u^{\beta}u^{\alpha}\right)_{,\beta} = -p_{,\alpha}. \tag{7.2}$$

Note that we have inserted $\rho = 1$ everywhere. The continuity equation is a constraint on the possible velocity field, and the pressure must be such that the divergence-free constraint is satisfied. Another problem is that there is no equation containing the time-derivative of the pressure, i.e. there is no equation yielding the new pressure. The pressure-correction approach is a way to solve these problems. We apply the so-called discrete pressure-correction method, in which first the time and space discretisation is derived and afterwards the pressure-correction. By doing this there is no need to define boundary conditions for the pressure equation (which are not part of the original problem). The necessary boundary conditions have already been incorporated in the discretization.

7.2 Solution algorithm

Discretisation in space and time of (7.1)–(7.2) can be written symbolically as:

$$D\mathbf{u}^{n+1} = 0 \tag{7.3}$$

$$R\frac{\mathbf{u}^{n+1} - \mathbf{u}^{n}}{\Delta t} + C\mathbf{u}^{n+1} = -RGp^{n+1} \tag{7.4}$$

Remarks:

- an implicit Euler time-integration scheme has been used;

- D is the discrete divergence operator; R is an operator containing the area of the control volume ($R_{ij} = 0$ for $i \neq j$), C is the convection operator and G the gradient operator.
- $\mathbf{u}$ and p are the velocity and pressure solution vectors.

The predictor of the new velocity $\mathbf{u}^*$ is computed from:

$$R\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} + C\mathbf{u}^* = -RGp^n. \tag{7.5}$$

Subtracting (7.5) from (7.4) yields:

$$R\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} + C(\mathbf{u}^{n+1} - \mathbf{u}^*) = -RG(p^{n+1} - p^n). \tag{7.6}$$

Postulating

$$C(\mathbf{u}^{n+1} - \mathbf{u}^*) = 0 \tag{7.7}$$

and defining the pressure-correction δp as:

$$\delta p = p^{n+1} - p^n, \tag{7.8}$$

we arrive at:

$$\mathbf{u}^{n+1} - \mathbf{u}^* = -\Delta t G\delta p. \tag{7.9}$$

Inserting (7.3) into (7.9) yields the pressure-correction equation:

$$\Delta t DG\delta p = D\mathbf{u}^* \tag{7.10}$$

Knowing δp, we can compute the new velocity by rewriting (7.9) as:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t G\delta p \tag{7.11}$$

Hence, summarizing, the pressure-correction algorithm consists of three steps:

1. Compute the velocity predictor $\mathbf{u}^*$ from (7.5).
2. Compute the pressure-correction δt from (7.10).
3. Compute the new velocity $\mathbf{u}^{n+1}$ from (7.11).

In the next section, the discretisation of these equations is discussed.

7.3 Discretisation of the pressure-correction equations

In this section, the discretisation of equations (7.5), (7.10) and (7.11) is discussed.

The discretisation of (7.5) is discussed in Chapter 5.

Consider Figure 4.1. The pressure gradient at face k is computed from, see Section 5.7:

$$(Gp)_k = (\nabla p \cdot \mathbf{N})_k = \sum_{s=s(k)} \gamma_{k,s} p_s, \tag{7.12}$$

where the summation is over the gradient stencil of face k. For example, if the path-integral method is used, the stencil is given by: $s = s(k) = \{i, j, q, f, g, r\}$. Similar expressions hold for other faces. The divergence operator on a vector $\mathbf{v}$ located in cell i is defined by

$$(D\mathbf{v})_i = \sum_{e=e(i)} v_e \bar{l}_e \qquad v_e = \mathbf{v}_e \cdot \mathbf{N}_e \qquad \bar{l}_e = l_e(\mathbf{n}_e \cdot \mathbf{N}_e), \tag{7.13}$$

and $e = e(i) = \{k, l, o\}$. Putting this together leads for the pressure-correction equation (7.10) to:

$$\Delta t \sum_e (\nabla \delta p \cdot \mathbf{N})_e \bar{l}_e = \sum_e u_e^* \bar{l}_e. \tag{7.14}$$

The discretisation of (7.11) is trivial:

$$u_k^{n+1} = u_k^* - \Delta t \sum_{s=s(k)} \gamma_{k,s} (\delta p)_s. \tag{7.15}$$

One question is, how do the boundary conditions enter the discretisation? Assume that face k in Figure 4.1 is a boundary face.
First consider the case that u_k^{n+1} is given at this boundary face. Then there is obviously no need to compute the pressure gradient at this face in the momentum equation, hence we put $\gamma_{k,s} = 0$ for all possible s. Hence, $u_k^* = u_k^{n+1}$. As a consequence, the contribution of face k to the summation in both sides of (7.14) is zero. Furthermore, the gradient term in (7.15) equals zero, as it should.
Now consider the case that u_k is not given. In this situation, the pressure gradient at face k is computed using the procedure discussed in Section (5.7.7), and the remainder of the algorithm looks identical to the algorithm as discussed before.

Diagonal dominancy of the pressure-correction equation
The ith row of the pressure-correction matrix follows from the left-hand side of (7.14). Using (5.66) we see that the sum S_i of all matrix elements in the ith row adds up to zero:

$$S_i = \Delta t \sum_e \bar{l}_e \sum_{s=s(e)} \gamma_{e,s} = 0. \tag{7.16}$$

Only for boundary cells with a boundary face at which the pressure is given, this does not hold. Let at boundary face k the pressure p_k be given, then the pressure gradient at this face is computed from, see equation (5.91):

$$(\nabla p \cdot \mathbf{N})_k = r + \gamma_{k,i} p_i \qquad r = \frac{p_k}{|\mathbf{x}_k - \mathbf{x}_i|} \qquad \gamma_{k,i} = \frac{-1}{|\mathbf{x}_k - \mathbf{x}_i|}. \tag{7.17}$$

This leads to

$$S_i = \Delta t \sum_{e=e(i)} \bar{l}_e \sum_{s=s(e)} \gamma_{e,s} + \Delta t \bar{l}_k \gamma_{k,i} < 0, \tag{7.18}$$

where the summation over $e = e(i)$ is over the internal faces.

Chapter 8

Mach-uniform pressure-correction

8.1 Introduction

With respect to the Mach-uniform pressure-correction method, the following remarks are made:

- the corresponding equations are discussed in Section 2.3.2.
- the aim is to be able to compute, efficiently and accurately, incompressible and compressible flows, or flows in which both incompressible and compressible parts are present, with one algorithm.

8.2 Solution algorithm & pressure-correction equation

The following solution algorithm is proposed:

1. The new density follows from:

$$\frac{\rho^{n+1} - \rho^n}{\Delta t} + \nabla \cdot \mathbf{u}^n \rho^{n+1} = 0. \tag{8.1}$$

2. The predictor of the momentum follows from:

$$\frac{\mathbf{m}^* - \mathbf{m}^n}{\Delta t} + \nabla \cdot \mathbf{u}^n \mathbf{m}^* = -\nabla p^n. \tag{8.2}$$

3. Compute $\mathbf{u}^*$.
4. From (2.69) a suitable pressure-correction equation will be derived. This means that we obtain the new pressure and the corrected momentum and velocity. This will be discussed below.
5. The new enthalpy follows from (2.67).

The relation between the predictor of the momentum and the predictor of the velocity is defined by:

$$\mathbf{m}^* = \rho^{n+1} \mathbf{u}^*. \tag{8.3}$$

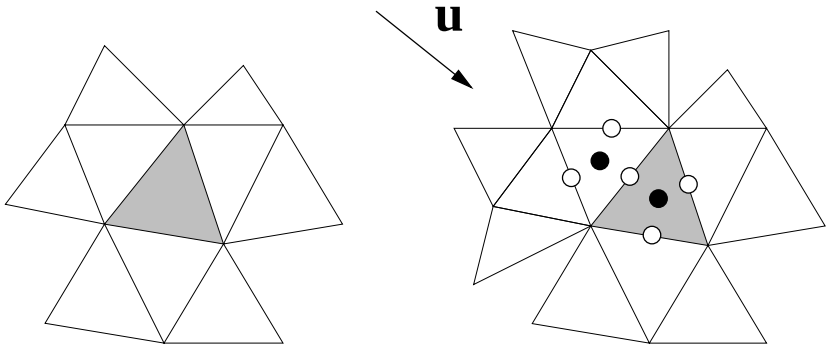

Figure 8.1: Pressure-correction stencil for incompressible computations (a). Pressure-correction stencil after discretization of (8.7) (b).

The following pressure-correction is postulated:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho^{n+1}} \nabla \delta p, \tag{8.4}$$

where

$$\delta p = p^{n+1} - p^n. \tag{8.5}$$

Note that

$$\left(\rho \mathbf{u}^2\right)^{n+1} = \left(\mathbf{m}^2/\rho\right)^{n+1} = (\mathbf{m}^* - \Delta t \, \nabla \delta p)^2/\rho^{n+1}. \tag{8.6}$$

Inserting all this in (2.69) yields:

$$\begin{aligned} M_r^2 \Bigg\{ & \frac{\delta p}{\Delta t} + \frac{1}{2}(\gamma - 1) \frac{(\mathbf{m}^* - \Delta t \, \nabla \delta p)^2/\rho^{n+1} - (\mathbf{m}^n)^2/\rho^n}{\Delta t} + \\ + \quad & \nabla \cdot \left(\mathbf{u}^* - \frac{\Delta t}{\rho^{n+1}} \nabla \delta p \right) \left[\gamma (p^n + \delta p) + \frac{1}{2}(\gamma - 1)(\mathbf{m}^* - \Delta t \, \nabla \delta p)^2/\rho^{n+1} \right] \Bigg\} + \\ + \quad & \nabla \cdot \left(\mathbf{u}^* - \frac{\Delta t}{\rho^{n+1}} \nabla \delta p \right) = 0. \end{aligned} \tag{8.7}$$

The pressure-correction stencil that we use for incompressible computations is depicted in Figure 8.1a, where the shaded cell is the cell under consideration. We would like to keep this stencil the same for the discretized version of (8.7). In the second line of (8.7), a term representing the convected kinetic energy is present. The convected kinetic energy has to be evaluated in each cell, after which an upwind or central approximation should be applied. How the kinetic energy in a cell is obtained, will be discussed later, but it suffices here to state that the pressure gradient at its faces has to be evaluated. Suppose a first order upwind interpolation is used, and the velocity is directed as indicated in Figure 8.1b. This means that, in order to obtain the pressure-correction equation for the shaded cell, the kinetic energy

has to be evaluated in the cells indicated by a $\bullet$. In order to obtain the kinetic energy in these cells, the pressure gradient at the faces indicated by $\circ$ has to be evaluated. As a consequence, the stencil for the pressure-correction equation is enlarged, see Figure 8.1b, and this is what we did not want to happen.

Hence, some changes with respect to (8.7) have to be made. It is clear that the $\nabla \cdot \mathbf{u}$, i.e. the last line in (8.7), has to be discretized implicitly in order to have the scheme reduced to the standard incompressible pressure-correction scheme. In the derivation to arrive at (2.69), one notices that the $\nabla \cdot \mathbf{u}$ term stems from the convection velocity of the pressure, which means that we have to evaluate the convection velocity (second line of (8.7)) also at the new time level. If we take the kinetic energy at the $\star$ level instead of the new time level, this will not affect the scheme in the limit $M_r \downarrow 0$, and the standard 10 point stencil is maintained. The temporal accuracy is probably not too much affected. Hence, this leads to:

$$\begin{aligned} M_r^2 \Bigg\{ & \frac{\delta p}{\Delta t} + \frac{1}{2}(\gamma - 1) \frac{(\mathbf{m}^* - \Delta t \, \nabla \delta p)^2 / \rho^{n+1} - (\mathbf{m}^n)^2 / \rho^n}{\Delta t} + \\ & + \quad \nabla \cdot \left(\mathbf{u}^* - \frac{\Delta t}{\rho^{n+1}} \nabla \delta p \right) \left[\gamma (p^n + \delta p) + \frac{1}{2}(\gamma - 1)(\mathbf{m}^*)^2 / \rho^{n+1} \right] \Bigg\} + \\ & + \quad \nabla \cdot \left(\mathbf{u}^* - \frac{\Delta t}{\rho^{n+1}} \nabla \delta p \right) = 0. \end{aligned} \tag{8.8}$$

Linearization of this equation, i.e. leaving the terms linear in δp and omitting the higher order terms, and rearranging results in:

$$\begin{aligned} M_r^2 & \left\{ \frac{\delta p}{\Delta t} + \frac{1}{2}(\gamma - 1) \frac{\left[(\mathbf{m}^*)^2 - 2 \Delta t \, \mathbf{m}^* \cdot \nabla \delta p \right] / \rho^{n+1} - (\mathbf{m}^n)^2 / \rho^n}{\Delta t} \right\} + \\ & + \quad \nabla \cdot \mathbf{u}^* \left\{ 1 + \gamma M_r^2 (p^n + \delta p) + \frac{1}{2}(\gamma - 1) M_r^2 (\mathbf{m}^*)^2 / \rho^{n+1} \right\} - \\ & - \quad \nabla \cdot \frac{\Delta t}{\rho^{n+1}} \nabla \delta p \left\{ 1 + \gamma M_r^2 p^n + \frac{1}{2}(\gamma - 1) M_r^2 (\mathbf{m}^*)^2 / \rho^{n+1} \right\} = 0. \end{aligned} \tag{8.9}$$

Discretization of this equation will be discussed in the next section.

8.3 Discretization of the pressure-correction equation

We will derive a discretization of (8.9) in cell 1, both for the interior and boundary case, see Figure 8.2. Before the separate terms are discussed, we will discuss how to obtain vector quantities in cell-centers.

8.3.1 Evaluation of vector in cell-center

In the past we have devised several ways to obtain a vector in the cell-center. One of them came down to obtaining the vector at the three cell-faces by means of reconstruction using the normal vector components at the surrounding faces. This would result in the use of the normal vector components at all nine faces (in the interior case) that are depicted in Figure 8.2a. This is what we have to avoid, since this would lead, due to the evaluation of the vector $\nabla \delta p$ in the time derivate, to a larger stencil than given in Figure 8.1a. However, only

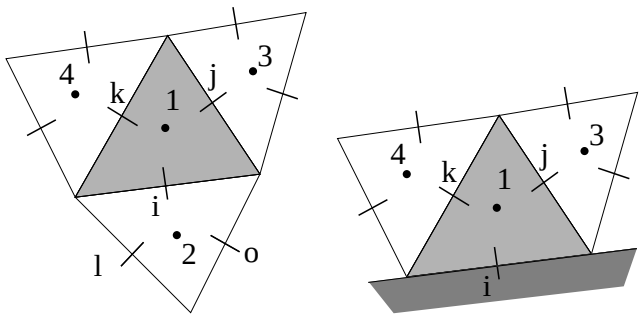

Figure 8.2: The CV for the pressure-correction equation is shaded. Interior (a) and boundary (b).

making use of the normal components at the three cell-faces does not enlarge the stencil. We will now show how this can be done. (See also Chapter 6)

Our aim is to find an approximation of the vector $\mathbf{v}$ in the cell-center of cell 1, making only use of given values of $v_e = \mathbf{v}_e \cdot \mathbf{N}_e$, with $e \in \{i, j, k\}$ the faces of cell 1 (see Figure 8.2). We choose $\mathbf{v}_1$ such that the least squares functional

$$\mathcal{F}(\mathbf{v}_1) = \sum_{e(1)} [v_e - (\mathbf{v} \cdot \mathbf{N}_e)]^2 \tag{8.10}$$

is minimal. Writing $\mathbf{v} = (v_x, v_y)$, the minimum of $\mathcal{F}(v_x, v_y)$ is found there where

$$\begin{aligned} \frac{\partial \mathcal{F}}{\partial v_x} = \sum_e -2N_{x,e}(v_e - v_x N_{x,e} - v_y N_{y,e}) &= 0 \\ \frac{\partial \mathcal{F}}{\partial v_y} = \sum_e -2N_{y,e}(v_e - v_x N_{x,e} - v_y N_{y,e}) &= 0. \end{aligned}$$

The summation over e represents summation over the three faces of cell 1. This is equivalent to solving the system $A\mathbf{v} = \mathbf{b}$, with

$$A = \begin{bmatrix} \sum_e N_{x,e}^2 & \sum_e N_{x,e}N_{y,e} \\ \sum_e N_{x,e}N_{y,e} & \sum_e N_{y,e}^2 \end{bmatrix}, \qquad \mathbf{b} = \sum_e v_e \mathbf{N}_e = \begin{bmatrix} \sum_e v_e N_{x,e} \\ \sum_e v_e N_{y,e} \end{bmatrix}. \tag{8.11}$$

Note that, because we restrict ourselves to the three normal components of the cell under consideration, this procedure needs not be modified near boundaries.

8.3.2 Discretization time derivative

The discretization of the the time derivative, i.e. the first line in equation (8.9) yields:

$$\Omega_1 M_r^2 \left\{ \frac{(\delta p)_1}{\Delta t} + \frac{1}{2}(\gamma - 1) \frac{\left[(\mathbf{m}_1^*)^2 - 2\Delta t \, (\mathbf{m}^* \cdot \nabla \delta p)_1 \right] / \rho_1^{n+1} - (\mathbf{m}_1^n)^2 / \rho_1^n}{\Delta t} \right\}, \tag{8.12}$$

where Ω_1 is the area of cell 1. The terms with (δp) are put in the matrix, whereas the other terms are put in the right-hand side. The terms $(\mathbf{m}_1^*)^2$ and $(\mathbf{m}_1^n)^2$ are obtained using the procedure described in Section 8.3.1. Only the term $(\nabla\delta p)_1$ in $(\mathbf{m}^* \cdot \nabla\delta p)_1$ poses some additional problems. The expression $(\nabla\delta p)_1$ is written in terms of $(\nabla\delta p \cdot \mathbf{N}_e)_e$, $e \in \{i,j,k\}$, as discussed in Section 8.3.1. As discussed elsewhere, at face e the normal pressure gradient is computed using

$$(\nabla\delta p \cdot \mathbf{N}_e)_e = \sum_{c(e)} \gamma_c (\delta p)_c, \tag{8.13}$$

where the summation is over the gradient stencil, which consists of the neighboring and next-neighboring cells of face e. With A as in (8.11), the expression $(\nabla\delta p)_1$ follows from

$$A(\nabla\delta p)_1 = \sum_e \mathbf{N}_e \sum_{c(e)} \gamma_c (\delta p)_c, \tag{8.14}$$

or, written out:

$$(\nabla\delta p)_1 = \frac{1}{a_{1,1}a_{2,2} - a_{1,2}a_{2,1}} \begin{bmatrix} \sum_{c(e)} \sum_e \{a_{2,2}N_{e,x} - \sum_e a_{1,2}N_{e,y}\}\gamma_c(\delta p)_c \\ \\ \sum_{c(e)} \sum_e \{-a_{2,1}N_{e,x} + \sum_e a_{1,1}N_{e,y}\}\gamma_c(\delta p)_c \end{bmatrix}. \tag{8.15}$$

Here we made use of

$$A^{-1} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ \\ a_{2,1} & a_{2,2} \end{bmatrix}^{-1} = \frac{1}{a_{1,1}a_{2,2} - a_{1,2}a_{2,1}} \begin{bmatrix} a_{2,2} & -a_{1,2} \\ \\ -a_{2,1} & a_{1,1} \end{bmatrix}. \tag{8.16}$$

Taking the inner product of $(\nabla\delta p)_1$ with $\mathbf{m}_1^*$ is trivial.

Boundary conditions
Consider now the boundary case, i.e. cell 1 is a boundary cell and face i is the boundary face, see Figure 8.2b. Two situations can occur:

- **The normal momentum component m_i is given.**
  In this case, there is no need to compute the pressure-correction term $(\nabla\delta p \cdot \mathbf{N})_i$, since it is zero. In the algorithm as given above, this means that the summation over e implies summation over only j and k. Another, mathematically equivalent way of seeing this, is by putting all coefficients γ_c to zero in the summation over $c(i)$. Doing this facilitates the use of `DO`-loops over all (i.e. internal and boundary) cells.

- **The normal momentum component m_i not given.**
  In this situation, the pressure-correction term needs also to be computed at the boundary face. In the algorithm as given above, this implies no changes, apart from the fact that a given pressure-difference at the boundary has to be put in the right-hand side.

8.3.3 Discretization convection term

The discretized convection term is given by

$$\sum_{e(1)} u_e^* \bar{l}_e \left\{ 1 + \gamma M_r^2 [p_e^n + (\delta p)_e] + \frac{1}{2}(\gamma - 1) M_r^2 (\mathbf{m}_e^*)^2 / \rho_e^{n+1} \right\}. \tag{8.17}$$

Here the summation is over the faces of cell 1. Because the complete term that is between curly braces originates from ρH, each term needs to interpolated (central or upwind) in an identical manner.

Internal faces
The projected normal velocity predictor at an internal face i, see Figure 8.2a, is computed using:

$$u_i^* = m_i^*/\rho_{i,av}^{n+1}, \tag{8.18}$$

where

$$\rho_{i,av} = \frac{\Omega_2}{\Omega_1+\Omega_2}\rho_1 + \frac{\Omega_1}{\Omega_1+\Omega_2}\rho_2. \tag{8.19}$$

If a first order upwind scheme is used, then, if $u_i\bar{l}_i > 0$, the pressure-correction matrix P and right-hand side r are changed as follows due to the contribution of face i:

$$\begin{aligned} P_{11} &:= P_{11} + \gamma M_r^2 u_i^* \bar{l}_i, \\ r_1 &:= r_1 - u_i^* \bar{l}_i \left\{ 1 + \gamma M_r^2 p_1^n + \frac{1}{2}(\gamma-1) M_r^2 (\mathbf{m}_1^*)^2/\rho_1^{n+1} \right\}. \end{aligned}$$

On the other hand, if $u_i\bar{l}_i < 0$, then

$$\begin{aligned} P_{12} &:= P_{12} + \gamma M_r^2 u_i^* \bar{l}_i, \\ r_1 &:= r_1 - u_i^* \bar{l}_i \left\{ 1 + \gamma M_r^2 p_2^n + \frac{1}{2}(\gamma-1) M_r^2 (\mathbf{m}_2^*)^2/\rho_2^{n+1} \right\}. \end{aligned}$$

The following central scheme is used:

$$\begin{aligned} P_{11} &:= P_{11} + \frac{1}{2}\gamma M_r^2 u_i^* \bar{l}_i, \\ P_{12} &:= P_{12} + \frac{1}{2}\gamma M_r^2 u_i^* \bar{l}_i, \\ r_1 &:= r_1 - u_i^* \bar{l}_i \left\{ 1 + \gamma M_r^2 p_i^n + \frac{1}{2}(\gamma-1) M_r^2 (\mathbf{m}_i^*)^2/\rho_i^{n+1} \right\}, \end{aligned}$$

where:

$$p_i^n = \frac{1}{2}(p_1^n + p_2^n), \qquad (\mathbf{m}_i^*)^2/\rho_i^{n+1} = \frac{1}{2}\left[(\mathbf{m}_1^*)^2/\rho_1^{n+1} + (\mathbf{m}_2^*)^2/\rho_2^{n+1} \right].$$

Boundary conditions
Consider boundary face i, see Figure 8.2b. The term

$$\frac{1}{2}(\gamma-1) M_r^2 u_i^* \bar{l}_i (\mathbf{m}_1^*)^2/\rho_1^{n+1} \tag{8.20}$$

is moved to the right-hand side. The normal velocity predictor, cf. (8.18), is computed with $\rho_{i,av} = \rho_1$ instead of (8.19). If both $\mathbf{m}$ and h are given at this face (e.g. inflow), then:

$$u_i^* \bar{l}_i \left\{ 1 + \gamma M_r^2 [p_i^n + (\delta p)_i] \right\} = u_i^{n+1} \bar{l}_i \left\{ 1 + \gamma M_r^2 p_i^{n+1} \right\} = u_i^{n+1} \bar{l}_i \rho_i^{n+1} h_i^{n+1} = m_i^{n+1} h_i^{n+1} \tag{8.21}$$

is moved to the right-hand side as well. If a central scheme is applied, and p at the boundary face i is given (e.g. outflow), then we write

$$u_i^* \bar{l}_i \left\{1 + \gamma M_r^2 [p_i^n + (\delta p)_i]\right\} = u_i^* \bar{l}_i \left\{1 + \frac{1}{2}\gamma M_r^2 [p_1^n + (\delta p)_1 + p_i^{n+1}]\right\}, \tag{8.22}$$

resulting in the following changes of the matrix-elements:

$$\begin{aligned} P_{11} &:= P_{11} + \frac{1}{2}\gamma M_r^2 u_i^* \bar{l}_i, \\ r_1 &:= r_1 - u_i^* \bar{l}_i \left\{1 + \frac{1}{2}\gamma M_r^2 (p_1^n + p_i^{n+1})\right\}. \end{aligned}$$

In all other situations (e.g. homogeneous Neumann conditions for enthalpy and pressure (free-slip walls), or first order upwind and pressure given), one-sided differences are used, resulting in the following changes of the matrix-elements:

$$\begin{aligned} P_{11} &:= P_{11} + \gamma M_r^2 u_i^* \bar{l}_i, \\ r_1 &:= r_1 - u_i^* \bar{l}_i \left\{1 + \gamma M_r^2 p_1^n\right\}. \end{aligned}$$

8.3.4 Discretization Laplacian term

The discretization of the Laplacian term is given by:

$$-\sum_{e(1)} \frac{\Delta t}{\rho_e^{n+1}} \bar{l}_e \left\{1 + \gamma M_r^2 p_e^n + \frac{1}{2}(\gamma - 1) M_r^2 (\mathbf{m}_e^*)^2 / \rho_e^{n+1}\right\} \sum_{c(e)} \gamma_c (\delta p)_c. \tag{8.23}$$

The quantities at face ρ_e, p_e and $(\mathbf{m}_e^*)^2/\rho_e$ are obtained using weighted averagings as in (8.19).

Boundary conditions

The prescribed values or, if the required quantities are not given, one-sided differences are used to obtain ρ_e, p_e and $(\mathbf{m}_e^*)^2/\rho_e$. At the faces at which the normal momentum component is given, the coefficients γ_c are put to zero, corresponding to zero correction at these faces. At the faces e at which the pressure is given, we have

$$(\nabla \cdot \delta p \cdot \mathbf{N})_e = \sum_{c(e)} \gamma_c (\delta p)_c = \gamma_e (\delta p)_e + \sum_{c(e) \neq e} \gamma_c (\delta p)_c, \tag{8.24}$$

with $(\delta p)_e = p_e^{n+1} - p_e^n$ the prescribed pressure difference at boundary face e. We assume that $(\delta p)_e = 0$, i.e. the pressure at the outflow boundary does not vary in time.

Chapter 9

Time-discretization

9.1 Introduction

The spatial discretization of the Navier-Stokes equations is discussed in Chapters 4, 5 and 6. This chapter is devoted to the time-integration, for which the θ-method, discussed in Section 9.2 is employed. Explicit time-integration, using an Euler forward scheme, is discussed in Section 9.3. The order in which the conservation laws are solved, is subject of Section 9.4. When considering a stationary problem, time-stepping is utilized to march towards the steady solution. The termination criterion, determining whether a solution is converged sufficiently to the steady state, is treated in Section 9.5.

9.2 The θ-method

In this section an implicit one-step time-integration method is presented, the θ-method. We have the following PDE for a primary Q:

$$\frac{\partial Q}{\partial t} + L_h Q = 0, \tag{9.1}$$

where L_h represents the spatial discretization. Let n be an integer representing the time-level, and let $0 < \theta \leq 1$, we then assume that a linear interpolation of the form

$$Q^{n+\theta} = Q^n + \theta(Q^{n+1} - Q^n) \tag{9.2}$$

suffices to relate Q^n, $Q^{n+\theta}$ and Q^{n+1}. This equation can be rewritten as

$$Q^{n+1} = \frac{1}{\theta} Q^{n+\theta} + (1 - \frac{1}{\theta}) Q^n. \tag{9.3}$$

Hence, knowing Q^n and $Q^{n+\theta}$ is sufficient to find Q^{n+1}. Quantity $Q^{n+\theta}$ is computed from

$$\frac{Q^{n+\theta} - Q^n}{\theta\tau} + L_h^n Q^{n+\theta} = 0. \tag{9.4}$$

(This is equivalent to using Euler backward with a time-step $(\theta\tau)$; it is also implemented this way). Inserting (9.2) in (9.4) leads to the more familiar

$$\frac{Q^{n+1} - Q^n}{\tau} + (1 - \theta) L_h^n Q^n + \theta L_h^n Q^{n+1} = 0, \tag{9.5}$$

which can be evaluated for $0 \leq \theta \leq 1$. For $\theta = 0$ we have Euler forward (explicit method), for $\theta = 1$ we have Euler backward (implicit method), and for $\theta = 1/2$ we have Crank-Nicolson (implicit method). The advantage of using (9.4), (9.3) instead of (9.5) is circumvention of the time-consuming matrix-vector multiplication $L_h^n Q^n$. Note that for $\theta = 0$ we cannot use (9.4), (9.3), and that for $\theta = 1$ equation (9.3) becomes trivial. The explicit method, $\theta = 0$, is discussed in Section 9.3.

The θ-method is employed for primary variables Q_1, $Q_2, \ldots$, which are $\mathbf{m}$, ρ and H in our case. Derived quantities q are functions of the primary variables, $q = f(Q_1, Q_2, \ldots)$, think of for example the velocity and the pressure. The question that then raises, is how to compute q^{n+1}: do we use $q^{n+\theta} = f(Q_1^{n+\theta}, Q_2^{n+\theta}, \ldots)$ and compute q^{n+1} from an interpolation of the form (9.3), or do we first obtain Q_1^{n+1}, $Q_2^{n+1}, \ldots$ and compute q^{n+1} from $q^{n+1} = f(Q_1^{n+1}, Q_2^{n+1}, \ldots)$?
First we will show that these two methods lead to different values for q^{n+1}, and then we will argue which solution procedure to take.
Assume that we have given Q_1^n, Q_2^n, $Q_1^{n+\theta}$ and $Q_2^{n+\theta}$, and that the derived variable q follows from

$$q = Q_1 Q_2. \tag{9.6}$$

The first method gives, with

$$q^{n+\theta} = Q_1^{n+\theta} Q_2^{n+\theta}, \tag{9.7}$$

for q^{n+1}:

$$q^{n+1} = \frac{1}{\theta} Q_1^{n+\theta} Q_2^{n+\theta} + (1 - \frac{1}{\theta}) Q_1^n Q_2^n. \tag{9.8}$$

The second method, with

$$Q_1^{n+1} = \frac{1}{\theta} Q_1^{n+\theta} + (1 - \frac{1}{\theta}) Q_1^n \qquad Q_2^{n+1} = \frac{1}{\theta} Q_2^{n+\theta} + (1 - \frac{1}{\theta}) Q_2^n \tag{9.9}$$

leads to

$$\begin{aligned} q^{n+1} &= Q_1^{n+1} Q_2^{n+1} = \left[\frac{1}{\theta} Q_1^{n+\theta} + (1 - \frac{1}{\theta}) Q_1^n\right] \left[\frac{1}{\theta} Q_2^{n+\theta} + (1 - \frac{1}{\theta}) Q_2^n\right] = \\ &= \frac{1}{\theta^2} Q_1^{n+\theta} Q_2^{n+\theta} + \frac{1}{\theta}(1 - \frac{1}{\theta})(Q_1^{n+\theta} Q_2^n + Q_1^n Q_2^{n+\theta}) + (1 - \frac{1}{\theta})^2 Q_1^n Q_2^n \neq \\ &\neq \frac{1}{\theta} Q_1^{n+\theta} Q_2^{n+\theta} + (1 - \frac{1}{\theta}) Q_1^n Q_2^n \end{aligned}$$

for $\theta \neq 1$.
The second method seems the best, since we want the derived variables to be consistent with the relations $q = f(Q_1, Q_2, \ldots)$ at the time-levels n, $(n+1), \ldots$. We are not interested in the solutions at intermediate stages $(n + \theta)$, but solely in solutions at time-levels n, $(n + 1), \ldots$. Then the second method seems the best, since we want the derived variables to be consistent with the relations $q = f(Q_1, Q_2, \ldots)$ at these time-levels.

9.3 Explicit time-integration

An implicit Euler time-integration of equation (9.1) leads to:

$$D(\mathbf{q}^{n+1} - \mathbf{q}^n) + C\mathbf{q}^{n+1} = \mathbf{r}. \tag{9.10}$$

An explicit Euler time-integration of equation (9.1) leads to:

$$D(\mathbf{q}^{n+1} - \mathbf{q}^n) + C\mathbf{q}^n = \tilde{\mathbf{r}}. \tag{9.11}$$

Vectors $\mathbf{q}^n$ and $\mathbf{q}^{n+1}$ refer to the solution vectors at time-levels n and $n+1$. D is the diagonal contribution resulting from the time-derivative. C is the operator resulting from the convection and possibly viscous term. The right-hand side vectors $\mathbf{r}$ and $\tilde{\mathbf{r}}$ contain the pressure gradient term (in case (9.1) is the momentum equation) and information w.r.t. the boundary conditions. Note that, with the current discretization, the only difference between $\mathbf{r}$ and $\tilde{\mathbf{r}}$ is in the values for the boundary conditions. Equation (9.10) is written as:

$$A^i\mathbf{q}^{n+1} = \mathbf{b}^i, \tag{9.12}$$

where superscript ^{i} refers to the word 'implicit', and

$$A^i = D + C \qquad \mathbf{b}^i = \mathbf{r} + D\mathbf{q}^n. \tag{9.13}$$

Equation (9.11) can be written as:

$$D\mathbf{q}^{n+1} + C\mathbf{q}^n = \tilde{\mathbf{b}}^i \qquad \tilde{\mathbf{b}}^i = \tilde{\mathbf{r}} + D\mathbf{q}^n, \tag{9.14}$$

or, equivalently:

$$\mathbf{q}^{n+1} = A^e\mathbf{q}^n + \mathbf{b}^e, \tag{9.15}$$

where superscript ^{e} refers to the word 'explicit', and

$$A^e = -D^{-1}C \qquad \mathbf{b}^e = D^{-1}\tilde{\mathbf{b}}^i. \tag{9.16}$$

Note that computation of D^{-1} is trivial, since D is a diagonal matrix.
In our software, which is focussed on implicit time-integration, we have at a certain moment matrices A^i, D and vector $\mathbf{b}^i$ at our disposal. Suppose that we want to do explicit time-integration, then we need A^e and $\mathbf{b}^e$. First we make sure that, in the case of explicit time-integration, we compute $\tilde{\mathbf{b}}^i$ instead of $\mathbf{b}^i$. This is easy, since the difference between these vectors lies solely in the boundary conditions. Quantities A^e and $\mathbf{b}^e$ are easily found using:

$$A^e = -D^{-1}C = -D^{-1}(A^i - D) = I - D^{-1}A^i, \tag{9.17}$$

with I the unit matrix. This means for matrix-elements A^e_{ij}:

$$A^e_{ij} = \delta_{ij} - d_{ii}^{-1}A^i_{ij}. \tag{9.18}$$

Furthermore,

$$\mathbf{b}^e = D^{-1}\tilde{\mathbf{b}}^i \tag{9.19}$$

is trivial.

9.4 Solution algorithm

Compressible approach
Having at time-level n all quantities, the quantities at time-level $(n+1)$ are computed in the following order:

1. the normal momentum components m^{n+1}, see Chapter 5;
2. the tangential momentum components $\tilde{m}^{n+1}$, see Section 5.8;
3. the density ρ^{n+1}, see Section 4.6.
4. the energy variable (e.g. H^{n+1}, h^{n+1}, E, ρH), see Section 4.5;
5. the pressure p^{n+1}, see Chapter 6.
6. the normal velocity components u^{n+1}, see Section 5.9.

For items 1, 3 and 4 a linear system has to be solved. For all other items, simple algebraic relations are used.

Incompressible approach
Having at time-level n all quantities, the quantities at time-level $(n+1)$ are computed in the following order:

1. the normal predictor velocity components u^*, see Chapter 5;
2. the new pressure p^{n+1}, see Section 7.3;
3. the new velocity components u^{n+1}, see Section 7.3;
4. the tangential velocity components $\tilde{u}^{n+1}$, see Section 5.8;

For items 1 and 2 a linear system has to be solved. For all other items, simple algebraic relations are used.

Mach uniform approach
Having at time-level n all quantities, the quantities at time-level $(n+1)$ are computed in the following order (see also Chapter 8:

1. the new densityρ^{n+1}, see Section 4.6.
2. the normal predictor momentum components m^*, see Chapter 5;
3. the normal predictor velocity u^*.
4. the new pressure p^{n+1};
5. the corrected momentum and velocity, see Section 7.3;
6. the tangential momentum components $\tilde{m}^{n+1}$, see Section 5.8;
7. the new enthalpy using the equation of state.

For items 1, 2 and 4 a linear system has to be solved. For all other items, simple algebraic relations are used.

9.5 Termination criterion for stationary problems

In order to monitor when the time stepping scheme has converged sufficiently to steady state, a termination criterion has been implemented. We define the 2-norm for a vector $\mathbf{u} = (u_1, \ldots, u_p)$ as follows:

$$||\mathbf{u}||_2 = \sqrt{\frac{\sum_{i=1}^{p} u_i^2}{p}}. \tag{9.20}$$

Let $\mathbf{u}^n$ be the solution vector of quantity u at time-level n with length p, equal to C, the number of cells, or E, the number of faces. We use the following termination criterion:

$$||\mathbf{u}^{n+1} - \mathbf{u}^n||_2 \leq \frac{1-\lambda}{\lambda}(\epsilon_1||\mathbf{u}^{n+1}||_2 + \epsilon_2) \tag{9.21}$$

with

$$\lambda = \frac{||\mathbf{u}^{n+1} - \mathbf{u}^n||_2}{||\mathbf{u}^n - \mathbf{u}^{n-1}||_2},$$

and ϵ_1 and ϵ_2 the relative and absolute accuracy parameters, to be given by the user. When equation (9.21) is satisfied for all three primary variables, the solution is assumed to be converged within the tolerances as given by the user.

Chapter 10

The linear solver

For information the linear solvers, we refer to the ISNaS Mathematical manual.

Chapter 11

Post-processing

The SEPRAN postprocessing routines require the quantities to be located at the vertices. Hence, due to the locations of the variables, interpolations are required. For scalar quantities, positioned at the cell-faces, these interpolations are discussed in 11.1. Section 11.2 is devoted to the interpolation of vector quantities, e.g. the momentum, which are positioned at the cell-faces. The postprocessing of derived variables is discussed in Section 11.3.

11.1 Postprocessing of scalar quantities

The scalar variables (ρ, p and an energy variable) are positioned in the cell-centers, see Figure 11.1. Let V be a vertex surrounded by N cells 1,...,N, and let Q_j be the value of scalar Q at the cell-center of cell j, then Q_V, the value of scalar Q at vertex Vi, is computed using

$$Q_V = \frac{\sum_{j=1}^{N} w_j Q_j}{\sum_{j=1}^{N} w_j}.$$

The weight-coefficients w_j are chosen to be the inverse of the distance between cell j and vertex V.

11.2 Postprocessing of vector quantities

The normal components of the momentum vector are positioned at the midpoints of the faces, see also Figure 11.2. At each of the faces the full momentum vector is reconstructed,

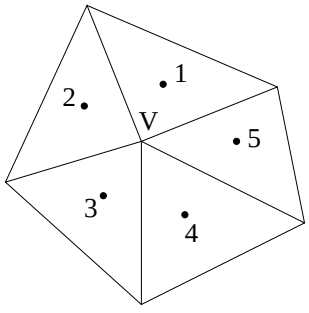

Figure 11.1: Vertex V surrounded by cells 1,...,5.

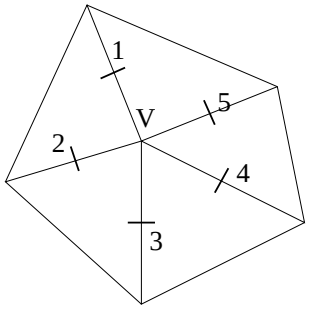

Figure 11.2: Vertex V surrounded by faces 1,...,5.

see Section 5.8. Interpolation to the vertices is then done using the same formulation as given in the previous section:

$$Q_V = \frac{\sum_{j=1}^{N} w_j Q_j}{\sum_{j=1}^{N} w_j}$$

where Q stands for the x- and y-component of the momentum. The weight-coefficients are now taken to be the inverse of the distance between the midpoint of face j and vertex V.

11.3 Postprocessing of derived quantities

Apart from the primary variables, some derived quantities are computed at the vertices as well.
The pressure at each vertex is computed by interpolating the cell-center values of the pressure, even though it is fundamentally more correct to insert the interpolated density and enthalpy in the equation of state.
For the treatment of the primary and derived energy variables, see Section 19.1.
In all vertices the velocity vector is obtained by dividing the interpolated momentum vector by the interpolated density.
The Mach number in the vertices results from the interpolated velocity and enthalpy, together with relation (2.45).

Chapter 12

Flow around profiles

12.1 Introduction

An application for which the TUDFINVOL often is used, is the computation of flows around airfoils. In this chapter some relevant information concerning this type of flows is gathered.

12.2 Initial and boundary conditions

In Figure 12.1 curves for a generic grid around an airfoil are shown. Since the SEPRAN grid-generator `general` is not capable of dealing with grids containing a hole, a slit s is required. The slit is used as a sort of boundary between the ends of a C-grid. The curves i, f and o stand for the inflow, (air)foil and outflow boundary, respectively.
The following boundary conditions are usually given for Euler flow:

- at the inflow curve i: the enthalpy $h = 1$, $|\mathbf{m}| = 1$ and α, the angle of incidence of the flow, usually called angle of attack;
- at the outflow curve o: $\sigma^{nn} = -1$ (i.e. $p = 1$) and $\partial h/\partial \mathbf{n} = 0$;
- at the airfoil f: $m = 0$ and $\partial h/\partial \mathbf{n} = 0$;
- at the slit s no boundary conditions are given.

Note that prescribing $\partial h/\partial \mathbf{n} = 0$, though mathematically not correct, is done to instruct the software that one-sided differences should be used.

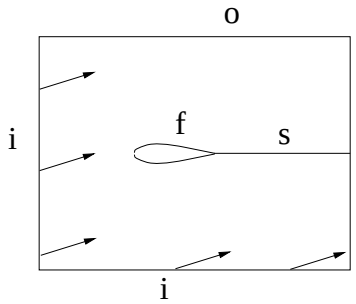

Figure 12.1: Curves in a generic grid for flow computations around an airfoil.

The following initial conditions are usually given for Euler flow: the momentum vector $\mathbf{m} = (m_x, m_y) = (|\mathbf{m}| \cos\alpha, |\mathbf{m}| \sin\alpha)$, the enthalpy $h = 1$ and pressure $p = 1$. Note that, though we use H and ρ as primary thermodynamic variables, we prescribe values for h and p. The reason for this is solely to remain compatible with the ISNaS input files.

12.3 Computation of lift and drag

The computation of the pressure coefficient is discussed in Section 12.3.1. In Section 12.3.2 the lift and drag coefficients are introduced. In Sections 12.3.3, 12.3.4 and 12.3.5 the numerical computation of these coefficients is treated.

12.3.1 Computation of pressure coefficient

The pressure-coefficient at the airfoil is defined as:

$$c_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2}. \tag{12.1}$$

The subscript ∞ stands for the value of the corresponding quantity far away (at infinity) from the airfoil, and p is the computed pressure at the airfoil. In dimensionless quantities, the definition for c_p becomes:

$$c_p = \frac{1}{\gamma M_r^2} \frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2}. \tag{12.2}$$

(In the rest of this section 12.3.1, all quantities are dimensionless). The dynamic pressure $\frac{1}{2}\rho_\infty u_\infty^2$ is computed using

$$\frac{1}{2}\rho_\infty u_\infty^2 = \frac{1}{2}\frac{\mathbf{m}_\infty \cdot \mathbf{m}_\infty}{\rho_\infty} = \frac{1}{2}\frac{|\mathbf{m}_\infty|^2}{\rho_\infty}. \tag{12.3}$$

The length of the momentum $|\mathbf{m}_\infty|$ is given, usually at the inlet, and ρ_∞ follows from the dimensionless equation of state

$$\rho_\infty = \frac{p_\infty}{h_\infty}, \tag{12.4}$$

where h_∞, usually at the inlet, and p_∞, usually at the outlet (or inlet, in supersonic computations), are given.

Given the scaled coordinates of the profile, together with the c_p-values at all vertices, the lift and drag coefficient can be computed, see the subsequent sections.

12.3.2 Definition of lift and drag coefficients

Given a 2D airfoil, see Figure 12.2, defined by the functions

$$y = \begin{cases} y_u(x) & \text{upper surface} \\ y_l(x) & \text{lower surface} \end{cases} \tag{12.5}$$

and $0 \leq x \leq c$, with c the length of the airfoil. Note that the x-axis passes through both the leading and trailing edge, and that the origin is positioned at the leading edge. Vector $\mathbf{R}$ represents the sum of all forces on the airfoil.
The components of force $\mathbf{R}$ in the x- and y-direction are respectively the tangential force T

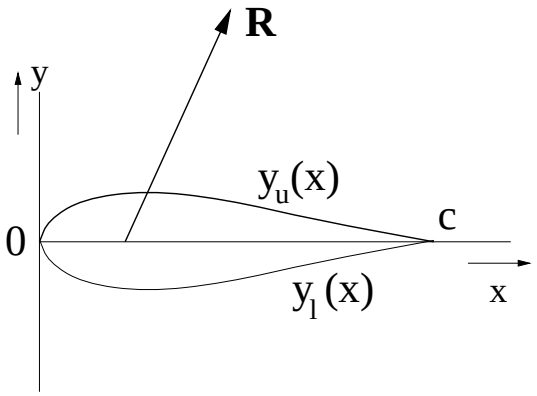

Figure 12.2: Airfoil.

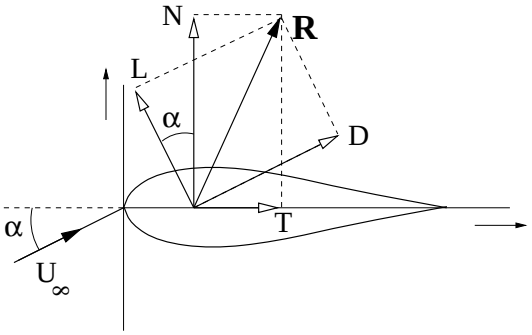

Figure 12.3: Forces on the airfoil.

and the normal force N. With $\mathbf{e}_t$ indicating the unit-vector of the tangential force and $\mathbf{e}_n$ the unit-vector of the normal force ($\mathbf{e}_t = \mathbf{e}_x$ and $\mathbf{e}_n = \mathbf{e}_y$), we write

$$\mathbf{R} = T\,\mathbf{e}_t + N\,\mathbf{e}_n.$$

The angle of attack α is defined as the angle between the chord line, i.e. the x-axis, and the flow direction. Similarly, the components of force $\mathbf{R}$ can be expanded in a component $\mathbf{e}_l$ perpendicular to the flow and a component $\mathbf{e}_d$ parallel to the flow. These components, see Figure 12.3, are respectively called L (lift) and D (drag), and consequently we can write

$$\mathbf{R} = L\,\mathbf{e}_l + D\,\mathbf{e}_d.$$

The relations between components T, N, L and D are

$$\begin{aligned} L &= N\cos\alpha - T\sin\alpha \\ D &= N\sin\alpha + T\cos\alpha. \end{aligned}$$

Division by $\frac{1}{2}\rho_\infty u_\infty^2 c$, with ρ_∞ and u_∞ representing the undisturbed density and velocity, yields relations between the force coefficients:

$$c_l = c_n\cos\alpha - c_t\sin\alpha \tag{12.6}$$

$$c_d = c_n\sin\alpha + c_t\cos\alpha. \tag{12.7}$$

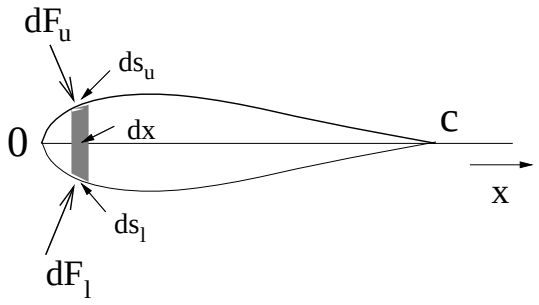

Figure 12.4: Forces dF_u and dF_l acting on element dx of the airfoil.

The next step is to determine c_n and c_t. At a small piece (thickness dx) forces dF_u and dF_l ($dF_u > 0$; $dF_l > 0$) are present, see Figure 12.4. Since we omit viscous effects for the moment, these forces are perpendicular to the profile. Piece dx has at the upper surface a length $(ds)_u$ and a slope with angle γ_u: $\gamma_u = \tan(dy/dx)_u$; at the lower surface this length is $(ds)_l$, with a slope angle γ_l: $\gamma_l = \tan(dy/dx)_l$.
The connection between forces dF_u and dF_l and the normal and tangential forces are given by:

- upper surface of the airfoil:

$$\begin{aligned} dN_u &= -dF_u \cos\gamma_u = -dF_u \left(\frac{dx}{ds}\right)_u \\ dT_u &= dF_u \sin\gamma_u = dF_u \left(\frac{dy}{ds}\right)_u ; \end{aligned}$$

- lower surface of the airfoil:

$$\begin{aligned} dN_l &= dF_l \cos\gamma_l = dF_l \left(\frac{dx}{ds}\right)_l \\ dT_l &= -dF_l \sin\gamma_l = -dF_l \left(\frac{dy}{ds}\right)_l . \end{aligned}$$

With $p_u(x)$ and $p_l(x)$ the pressure distribution at the upper and lower surface of the profile, respectively, the forces dF_u and dF_l follow from

$$\begin{aligned} dF_u &= p_u(x)(ds)_u = p_u(x) \left(\frac{ds}{dx}\right)_u dx \\ dF_l &= p_l(x)(ds)_l = p_l(x) \left(\frac{ds}{dx}\right)_l dx. \end{aligned}$$

The normal and tangential force at piece dx are:

$$\begin{aligned} dN &= dN_u + dN_l = (p_l - p_u)\, dx \\ dT &= dT_u + dT_l = \left(p_u \frac{dy_u}{dx} - p_l \frac{dy_l}{dx}\right) . \end{aligned}$$

Integration over $x \in [0, c]$ and division by $\frac{1}{2}\rho_\infty u_\infty^2 c$ results in

$$c_n = \frac{1}{c}\int_0^c (c_{p_l} - c_{p_u})\, dx \tag{12.8}$$

$$c_t = \frac{1}{c}\int_0^c \left(c_{p_u}\frac{dy_u}{dx} - c_{p_l}\frac{dy_l}{dx}\right), \tag{12.9}$$

with c_{p_u} and c_{p_l} the pressure coefficient at the upper respectively lower surface of the profile. The pressure coefficient is defined by

$$c_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2}.$$

Given (12.5), c_{p_u} and c_{p_l}, one can compute the lift and drag coefficient using relations (12.6), (12.7), (12.8) and (12.9).

12.3.3 Numerical computation of lift and drag coefficients

Given is a set of coordinates $(x_{u,i}, y_{u,i})$, with $i = 1, \ldots, N_u$, where N_u equals the number of vertices at the upper surface of the airfoil. We use the convention that $x_{u,i} < x_{u,i+1}$ for all possible i. Note that $(x_{u,1}, y_{u,1}) = (0, 0)$ and $(x_{u,N_u}, y_{u,N_u}) = (c, 0)$. In addition we have a set $c_{p_{u,i}} = c_{p_u}(x_{u,i})$. Furthermore we have similar sets $(x_{l,i}, y_{l,i})$ and $c_{p_{l,i}} = c_{p_u}(x_{l,i})$, with $i = 1, \ldots, N_l$.

12.3.4 Numerical integration of c_n

Equation (12.8) is first split into two parts:

$$c_n = \frac{1}{c}\int_0^c [c_{p_l}(x) - c_{p_u}(x)]\, dx = \frac{1}{c}\int_0^c c_{p_l}(x)\, dx - \frac{1}{c}\int_0^c c_{p_u}(x)\, dx.$$

Each of these parts is written symbolically as

$$\frac{1}{c}\int_0^c f(x)\, dx. \tag{12.10}$$

As mentioned above, we have given a set x_i and a set $f_i = f(x_i)$, with $i = 1, \ldots, N$ and $x_i < x_{i+1}$, where $x_1 = 0$ and $x_N = 1$.
Numerical integration of (12.10) is done using the trapezoidal rule:

$$\frac{1}{c}\int_0^c f(x)\, dx = \frac{1}{c}\sum_{i=1}^{N-1}\int_{x_i}^{x_{i+1}} f(x)\, dx \approx \frac{1}{c}\sum_{i=1}^{N-1}\frac{1}{2}(f_i + f_{i+1})(x_{i+1} - x_i). \tag{12.11}$$

12.3.5 Numerical integration of c_t

Equation (12.9) is first split into two parts:

$$c_t = \frac{1}{c}\int_0^c c_{p_u}(x)\frac{dy_u(x)}{dx}\, dx - \frac{1}{c}\int_0^c c_{p_l}(x)\frac{dy_l(x)}{dx}\, dx.$$

Each of these parts is written symbolically as

$$\frac{1}{c}\int_0^c f(x)\frac{dy(x)}{dx}\,dx. \tag{12.12}$$

As mentioned above, we have given a set $\{x_i, y_i\}$ and a set $f_i = f(x_i)$, with $i = 1, \ldots, N$ and $x_i < x_{i+1}$, where $x_1 = 0$ and $x_N = c$.
We write (12.12) as

$$\frac{1}{c}\int_0^c f(x)\frac{dy(x)}{dx}\,dx = \frac{1}{c}\sum_{i=1}^{N-1}\int_{x_i}^{x_{i+1}} f(x)\frac{dy(x)}{dx}\,dx. \tag{12.13}$$

Numerical integration of (12.13) is done by using the trapezoidal rule, together with a finite difference approximation for dy/dx:

$$\int_{x_i}^{x_{i+1}} f(x)\frac{dy(x)}{dx}\,dx \approx \frac{1}{2}\left(f_i + f_{i+1}\right)\frac{y_{i+1}-y_i}{x_{i+1}-x_i}\left(x_{i+1}-x_i\right) = \frac{1}{2}\left(f_i + f_{i+1}\right)(y_{i+1}-y_i). \tag{12.14}$$

A somewhat different approach, using partial integration, leads to the same final result:

$$\int_{x_i}^{x_{i+1}} f(x)\frac{dy(x)}{dx}\,dx = \left[f(x)y(x)\right]_{x_i}^{x_{i+1}} - \int_{x_i}^{x_{i+1}} \frac{df(x)}{dx}y(x)\,dx = \frac{1}{2}\left(f_i + f_{i+1}\right)(y_{i+1}-y_i). \tag{12.15}$$

Part II

Programmers Guide

Chapter 13

Data structure: mesh

We restrict ourselves to 2D meshes consisting of triangles. Some extensions to the SEPRAN data structure are made in order to be able to implement the TUDFINVOL code. These extensions, together with some often used SEPRAN data structure arrays, are discussed in this chapter.
These extensions and often-used-arrays are discussed in Section 13.1. To take into account the presence of grid boundaries, some additional arrays and constants must be defined: see Section 13.2. How to do appropriate calls to the grid-related arrays and find the grid-related constants mentioned in this chapter, is summarized in Section 13.3.

13.1 Vertex array, face-based and cell-based data structures

The data structure of the mesh contains mainly connectivity information, i.e. provides the necessary information to connect grid components (cells, faces) to adjacent components. Two fundamentally different mesh related arrays must be constructed: *face-based* for the momentum equation, and *cell-based* for the convection-diffusion equations. In addition to these two data structures, we need to have a *vertex* array containing the coordinates of all vertices. Making use of these, arrays containing the *lengths* of the faces and the *areas* of the cells are created. These parts of the datastructure will be discussed. For more information we refer to the SEPRAN Programmers Guide, Section 24.2.
It must be noted that we use a finite-volume terminology rather than a finite-element terminology: we use the word 'cell' and 'face' where finite-element terminology uses 'element' and 'edge'.

Vertex array: `coor`
The vertex array contains the Cartesian coordinates $\mathbf{v}_i = (v_{ix}, v_{iy})$ of all vertices v_i. The vertex array is a real array `coor(2,npoint)` in the SEPRAN data structure, where `npoint` is the number of vertices, and `npoint` = kmesh(8). The memory management number of array `coor` is `kmesh`(23). Note that V = `npoint`, in equation (3.2).

Cell-based data structure: `cellfv`
The memory management sequence number of array `cellfv` is `kmesh( kmesh(48)-1+32 ) + nelem` in integer array IBUFFR. For more info, see SEPRAN Programmers Guide, Section Array KMESH. In the case of 2D triangles, array `cellfv` is an integer (0:3 $\times$ `nelem`)-array where

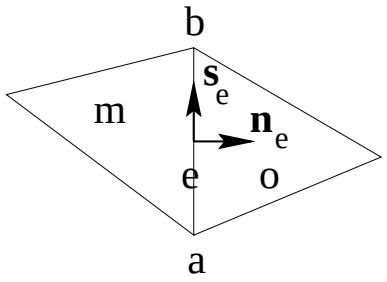

Figure 13.1: Convention with respect to normal and tangential vector. Adjacent cell-numbers are m and o, and vertex-numbers are a and b

the first number (0,*) refers to the number of faces (always 3 in this case), and the other three numbers ((1,*), (2,*) and (3,*)) refer to the face numbers of each cell. `nelem` is the total number of cells (`nelem`=kmesh(9)). The following conventions in storing `cellfv` are adopted:

- the order in storing the face numbers corresponds to a counterclockwise contour.
- the face number is stored as a positive or negative number depending on the direction of the face.

Hence, the face-numbers are the **absolute values** of the elements in array `cellfv`. Note that C = `nelem`, in equations (3.1) and (3.2).

Face-based data structure: `face`
The memory management sequence number of array `face` is kmesh(kmesh(50)-1+21) in array IBUFFR. Array `face` is an integer (6 $\times$ `nfaces`)-array, where `nfaces` stands for the total number of faces, `nfaces` = kmesh(kmesh(48)-1+4), with the following contents:

- `face`(1,e) and `face`(2,e) contain the vertex numbers that define face e.
- `face`(3,e) and `face`(5,e) contain the cell numbers of the cells adjacent to face e.
- `face`(4,e) and `face`(6,e) contain the local face numbers of respectively cells `face`(3,e) and `face`(5,e). The local face number is the column-position (1,2 or 3) in which the face is stored for the corresponding cell in array `cellfv`.

Note that E = `nfaces`, in equation (3.2).
The following conventions are made in storing array `face`:

- `face`(3,e) $<$ `face`(5,e) always.
- the cell with cell number `face`(3,e) must always lie 'left' of the vector defined by (`face`(2,e) - `face`(1,e)), and consequently the cell with cell number `face`(5,e) lies always 'right' of this vector. The normal vector of (`face`(2,e) - `face`(1,e)) that is obtained after rotating this vector by $\pi/2$ radials in the clockwise direction, points towards the cell that lies 'right'. The definition of what is 'left' now is trivial. Note that due to this convention we cannot interchange `face`(1,e) and `face`(2,e).

In Figure 13.1 the normal and tangential vector are drawn for face e. Note that `face`(1,e) = a, `face`(2,e) = b, `face`(3,e) = m and `face`(5,e) = o. Note that $m < o$.

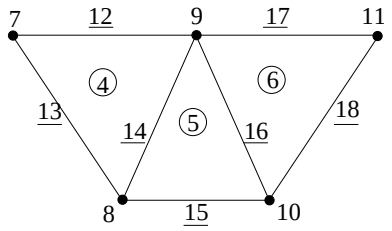

Figure 13.2: Example of a mesh. Cell numbers are in circles, face numbers are underlined, and the other numbers are vertex identification numbers.

In the case that face e is a boundary face, `face`(5,e) is put to (`nelem`+1), and `face`(6,e) is put to zero.

To illustrate the use of arrays `cellfv` and `face`, consider Figure 13.2. Array `cellfv` has the following contents (the face numbers may be permutated in a cyclic manner):

cell 4: `-12  13  14`
cell 5: `-14  15  -16`
cell 6: `18  -17  16`.

Assuming that faces 12, 13, 15, 17 and 18 are boundary faces and that[1] `nelem`=6, array `face` has the following contents:

face 12: `9  7  4  1  7  0`
face 13: `7  8  4  2  7  0`
face 14: `8  9  4  3  5  1`
face 15: `8  10  5  2  7  0`
face 16: `10  9  5  3  6  3`
face 17: `11  9  6  2  7  0`
face 18: `10  11  6  1  7  0`.

Recall that on every face e a unique choice for the direction of the normal $\mathbf{N}_e$ has to be made for the momentum equation, see Section 5.2. Our decision is to let $\mathbf{N}_e$ point from the cell with the smaller identification integer towards the cell with the larger identification integer. Note that due to this convention, the normal on boundary faces is always pointing outwards. Since the direction of translation vector $\mathbf{s}_e$ = `face`(2,e) - `face`(1,e) is related to the identification integers of the cells, the normal vector $\mathbf{N}_e$ can be obtained in a unique manner from $\mathbf{s}_e$ by taking into account the following three conditions:

$$\mathbf{s}_e \cdot \mathbf{n}_e = 0, \qquad |\mathbf{N}_e| = 1 \qquad \text{and} \qquad (\mathbf{s}_e \times \mathbf{N}_e)_z < 0. \tag{13.1}$$

The third condition defines the direction of $\mathbf{N}_e$ with respect to $\mathbf{s}_e$. Writing in Cartesian coordinates

$$\mathbf{s}_e = (s_x, s_y), \tag{13.2}$$

we arrive at

$$\mathbf{N}_e = (s_y, -s_x)/|\mathbf{s}_e|. \tag{13.3}$$

[1] Note that cells 1, 2 and 3 are not drawn. The reason for this, is to avoid confusion between the local face numbers (always 1,2 or 3) and the cell numbers. This example hence serves only to give an illustration.

Note that the direction of the normal vector is obtained by rotating the translation vector $\mathbf{s}_e$ over an angle $\pi/2$ in clockwise direction, see also Figure 13.1.

Array containing lengths of faces: `lengthf`
From the arrays `face` and `coor` we can obtain the lengths of all faces, using $l_e = |\mathbf{s}_e| = \sqrt{s_{e,x}^2 + s_{e,y}^2}$. Array `lengthf`, with length `nfaces`, contains the lenght of all faces: `lengthf(i)` contains the length of face i.

Array containing areas of cells: `areafv`
From the arrays `cellfv`, `face` and `coor` we can obtain the areas of all cells, by using:

$$\Omega = \frac{1}{2}|\mathbf{s}_a \times \mathbf{s}_b| = \frac{1}{2}|\mathbf{s}_a \times \mathbf{s}_c| = \frac{1}{2}|\mathbf{s}_b \times \mathbf{s}_c|, \tag{13.4}$$

where $\mathbf{s}_a$, $\mathbf{s}_b$ and $\mathbf{s}_c$ are the three translation vectors of the cell faces a, b and c.

In addition, the following quantities can be obtained. Note that these quantities are not stored, hence they have to be computed each time when they are needed.

1. the midpoint of a face follows from $\mathbf{e} = (\mathbf{v}_a + \mathbf{v}_b)/2$, where $\mathbf{v}_a$ and $\mathbf{v}_b$ are the coordinates of the vertices of the face.

2. the uniquely defined normal vector $\mathbf{N}_e$ of face e; see discussion above.

3. the inner product $(\mathbf{n}_e \cdot \mathbf{N}_e)$, where $\mathbf{n}_e$ is the normal pointing outwards of the cell under consideration, and $\mathbf{N}_e$ the uniquely defined normal vector. Say that m stands for the cell under consideration, and n is the cell that lies adjacent to face e. If the cell number of $m < n$, then $(\mathbf{n}_e \cdot \mathbf{N}_e) = 1$, else $(\mathbf{n}_e \cdot \mathbf{N}_e) = -1$. For example, if we consider cell 5 in Figure 13.2, then $(\mathbf{n}_{14} \cdot \mathbf{N}_{14}) = -1$ and $(\mathbf{n}_{16} \cdot \mathbf{N}_{16}) = 1$.

4. the coordinates of the cell-centroid $\mathbf{c}$ from

$$\mathbf{c} = \frac{1}{3}(\mathbf{v}_i + \mathbf{v}_j + \mathbf{v}_k), \tag{13.5}$$

   where $\mathbf{v}_i$, $\mathbf{v}_j$ and $\mathbf{v}_k$ are the coordinates of the three cell vertices.

13.2 Boundary treatment in the data structure

Now we are going to discuss the influence of the presence of boundaries in the data structure. In Sections 13.2.1 and 13.2.2 the cells and faces are divided into subgroups: internal cells and boundary cells, and internal faces, quasi internal faces and boundary faces. Additional reordering arrays are created, and discussed.

13.2.1 Cells and boundaries

As discussed in Section 3.2, we distinguish between internal and boundary cells.
In the software, with `nelem` = kmesh(9) we intend the total number of cells C and with `nelemi` = kmesh(kmesh(50)-1+4) we intend the total number of internal cells C_i.
In integer array `iconcell(nelem)` we have rearranged the cell numbers such, that the first `nelemi` elements refer to internal cells, and the next (`nelemi` + 1) to `nelem` elements to boundary cells. The memory management sequence number of array `iconcell` is kmesh(kmesh(50)-1+22).

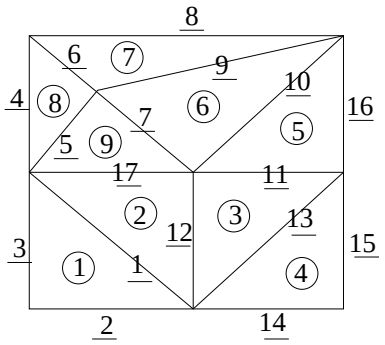

Figure 13.3: Example of a mesh in the vicinity of boundaries. Cell numbers are in circles and face numbers are underlined.

13.2.2 Faces and boundaries

As discussed in Section 3.2, we distinguish between internal and boundary faces. Furthermore, the internal faces are subdivided in real internal faces and quasi internal faces.
In the software, with `nfaces` = kmesh(kmesh(48)-1+4) we intend the total number of faces E, `nfacesi` = kmesh(kmesh(50)-1+2) stands for the number of internal faces E_i, and `nfacesir` = kmesh(kmesh(50)-1+3) represents the number of real internal faces E_{ir}. The number of boundary faces: $E_b = (E - E_i)$. The number of quasi internal faces: $E_q = (E_i - E_{ir})$.
In integer array `iconface(nfaces)` we have rearranged the face numbers such, that the first E_b elements of this array correspond to boundary faces, the next $(E_b + 1)$ to $(E_b + E_q)$ elements to quasi internal faces, and the final (E_b+E_q+1) to E elements to real internal faces.
In addition, we have stored the array `convarray(nfaces)`, which is the inverse of `iconface(nfaces)`. This array is stored immediately after array `iconface`. This means that, when `i` = `iconface(j)`, then `j` = `convarray(i)`. The use of this is the following: when doing a loop over all boundary faces, we do a loop over the elements `j` = 1 to E_b in array `iconface`. Storing relevant info then is done in some array, call it `boundary_info`, with a length equal to E_b. When we know the face number `i` of a face, we use array `convarray` to find the corresponding position in array `iconface`, i.e. the position in array `boundary_info`.

To give an example of the use of arrays `iconcell`, `iconface` and `convarray`, consider Figure 13.3. The internal cells are 2, 3, 6 and 9. Consequently, the boundary cells are 1, 4, 5, 7 and 8. The internal faces are 1, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 17, of which the faces 7, 12 and 17 are real internal faces. Consequently, the quasi internal faces are 1, 5, 6, 8, 9, 10, 11 and 13. Consequently, the boundary faces are 2, 3, 4, 8, 14, 15 and 16. This leads to the following mesh-related constants: $C = 9$; $C_i = 4$; $C_b = 5$; $E = 17$; $E_i = 10$, $E_{ir} = 3$, $E_b = 7$; $V = 9$. The relations 3.1 and 3.2 ($H = 0$) are satisfied.
The array `iconcell` is given by:

```
%    2 3 6 9    %    1 4 5 7 8    %
```

(where the parts between % may be permutated).
The array `iconface` is given by:

```
%   2 3 4 8 14 15 16    %    7 12 17    %    1 5 6 9 10 11 13    %
```

(where the parts between % may be permutated).
The array `convarray` is given by:

```
11 1 2 3 12 13 8 4 14 15 16 9 17 5 6 7 10.
```

13.3 Software implementation of the data structure

To obtain an unstructured mesh, the appropriate SEPRAN routines are used or adapted (if necessary). In order to obtain the mesh related quantities as described above, the author has written some new procedures (note that generation of array `cellfv` is not implemented by the author).

- array `face` is created by routine `mshface.f`.
- arrays `lengthf` and `areafv` are created by routine `mshgeom.f`.
- arrays `iconcell` and `iconface`, together with the numbers `nelemi`, `nfacesi` and `nfacesir`, are created by routine `mshrenum.f`.

The way to find the starting-addresses (`ip***`) of these arrays, is to use the procedure `iniactmk.f`, see Section 22.9.1 in the SEPRAN Guide called *Subroutine INIACTMK*. The number indicated with `mm***` are memory management sequence numbers. The dummy-parameter, required in the call to `iniactmk`, must be declared as a one-dimensional integer array, i.e. `dummy(1)`.

Some important numerical values (note that it is trivial to compute other numerical values like the number of boundary faces or the number of quasi internal faces):

```
ndim = kmesh(6).
npoint = kmesh(8).
nelem = kmesh(9).
ncurvs = kmesh(11).
nfaces = kmesh( kmesh(48)-1+4 ).
nfacesi = kmesh( kmesh(50)-1+2 ).
nfacesir = kmesh( kmesh(50)-1+3 )
nelemi = kmesh( kmesh(50)-1+4 ).
```

- Find array `coor`:
  ```
  call iniactmk ( ibuffr, 1, 23, kmesh, dummy, mmcoor )
  ipcoor = inidgt( mmcoor ),
  ```
  and give `buffer(ipcoor)` in the `call`-statement. Length of this array: 2 x `npoint`.
- Find array `face`:
  ```
  call iniactmk ( ibuffr, 4, 21, kmesh, dummy, mmface )
  ipface = iniget( mmface ) ,
  ```
  and give `ibuffr( ipface )` in the `call`-statement. Length of this array: 6 x `nfaces`.
- Find array `iconcell`:
  ```
  call iniactmk ( ibuffr, 4, 22, kmesh, dummy, mmconcel )
  ipconcel = iniget( mmconcel ) ,
  ```
  and give `ibuffr( ipconcel )` in the `call`-statement. Length of this array: `nelem`.
- Find array `iconface`:
  ```
  call iniactmk ( ibuffr, 4, 23, kmesh, dummy, mmconfac )
  ipconfac = iniget( mmconfac ) ,
  ```
  and give `ibuffr( ipconfac )` in the `call`-statement. Length of this array: `nfaces`.

- Find array convarray:
  ```
  call iniactmk ( ibuffr, 4, 23, kmesh, dummy, mmconfac )
  ipconfac = iniget( mmconfac ) ,
  ipconv = ipconfac + nfaces,
  ```
  and give `ibuffr( ipconv )` in the `call`-statement. Length of this array: `nfaces`. Note that this array is the inverse of array `iconface`, and that it is stored immediately next to (behind) `iconface`.

- Find array `lengthf`:
  ```
  call iniactmk ( ibuffr, 4, 24, kmesh, dummy, mmlengthf )
  iplengthf = inidgt( mmlengthf ) ,
  ```
  and give `buffer( iplengthf )` in the `call`-statement. Length of this array: 6 x `nfaces`.

- Find array `areafv`: `call iniactmk ( ibuffr, 4, 25, kmesh, dummy, mmareafv )`
  `ipareafv = inidgt( mmareafv ) ,`
  and give `buffer( ipareafv )` in the `call`-statement. Length of this array: `nelem`.

- For the `cellfv`-array, the following actions have to be performed:
  ```
  call iniactmk ( ibuffr, 2, 32, kmesh, dummy, mmcellfv )
  ipcellfv = iniget( mmcellfv ) ,
  ```
  and give `ibuffr( ipcellfv + nelem )` in the `call`-statement. In the routine itself, array `cellfv` must be declared as:
  ```
  cellfv(0:3,nelem),
  ```
  so that the elements `cellfv(1,*)`, `cellfv(2,*)` and `cellfv(3,*)` correspond to the faces. The number `cellfv(0,*)` correspond to the number of faces of the element under consideration, hence is always equal to 3 when dealing with triangles. Note that the mentioned procedure works only for meshes consisting *solely* out of triangles.
  The reason to do this so complicatedly, is that array `cellfv` is actually KMESH part aa, NMESH part b; see SEPRAN Programmers Guide. Recall that we consider only 2D triangles, and that we use a finite volume terminology rather than a finite element terminology (see the beginning of Section 13.1).

Chapter 14

Datastructures

In Section 14.1 the reference numbers to each unknown is given. Furthermore, some definitions concerning the total number of unknowns is given. The storage of the matrix, right-hand side, solution and matrix structure is discussed in Section 14.2. Array KFINVOL, containing very general information about array IINPUT and gradient arrays, is discussed in Section 14.3. The rest of the chapter is devoted to the description of arrays containing boundary-condition information and other parameters.

14.1 Conventions

At this moment we can only deal with coupled momentum equations (`icoupled` = 0; `icoupled` is at position 76 in array IINPUT, see ISNaS Programmers Guide Section 16.9). Concerning the type of flows, we can deal with fully compressible flows (`mcom` = 3; `mcom` is at position 1 in array IINCOM, see ISNaS Programmers Guide Section 16.9), incompressible flows (`mcom` = 0) and Mach-uniform flows (`mcom` = 1). With parameter `eq` we refer to an unknown. The following ordening of the unknowns/solutions is adopted:

- `eq` = 1,..., `ndim`: in the coupled case the momentum component m normal to the faces is stored at `eq` = 1 and the numbers `eq` = 2, ..., `ndim` refer to tangential momentum components. In the 2D-case (`ndim` = 2), `eq` = 2 refers to $\tilde{m}$. `ndim` is the number of dimensions (stored at position 57 in IINPUT).
- `eq` = (`ndim`+1): pressure.
- `eq` = (`ndim`+2), ..., (`ndim`+1+`ntrnsp`): transport variables. Position `eq` = (`ndim`+2) is reserved for the energy variable (e.g., enthalpy, total enthalpy, total energy, ...). `ntrnsp` is the number of transport variables (stored at position 63 in IINPUT).
- `eq` = (`ndim`+2+`ntrnsp`), ..., (`ndim`+1+`ntrnsp`+`nturb`): turbulence quantities. `nturb` is the number of turbulence equations (stored at position 70 in IINPUT).
- `eq` = (`ndim`+2+`ntrnsp`+`nturb`), ..., (`ndim`+1+`ntrnsp`+`nturb`+`nvcextra`): derived quantities. Derived quantities are quantities that can be computed by simple relations from the primary variables. Storing derived quantities hence is not strictly necessary, but for ease of programming they are stored nevertheless. At this moment we have `nvcextra` = 2 for compressible flows and `nvcextra` = 0 for incompressible flows. In the case of compressible flows, the first position (i.e. `eq` = (`ndim`+1+ `ntrnsp`+`nturb`+1)) refers to the density, and the second position to the velocity.

Furthermore, in the code we use sometimes the following variables:

- Number of transport variables: `ntrans` = (`ntrnsp` + `nturb`);
- Number of unknowns: `numunk` = (`ndim`+1+`ntrans`);
- Number of degrees of freedom: `ndegfd` = `numunk`.
- Number of solutions: `numsol` = (`ndim`+1+`ntrans`+ `nvcextra`).

At this moment, the compressible (i.e. `mcom` = 0,1) part of the code is suited for `ndim` = 2, `ntrnsp` = 1 (energy variable), `nturb` = 0 and `nvcextra` = 2. This leads to the following ordening:

- `eq` = 1: normal momentum component m,
- `eq` = 2: tangential momentum component $\tilde{m}$,
- `eq` = 3: pressure p,
- `eq` = 4: energy variable (e.g. h, H, E, ...), (when `mcom` = 1, then this has to be h).

- `eq` = 5: density ρ,
- `eq` = 6: normal velocity component u,

and the following constants: `ndim` = 2, `ntrnsp` = 1, `nturb` = 0, `nvcextra` = 2, `ntrans` = 1, `numunk` = `ndegfd` = 4 and `numsol` = 6.

The incompressible part of the code is suited for `ndim` = 2, `ntrnsp` = 0 (no energy variable), `nturb` = 0 and `nvcextra` = 0. This leads to the following ordening:

- `eq` = 1: normal velocity component u,
- `eq` = 2: tangential velocity component $\tilde{u}$,
- `eq` = 3: pressure p,

14.2 Matrix and solution arrays

14.2.1 Array isol

Information concerning the solution arrays is stored in integer array `isol(5,numsol,ntimlv+1)`, where `numsol` is defined in Section 14.1, and `ntimlv` is the number of time-levels for which the solution remains stored. `ntimlv` may never be smaller than 2; at this moment we have `ntimlv` = 2. Array `isol` is initialized in routine `fvincnd.f`.

- `isol(1,i,j)` refers to memory management number of equation number `eq` = `i` (see Section 14.1) at a certain time-level. `j` = `ntimlv` - 1 = 1 refers to the 'old' solution, and `j` = `ntimlv` = 2 refers to the 'new' (latest obtained) solution. `j` = `ntimlv` + 1 = 3 is used for the post-processing.
- `isol(2,i,j)` = 127 for the normal and tangential momentum (`i` = 1,..., `ndim`, and all `j`), and `isol(2,i,j)` = 116 for all other (scalar) quantities (`i` = `ndim` + 1,...,`numsol`, and all `j`)
- `isol(3,i,j)` = 0 for all `i` and `j`.
- `isol(4,i,j)` = 0 for the normal and tangential momentum (`i` = 1,..., `ndim`, and all `j`), and `isol(4,i,j)` = 1001 for all other quantities (`i` = `ndim` + 1,...,`numsol`, and all `j`)
- `isol(5,i,j)` equals `nusol`, the length of the stored solution vector. Hence, if `i` refers to a momentum or velocity solution, `nusol` = `nfaces`. If `i` refers to a transport or turbulence solution, then `nusol` = `nelem`.

For a more complete description of `isol`, see SEPRAN Programmer's Guide Section 24.4. For the actual content of the arrays to which the memory management numbers `isol(1,i,j)` refers, see Section 17.3.

14.2.2 Arrays matrix and irhsd

The integer arrays `matrix(5)` and `irhsd(5)`, refering to the matrix resp. right-hand side, are descriped in the SEPRAN Programmer's Guide as well (the sections on Array MATR and Array IRHSD, resp. 24.7 and 24.5). The matrix is stored using JMETOD = 6. In short, the arrays have the following content:

- `matrix(1)` = 806 (momentum equation) or 1806 (scalar equation).
- `matrix(2)` = 103.
- `matrix(3)`: memory management number refering to actual matrix.
- `matrix(4)` = 0.
- `matrix(5)` = 0.
- `irhsd(1)`: memory management number refering to actual right-hand side
- `irhsd(2)` = 116 (scalar equation) or 127 (momentum equation).

- `irhsd(3)` = 0.
- `irhsd(4)` = 1.
- `irhsd(5)`: length of right-hand side vector (i.e. `nfaces` for momentum equation, and `nelem` for scalar equation).

In the computer code, these arrays, as well as the large arrays containing the matrix and the right-hand side, are created at every time-level and for every equation in routine `fvbldmat.f`, and after having solved the matrix equation, these arrays are deleted in routine `fvsbstep.f`, see also Section 17.2.

14.2.3 Array intmat

Information concerning the structure of the matrices is stored in integer array `intmat(5,numunk)`. For definition of `numunk`, see Section 14.1. Storage scheme JMETOD = 6 is used. Short description of content (see also SEPRAN Programmers Guide Section 24.6):

- `intmat(1,i)` = 806 (momentum equation) or 1806 (scalar equation).
- `intmat(2,i)` = 102
- `intmat(3,i)` = memory management number refering to actual matrix structure.
- `intmat(4,i)` = 0
- `intmat(5,i)`: length of vector in which matrix itself is stored. For the momentum matrix this length is called `lmstrmom` and for the scalar matrix this length is called `lmstrcvdf`.

At this moment we can account for only two matrix-structures: one for the momentum, and one for the scalar variables. Hence, for the density, enthalpy and all other scalar variables we use the same matrix structure as for the pressure correction (hence `intmat(i,ndim+1)` = `intmat(i,ndim+2)` = `intmat(i,ndim+...)` for all `i`). Matrix `intmat(5,numunk)` is created in routine `fvstrmat.f`.
(It must be noted that, when building a matrix for the density, having equation-number (`numunk` + 1), we don't have any position left in `intmat`. But, since `intmat` is the same for all scalar quantities, we will use the enthalpy-positions in `intmat` instead.)

14.3 Array KFINVOL

Array KFINVOL contains general information that is required to build the matrices. The positions in KFINVOL have the following meaning:

Pos. 1 Declared length of array KFINVOL (to be filled in by user).

Pos. 2 1000. This number is used to indicate that this is an array of structure KFINVOL.

Pos. 3 Actual length of array KFINVOL.

Pos. 4 0.

Pos. 5 0.

Pos. 6 mmiinput. Memory management sequence number of integer array IINPUT.

Pos. 7 mmrinput. Memory management sequence number of real array RINPUT.

Pos. 8 mmcoefs. Memory management sequence number of integer array IINCOFS; see Section 14.4.

Pos. 9 mmigrfac. Memory management sequence number of integer array IGRFAC, containing for each face the 6-points stencil required to evaluate gradients with e.g. the path-integral method; see Chapter 21.

Pos. 10 mmrgrfac. Memory management sequence number of real array RGRFAC, containing the weights corresponding to the elements in IGRFAC; see Chapter 21.

Pos. 11 mmibndcon. Memory management sequence number of integer array IBNDCON containing integer information with respect to the boundary conditions, see Section 14.5.3.

Pos. 12 mmrbndcon. Memory management sequence number of real array RBNDCON containing real information with respect to the boundary conditions, see Section 14.5.4.

Pos. 13 mmlinpol. Memory management sequence number of real array LINPOL, containing linear interpolation coefficients required to compute quantities at faces when the quantities are located at the cell-centres (e.g. density); see Section 16.4.

Pos. 14 mmitancomp. Memory management sequence number of integer array ITANCOMP, containing the four surrounding faces of the face under consideration; see Section 16.5.

Pos. 15 mmrtancomp. Memory management sequence number of real array RTANCOMP, containing the weights in order to compute the tangential component; see Section 16.5.

Pos. 16 mmgradvel. Memory management sequence number of real array GRADVEL, containing the weights in order to compute the velocity gradient, see Section 16.6.

Pos. 17 mmreconstnor. Memory management sequence number of real array RECONSTNOR, containing the reconstruction coefficients for a normal vector in terms of surrounding normals. See Section 16.7.

Pos. 18 mmreconsttan. Memory management sequence number of real array RECONSTTAN, containing the reconstruction coefficients for a tangential vector in terms of surrounding normals. See Section 16.7.

Pos. 19 mmnormal. Memory management sequence number of real array NORMAL, containing the normal vector at each face. See Section 13.1.

14.4 Coefficient arrays

Information with respect to the coefficients, see ISNaS Programmers Guide Section 16.10, is stored in arrays `iincofs(ncoefs,ndegfd)` and `rincofs(ncoefs,ndegfd)`. For every equation, at every time-level, a real array `coefs(ncoefs,nelem)` is filled. This filling is done in routine `fvcoefs.f`. Array `coefs` contains information with respect to coefficients (may be functions of time and/or space) for each cell. Then routine `fvscalarfaces.f` uses those values to interpolate the viscosity in face centers, which is needed for the momentum equation. It also calculates the density in face centers, but this is calculated from the array `dens` because the density is not the coefficient but a calculated value. Those values are stored in array `facecoefs`.

For the `tudfinvol`-code we have added some additional terms in array `coefs` for the convection-diffusion equation. To this aim we had to increase integer `ncoefs` from 10 to 11 (in `fvreadal.f` and `fvinput.f`). Let the convection-diffusion equation be of the following form:

$$\frac{\partial a\phi}{\partial t} + \nabla \cdot (b\mathbf{u}\phi) - \nabla \cdot (c\nabla\phi) + d\phi = g, \tag{14.1}$$

where a, b, c, d and g are the coefficients. For discretization of this equation, see Section 4.2. The positions `j` and `i`, with `j` running from 1 to 11, and `i` from 1 to `nelem`, in array `coefs(j,i)`, contain:

convection diffusion equation

- `i` $= 1$: capacity at new time-level a_i^{n+1}
- `i` $= 2$: constant d_i^{n+1}
- `i` $= 3$: source f_i^{n+1}
- `i` $= 4$: scalar diffusion c_i^{n+1}
- `i` $= 5$: scalar diff$_x$ y
- `i` $= 6$: scalar diff$_y$ y
- `i` $= 7$: scalar diff$_x$ z
- `i` $= 8$: scalar diff$_y$ z
- `i` $= 9$: scalar diff$_z$ z
- `i` $= 10$: convection term b_i^{n+1}
- `i` $= 11$: capacity at old time-level a_i^n

momentum equation

- `i` $= 1$: density
- `i` $= 2$: viscosity

Array `facecoefs(j,i)` contains the density for `i` = 1 and viscosity for `i` = 2.

The energy equation, see Section 4.5, and the continuity equation, see Section 4.6 are considered as special forms of the convection-diffusion equation. Note that the source term f is an external source and *not* the term $\Omega_i g_i$ in relation (4.2).

14.5 Arrays with respect to the boundary conditions

With respect to boundary conditions, we distinguish between four types of arrays:

- **iinbc** Array containing integer information of the boundary conditions.
- **rinbc** Array containing real information of the boundary conditions.
- **ibndcon** Array containing integer information of the boundary conditions, for every boundary face, for the equation under consideration.
- **rbndcon** Array containing real information of the boundary conditions, for every boundary face, for the equation under consideration.

Arrays `iinbc` and `rinbc` are stored in arrays `iinput` and `rinput`, see ISNaS Programmers Guide Section 16.11. Note that the content of arrays `iinbc` and `rinbc` differs from the content in ISNaS.
At every time-level, for every equation, arrays `ibndcon` and `rbndcon` are filled again. These arrays contain, respectively, integer and real information concerning the type and numerical value(s) of the boundary conditions at each face, for a certain equation at a certain time-level. Filling these arrays is done in routine `fvbndcon.f`. Arrays `ibndcon` and `rbndcon` are used in the computation of the matrix elements. The contents of the arrays `iinbc`, `rinbc`, `ibndcon` and `rbndcon` is discussed subsequently.

14.5.1 Array IINBC

Array `iinbc` is an integer array of dimension (`ncurvs` x `ndegfd` x 3), where `ncurvs` is the number of curves used to define the mesh (in mesh-input-file), and `ndegfd` is the number of degrees of freedom.
Element `iinbc(icurnr,j,k)` refers to curve-number `icurnr` and the `j`th degree of freedom (see description of parameter `eq` in Section 14.1). Parameter `k` runs from 1 to 3:

`k` = 1: gives type of boundary condition, see Section 16.11 of the ISNaS programmers guide. For example, if `iinbc(icurnr,1,1)` = 4, then at curve `icurnr` we have the normal and tangential momentum given for the momentum equation. For example, if `iinbc(icurnr,ndim+2,1)` = 1, then at curve-number `icurnr` we have a Dirichlet condition for the enthalpy. When `iinbc(icurnr,j,1)` = 0, then no boundary condition for variable `j` is given. This occurs for example at inner curves.

`k` = 2: refers to the value of the boundary condition (see ISNaS Programmers Guide Section 16.11.3). If `iinbc(icurnr,j,2)` = -1, then the boundary condition is constant over the curve and in time. The value of the boundary condition is then stored in `rinbc(icurnr,j,1)`. If `iinbc(icurnr,j,2)` > 0, then the boundary condition is a function of the position and/of time. The value of the boundary condition must then be computed.

`k` = 3: refers to another value of the boundary condition (see ISNaS Programmers Guide Section 16.11.3). This value is needed for example in case of a Robbins type of boundary condition. If `iinbc(icurnr,j,3)` = -1, then the boundary condition is constant over the curve and in time. The value of the boundary condition is then stored in `rinbc(icurnr,j,2)`. If `iinbc(icurnr,j,3)` > 0, then the boundary condition is a function of the position and/of time. The value of the boundary condition must then be computed.

14.5.2 Array RINBC

Array `rinbc` is a real array of dimension (`ncurvs` x `ndegfd` x 2), where `ncurvs` is the number of curves used to define the mesh (in mesh-input-file), and `ndegfd` is the number of degrees of freedom.
Array `rinbc` is used to store real constants for the computation of boundary conditions for each curve and for each equation. The contents are defined by array `iinbc`, see previous section.
For example, let **m** be given at curve `icurnr` (i.e. `iinbc(icurnr,1,1)` = 4), then we have: m_n = `rinbc(icurnr,1,1)`; $\tilde{m}$ = `rinbc(icurnr,2,1)`.

14.5.3 Array IBNDCON

Array `ibndcon` is an integer array with length `nfacesb`, where `nfacesb` stands for the number of boundary faces (hence, `nfacesb` = `nfaces` - `nfacesi`, with `nfaces` and `nfacesi` the total number of faces and the number of internal faces, respectively).
The `(ibouface)`th element of array `ibndcon`, i.e. `ibndcon(ibouface)`, refers to the (boundary) face with number `iface` in array `face` (see KMESH part y). Note that $1 \leq$ `ibouface` $\leq$ `nfacesb`. The relations between the integers `ibouface` and `iface` are:

```
iface = iconface(ibouface);
ibouface = convarray(iface),
```

with `convarray` the inverse of array `iconface`, a part of KMESH part y, see also Section 13.2.2.
Array `ibndcon` contains for each boundary face an integer indicating the type of boundary condition. For every face `ibouface` lying on curve `icurnr`, we have `ibndcon(ibouface)` = `iinbc(icurnr,eq,1)`, for equation `eq`. For a description, see Section 16.11.3 in the ISNaS Programmers Guide. In short, the following types of boundary conditions are implemented, together with their corresponding integer numbers in `ibndcon`:

Momentum equation

4 Momentum vector $\mathbf{m} = (\tilde{m}, m)$ is given (Dirichlet).

5 Normal momentum m and tangential stress σ^{tt} given.

6 Normal stress σ^{nn} and tangential momentum $\tilde{m}$ given.

7 Stress ($\sigma^{nn} = -p$ and σ^{tt}) given (Neumann).

10 Cartesian momentum components $\mathbf{m} = (m_x, m_y)$ given (Dirichlet).

11 Length of momentum vector $|\mathbf{m}|$ and angle of inflow α given (Dirichlet). The angle of inflow is the angle between the positive x-axis and the direction of the flow, being positive in the counterclockwise direction. For profile flow, the angle of inflow is usually called angle of attack.

14 Momentum vector $\mathbf{m} = (\tilde{m}, m)$ and $\sigma^{nn} = -p$ are given (supersonic inflow).

15 Cartesian momentum components $\mathbf{m} = (m_x, m_y)$ and $\sigma^{nn} = -p$ are given (supersonic inflow).

16 Length of momentum vector $|\mathbf{m}|$, angle of inflow α and $\sigma^{nn} = -p$ are given (supersonic inflow).

17 Nothing is given (supersonic outflow).

Note that items 10 and 11 are equivalent to item 4, and that items 15 and 16 are equivalent to item 14. In routine `fvbndc01.f` boundary conditions of type 10 and 11 are transformed to type 4, and boundary conditions of type 15 and 16 are transformed to type 14.

Scalar equation

1 Scalar is given (Dirichlet).

2 Robbins boundary condition ($a\phi + (k\nabla\phi) \cdot \mathbf{n} = f$) for scalar. At this moment only homogeneous Neumann implemented ($(k\nabla\phi) \cdot \mathbf{n} = 0$, or, equivalently, $a = f = 0$).

For the continuity equation, with density as primary variable, no explicit boundary conditions are given. For the discretization, see also Section 4.6.1, a homogeneous Neumann boundary condition at all boundary faces (`ibndcon` $= 2$, for all positions) is when the momentum vector is not given. If the momentum vector is given, a Dirichlet condition for the density is given ($\rho = m/u$, where the normal momentum component m follows from the given Dirichlet condition for the momentum vector, and the normal velocity u is the most recently computed value).
When the enthalpy h is *not* the primary variable (`menergy` $\neq 1$), then at the boundaries a Dirichlet condition holds when both the enthalpy and the momentum vector are given, see also the end of Section 14.5.4.

Pressure-correction equation
No boundary conditions are required for the incompressible pressure-correction equation.
For the compressible pressure-correction, this is not the case, see also the discussion in Chapter 8. The following contents for the corresponding arrays is implemented: From `iinbc(1,ndim+2,1)` we can see whether a Dirichlet or homogeneous boundary condition for the enthalpy is given (values for `iinbc(1,ndim+2,1)` are 1 and 2 respectively). From `iinbc(1,1,1)` we can derive whether the pressure (actually σ^{nn} is given at the boundary or not. The values for `ibndcon` are given here below:

1 Dirichlet condition for the enthalpy, no condition for the pressure given. We have: `rbndcon(1,ibouface,1)` $= h$, with h the given enthalpy.

2 Neumann condition for enthalpy and pressure.

3 Both pressure and enthalpy given, and stored as follows: `rbndcon(1,ibouface,1)` $= h$ and `rbndcon(2,ibouface,1)` $= \sigma^{nn}$.

4 Pressure given and Neumann for enthalpy. The given pressure is stored in: `rbndcon(2,ibouface,1)` $= \sigma^{nn}$.

14.5.4 Array RBNDCON

Array rbndcon is a real array of dimension (2 x nfacesb x ndim), where nfacesb is the number of boundary faces, and ndim is the dimension of the problem. The (ibouface)th element of array rbndcon, i.e. rbndcon(i,ibouface,j), with i = 1,2, and j = 1,...,ndim, refers to the (boundary) face with number iface in array face (see KMESH part y). Note that $1 \leq$ ibouface $\leq$ nfacesb. The relations between the integers ibouface and iface are:

iface = iconface(ibouface);

ibouface = convarray(iface),

with convarray the inverse of array iconface, a part of KMESH part y, see also Section 13.2.2.
Array rbndcon contains for every boundary face the value(s) of the boundary conditions.

For all equations, except for the momentum equation, only the elements rbndcon(i,ibouface,1), with i = 1,2, are filled. For every face ibouface lying on curve icurnr, we have, in case of constant boundary conditions, rbndcon(i,ibouface,1) = rinbc(icurnr,eq,i) (with i = 1,2), for all eq except the momentum equation. In case that the boundary conditions depend on position and/or time, the values of rbndcon(i,ibouface,1) are computed.

For the momentum equation, all elements of rbndcon are used. For every face ibouface lying on curve icurnr, we have, in case of constant boundary conditions, rbndcon(i,ibouface,j) = rinbc(icurnr,j,i), for i = 1,2, and j = 1,...,ndim. In case that the boundary conditions depend on position and/or time, the values of rbndcon(i,ibouface,j) are computed.
For example, assume that at boundary-face ibouface, lying at curve icurnr, both the normal momentum m and tangential momentum $\tilde{m}$ are given and are constant. Furthermore, assume that ndim=2. We then have:

iinbc(icurnr,1,1) = 4; iinbc(icurnr,1,2) = -1; iinbc(icurnr,1,3) = 0;

iinbc(icurnr,2,1) = 4; iinbc(icurnr,2,2) = -1; iinbc(icurnr,2,3) = 0;

rinbc(icurnr,1,1) = m; rinbc(icurnr,1,2) = 0;

rinbc(icurnr,2,1) = $\tilde{m}$; rinbc(icurnr,2,2) = 0;

leading to:

ibndcon(ibouface) = 4; rbndcon(1,ibouface,1) = m; rbndcon(2,ibouface,1) = 0;

rbndcon(1,ibouface,2) = $\tilde{m}$; rbndcon(2,ibouface,2) = 0.

Filling array rbndcon is done in routine fvbndcon.f.

A systematic overview of the contents of array rbndcon, for face ibouface, with ndim = 2, with given values of ibndcon (see previous section).

Momentum equation

4 rbndcon(1,ibouface,1) = m; rbndcon(2,ibouface,1) = 0;
rbndcon(1,ibouface,2) = $\tilde{m}$; rbndcon(2,ibouface,2) = 0.

5 rbndcon(1,ibouface,1) = m; rbndcon(2,ibouface,1) = 0;
rbndcon(1,ibouface,2) = σ^{tt}; rbndcon(2,ibouface,2) = 0.

6 rbndcon(1,ibouface,1) = σ^{nn}; rbndcon(2,ibouface,1) = 0;
rbndcon(1,ibouface,2) = $\tilde{m}$; rbndcon(2,ibouface,2) = 0.

7 `rbndcon(1,ibouface,1)` $= \sigma^{nn}$; `rbndcon(2,ibouface,1)` $= 0$;
`rbndcon(1,ibouface,2)` $= \sigma^{tt}$; `rbndcon(2,ibouface,2)` $= 0$;

14 `rbndcon(1,ibouface,1)` $= m$; `rbndcon(2,ibouface,1)` $= \sigma^{nn}$;
`rbndcon(1,ibouface,2)` $= \tilde{m}$; `rbndcon(2,ibouface,2)` $= 0$.

Scalar equation

1 `rbndcon(1,ibouface,1)` $= \phi^{n+1}$ (at boundary); `rbndcon(2,ibouface,1)` $= 0$;
`rbndcon(1,ibouface,2)` $= 0$; `rbndcon(2,ibouface,2)` $= 0$.

2 `rbndcon(1,ibouface,1)` $= a$; `rbndcon(2,ibouface,1)` $= f$;
`rbndcon(1,ibouface,2)` $= 0$; `rbndcon(2,ibouface,2)` $= 0$.
Note that, for the moment, we have implemented homogeneous Neumann, i.e. $a = f = 0$.

For the continuity equation, in order to be consistent with the discretization, we have inserted a homogeneous Neumann condition at all boundaries.
When the total enthalpy is a primary variable, we have (of course) `rbndcon(1,ibouface,1)` $= H^{n+1}$ at the boundary in case of a Dirichlet condition. As mentioned in Section 14.5.3, a Dirichlet condition for H only holds when both h as the momentum vector $\mathbf{m}$ or length of momentum vector $|\mathbf{m}|$ are given at the boundary. Note that $(\mathbf{m} \cdot \mathbf{m}) = |\mathbf{m}|^2$. The required density ρ is taken to be the density at in the adjacent boundary cell, and then we compute H at the boundary using

$$H = h + \frac{1}{2}(\gamma - 1) M_r^2 \frac{(\mathbf{m} \cdot \mathbf{m})}{\rho^2}. \tag{14.2}$$

This is done in routine `fvbndc02.f`. For other primary energy variables, see Section 2.3.3 for equations that relate this energy variable with h and $\mathbf{m}$, a similar approach is followed.

Chapter 15

Main structure of the software

In this chapter a general overview of the software is given. Note that a lot of things are omitted for reasons of clarity: they will be discussed further on in this manual.

15.1 Main program: tudfinvol.f

The main program `tudfinvol.f` initializes the buffer-arrays `buffer` and `ibuffr` with an `equivalence`-statement. The total length of the buffer-array is set to $5000000 = 5 \cdot 10^6$ positions.
The program `tudfinvol.f` calls the routine `fvcomput.f`.

15.2 Time-loop: fvcomput.f

The routine `fvcomput.f` contains the time-loop.
In `fvstart.f` initializations etc. are performed. As long as the actual time `tact` is smaller

Algorithm 1 fvcomput.f

```
call fvstart
while (t < t^end) and (not-converged) do
  call fvtstep
end while
```

than the end-time `tend` given by the user, and as long the solution is not converged, then the time-stepping continues, in `fvtstep.f`.

15.3 Initializations: fvstart.f

Before the time-loop commences, all sorts of arrays concerning the mesh, data structure, initial and boundary conditions etc. etc. have to be put in appropriate arrays. This is all done in routine `fvstart.f`. For more information, see Chapter 16.

15.4 Time-stepping: fvtstep.f

Routine `fvtstep.f` is designed such that it contains exactly one time-step of the time-stepping procedure. In each time-step a certain number of linear systems is built and solved. At this moment we assume that the θ-method or explicit Euler is applied to do the time-integration, and that no fractional time-stepping or sub-stepping are used. For more information, see Chapter 17.

Chapter 16

Before time-stepping

In this Chapter we discuss the work to be performed before the time-stepping for the unstructured finite volume solver can take place. This work consists of, among others, a check of the grid, the creating and filling of certain arrays, reading the input files, setting the initial conditions and so on. We have copied from the ISNAS software as much as possible, and in those cases references to the ISNAS Programmers Guide will suffice. Therefore, in this chapter the main focus lies on the aspects that are different from ISNAS. The main routine dealing with the work before the time-stepping starts, is routine `fvstart.f`, see below. The subroutines of `fvstart.f` are discussed in Section 16.1 and the following section.
The time-stepping itself is discussed in Chapters 17 and 18. In routine `fvstart.f`, see the al-

Algorithm 2 fvstart.f

some initializations
check the mesh (`fvmshchk.f`)
read the input (`fvinput.f`)
determine the structure of the matrices (`fvstrmat.f`)
Compute the normal vectors at each face (`fvnormal.f`)
compute the gradient coefficients (`fvgrad.f`)
compute the weighted interpolation coefficients (`fvlinpol.f`)
compute the interpolation coefficients for the tangential momentum (`fvtancomp`)
compute the weight coefficients needed to compute the velocity gradients (`fvgradvel`).
compute the reconstruction coefficients (`fvreconstcoef.f`)
store the initial conditions (`fvincnd.f`)
print some information concerning the grid used (`fvpringridinfo.f`)

gorithm given in this section, several subroutines are called. They are discussed, respectively, in Sections 16.1 to 16.8.

16.1 Checking mesh: fvmshchk.f

Routine `fvmshchk.f` checks whether the mesh is suited for the discretization, see Section 3.3.

16.2 Reading input: fvinput.f

Routine `fvinput.f` reads the problem inputfile for the program TUDFINVOL. The input is recognized in exactly the same way as in ISNaS.

16.3 Determination of structure of matrices: fvstrmat.f

Routine `fvstrmat.f` creates two distinct matrix-structures: one for the scalar equations, and one for the momentum. This is done in subroutines `fvstrcvdf` and `fvstrmom` respectively. Note that for all scalar equations (energy, density, pressure-correction) the same matrixstructure (i.e. stencil) is used.
The necessary info concerning these matrix-structures is stored in the arrays `intmat(5,ndegfd)` (Section 14.2.3), `istrcvdf(listrcvdf)` and `istrmom(listrmom)`. The arrays `istrcvdf(listrcvdf)` and `istrmom(listrmom)` contain the positions of the essential non-zero elements of the matrices for the scalar respectively momentum matrix. We use `JMETOD` = 6, (see SEPRAN Programmers Guide, Section 24.6 on INTMAT) because the matrices are real and non-symmetric, but have a symmetric profile. This means that only the diagonal and lower triangular elements are stored in arrays `istrmom` and `istrcvdf`. The creation of these arrays, with respectively lengths `listrmom` and `listrcvdf`, is done in routines `fvstrmom.f` and `fvstrcvdf.f`. In these routines, first the complete stencil is determined and after that the connectivities are ordened such as to satisfy the requirements belonging to `JMETOD` = 6 (integer storage in increasing order; only diagonal and lower triangular connectivities are stored). Furthermore, these routines compute the lengths `lmstrmom` and `lmstrcvdf` of the real arrays `mstrmom` and `mstrcvdf` that will contain the matrix-elements. These lengths are computed from:

```
lmstrmom  = nfaces + 1 + 2*istrmom(nfaces+1)
lmstrcvdf = nelem  + 1 + 2*istrcvdf(nelem+1).
```

These lengths are stored in resp. `intmat(5,1)` and `intmat(5,ndim+1)`.
With respect to the stencil, the following must be said. The stencil for the scalar variables, see Figure 16.1A, consists of at most[1] 10 points. This is sufficient for Euler and Navier-Stokes computations, and suited for higher order upwind as well. The stencil for the momentum equation, see Figure 16.1B, contains at this moment 29 points: enough for Euler and Navier-Stokes, with higher order upwind.

[1] In the vicinity of the boundaries or high-aspect ratio cells this number may be smaller

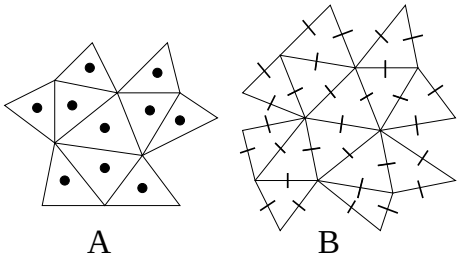

Figure 16.1: The stencil for scalar equations and the momentum equation.

16.3.1 Filling of the matrices: fvmfilmat.f

In the routine that fills the matrix, the following problem is met: given the integer array `istrcvdf` or `istrmom`, and given a matrixelement A_{ij} and the positions i and j, what is the position of this element in array `mstrcvdf` (or `mstrmom`)? Note that problem occurs when we have computed the matrix-elements. For more information concerning these computations, see Chapter 18.
Let `nusol` = `nelem` for the scalar matrix, and `nusol` = `nfaces` for the momentum matrix. There are three cases to be considered (for `mstrmom` this is completely similar):

- $i = j$ (diagonal element). This element is stored at the ith position of the matrix array `mstrcvdf`.

- $i > j$ (lower triangular element). The number of essential non-zero, lower triangular, elements in the ith row is given by
  `ncolumn = istrcvdf(i+1) - istrcvdf(i)`. (Completely similar for momentum matrix)
  This means that there is exactly one integer k, $k = 1, \ldots,$ `ncolumn`, for which $j =$ `istrcvdf(istrcvdf(i)+nusol+1+k)` holds. The matrix-element A_{ij} is stored at position `(istrcvdf(i)+nusol+k)` in the real matrix array.

- $i < j$ (upper triangular element). Let x be the position in the real matrix array of element A_{ji} (this position is determined by the procedure given above), then the element A_{ij} is stored at position (x + `istrcvdf(nusol+1)`) of the real matrix array.

16.4 Weighted interpolation to obtain thermodynamic quantities at faces: fvlinpol.f

The aim of interpolating scalar quantities like density and enthalpy, is to obtain their values at the midpoint of the cell faces, see Section 4.4.2 for how this should be done.
Scalar ϕ, at face i with adjacent cells q and r, is computed from (4.16), which we write here as:

$$\phi_i = \alpha_r \phi_r + \alpha_q \phi_q. \tag{16.1}$$

Recalling that r and q are integer cell numbers (Section 13.1), and noting that $\alpha_r + \alpha_q = 1$, we can write:

$$\phi_i = \begin{cases} \phi_r + \sigma_i(\phi_q - \phi_r) & \text{if } q > r \\ \phi_q + \sigma_i(\phi_r - \phi_q) & \text{if } r > q \end{cases} \tag{16.2}$$

So

$$\sigma_i = \begin{cases} \alpha_q & \text{if } q > r \\ \alpha_r & \text{if } r > q \end{cases} \tag{16.3}$$

In this way, we have to store per face only one quantity (σ_i) to do the weighted interpolation unambiguously.
When face i is a boundary face and cell r the boundary cell, then we replace the weighted interpolation by: $\phi_i = \phi_r$. Since cell q does not exist, but is referred to as having a number (`nelem` + 1), see Section 13.1), we always have $q > r$. The weighted interpolation hence still can be done by using $\sigma_i = 0$.
The quantities σ_i are computed using this formula for all internal faces i, and are put to zero for all boundary faces. This computation is done in `fvlinpol.f`, and the coefficients σ_i are put in array `linpol(nfaces)`.

16.5 Interpolation coefficients for tangential momentum: fvtancomp.f

The weight coefficients and the stencil, as discussed in Section 5.8, are stored in arrays RTANCOMP and ITANCOMP. User has a possibility to chose between the clasical and the alternative reconstruction by using the keyword `alter_reconstr`.

16.6 Weight coefficients to compute velocity gradients: fvgradvel.f

TO BE FILLED IN

16.7 Reconstruction coefficients: fvreconstcoef.f

Consider Figure 16.2. The reconstruction procedure as discussed in 5.4.2 gives us the following

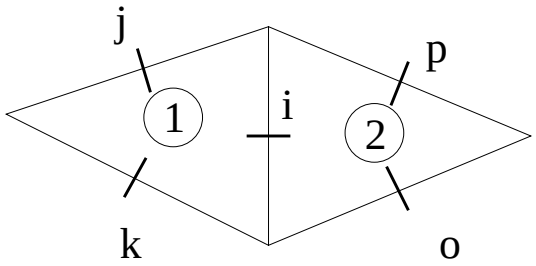

Figure 16.2: Face i is surrounding by four other faces.

relations:

$$\begin{aligned} \mathbf{N}_i &= \eta_j \mathbf{N}_j + \eta_k \mathbf{N}_k = \eta_o \mathbf{N}_o + \eta_p \mathbf{N}_p \\ \mathbf{t}_i &= \xi_j \mathbf{N}_j + \xi_k \mathbf{N}_k = \xi_o \mathbf{N}_o + \xi_p \mathbf{N}_p \end{aligned}$$

Let the cell-number of the left cell be smaller than the cell-number of the right cell. The arrays `reconstnor` and `reconsttan` contain:
`reconstnor(1,i)` $= \eta_j$; `reconstnor(2,i)` $= \eta_k$; `reconstnor(3,i)` $= \eta_o$; `reconstnor(4,i)` $= \eta_p$;
`reconsttan(1,i)` $= \xi_j$; `reconsttan(2,i)` $= \xi_k$; `reconsttan(3,i)` $= \xi_o$; `reconsttan(4,i)` $= \xi_p$.

16.8 Prescribing initial conditions: fvincnd.f

Routine `fvincnd.f` sets the initial conditions for program TUDFINVOL in the appropriate arrays. Memory management numbers are put in array `isol(5,numsol,3)`, see Section 14.2.1.

Compressible flow
In the input files we give (dimensionless) initial conditions for the momentum vector $\mathbf{m}$, enthalpy h and pressure p. As primary variables we use $\mathbf{m}$, density ρ and an energy variable or δp. Note that when h is not the energy variable, this energy variable must be computed using relations as given in Section 2.3.3. Computing ρ from a given p and h is done using the equation of state. Routine `fvinctoth.f` makes use of the relations as given in Section 6.1. The method described in Chapter 6 is used to do the required interpolation.

Incompressible flow
In the input files we give (dimensionless) initial conditions for the primary variables: velocity vector $\mathbf{m}$ and pressure p.

The outlines of `fvincnd.f` is given in the algorithm presented in this Section.

Algorithm 3 Set initial conditions (fvincnd.f)

Create solution arrays
Put m and $\tilde{m}$ at all faces (`fvincu2.f`)
Put scalar quantities in all cells (`fvinct2.f`)
if Compressible flow **then**
  Compute ρ in all cells (`fveqsta03.f`)
  if h is not the primary energy variable **then**
    Compute primary energy variable in all cells (`fvinctoth.f`)
  end if
end if

Chapter 17

Time-stepping

17.1 Routine fvtstep.f

Routine `fvtstep.f` contains one time-step, see also Section 15.4.
Routine `fvsbstep.f` builds and solves the matrix for each primary variable. The order in

Algorithm 4 fvtstep.f

for all primary variables **do**
  build matrix and solve linear system (`fvsbstep`)
end for
check for convergence (`fvstopcr`)
extrapolate values to next time-level (`fvthetaextrpol`)
if $t = t^{end}$ or (requested) or (converged) **then**
  do postprocessing (`fvpost`)
end if
shift solution arrays to be ready for next time-level (`fvshift`)

which the equations are solved, is discussed in Section 9.4. For a more thorough description of `fvsbstep.f`, see Section 17.2.
After that all equations at one time-level are solved, the convergence of the solution is checked in `fvstopcr.f`, using the termination criterion as given in Section 9.5.
Routine `fvthetaextrpol.f` extrapolates the solutions, having the solutions at time-levels n and $(n+\theta)$, to time-level $(n+1)$. Equation (9.3) is used to this aim. After that all primary variables are extrapolated to time-level $(n+1)$, the derived quantities (velocity, pressure) are computed. See for the discussion in Section 9.2 why first the primary variables have to be extrapolated before the derived quantities can be computed. For the computation of the velocity at the faces, see Section 5.9. The computation of the pressure, from the equation of state (2.40), is done in `fveqsta02.f`. When h is the primary energy variable, this is trivial. When we have another primary energy variable, see Chapter 6.
Routine `fvpost.f` does the post-processing, see Chapter 19.
Routine `fvshift.f` does some rearrangements in the solution arrays, such that they are fit to enter the new time-level. To be more precise, array `isol(*,*,1)` and `isol(*,*,2)` contain, when entering routine `fvshift.f`, the solutions at respectively time-levels n and $(n+1)$. In `fvshift.f` the new time-level $N = (n+1)$, the solution at the new $N = (n+1)$, i.e.

`isol(*,*,2)`, is copied to its 'new' position, i.e. `isol(*,*,1)`.

17.2 Routine fvsbstep.f

Routine `fvsbstep.f` performs a part of the time-stepping. To be more precise, `fvsbstep.f` builds and solves a matrix-equation for one primary variable. In some cases, to be discussed below, it performs some additional things.
Routine `fvsbstep.f` starts with filling arrays related to the boundary conditions (arrays `ibndcon` and `rbndcon`, using procedure `fvbndcon.f`, see also Section 14.5).
Then it fills the relevant coefficient-arrays in routine `fvcoefs.f`, see Section 14.4.
The matrix-equation is built in routine `fvbldmat.f`. The matrix and right-hand side for a scalar variable (density and energy variable) is made in `fvcvdf.f`. For the momentum equation, the matrix and right-hand side are made in `fvprmom.f`. These things will be discussed in much more detail in Chapter 18.
The matrix-equation is solved, for implicit time-integration, in routine `fvsolve.f`. When considering explicit time-integration, a matrix-vector multiplication and a vector addition (see Section 9.3) must be done.
The so-called additional things are specified below.

Algorithm 5 fvsbstep.f

Some initializations
Fill arrays concerning boundary conditions (`fvbndcon`)
Fill arrays concerning with coefficients (`fvcoefs`)
Build matrix-vector equation (`fvbldmat`)
if Implicit time-integration **then**
  Solve linear system (`fvsolve`)
else
  Do matrix-vector multiplication (`maver` and `fvaddvec`)
end if
if condition **then**
  do additional things
end if

To be more precise, see the comment statements in `fvsbstep.f`.

17.3 Content of solution arrays during time-stepping

The content of the arrays to which the memory management numbers `isol(1,eq,itimlv)`, with `eq` an equation number (Section 14.1) and `itimlv` = 1,..., `ntimlv`, refer, is now being discussed. For the content of array `isol`, see also Section 14.2.1. We have implemented only for `ntimlv` = 2.

Compressible flows
At the beginning of the time-loop to obtain quantities at time-level $n+1$ (so all quantities up to time-level n are known) the memory management numbers refer to the following quantities (stored in solution arrays):

- `isol(1,1,1)`: m^n;
  `isol(1,1,2)`: idem.
- `isol(1,2,1)`: $\tilde{m}^n$;
  `isol(1,2,2)`: idem.
- `isol(1,ndim+1,1)`: p^n;
  `isol(1,ndim+1,2)`: idem.
- `isol(1,ndim+2,1)`: energy variable (h^n, H^n or $(\rho H)^n$, ...); when considering the explicit scheme we have here $\phi = (\rho H)^n$
  `isol(1,ndim+2,2)`: idem.
- `isol(1,ndim+1+ntrnsp+nturb+1,1)`: ρ^n;
  `isol(1,ndim+1+ntrnsp+nturb+1,2)`: idem.
- `isol(1,ndim+1+ntrnsp+nturb+2,1)`: u^n;
  `isol(1,ndim+1+ntrnsp+nturb+2,2)`: idem.

The changes in the stored solution vectors for every part of the time-stepping procedure, see Section 9.4, will now be discussed.
When the theta-method, see Section 9.2, is employed to do the time-stepping, the content of the arrays is as follows.

1. Computation of the normal momentum.
   `isol(1,1,1)` still refers to m^n.
   `isol(1,1,2)` now contains $m^{n+\theta}$.
2. Computation of tangential momentum.
   `isol(1,2,1)` still refers to $\tilde{m}^n$.
   `isol(1,2,2)` now contains $\tilde{m}^{n+\theta}$.
3. Computation of new density.
   `isol(1,ndim+1+ntrnsp+nturb+1,1)` still refers to ρ^n.
   `isol(1,ndim+1+ntrnsp+nturb+1,2)` now contains $\rho^{n+\theta}$.
4. Computation of new energy variable, say ϕ.
   `isol(1,ndim+2,1)` still refers to ϕ^n.
   `isol(1,ndim+2,2)` now contains $\phi^{n+\theta}$.

5. Computation of new pressure.
   `isol(1,ndim+1,1)` still refers to p^n.
   `isol(1,ndim+1,2)` now contains p^{n+1}.

6. Computation of new velocity.
   `isol(1,ndim+1+ntrnsp+nturb+2,1)` still refers to u^n.
   `isol(1,ndim+1+ntrnsp+nturb+2,2)` now contains u^{n+1}.

Between items 4 and 5, routine `fvthetaextrpol.f` is used to extrapolate the solutions from time-level $(n+\theta)$ to time-level $(n+1)$; see Section 17.1. The derived quantities (pressure and velocity) are then obtained immediately at time-level $(n+1)$.
When the explicit Euler scheme is used, see Sections 2.3.4 and 4.5.6, the content of the arrays is as follows:

1. Computation of the normal momentum.
   `isol(1,1,1)` still refers to m^n.
   `isol(1,1,2)` now contains m^{n+1}.

2. Computation of tangential momentum.
   `isol(1,2,1)` still refers to $\tilde{m}^n$.
   `isol(1,2,2)` now contains $\tilde{m}^{n+1}$.

3. Computation of new density.
   `isol(1,ndim+1+ntrnsp+nturb+1,1)` still refers to ρ^n.
   `isol(1,ndim+1+ntrnsp+nturb+1,2)` now contains ρ^{n+1}.

4. Computation of new energy variable q^{n+1}; $q = \rho E$.
   `isol(1,ndim+2,1)` still refers to $\phi^n = (\rho H)^n$.
   `isol(1,ndim+2,2)` now contains $q^{n+1} = (\rho E)^{n+1}$.

5. Computation of new pressure.
   `isol(1,ndim+1,1)` still refers to p^n.
   `isol(1,ndim+1,2)` now contains p^{n+1}. Furthermore, q is transformed to ϕ, see Section 18.3.

6. Computation of new velocity.
   `isol(1,ndim+1+ntrnsp+nturb+2,1)` still refers to u^n.
   `isol(1,ndim+1+ntrnsp+nturb+2,2)` now contains u^{n+1}.

Incompressible flows
At the beginning of the time-loop to obtain quantities at time-level $n+1$ (so all quantities up to time-level n are known) the memory management numbers refer to the following quantities (stored in solution arrays):

- `isol(1,1,1)`: u^n;
  `isol(1,1,2)`: idem.

- `isol(1,2,1)`: $\tilde{u}^n$;
  `isol(1,2,2)`: idem.

- `isol(1,ndim+1,1)`: p^n;
  `isol(1,ndim+1,2)`: $\delta p = p^n - p^{n-1}$.

Note that at `isol(1,ndim+1,2)` we have stored δp, and not p^n. The reason is that the linear solver wants to use this vector as start-vector of the GMRES-iterations when solving the pressure-correction equation.
The changes in the stored solution vectors for every part of the time-stepping procedure, see Section 9.4, will now be discussed.
When the theta-method, see Section 9.2, is employed to do the time-stepping, the content of the arrays is as follows. Usually, only $\theta = 1$ is used.

1. Computation of the predictor of the velocity.
   `isol(1,1,1)` still refers to u^n.
   `isol(1,1,2)` now contains u^*.

2. Computation of pressure-correction.
   `isol(1,ndim+1,1)` still refers to p^n.
   `isol(1,ndim+1,2)` now contains $\delta p = p^{n+\theta} - p^n$.

3. Computation of new velocity by corrector step.
   `isol(1,1,1)` still refers to u^n.
   `isol(1,1,2)` now contains $u^{n+\theta}$

4. Computation of tangential velocity.
   `isol(1,2,1)` still refers to $\tilde{u}^n$.
   `isol(1,2,2)` now contains $\tilde{u}^{n+\theta}$.

Routine `fvthetaextrpol.f` is used to extrapolate the solutions from time-level $(n+\theta)$ to time-level $(n+1)$; see Section 17.1. Concerning the pressure solution arrays, it makes sure that `isol(1,ndim+1,1)` contains p^{n+1} and `isol(1,ndim+1,2)` contains $\delta p = p^{n+1} - p^n$. As a consequence, `fvshift.f` does not need to move the content from the pressure solution arrays anymore. For the postprocessing, we want to have the solution p^{n+1} in `isol(1,ndim+1,2)`. This is done in `fvpost.f` with help of `fvchprincr.f`.

Mach-uniform flows
At the beginning of the time-loop to obtain quantities at time-level $n+1$ (so all quantities up to time-level n are known) the memory management numbers refer to the following quantities (stored in solution arrays):

- `isol(1,1,1)`: m^n;
  `isol(1,1,2)`: idem.

- `isol(1,2,1)`: $\tilde{m}^n$;
  `isol(1,2,2)`: idem.

- `isol(1,ndim+1,1)`: p^n;
  `isol(1,ndim+1,2)`: idem.

- `isol(1,ndim+2,1)`: h^n
  `isol(1,ndim+2,2)`: idem.

- `isol(1,ndim+1+ntrnsp+nturb+1,1)`: ρ^n;
  `isol(1,ndim+1+ntrnsp+nturb+1,2)`: idem.

- `isol(1,ndim+1+ntrnsp+nturb+2,1)`: u^n;
  `isol(1,ndim+1+ntrnsp+nturb+2,2)`: idem.

The changes in the stored solution vectors for every part of the time-stepping procedure, see Section 9.4, will now be discussed.
When the theta-method, see Section 9.2, is employed to do the time-stepping, the content of the arrays is as follows.

1. Computation of the density.
   `isol(1,ndim+1+ntrnsp+nturb+1,1)` still refers to ρ^n.
   `isol(1,ndim+1+ntrnsp+nturb+1,2)` now contains ρ^{n+1}.

2. Computation of the predictor of the momentum.
   `isol(1,1,1)` still refers to m^n.
   `isol(1,1,2)` now contains m^*.

3. Computation of the predictor of the velocity.
   `isol(1,ndim+1+ntrnsp+nturb+2,1)` still refers to u^n.
   `isol(1,ndim+1+ntrnsp+nturb+2,2)` now contains u^*.

4. Computation of new pressure.
   `isol(1,ndim+1,1)` still refers to p^n.
   `isol(1,ndim+1,2)` now contains δp.

5. Correction of the momentum.
   `isol(1,1,1)` still refers to m^n.
   `isol(1,1,2)` now contains m^{n+1}.

6. Computation of tangential momentum.
   `isol(1,2,1)` still refers to $\tilde{m}^n$.
   `isol(1,2,2)` now contains $\tilde{m}^{n+1}$.

7. Computation of the new velocity.
   `isol(1,ndim+1+ntrnsp+nturb+2,1)` still refers to u^n.
   `isol(1,ndim+1+ntrnsp+nturb+2,2)` now contains u^{n+1}.

8. Computation of new enthalpy h.
   `isol(1,ndim+2,1)` still refers to h^n.
   `isol(1,ndim+2,2)` now contains h^{n+1}.

Between items 6 and 7, routine `fvthetaextrpol.f` is used to extrapolate the solutions from time-level $(n+\theta)$ to time-level $(n+1)$; see Section 17.1. The derived quantities (pressure and velocity) are then obtained immediately at time-level $(n+1)$.

Chapter 18

Building matrices

In this chapter the building of the matrices and right-hand sides is described in great detail. With 'building of the matrix', we mean computation of the matrix-elements and computation of the right-hand sides. The implementation of a general convection-diffusion equation of a scalar is discussed in Section 18.1. A distinction is made between internal and boundary cells. Note that the continuity equation and energy equation are specific forms of the general convection-diffusion equation. The implementation of the momentum equation is discussed in Section 18.2. A distinction is made between real internal, quasi internal and boundary faces. The computation of the pressure is discussed in Section 18.3. The computation of the velocity is discussed in Section 18.4. The computation of the tangential momentum component is discussed in Section 18.5. Numerical evaluation of a term in the conservative energy equation is discussed in Section 18.6.

18.1 General convection-diffusion equation

All procedures concerning the evaluation of scalar ϕ^{n+1}, using the conservative form of the convection-diffusion equation, are put together here. For a mathematical derivation of the discretization, we refer the reader to read Chapter 4.
We build the matrix using an implicit time-integration scheme. Each term in the equation (time derivative, convection term, diffusion term) is evaluated in another routine, see Algorithm 6. In the case that we want to do explicit time-integration, we still build the matrix

Algorithm 6 Build matrix for scalar(n+1) (fvcvdf.f)

Store structure of upper matrix in temporary arrays (`fvintmat.f`)
Add source term to right-hand side (`fvcvdfsrc.f`)
Store time derivative part into matrix (`fvcvdftim.f`)
Store convection term into matrix (`fvcvdfconv`)
Compute contributions to the right-hand side (`fvcvdfrhsd`)
if $\theta = 0$ **then**
  Make matrix and right-hand side ready for explicit time-integration (`fvexplicitmat`).
end if

using implicit time-integration, but we do some manipulation with the matrix and right-hand side in `fvexplicitmat`.

18.1.1 Convection-diffusion equation: source term

With $q(\mathbf{x}, t)$ a source term, the following term is added to the right-hand side in `fvcvdfsrc.f`:

$$b_i := b_i + \Omega_i q_i, \tag{18.1}$$

with i the cell under consideration and $q_i = q(\mathbf{x}_i, t)$ at the appropriate time t.

18.1.2 Convection-diffusion equation: time derivative term

In routine `fvcvdftim.f` the time derivative term is computed and inserted in the matrix and right-hand side.

The contribution for cell i to the diagonal term is:

$$A_{ii} := A_{ii} + \frac{\Omega_i a_i^{n+1}}{\tau} \tag{18.2}$$

The contribution to the right-hand side is:

$$b_i := b_i + \frac{\Omega_i a_i^n \phi_i^n}{\tau} \tag{18.3}$$

18.1.3 Convection diffusion equation: convection term

In routine `fvcvdfconv.f` the convection term is computed and inserted in the right-hand side. For mathematical information we refer to Chapter 4. In routine `fvcvdfconv.f` the convection term for the convection diffusion equation is computed. Here below we discuss the

implementation.
For a discussion of the implementation, we refer to the pseudo-code and comment-statements in `fvcvdfconv.f`.

18.1.4 Convection diffusion equation: contribution to rhsd

In Sections 4.5 and 4.6 we see that the term $\Omega_i g_i^{n,n+1}$ has to be added to the right-hand side. Define:

$$T_i^n = \frac{\Omega_i}{\tau}\frac{(\mathbf{m}\cdot\mathbf{m})^n}{\rho_i^n} \tag{18.4}$$

$$T_i^{n+1} = \frac{\Omega_i}{\tau}\frac{(\mathbf{m}\cdot\mathbf{m})^{n+1}}{\rho_i^{n+1}} \tag{18.5}$$

$$C_i^{n+1} = \sum_e m_e^{n+1}\frac{(\mathbf{m}\cdot\mathbf{m})_e^{n+1}}{(\rho_e^{n+1})^2}\bar{l}_e, \tag{18.6}$$

then we see that the corresponding expressions in (4.20), (4.21), (4.23) and (4.25) can be written in the form:

$$\Omega_i g_i^{n,n+1} = C_t(T_i^{n+1} - T_i^n) + C_c C_i^{n+1} \tag{18.7}$$

with C_t and C_c constants. Note that the corresponding expression in (4.29) is trivial. For more info we refer to the pseudo code and comments in `fvcvdfrhsd.f`.

18.1.5 Convection diffusion equation: explicit time integration

In the case that the user wishes to do explicit Euler time integration, we still will build the matrix and right-hand side as if we were to do implicit Euler time integration. Routine `fvexplicitmat` then changes the matrix and rigth-hand side such that they can be used for explicit time integration. For a mathematical discussion on this topic we refer to Section 9.3.

18.2 Momentum equation

All procedures concerning the evaluation of m^{n+1} are put together here. For a mathematical derivation of the discretization, we refer the reader to Chapter 5. It must be noted that the unknown is the primary variable m (projected momentum), and not the flux (equal to $(m \times l)$, where l is length of the face) as one does in ISNAS.
There are three kinds of faces, see Section 3.2: real internal faces, quasi internal faces and boundary faces.
We build the matrix using an implicit time-integration scheme. Each term in the equation (time derivative, convection term, viscous term, pressure gradient) is evaluated in another routine, see Algorithm 7. In the case that we want to do explicit time-integration, we still

Algorithm 7 Build matrix for m(n+1) (fvprmom.f)

Calculate the scalars at the faces (`fvscalarfaces.f`)
Store structure of upper matrix in temporary arrays (`fvintmat.f`)
Compute source term for right-hand side (`fvmomsrc.f`)
Store viscosity part into matrix (`fvviscmat.f`)
Store time derivative part into matrix (`fvmomtim.f`)
Store projected pressure gradient term in right hand side (`fvpresgrad`)
Store convection term into matrix (`fvmomconv`)
Store boundary conditions into matrix and right-hand side (`fvmombound.f`)
if $\theta = 0$ **then**
  Make matrix and right-hand side ready for explicit time-integration (`fvexplicitmat`).
end if

build the matrix using implicit time-integration, but we do some manipulation with the matrix and right-hand side in `fvexplicitmat`.

18.2.1 Momentum equation: scalars at the faces

Subroutine `fvscalarfaces.f` calculates the density and the viscosity at the face centers by the linear interpolation. There is a switch `areaweighted` in this subroutine that allows to chose between the area-weighted interpolation and arithmetic averaging. Density is calculated from the array `dens`, and the velocity is calculated from `coefs`. Those interpolated scalars are stored in the array `facecoefs`.

18.2.2 Momentum equation: source term

Let $\mathbf{f}$ be a source term force. Let the control volume for face i be $\Omega_i = \Omega_1 \cup \Omega_2$, where Ω_1 and Ω_2 are in distinct triangles. When integration is done over whole triangles, then Ω_1 and Ω_2 are these triangles. When integration is done over half triangles, then Ω_1 and Ω_2 are these half triangles. The term added to the right-hand side is:

$$\Omega_i \mathbf{f}_i \cdot \mathbf{N}_i = \Omega_1 \mathbf{f}_1 \cdot \mathbf{N}_1 + \Omega_2 \mathbf{f}_2 \cdot \mathbf{N}_2. \tag{18.8}$$

This is done in routine `fvmomsrc.f`.

18.2.3 Momentum equation: viscous term

In routine `fvviscmat.f` the viscous term is computed and inserted in the matrix and the right-hand side.

Algorithm 8 Viscous term (fvviscmat.f)

if Integration over whole triangles **then**
  if Incompressible case **then**
    Compute viscous term for internal and quasi-internal faces (`fvviscmatiq2s.f`)
    Compute viscous term for boundary faces (`fvviscmatb2s.f`)
  else
    Compute viscous term for internal and quasi-internal faces (`fvviscmatiq2.f`)
    Compute viscous term for boundary faces (`fvviscmatb2.f`)
  end if
else if Integration over half triangles **then**
  Compute viscous term for internal faces (`fvviscmati.f`)
  Compute viscous term for quasi-internal faces (`fvviscmatq.f`)
end if

Viscous term: integration over whole triangles

For mathematical information we refer to Section 5.2.1. Routines `fvviscmatiq2s.f` and `fvviscmatb2s.f` calculate the viscous term in simplified incompressible case, when viscosity is constant and whole viscous term becomes a Laplacian. Only Dirichlet boundaty conditions had been implemented. Those routines serve mainly for experimental purposes, and presently such mode can be activated by setting `simplemode = .true.` in routine `fvviscmat.f`.

Routines `fvviscmatiq2.f` and `fvviscmatb2.f` normaly calculate the viscous term on whole triangles. The global structure of all those routines is the same.

Viscous term: integration over half triangles

Only Dirichlet boundary conditions had been implemented. Since the momentum is given at the boundary, there is no need to treat the boundary cells.

Algorithm 9 fvviscmatiq2.f

```
for all internal and quasi-internal faces iface do
  Find the two elements corresponding to iface
  Compute the normal to the face
  for both elements ielem do
    for all three faces jface in ielem exept iface do
      Get μ
      if tangential stress is given then
        Add contribution to rhsd
        if normal stress is given then
          Add it to the rhsd
          Go to next jface
        else
          Calculate multiplication factors that relate the stress to the gradient
        end if
      else
        Calculate multiplication factors that relate the stress to the gradient
      end if
      for all three or four faces kface around jface do
        Calculate normal and tangential vectors
        Calculate multiplication factors for gradient
        Fill part of stencil corresponding to normal component
        Fill part of stencil corresponding to tangential component
      end for
    end for
  end for
  Store the stencil in the matrix
end for
```

Algorithm 10 fvviscmatb2.f

```
for all boundry faces iface do
  Find the element ielem corresponding to iface
  Compute the normal to the face
  for all three faces jface in ielem exept iface do
    Get μ
    if tangential stress is given then
      Add contribution to rhsd
      if normal stress is given then
        Add it to the rhsd
        Go to next jface
      else
        Calculate multiplication factors that relate the stress to the gradient
      end if
    else
      Calculate multiplication factors that relate the stress to the gradient
    end if
    for all three or four faces kface around jface do
      Calculate normal and tangential vectors
      Calculate multiplication factors for gradient
      Fill part of stencil corresponding to normal component
      Fill part of stencil corresponding to tangential component
    end for
  end for
  Store the stencil in the matrix
end for
```

Algorithm 11 fvviscmati.f, fvviscmatq.f

```
for all internal (or quasi-internal) faces iface do
  Find the two elements corresponding to iface
  Compute the normal to the face
  for both elements ielem do
    Get μ
    for all three faces jface in ielem do
      Calculate multiplication factors that relate the stress to the gradient
      for all three or four faces kface around jface do
        Calculate normal and tangential vectors
        Calculate multiplication factors for gradient
        Fill part of stencil corresponding to normal component
        Fill part of stencil corresponding to tangential component
      end for
    end for
  end for
  Store the stencil in the matrix
end for
```

18.2.4 Momentum equation: time derivative term

In routine `fvmomtim.f` the time derivative term is computed and inserted in the matrix and right-hand side.

Algorithm 12 Time derivative (fvmomtim.f)

Time derivative for internal faces (`fvmomtimi.f`)
Time derivative for boundary faces (`fvmomtimb.f`)

Time derivative for internal faces: fvmomtimi.f
Let cells 1 and 2 be the adjacent cells of face i. The area of control volume follows from

$$\Omega_i = C(\Omega_1 + \Omega_2), \tag{18.9}$$

with $C = 1$ when one integrates over whole triangles, and $C = 1/2$ when one integrates over half triangles. The contribution to the diagonal of the matrix is:

$$A_{ii} := A_{ii} + \frac{\Omega_i}{\tau}. \tag{18.10}$$

The contribution to the right-hand side is:

$$b_i := b_i + \frac{\Omega_i m_i^n}{\tau}. \tag{18.11}$$

In addition, we store the contribution to the diagonal of the matrix, i.e. the term $D_{ii} = \Omega_i/\tau$, since this is required in the case of explicit time-integration, see Section 18.2.8.

Time derivative for boundary faces: fvmomtimb.f
If at boundary face i the normal momentum component is given, we put $A_{ii} = D_{ii} = 1$ and we leave $A_{ij} = 0$ for $j \neq i$. Furthermore we put $b_i = m_i^{n+1}$. If at boundary face i the normal momentum component is not given, and let cell 1 be the adjacent boundary cell, then

$$\Omega_i = C\Omega_1. \tag{18.12}$$

The contribution to the diagonal and the right-hand side are then the same as given above.

18.2.5 Momentum equation: pressure term

In routine `fvpresgrad.f` the projected pressure gradient term is computed and inserted in the right-hand side.

Algorithm 13 Pressure gradient (fvpresgrad.f)

Pressure gradient at internal faces (`fvpresgradi.f`)
Pressure gradient at boundary faces (`fvpresgradb.f`)

Pressure gradient at internal faces: fvpresgradi.f
With (18.9) giving the area of the control volume, the right-hand side for face i is altered as follows:

$$b_i := b_i - \frac{\Omega_i}{\gamma M_r^2} \sum_j \gamma_j p_j. \tag{18.13}$$

The factor $1/\gamma M_r^2$ is absent (hence, equal to 1) in the incompressible case. The computation of the coefficients γ_j and the corresponding stencil is discussed in Section 5.7. These coefficients and the corresponding stencil are stored in arrays `rgrfac` and `igrfac`, see Chapter 21.

Pressure gradient at boundary faces: fvpresgradb.f
When at boundary face i the momentum is given, then, obviously, the pressure gradient need not be calculated. When the momentum is not given, see Section 5.7.7.

18.2.6 Momentum equation: convection term

In routine `fvmomconv.f` the convection term is computed and inserted in the right-hand side.

Algorithm 14 Convection term (fvmomconv.f)

if Integration over whole triangles **then**
  Compute convection term when integrating over whole triangles (`fvmomconvwhol.f`)
else if Integration over half triangles **then**
  Compute convection term when integrating over half triangles (`fvmomconvhalf.f`)
end if

Convection term: integration over whole triangles

For mathematical information we refer to Sections 5.2.1 and 5.4. In routine `fvmomconvwhol.f` the convection term for the momentum equation is computed, when the user has specified that we wants to do the integration over whole triangles. Routine `fvmomconvwhol.f` calls two subroutines, see Algorithm 15, dealing with respectively internal and boundary faces.

Algorithm 15 Convection term when integration over whole triangles (fvmomconvwhol.f)

Convection term for internal faces (`fvmomconvwholi.f`)
Convection term for boundary faces (`fvmomconvwholb.f`)

Convection term for internal faces: fvmomconvwholi.f

For the mathematical discussion we refer to the corresponding sections in Chapter 5. The algorithm used in the routine for internal faces is given in Algorithm 16. For more details we refer to the comment-statements in the routine itself.

Algorithm 16 fvmomconvwholi.f)

for all internal faces i **do**
  Determine the four face-numbers e of the control volume (CV)
  Determine the rest of the stencil, the cell-numbers and whether one of more CV-faces are boundary faces
  Determine $\bar{l}_e$ for all faces e
  Compute $(\mathbf{N}_e \cdot \mathbf{N}_i)$ and $(\mathbf{t}_e \cdot \mathbf{N}_i)$ for all faces e
  Compute $(\mathbf{u}_e \cdot \mathbf{N}_e)$ for all faces e
  Compute contributions to the matrix and the right-hand side. Note that this depends on the type of upwind
  Insert the matrix-contributions in the matrix
end for

Convection term for boundary faces: fvmomconvwholb.f

For the mathematical discussion we refer to the corresponding sections in Chapter 5. The algorithm used in the routine for boundary faces is given in Algorithm 17. For more details we refer to the comment-statements in the routine itself.

Algorithm 17 fvmomconvwholb.f)

for all boundary faces i **do**
Determine the two other face-numbers e of the control volume (CV)
Determine the rest of the stencil, the cell-numbers and whether another CV-face is a boundary face
Determine $\bar{l}$ for i and both faces e
Compute $(\mathbf{N}_e \cdot \mathbf{N}_i)$ and $(\mathbf{t}_e \cdot \mathbf{N}_i)$ for both faces e, and $(\mathbf{N}_i \cdot \mathbf{N}_i) = 1$ and $(\mathbf{t}_i \cdot \mathbf{N}_i) = 0$.
Compute $(\mathbf{u}_e \cdot \mathbf{N}_e)$ for i and both faces e
Compute contributions to the matrix and the right-hand side. Note that this depends on the type of upwind
Insert the matrix-contributions in the matrix
end for

Convection term: integration over half triangles

For mathematical information we refer to Sections 5.2.2 and 5.5. In routine `fvmomconvhalf.f` the convection term for the momentum equation is computed, when the user has specified that we wants to do the integration over half triangles. Routine `fvmomconvwhol.f` calls three subroutines, see Algorithm 18, dealing with respectively internal, boundary and quasi internal faces.

Algorithm 18 Convection term when integration over half triangles (fvmomconvhalf.f)

Convection term for internal faces (`fvmomconvhalfi.f`)
Convection term for boundary faces (`fvmomconvhalfb.f`)
Convection term for quasi internal faces (`fvmomconvhalfq.f`)

Convection term for internal faces: fvmomconvhalfi.f

For the mathematical discussion we refer to the corresponding sections in Chapter 5. The algorithm used in the routine for all internal faces is given in Algorithm 19. Note that routine `fvmomconvhalfq.f`, working on all quasi internal faces, overwrites in certain cases the matrix elements and right-hand side obtained in `fvmomconvhalfi.f` For more details we refer to the comment-statements in the routine itself.

Algorithm 19 fvmomconvhalfi.f)

for all internal faces i **do**
  Determine the face-numbers of the four surrounding faces and the cell-numbers (indicated with 1 and 2)
  Compute $l_i(\mathbf{u}_1 \cdot \mathbf{N}_i)$ and $l_i(\mathbf{u}_2 \cdot \mathbf{N}_i)$
  Compute contributions to the matrix. Note that this depends on the type of upwind used
  Insert the matrix-contributions in the matrix
end for

Convection term for boundary faces: fvmomconvhalfb.f

For the mathematical discussion we refer to the corresponding sections in Chapter 5. The algorithm used in the routine for boundary faces is given in Algorithm 20. For more details we refer to the comment-statements in the routine itself.

Convection term for quasi internal faces: fvmomconvhalfq.f

For the mathematical discussion we refer to the corresponding sections in Chapter 5. The algorithm used in the routine for quasi internal faces is given in Algorithm 21. For more details we refer to the comment-statements in the routine itself.

Algorithm 20 fvmomconvhalfb.f)

for all boundary faces i **do**
  if m_i is given **then**
    Do nothing
  else
    Determine the face-numbers of the two surrounding faces and the cell-number (indicated with 1)
    Compute $l_i(\mathbf{u}_1 \cdot \mathbf{N}_i)$
    Compute contributions to the matrix.
    Insert the matrix-contributions in the matrix
  end if
end for

Algorithm 21 fvmomconvhalfq.f)

for all quasi internal faces i **do**
  Determine whether at one or more of the four surrounding faces the momentum vector is given
  if At at least one surrounding face **m** is given **then**
    Compute contributions to the matrix and right-hand side.
    Insert the matrix-contributions in the matrix
  end if
end for

18.2.7 Momentum equation: boundary conditions

If at boundary face i the normal momentum is given, we put $A_{ii} = D_{ii} = 1$ and $A_{ij} = 0$ for $j \neq i$. This is done in routine `fvmombound.f`.

Contibutions of other types of boundary conditions to viscous term are treated in routines `fvviscmatiq2.f` and `fvviscmatb2.f`

18.2.8 Momentum equation: explicit time integration

In the case that the user wishes to do explicit Euler time integration, we still will build the matrix and right-hand side as if we were to do implicit Euler time integration. Routine `fvexplicitmat` then changes the matrix and rigth-hand side such that they can be used for explicit time integration. For a mathematical discussion on this topic we refer to Section 9.3.

18.3 Computation of pressure

In the compressible situation, the pressure follows from the equation of state, see Section 18.3.1. In the incompressible case, the pressure follows from the pressure-correction approach, see Section 18.3.2.

18.3.1 Compressible case

The scaled equation of state reads (in dimensionless quantities) for several primary energy variables is given in Section 2.3.3. In procedure `fveqsta02.f` the new pressure p^{n+1} for every cell is obtained from the equation of state. With h as primary variable, it is trivial. When h is not a primary variable, see Chapter 6 for some necessary information. In the situation of a supersonic inflow boundary, the pressure is given at the inflow boundary. This is implemented by putting the pressure in all boundary cells adjacent to the inflow boundary, equal to the prescribed pressure (routine `fvinletpres.f`).
When dealing with explicit time-integration, the variable `enth` entering routine `fveqsta02.f` is q, equation (2.80). When leaving this routine, variable `enth` stands for ϕ, computed using (2.84).

18.3.2 Incompressible case

For a mathematical formulation of the pressure-correction equation, see Sections 7.3 or Chapter 8, see routine `fvpres.f`.

18.4 Computation of velocity

The normal velocity component is computed in routine `fvvelo01.f`, using the method as given in Section 5.9.

The correction to the velocity u^*, in the pressure-correction approach, is computed in `fvvelcor.f`.

18.5 Computation of tangential momentum

The tangential momentum component is computed on every face using the (known) normal momentum components at every face. The procedure as given in Section 5.8 is employed. The routine that computes the tangential momentum component, is `fvtanmom.f`.

18.6 Numerical evaluation of integral in conservative energy equation

In the energy equation, see Section 18.1, the following term appears in the right-hand side, in term Q:

$$\int_{\Omega_i} \left(\frac{(\mathbf{m} \cdot \mathbf{m})^{n+1}}{\tau \rho^{n+1}} - \frac{(\mathbf{m} \cdot \mathbf{m})^{n}}{\tau \rho^{n}} \right) d\Omega. \tag{18.14}$$

The numerical evaluation of this term is explained in Section 4.5.1, and is implemented in routine `fvbengy.f`.

Chapter 19

Postprocessing

After a certain, by the user defined, number of time-steps, the 'progress' of the computed solution has to be stored. This storing, together with the computation of derived quantities and plotting, is called postprocessing.
Due to the staggered placement of the variables, we have to interpolate all (scalar and vector) quantities to the vertices. This is subject of Section 19.1. The postprocessing itself is divided into two levels, and this is discussed in Section 19.2.

19.1 Interpolation to vertices

The interpolation of scalars (ρ, p and H or h) to vertices is discussed in Section 11.1. The routine doing this interpolation is routine `fvposts.f`. The interpolation of the momentum vector $\mathbf{m}$ to vertices is discussed in Section 11.1. The routine doing this interpolation is routine `fvpostm.f`. In routine `fvpostuv.f` the velocity-vector at the vertices is computed, using simply $u_x = m_x/\rho$ and $u_y = m_y/\rho$, where m_x, m_y and ρ are the interpolated values at the vertices.

A special treatment is made for the total enthalpy. Since we want a unique energy variable in the postprocessing (i.e. we do not want to have `IF`-statements everywhere), in all vertices the enthalpy h is stored. When we have another primary energy variable, this is first transformed to h in all vertices using equations like (2.38) This is done in routine `fvtransenth.f`.

19.2 Two levels of postprocessing

The postprocessing is divided into two levels.

1. Level 1. At (almost) every time-level we write some quantities to file. This must be restricted to a few numbers each time-step in order to avoid memory problems. Think of quantities like lift, number of supersonic vertices etcetera. The aim of this is primarily to check convergence.

2. Level 2. At a few ($\mathcal{O}(10)$) time-levels the complete solution is written to file, to be put in `seppost.f`. Note that this requires much memory, since arrays with length equal to the number of vertices have to be stored.

In Algorithm 22 the postprocessing routine `fvpost.f` is described schematically. Postprocessing level 1 is discussed in Section 19.3, and postprocessing level 2 is discussed in Section 19.4.

Algorithm 22 Postprocessing: fvpost.f

Interpolate quantities ($\mathbf{m}$, ρ, h, p, $\mathbf{u}$) to vertices
if *freqout* is true **then**
  Postprocessing level 1
end if
if *writeoutput* is true **then**
  Postprocessing level 2
end if

19.3 Postprocessing level 1

At (almost) every time-level some quantities are written to file. At which time-levels, and which quantities, depends on what the user specifies in the inputfile `file.prb`, under keyword `FREQUENT_OUTPUT`, see ISNaS Users Guide Section 5.15.
Postprocessing level 1 is controlled by Algorithm 23, a part of `fvpost.f`.
Arrays `ipost` and `rpost` contain relevant information, which is in fact also stored in other

Algorithm 23 Postprocessing level 1

Compute desired quantities
Write desired quantities to file

arrays. These arrays are filled by routine `fvpostia.f`, a subroutine of `fvreadbody.f`. Arrays `ipost` and `rpost` are put in COMMON-blocks `SPcommon/cpost1` and `SPcommon/cpost2` respectively. These arrays are also used for `seppost.f`; see also Section 19.4.

Array ipost: i=1: Length array `ipost` (now: 10).

i=2: Length array `rpost` (now: 10).

i=3: 1 (flow around profile) or 0 (no profile flow).

i=4: curve number of upper surface of profile (only when `ipost(3)` = 1).

i=5: curve number of lower surface of profile (only when `ipost(3)` = 1).

i=7: `mcom`.

Array rpost: i=1: γ

i=2: M_r

i=3: p_∞

i=4: h_∞

i=5: $|\mathbf{m}_\infty|$

i=6: c (length of airfoil)

i=7: α (angle of attack)

The computation of the desired quantities (lift, drag, number of supersonic vertices, position of the sonic points, residuals) is discussed in the following subsections: 19.3.1, 19.3.2, 19.3.3 and 19.3.4. Writing the desired quantities to file `sepcomp.freq` is done in routine `fvwrite.f`.

19.3.1 Computation of lift and drag coefficient

Routine `fvpslift.f` computes the lift and drag coefficient c_l and c_d (`rlift` and `rdrag`). Note

Algorithm 24 fvpslift.f

Find number of vertices at upper and lower surface (`fvlnodprf.f`)
Find vertex numbers at upper and lower surface (`fvnnodprf.f`)
Compute coordinates of vertices at airfoil, and pressure coefficient c_p in each vertex at airfoil (`fvpslift01`)
Compute lift and drag coefficient (`fvpslift02`)

that the leading edge has to be in the origin always, and the trailing edge at position $(0, c)$. The pressure-coefficient at the airfoil is computed using the equations given in Section 12.3.1. Note that ρ_∞, h_∞, p_∞ and $|\mathbf{m}_\infty|$ are present in array `rpost`.

Given the scaled coordinates of the profile, together with the c_p-values at all vertices, the lift and drag coefficient can be computed, routine `fvpslift02.f`. More information on this subject can be found in Section 12.3.

19.3.2 Minimum and maximum Mach number, and number of supersonic vertices

Routine `fvpsmimxmach.f` computes, depending on input-parameter `ichoice`, the number of supersonic vertices, i.e. the number of vertices in the domain with a Mach number larger than 1 (integer `nsupersvert`), or the minimum and maximum Mach number in the domain (real array `mimxmach`). The Mach number in a vertex is computed using (2.45), where we use $u^2 = (u_x^2 + u_y^2)$.

19.3.3 Position of sonic points

Routine `fvpssonic.f` computes the positions, the scaled x-coordinate to be more precise, of the sonic points at the airfoil, and puts them in array `rpossonic(4)`. Sonic points are points with Mach number equal to 1. In this array, the first two positions are reserved for the sonic points at the upper part of the airfoil, and the last two positions are reserved for the sonic points at the lower part of the airfoil. Routines `fvlnodprf` and `fvnnodprf`, discussed already in 19.3.1, find the numbers of the vertices at the airfoil. Routine `fvpssonic01` computes then the position of the sonic points at the airfoil. This is done as follows: in all vertices at the airfoil the Mach number is computed using equation (2.45). Say that at x_i the Mach number M_i is subsonic/supersonic and at x_{i+1} the Mach number M_{i+1} is supersonic/subsonic, hence a sonic point is present in the domain $[x_i, x_{i+1}]$. Then we assume the Mach number, for $x_i \leq x \leq x_{i+1}$, to behave as

$$M = ax + b, \tag{19.1}$$

with

$$a = \frac{M_{i+1} - M_i}{x_{i+1} - x_i} \qquad b = M_i - ax_i = M_{i+1} - ax_{i+1}. \tag{19.2}$$

The position of the sonic point (M=1) then follows from:

$$x_{sp} = \frac{1-b}{a}. \tag{19.3}$$

19.3.4 Computation of residuals

Another quantity of importance, is the residual (the difference between two subsequent solution vectors in a certain norm). How smaller the residual, the closer to the stationary solution. Therefore the time-history of the residual can be used for convergence investigation.
At this moment we have, for $||\rho^{n+1} - \rho^n||$, $||m^{n+1} - m^n||$ and $||h^{n+1} - h^n||$ (n: time-level), the following norms, with $u = (u_1, u_2, \ldots, u_p)$, implemented, in routine `fvnorm.f`:

- 1 norm:

$$||u||_1 = \frac{\sum_{i=1}^{p} |u_i|}{p} \tag{19.4}$$

- 2 norm:

$$||u||_2 = \sqrt{\frac{\sum_{i=1}^{p} u_i^2}{p}} \tag{19.5}$$

- ∞ norm:

$$||u||_\infty = \max(|u_i|) \tag{19.6}$$

19.4 Postprocessing level 2

In `fvpost.f` at a few time-levels the whole solution (interpolated to the vertices) is written to files `sepcomp.inf` and `sepcomp.out`. In `seppost.f` then all desired variables in the vertices are computed, using, among others, the routines mentioned in this section. The connecting routine is `insp01.f`. The computation of the Mach number is discussed in Section 19.4.1. The computation of the total enthalpy is discussed in Section 19.4.2. The computation of the stagnation pressure is discussed in Section 19.4.3.

19.4.1 Computation of Mach number

The Mach number is computed using routine `prmachno.f`, using relation (2.45).

19.4.2 Computation of total enthalpy

The dimensionless total enthalpy H is computed using relation (2.38) in routine `prtotenth.f`.

19.4.3 Computation of stagnation pressure

The stagnation pressure p_0, following from

$$p_0 = p\left[1 + \frac{1}{2}(\gamma - 1)M^2\right]^{\gamma/\gamma-1}, \tag{19.7}$$

is computed in routine `prstagpres.f`.

19.4.4 Computation of pressure coefficient c_p

The pressure coefficient c_p, see Section 12.3.1 is computed in routine `prcpcoef.f`.

19.4.5 Computation of entropy S

The entropy S is computed in routine `prentropy.f`, using the relation

$$S = \ln(p/\rho^\gamma), \tag{19.8}$$

where p and ρ are dimensionless.
For the Mach-uniform pressure-correction formulation, another non-dimensionalization has been used, resulting in the following expression for the entropy:

$$S = \ln\left(\frac{1 + \gamma M_r^2 p}{\rho^\gamma}\right). \tag{19.9}$$

Chapter 20

General print routines

In this section some print-routines are described. All print-routines for the finite-volume unstructured flow solver start with the letters `fvprin`.

20.1 Subroutine fvprinbc

Description :

Prints the content of arrays `iinbc` and `rinbc`, arrays dealing with the boundary conditions.

Heading :

```
subroutine fvprinbc ( iinbc, rinbc, ncurvs, ndegfd )
```

Parameters :

iinbc : Array containing integer information of the boundary conditions

rinbc : Array containing real information of the boundary conditions

ncurvs: Number of curves; given in the mesh-input-file

ndegfd: Number of degrees of freedom

Output :

Arrays `iinbc` and `rinbc`.

Position where to insert in the software :

In `fvinput.f` after the call to `fvreadal.f`. Insert `ndegfd = ibuffr(ipiinput-1+55)` before calling to `fvprinbc`.

20.2 Subroutine fvprinbnd

Description :

Prints the content of arrays `ibndcon` and `rbndcon`, arrays containing info of the boundary conditions for the equation under consideration.

Heading :

```
subroutine fvprinbnd ( ibuffr, kmesh, ibndcon,
                       rbndcon, nfacesb, ndim, iseq )
```

Parameters :

ibuffr : Array containing integer information of the problem

kmesh : Array containing mesh-information.

convarray: Array containing the inverse of array `iconface`, see KMESH part y.

ibndcon : Array containing integer information of the boundary conditions, for every boundary face, for the equation under consideration.

rbndcon : Array containing integer information of the boundary conditions, for every boundary face, for the equation under consideration.

nfacesb : Number of boundary faces.

ndim : Dimension of the problem

iseq : Number of the equation under consideration.

Output :

Arrays `ibndcon` and `rbndcon`.

Position where to insert in the software :

In routine `fvbldmat.f`, before the call to the equation under consideration.

20.3 Subroutine fvprincv

Description :
Prints face-number per curve.

Heading :
`subroutine fvprincv ( ibuffr, kmesh )`

Parameters :

ibuffr: Array containing integer information of the problem

kmesh : Array containing mesh-information.

Output :
Prints per curve, the face-numbers on that curve.

Position where to insert in the software :
At the end of `fvstart.f`.

20.4 Subroutine fvprinin

Description :
Prints the contents of arrays `iinput` and `rinput`.

Heading :
`subroutine fvprinin ( iinput, rinput, kmesh )`

Parameters :

iinput: Array containing integer user input.

rinput: Array containing real user input

kmesh : Array containing mesh-information.

Output :
Prints numerical constants with respect to the problem under consideration; prints arrays `iincof` and `rincof`; prints arrays `iincnd` and `rincnd`; prints arrays `iintim` and `rintim`; prints arrays `iinsol` and `rinsol`; prints arrays `iindsc` and `rindsc`; prints arrays `iintur` and `rintur`; prints arrays `iincom` and `rincom`; prints arrays `iinseq` and `rinseq`, and prints arrays `iincav` and `rincav`.

Position where to insert in the software :
At the end of `fvstart.f`. Insert then in `fvstart.f` :
`call fvprinin ( ibuffr(ipiinput), buffer(iprinput), kmesh )`.

20.5 Subroutine fvprinms

Description :

Prints the contents of KMESH part y and some additional info. In fact, it prints mesh-related information that is used for the unstructured finite-volume solver.

Heading :

```
subroutine fvprinms ( ibuffr, buffer, kmesh )
```

Parameters :

ibuffr: Array containing integer information of the problem

buffer: Array containing real information of the problem

kmesh : Array containing mesh-information.

Output :

Prints numerical constants with respect to the mesh; prints array `face`; prints array `iconcell`; prints array `iconface`; prints array `lengthf`; prints array `areafv`; prints array `cellfv`; prints array `coor` (the coordinates of the vertices); prints area of largest and smallest cell, together with its coordinates, and prints lenght of largest and smallest faces, together with its coordinates.

Position where to insert in the software :

At the end of `fvstart.f`.

20.6 Subroutine fvprinmt

Description :
Prints the matrix-structure (see Section 16.3), the matrix itself and the right-hand side.

Heading :

```
subroutine fvprinmt ( lmstrmat, nusol, istrmat,
                      mstrmat, rhsd, eq )
```

Parameters :

lmstrmat: Length of array `mstrmat`

nusol : Length of array `rhsd` (always `nelem` or `nfaces`).

istrmat : Integer array containing matrix structure.

mstrmat : Real array containing matrix elements.

rhsd : Real array containing right-hand side.

eq : Integer indicating equation number; see Section 14.1.

Output :
Prints non-zero elements of the matrix.

Position where to insert in the software :
At the end of routine `fvcvdf.f` or `fvprmom.f`, depending on the equation you want to consider. In `fvcvdf.f`:

```
call fvprinmt ( lmstrcvdf, nelem, ibuffr(ipistrcvdf),
                buffer(ipmstrcvdf), buffer(ipirhsd), 4 )
```

or, in `fvprmom.f`:

```
call fvprinmt ( lmstrmom, nfaces, ibuffr(ipistrmom),
                buffer(ipmstrmom), buffer(ipirhsd), 1 )
```

20.7 Subroutine fvprinv

Description :
Prints the solution vector.

Heading :

```
subroutine fvprinv (ibuffr, buffer, kfinvol, eq, isol, numsol, ntimlv, ichoice)
```

Parameters :

ibuffr: Array containing integer information of the problem

buffer: Array containing real information of the problem

kfinvol: Array containing integer information with respect to memory management numbers; see Section 14.3.

eq : Integer indicating equation number; see Section 14.1.

isol : Array with integer information concerning the solution vector; see Section 14.2.1.

numsol: Number of solutions; see Section 14.1.

ntimlv: Number of time-levels for which the solution is stored, is usually 2. See Section 14.2.1.

ichoice: Choice parameter.

Output :
`ichoice` $= 0$. Prints the length of the solution vector of the equation under consideration, the solution vector itself, and the solution vector at the previous time-level.
`ichoice` $\neq 0$. Prints quantities of solution vector at the new time-level, using routine `fvprinv1.f`. The parameter `ichoice` is then passed to `fvprinv1.f`.

Position where to insert in the software :
In `fvshift.f`, after the call to `fvvelo01.f`.

20.8 Subroutine fvprinv1

Description :

Prints (quantities of) a vector containing real elements.

Heading :

```
subroutine fvprinv1 ( vector, length, ichoice )
```

Parameters :

vector: Vector, containing real elements, under consideration

length: Length of the vector

ichoice: Integer to be chosen by user

`ichoice` = 1: prints average of the absolute values of all vector-components (i.e. $\sum_{i=1}^{n} |x_i|/n$), and the maximum of the absolute value of all vector components (i.e. $\max\{|x_1|, \ldots, |x_n|\}$). With n the length of the vector is intended.
`ichoice` = 2: prints average of all vector-components (i.e. $\sum_{i=1}^{n} x_i/n$) and prints the maximum of all vector components (i.e. $\max\{x_1, \ldots, x_n\}$).
`ichoice` = 3: prints absolute values of all vector components (i.e. $|x_1|, \ldots, |x_n|$).
`ichoice` = 4: prints all vector components (i.e. $x_1, \ldots, x_n$).

Output :

Prints the length of the solution vector of the equation under consideration, the solution vector itself, and the solution vector at the previous time-level.

Position where to insert in the software :

In `fvsbstep.f`, after the call to `fvsolve.f`.

20.9 Subroutine fvprinv2

Description :

Prints (quantities of) a vector containing of integer elements. Routine `fvprinv2.f` is the integer version of `fvprinv1.f`.

20.10 Subroutine fvcheck

Description :

A routine that can be used to check whether the matrix is built correctly.
The exact matrix A_{ex}, right-hand side $\mathbf{b}_{ex}$ and solution-vector $\mathbf{x}_{ex}$ satisfy: $A_{ex}\mathbf{x}_{ex} = \mathbf{b}_{ex}$. The computed matrix A_{comp} and right-hand side $\mathbf{b}_{comp}$ lead to solution-vector $\mathbf{x}_{comp}$, according to $A_{comp}\mathbf{x}_{comp} = \mathbf{b}_{comp}$. (Of course, we want $A_{comp} = A_{ex}$ and $\mathbf{b}_{comp} = \mathbf{b}_{ex}$, leading to $\mathbf{x}_{comp} = \mathbf{x}_{ex}$.) Suppose we know the exact solution of a certain problem, i.e. we know $\mathbf{x}_{ex}$ and $\mathbf{b}_{ex}$. Then we can check whether the matrix A_{comp} is computed correctly, by considering $(\mathbf{b}_{ex} - A_{comp}\mathbf{x}_{ex}) = \mathbf{r}$. The elements in vector $\mathbf{r}$ that are not equal to zero, correspond to either i) elements (cells or faces) for which the matrix is not built correctly, or to ii) elements (cells or faces) for which the matrix cannot represent the exact solution.
The error of solution i is defined as: $e_i = |x_{i,comp} - x_{i,ex}|$. The average error follows from: $\bar{e} = \sum_{i=1}^{n} e_i/n$, with n the dimension of vector $\mathbf{x}$. The maximum error e_{max} follows from: $e_{max} = \max(e_i)$.
Note that this whole procedure has only sense, when we give the exact solution as initial condition, and that we consider solely the computation after **one** time-step.

Heading :

```
subroutine fvcheck ( ibuffr, buffer, kfinvol, matrix, isol, intmat,
                     numsol, ntimlv, numunk, irhsd, kprob, eq,
                     ipar )
```

Parameters :

ibuffr: Array containing integer information of the problem

buffer: Array containing real information of the problem

kfinvol: Array containing integer information with respect to memory management numbers; see Section 14.3.

matrix: Integer array with information concerning the matrix; see Section 14.2.2.

isol : Array with integer information concerning the solution vector; see Section 14.2.1.

intmat: Array with integer information concerning matrix storage; see Section 14.2.3.

numsol: Number of solutions; see Section 14.1.

ntimlv: Number of time-levels for which the solution is stored, is usually 2. See Section 14.2.1.

numunk: Number of unknowns; see Section 14.1.

irhsd : Integer array with information concerning the right-hand side; see Section 14.2.2.

kprob : Integer array with information concerning problem-definition.

eq : Integer indicating equation number; see Section 14.1.

ipar : Integer to be chosen by user:
`ipar` = 1: prints $\mathbf{b}_{ex}$ and $\mathbf{b}_{comp}$.

`ipar` = 2: prints ($\mathbf{b}_{ex} - A_{comp}\mathbf{x}_{ex}$). Note that $\mathbf{b}_{ex}$ and $\mathbf{x}_{ex}$ must be known.
`ipar` = 3: prints $\bar{e}$ and e_{max}. Note that $\mathbf{x}_{ex}$ must be known.
`ipar` = 4: prints $\mathbf{e}$. Note that $\mathbf{x}_{ex}$ must be known.
`ipar` = 5: prints $\bar{r}$ and r_{max}. Note that $\mathbf{x}_{ex}$ must be known.

Output :

Depending on parameter `ipar`, the arrays $\mathbf{b}_{ex}$ and $\mathbf{b}_{comp}$, or ($\mathbf{b}_{ex} - A_{comp}\mathbf{x}_{ex}$) are printed.

Position where to insert in the software :

In routine `fvsbstep.f`, after the call to `fvsolve.f`.

Chapter 21

Computation of gradients

The methods discussed in Section 5.7, are ways of computing first order or second order derivatives. In this chapter, its implementation for computation of first order derivatives, of the form $(\nabla p \cdot \mathbf{N})_i$, where i is a face, and variable p is positioned at the cell-centers, is discussed.

21.1 Computation of gradients in the software

The main routine for the computation of everything that is related to the computation of gradients, is routine `fvgrad.f`.
For all faces (internal and boundary), we compute the coefficients γ once, and store them in array `rgrfac(6,nfaces)`. The second index runs from 1 to `nfaces`, and refers to the face. In addition, we store the required cell-numbers in `igrfac(6,nfaces)`. The elements `igrfac` and `rgrfac` are computed in routine `fvgradif.f`. The ordering of the cell numbers to which the elements correspond is the same as in Figure 21.1 for the path-integral method for a six-point, four-point and three point-stencil and the auxiliary point method. For the four-point method, the numbering is like the one in Figure 5.7, where a corresponds to `igrfac(1,i)`, b to `igrfac(2,i)`, and so on (hence `igrfac(5,i)` and `igrfac(6,i)` remain empty). Note that, since we have only implemented the gradient methods for internal faces, the columns in `rgrfac` and `igrfac` corresponding to boundary faces, are put to zero. The actual computation of the pressure gradient coefficients `rgrfac(6,nfaces)` is done in routine `fvgrad01.f`. Hence, with `prescv(k)` the pressure at cell k and i the face-number, the surrounding cell-numbers follow from `igrfac(1,i)`,...,`igrfac(6,i)`, and we find that:

$$(\text{grad}\, p)_i \cdot \mathbf{N}_i = \texttt{rgrfac(1,i)} \cdot \texttt{prescv(igrfac(1,i))} + \ldots + \\ + \texttt{rgrfac(6,i)} \cdot \texttt{prescv(igrfac(6,i))}.$$

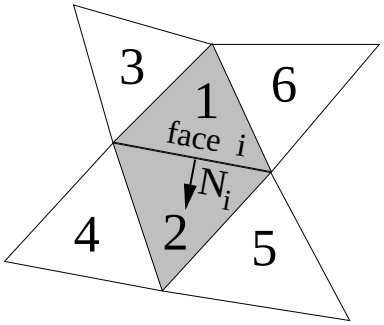

Figure 21.1: Situation for path-integral method.

21.2 Implementation of path-integral method for a six-point stencil

In the software, the path-integral method using a six-point stencil is chosen to be the default choice for computing gradients. It is indicated by parameter `ipresgrad` = 0.
Repeating the equations to arrive at $(\nabla p)_i$ from Section 5.7.1, see also Figure 21.1:

$$\begin{aligned} p_2 - p_1 &= (\nabla p)_i \cdot (\mathbf{x}_2 - \mathbf{x}_1) \\ p_3 - p_6 + p_4 - p_5 &= (\nabla p)_i \cdot (\mathbf{x}_3 - \mathbf{x}_6 + \mathbf{x}_4 - \mathbf{x}_5). \end{aligned} \tag{21.1}$$

Note that the second equation is a combination of equations

$$\begin{aligned} p_3 - p_5 &= (\nabla p)_i \cdot (\mathbf{x}_3 - \mathbf{x}_5) \\ p_4 - p_6 &= (\nabla p)_i \cdot (\mathbf{x}_4 - \mathbf{x}_6). \end{aligned} \tag{21.2}$$

Difficulties arise when one or more cells of the group $\{3, 4, 5, 6\}$ are absent. For example, when cell 3 is absent, we replace the first equation of (21.2) by $p_1 - p_5 = (\nabla p)_i \cdot (\mathbf{x}_1 - \mathbf{x}_5)$. If we now define the following things:

- if cell 3 is absent, then we put $\mathbf{x}_3 = \mathbf{x}_1$ and $p_3 = p_1$.
- if cell 4 is absent, then we put $\mathbf{x}_4 = \mathbf{x}_2$ and $p_4 = p_2$.
- if cell 5 is absent, then we put $\mathbf{x}_5 = \mathbf{x}_2$ and $p_5 = p_2$.
- if cell 6 is absent, then we put $\mathbf{x}_6 = \mathbf{x}_1$ and $p_6 = p_1$.

In doing this, we can incorporate automatically the possible absence of cells by rewritting (21.2) as:

$$\begin{aligned} p_3 - p_1 + p_1 - p_5 &= (\nabla p)_i \cdot (\mathbf{x}_3 - \mathbf{x}_1 + \mathbf{x}_1 - \mathbf{x}_5) \\ p_4 - p_2 + p_2 - p_6 &= (\nabla p)_i \cdot (\mathbf{x}_4 - \mathbf{x}_2 + \mathbf{x}_2 - \mathbf{x}_6). \end{aligned} \tag{21.3}$$

Using this, equations (21.1) can be rewritten as:

$$\begin{aligned} p_2 - p_1 &= (\nabla p)_i \cdot (\mathbf{x}_2 - \mathbf{x}_1) \\ p_3 - p_1 + p_1 - p_5 + p_4 - p_2 + p_2 - p_6 &= (\nabla p)_i \cdot (\mathbf{x}_3 - \mathbf{x}_1 + \mathbf{x}_1 - \mathbf{x}_5 + \mathbf{x}_4 - \mathbf{x}_2 + \mathbf{x}_2 - \mathbf{x}_6). \end{aligned} \tag{21.4}$$

Writing this set of equations as

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} \partial p/\partial x \\ \partial p/\partial y \end{bmatrix} = \begin{pmatrix} p_2 - p_1 \\ p_3 - p_6 + p_4 - p_5 \end{pmatrix}, \tag{21.5}$$

where

$$\begin{aligned} a_{11} &= x_2 - x_1 & a_{12} &= y_2 - y_1 \\ a_{21} &= x_3 - x_1 + x_1 - x_5 + x_4 - x_2 + x_2 - x_6 & a_{22} &= y_3 - y_1 + y_1 - y_5 + y_4 - y_2 + y_2 - y_6. \end{aligned} \tag{21.6}$$

Solution of this linear system leads to pressure-gradient $(\nabla p)_i$ in terms of matrix-coefficients $a_{11}, \ldots, a_{22}$ and pressure-values $p_1, \ldots, p_6$. The inner product of $(\nabla p)_i$ with respect to $\mathbf{N}_i = (N_x, N_y)$ then results in

$$(\nabla p)_i \cdot \mathbf{N}_i = \sum_{j=1}^{6} \gamma_j p_j, \tag{21.7}$$

where the coefficients γ follow from

$$\begin{aligned} -\gamma_1 = \gamma_2 &= \frac{a_{22}N_x - a_{21}N_y}{a_{11}a_{22} - a_{12}a_{21}} \\ \gamma_3 = \gamma_4 = -\gamma_5 = -\gamma_6 &= \frac{a_{11}N_y - a_{12}N_x}{a_{11}a_{22} - a_{12}a_{21}}. \end{aligned} \tag{21.8}$$

Note that consistency is satisfied:

$$0 = \sum_{j=1}^{6} \gamma_j. \tag{21.9}$$

When, for example, cell 3 is absent, the coefficients γ' that must be used to compute the pressure gradient, are changed according to:

$$\gamma_1' = \gamma_1 + \gamma_3 \qquad \gamma_3' = 0. \tag{21.10}$$

Similar relations hold when other cells are absent.

Chapter 22

Miscellaneous

In this section some miscellaneous aspect are gathered. In Section 22.1 an example is given how to insert boundary conditions that depend on time and/or position, or initial conditions that depend on position in the input-files. In Section 22.2 some remarks are made concerning testing of the software in cases that an exact solution is available. In Section 22.3 a description is given of routines that compare the exact tangential momentum at each face (note: this means that the exact solution must be known) with the interpolated tangential momentum.

22.1 Variable boundary and initial conditions

In many occasions the boundary and/or initial conditions will be functions of time and/or place. (Of course, the initial conditions may only depend on place, and not on time). Here below the inputfiles of a simple example (flow in a channel, with $h = 1$, $M = 0$, $\rho = 1$, $\mathbf{m} = \mathbf{u} = (x, -y)$) are given.

Mesh (file-name `channel.msh`):

```
      mesh2d
 points
    p1=(0,0)
    p2=(3,0)
    p3=(3,3)
    p4=(0,3)
 curves
    c1 = line1(p1,p2,nelm=3)
    c2 = line1(p2,p3,nelm=3)
    c3 = line1(p3,p4,nelm=3)
    c4 = line1(p4,p1,nelm=3)
 surfaces
    s1 = quadrilateral3(c1,c2,c3,c4)
       *  refine 1
 check_level = 2
 plot(jmark=3, ren_plot)
      end
```

Input (file-name `channel.prb`):

```
      *
      *   Input for the channel problem
      *
      number_of_transport_equations = 1
      compressible
  mach = 0.0d0
      initial_conditions
 u_momentum = func = 1
 v_momentum = func = 2
 pressure = 0.0d0
 transport 1 =  1.0d0
      time_integration
 tinit = 0
 tend =  20
 tstep = 0.01d0
 theta = 1
 rel_stationary_accuracy = 1d-2
      boundary_conditions
 curve 1 to 4: ux = func = 1, uy = func = 2,
```

```
                      transport 1 = dirichlet = 1.0d0
     coefficients
 momentum_equations
    mu    = 0.0d0
     linear_solver
 momentum_equations
    amount_of_output = 0
    relaccuracy = 1d-6
    divaccuracy = 0
 pressure_equations
    amount_of_output = 0
    divaccuracy = 0
    startvector = zero
 transport_equation = 1
    amount_of_output = 0
    divaccuracy = 0
    startvector = zero
```

File containing functions (file-name `channel.f`):

```
      program channel
      implicit none

      integer pbuffr
      parameter( pbuffr = 5000000 )
      integer ibuffr
      double precision buffer(pbuffr/2)
      common ibuffr(pbuffr)
      equivalence ( ibuffr(1), buffer(1) )

      call fvcomput ( ibuffr, buffer, pbuffr )

      end

c     --- Function with respect to boundary conditions

      function usfunb ( ichoice, x, y, z, t )

c     User written function subroutine. It gives
c     the user the opportunity to define a
c     boundary condition as a function of space
c     and time.

      implicit none

      double precision usfunb, x, y, z, t
      integer ichoice
```

```
c     x        i    x-coordinate
c     y        i    y-coordinate
c     t        i    actual time
c     ichoice  i    choice parameter given by the user input
c     usfunb   o    computed boundary condition

         if ( ichoice.eq.1 ) then
            usfunb = x
 else if ( ichoice.eq.2 ) then
    usfunb = -y
         end if

      end

c     --- Function with respect to initial conditions

      function usfuni ( ichoice, x, y, z )

c     User written function subroutine. It gives
c     the user the opportunity to define a
c     coefficient as a function of space
c     and time.

      implicit none

      double precision usfuni, x, y, z
      integer ichoice

c     x        i    x-coordinate
c     y        i    y-coordinate
c     z        i    z-coordinate
c     ichoice  i    choice parameter given by the user input
c     usfunc   o    computed coefficient

      if ( ichoice.eq.1 ) then
         usfuni = x
      else if ( ichoice.eq.2 ) then
         usfuni = -y
      end if

      end
```

22.2 Exact solution is known

There are two distinct cases to be distinguished, discussed in two subsections.

22.2.1 Case 1

For testing purposes, problems for which exact solutions are available, are used. In this section an example is given of how to incorporate the exact solution in the software such that the testing can take place.

In the example of the previous section, a stationary solution ($\partial\mathbf{m}/\partial t = \mathbf{0}$) is found when the right-hand equals the vector $\mathbf{f} = (x, y)$. The momentum equation then reads:

$$\frac{\partial \mathbf{m}}{\partial t} + \nabla(\mathbf{um}) = \mathbf{f} \tag{22.1}$$

This means that the pressure is given by $p(x, y) = p_0 - \frac{1}{2}x^2 - \frac{1}{2}y^2$, with p_0 an arbitrary constant. The right-hand side of the discretized momentum equation (projection on normal $\mathbf{n}$; integration over area Ω) equals: $-\Omega\nabla p \cdot \mathbf{n} = -\Omega(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}) \cdot (n_x, n_y) = \Omega(xn_x + yn_y)$. Note that, in the fully compressible case, the right-hand side has to be multiplied by $1/\gamma M_r^2$. The vector-function $\mathbf{f}$ is inserted in file `channel.f` as follows:

```
c     --- Function with respect to right-hand side

      function usfunc ( ichoice, x, y, z, t )

c     User written function subroutine. It gives
c     the user the opportunity to define a
c     coefficient as a function of space
c     and time.

      implicit none

      double precision usfunc, x, y, z, t
      integer ichoice

c     t        i    actual time
c     x        i    x-coordinate
c     y        i    y-coordinate
c     z        i    z-coordinate
c     ichoice  i    choice parameter given by the user input
c     usfunc   o    computed right-hand side

      if ( ichoice.eq.1 ) then
         usfunc = x
      else if ( ichoice.eq.2 ) then
         usfunc = y
      end if
```

```
      end
```

The routine adding the number -$\Omega \nabla p \cdot \mathbf{n}$ to the right-hand side is called `fvmkrhsd.f`, which must be put at an appropriate position (after initialization of the right-hand side vector, and before the actual solving of the matrix-vector equation) in `fvprmom.f`.
In the input-file `channel.prb` the following statement has to be added:

```
coefficients
   momentum_equations
      force1 = func = 1
      force2 = func = 2
```

22.2.2 Case 2

In case that the exact solution is known, then one is interested in the error, defined as the difference between the exact and numerical solution. Define $\mathbf{e} = (e_1, \ldots, e_N)$ as the error, with N the total number of unknowns (equal to the number of cells for the pressure and equal to the number of faces for the velocity). The following norms are defined:

- L_∞-norm: $|\mathbf{e}|_\infty = \max_i |e_i|$.
- L_2-norm: $|\mathbf{e}|_2 = \sqrt{\sum_i e_i^2/N}$.
- Weighted L_2-norm: $|\mathbf{e}|_2 = \sqrt{\sum_i \Omega_i e_i^2 / \sum_i \Omega_i}$, where Ω_i represents the area of the control volume.

In order to compute these errors, remark the following:

- insert the key-word `exacterror` as subkeyword of `time\_integration`.
- put the exact solution as a function `usfune ( ichoice, x, y, z )` in the main program. The following convention is adopted: `ichoice` = 1 corresponds to the x-component of the velocity; `ichoice` = 2 corresponds to the y-component of the velocity; `ichoice` = 3 corresponds to the pressure.
- until so far it is only implemented for 2D incompressible flows.

An example is the testproblem `pois\_unstr02`

22.3 Exact and computed tangential momentum

The computation of tangential momentum components $\tilde{m}$ using surrounding normal momentum components m is discussed in the mathematical sections. In order to see how accurate the interpolations reproduce the exact tangential momentum, routine `fvtmom01.f` has been written. The routine `fvtmom01.f` yields for every face at which the tangential momentum is not prescribed, the absolute value of the reconstruction error, i.e. $d = |\tilde{m}_{exact} - \tilde{m}_{exact}|$. This quanity is put in the array `diftmom(nfaces)`. Inserting this piece of code goes as follows, by making the following changes in the software:

- Add in `fvprmom.f` the following items at the appropriate positions:

```
call ini080 ( ibuffr, mmdiftmom, nfaces, 'diftmom' )

ipdiftmom = inidgt( mmdiftmom )

do i = 1, nfaces
   buffer(ipdiftmom+i-1) = 0.0d0
end do

call ini066 ( ibuffr, mmdiftmom )
```

Furthermore, the array `diftmom` must be given in the CALL to `fvprmomr` and `fvprmomq` by using: `buffer(ipdiftmom)`.

- Add array `diftmomint(nfaces)` to the parameter-lists of `fvprmomr` and `fvprmomq`.
- The CALL to `fvtmom01` must be done after the CALL to `fvupmomr` and `fvupmomq` in routines `fvprmomr` and `fvprmomq`.
- Add in `fvprmomr.f` the quantities `imombnd(1)` = ... = `imombnd(4)` = 0.
- Insert a write-statement after the call to `fvprmomb`, for example `call fvprinv1 ( buffer(ipdiftmom), n` (note that this must be done before the statement `call ini066 ( ibuffr, mmdiftmom )`.

Chapter 23

Appendix

No content yet.

———————— Backmatter of the document

Bibliography

[1] H. Bijl and P. Wesseling. A unified method for computing incompressible and compressible flows in boundary-fitted coordinates. *J. Comp. Phys.*, 141:153–173, 1998.

[2] D.J. Ewing, A.J. Fawkes, and J.R. Griffiths. Rules governing the numbers of nodes and elements in a finite element mesh. *Int. J. Num. Meth. in Eng.*, 2:597, 1970.

[3] F.H. Harlow and J.E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids*, 8:2182–2189, 1965.

[4] C. Hirsch. *Numerical Computation of Internal and External Flows. Vol.2: Computational Methods for Inviscid and Viscous Flows.* Wiley, Chichester, 1990.

[5] J.C. Tannehill, D.A. Anderson, and R.H. Pletcher. *Computational Fluid Dynamics and Heat Transfer.* Taylor and Francis, London, 1997.

[6] F. Vermolen and K. Vuik. A vector valued Stefan problem from aluminium industry. Report MAS-R 9814, CWI, Amsterdam, 1998.

[7] P. Wesseling, A. Segal, and C.G.M. Kassels. Computing flows on general three-dimensional nonsmooth staggered grids. *J. Comp. Phys.*, 149:333–362, 1999.

[8] P. Wesseling, A. Segal, C.G.M. Kassels, and H. Bijl. Computing flows on general two-dimensional nonsmooth staggered grids. *J. Eng. Math.*, 34:21–44, 1998.

[9] P. Wesseling, M. Zijlema, A. Segal, and C.G.M Kassels. Computation of turbulent flow in general domains. *Math. Comp. Sim.*, 44:369–385, 1997.

[10] M. Zijlema and P. Wesseling. Higher-order flux-limiting schemes for the finite volume computation of incompressible flow. *Int. J. Comp. Fluid Dyn.*, 9:89–109, 1998.