Whitepaper:
Incompressible Navier-Stokes equations using Finite Volume spatial discretization

April, 2020

Jan Vermaak

Rev 0.00

Contents

List of Figures

List of Tables

1 The incompressible Navier-Stokes equations

Conservation of mass (continuity equation):

$$\frac{\partial\rho}{\partial t}+\boldsymbol{\nabla}\boldsymbol{\cdot}(\rho\mathbf{u})=0 \tag{1.1}$$

where ρ is the fluid density, $\mathbf{u}$ is the fluid velocity vector with components $\mathbf{u}=[u_x,u_y,u_z]$.

Conversion of momentum:

$$\frac{\partial(\rho\mathbf{u})}{dt}+\boldsymbol{\nabla}\boldsymbol{\cdot}(\rho\mathbf{u}\otimes\mathbf{u})=\mathbf{f}-\boldsymbol{\nabla}p+\boldsymbol{\nabla}\boldsymbol{\cdot}\boldsymbol{\tau} \tag{1.2}$$

where $\mathbf{f}$ is a body force vector (gravity,magnetism,etc.), p is scalar static pressure, $\boldsymbol{\tau}$ is the viscous shear-stress forces defined as

$$\boldsymbol{\tau}=\boldsymbol{\nabla}\boldsymbol{\cdot}\{\mu\boldsymbol{\nabla}\mathbf{u}\}+\boldsymbol{\nabla}\boldsymbol{\cdot}\{\mu(\boldsymbol{\nabla}\mathbf{u})^T\}-\frac{2}{3}\boldsymbol{\nabla}(\mu\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}).$$

The components of $\boldsymbol{\tau}$ can also be denoted with indices along $i,j=1,2,3$. The entries of a combined tensor representation is defined by

$$\tau_{ij}=\mu\left[\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}-\frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right] \qquad k=1,2,3$$

with $x_0=x$, $x_1=y$ and $x_2=z$. This tensor defines the shear-stress forces along the six cartesian planes as

$$\begin{aligned}
\tau_{xx}&=\frac{2}{3}\mu\left(2\frac{\partial u_x}{\partial x}-\frac{\partial u_y}{\partial y}-\frac{\partial u_z}{\partial z}\right)\\
\tau_{yy}&=\frac{2}{3}\mu\left(-\frac{\partial u_x}{\partial x}+2\frac{\partial u_y}{\partial y}-\frac{\partial u_z}{\partial z}\right)\\
\tau_{zz}&=\frac{2}{3}\mu\left(-\frac{\partial u_x}{\partial x}-\frac{\partial u_y}{\partial y}+2\frac{\partial u_z}{\partial z}\right)\\
\tau_{xy}&=\tau_{yx}=\mu\left(\frac{\partial u_y}{\partial x}+\frac{\partial u_x}{\partial y}\right)\\
\tau_{xz}&=\tau_{zx}=\mu\left(\frac{\partial u_z}{\partial x}+\frac{\partial u_x}{\partial z}\right)\\
\tau_{yz}&=\tau_{zy}=\mu\left(\frac{\partial u_z}{\partial y}+\frac{\partial u_y}{\partial z}\right)
\end{aligned}$$

or more compactly

$$\tau=\begin{bmatrix}\tau_{xx}&\tau_{xy}&\tau_{xz}\\ \tau_{yx}&\tau_{yy}&\tau_{yz}\\ \tau_{zx}&\tau_{zy}&\tau_{zz}\end{bmatrix}$$

2 Some necessary tools

2.1 Tensors

2.1.1 Tensor product of two vectors, a⊗b

Also called the dyadic product. In many textbooks the general notation $\{\mathbf{uu}\}$ or $\{\boldsymbol{\nabla}\mathbf{u}\}$ is adopted which might not be intuitive to scholars unfamiliar with tensors. In this notation the $\{\}$ indicates that the terms within forms a tensor. The more technically correct representation would be the **tensor product notation** where we take the tensor product of two vectors, i.e., **a** and **b**, both of the same dimension N, resulting in tensor of order 2, as

$$\{\mathbf{ab}\} = \mathbf{a}\otimes\mathbf{b} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} \otimes \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{N-1} \end{bmatrix} = \begin{bmatrix} a_0 b_0 & a_0 b_1 & \dots & a_0 b_{N-1} \\ a_1 b_0 & a_1 b_1 & \dots & a_1 b_{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1} b_0 & a_{N-1} b_1 & \dots & a_{N-1} b_{N-1} \end{bmatrix}. \tag{2.1}$$

Note here that the "order" of the tensor is also sometimes referred to as the tensor's rank. A simple definition of "rank" is the number of indices in a given dimension required to address the components of a tensor. Therefore the above tensor has dimension N and rank 2. A scalar has dimension 1 rank 0 because we do not need any index, and a vector has dimension N and rank 1 because we only need a single index. In fluid mechanics we mostly deal with tensors of rank 2.

In ChiTech this functionality is:

```
chi_mesh::Vector3 a;
chi_mesh::Vector3 b;
//
//Some code
//
chi_mesh::TensorRank2Dim3 c = a.OTimes(b);
```

2.1.2 Vector inverse, $\mathbf{a}^{-1}$

There is not a common notation for taking the component-wise inverse of a vector, however, the notation $\mathbf{a}^{-1}$ will be used to denote

$$\mathbf{a}^{-1} = \begin{bmatrix} \frac{1}{a_0} \\ \frac{1}{a_1} \\ \vdots \\ \frac{1}{a_{N-1}} \end{bmatrix} \tag{2.2}$$

In ChiTech this functionality is:

```
chi_mesh::Vector3 a;
chi_mesh::Vector3 a_inv = a.Inverse();
```

To avoid division by zero there are also safe versions:

```
chi_mesh::Vector3 a;
chi_mesh::Vector3 b = a.InverseZeroIfSmaller(1.0e-12);
chi_mesh::Vector3 c = a.InverseOneIfSmaller(1.0e-12);
```

2.1.3 Component-wise multiplication of vectors, ab

Also called the Hadamard-product which is the component-wise multiplication of two vectors or two matrices, where both operands need to be the same size. The notation for a Hadamard-product is the symbol $\circ$ or $\odot$. The tensor product notation is also sometimes confusingly mixed with component-wise multiplication (i.e., $\mathbf{ab}$ vs $\{\mathbf{ab}\}$), however the component-wise multiplication is

$$\mathbf{ab} = \mathbf{a}\circ\mathbf{b} = \mathbf{a}\odot\mathbf{b} = \begin{bmatrix} a_0 b_0 \\ a_1 b_1 \\ \vdots \\ a_{N-1} b_{N-1} \end{bmatrix} \tag{2.3}$$

In ChiTech this functionality is:

```
chi_mesh::Vector3 a;
chi_mesh::Vector3 b;
//
//Some code
//
chi_mesh::Vector3 c = a*b;
```

2.1.4 Component-wise division of vectors, $\mathbf{a}^{-1}\mathbf{b}$

Also called Hadamard division where again both operands need to be of the same size. This is another instance of component-wise operations and involves the use of a vector-inverse, $\mathbf{a}^{-1}\mathbf{b}$, or simply written as division, $\mathbf{b}/\mathbf{a}$, in which case

$$\mathbf{a}^{-1}\mathbf{b} = \mathbf{b}/\mathbf{a} = \mathbf{b}\oslash\mathbf{a} = \begin{bmatrix} b_0/a_0 \\ b_1/a_1 \\ \vdots \\ b_{N-1}/a_{N-1} \end{bmatrix}. \tag{2.4}$$

Again notice the absence of the tensor notation, $\{\}$.

In ChiTech this functionality is:

```
chi_mesh::Vector3 a;
chi_mesh::Vector3 b;
//
//Some code
//
chi_mesh::Vector3 c = a/b;
```

2.1.5 Commutativity of component-wise operations

The following commutativity rules apply to component-wise operations:

$$\begin{aligned} \alpha(\mathbf{ab}) &= (\alpha\mathbf{a})\mathbf{b} = \mathbf{a}(\alpha\mathbf{b}) \\ \alpha(\mathbf{a}^{-1}\mathbf{b}) &= (\alpha\mathbf{a}^{-1})\mathbf{b} = \mathbf{a}^{-1}(\alpha\mathbf{b}) \\ \mathbf{c}\bullet(\mathbf{ab}) &= (\mathbf{c}\bullet\mathbf{a})\mathbf{b} = \mathbf{a}(\mathbf{c}\bullet\mathbf{b}) \\ \mathbf{c}\bullet(\mathbf{a}^{-1}\mathbf{b}) &= (\mathbf{c}\bullet\mathbf{a}^{-1})\mathbf{b} = \mathbf{a}^{-1}(\mathbf{c}\bullet\mathbf{b}) \\ \alpha\mathbf{c}\bullet(\mathbf{ab}) &= (\alpha\mathbf{c}\bullet\mathbf{a})\mathbf{b} = \mathbf{a}(\mathbf{c}\bullet\alpha\mathbf{b}) = \mathbf{c}\bullet(\alpha\mathbf{ab}) \end{aligned} \tag{2.5}$$

where α is any scalar.

2.1.6 Dot-product of a vector with a tensor, $\mathbf{a}\bullet\{t\}$

Under the same topic of tensor product notation we can also discuss the **dot product of scalar and a rank 2 tensor**. The dot-product of a vector **a** and a tensor $\{\mathbf{t}\}$, commonly written as $\mathbf{a}\bullet\{\mathbf{t}\}$, which results in a vector of size N, can be understood using one of two thought patterns:

- Thought pattern 1: Classical component-wise dot product

$$\begin{aligned} \mathbf{a}\bullet\{\mathbf{t}\} &= \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} \bullet \begin{bmatrix} t_{00} & t_{01} & \dots & t_{0(N-1)} \\ t_{10} & t_{11} & \dots & t_{1(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ t_{(N-1)0} & t_{(N-1)1} & \dots & t_{(N-1)(N-1)} \end{bmatrix} \\ &= \begin{bmatrix} a_0 t_{00} & + & a_1 t_{01} & + & \dots & + & a_{N-1} t_{0(N-1)} \\ a_0 t_{10} & + & a_1 t_{11} & + & \dots & + & a_{N-1} t_{1(N-1)} \\ \vdots & + & \vdots & + & \ddots & + & \vdots \\ a_0 t_{(N-1)0} & + & a_1 t_{(N-1)1} & + & \dots & + & a_{N-1} t_{(N-1)(N-1)} \end{bmatrix} \end{aligned} \tag{2.6}$$

- Thought pattern 2 (preferred): Dot product of vectors

$$\mathbf{a}\bullet\{\mathbf{t}\} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} \bullet \begin{bmatrix} \mathbf{t}_0 \\ \mathbf{t}_1 \\ \vdots \\ \mathbf{t}_{(N-1)} \end{bmatrix} = \begin{bmatrix} \mathbf{a}\cdot\mathbf{t}_0 \\ \mathbf{a}\cdot\mathbf{t}_1 \\ \vdots \\ \mathbf{a}\cdot\mathbf{t}_{(N-1)} \end{bmatrix} \tag{2.7}$$

where the latter thought pattern requires the rank 2 tensor two be represented as a vector of vectors (or a matrix if you prefer).

In ChiTech this functionality is:

```
chi_mesh::Vector3 a;
chi_mesh::TensorRank2Dim3 t;
//
//Some code
//
chi_mesh::Vector3 b = a.Dot(t);
```

2.1.7 Dot-product of a vector with a tensor product of two other vectors, $\mathbf{a}\bullet\{\mathbf{b}\otimes\mathbf{c}\}$

A special case relating to the dot product of a vector and a tensor is the notation $\mathbf{a} \cdot \{\mathbf{b}\otimes\mathbf{c}\}$, which commonly arises from an area vector's dot product with the velocity tensor (i.e. $\mathbf{A} \cdot \{\mathbf{u}\otimes\mathbf{u}\}$). In such a case the non-linearity of $\mathbf{u}\otimes\mathbf{u}$ is removed by lagging the velocity as $\mathbf{A} \cdot \{\mathbf{u}^{(n)}\mathbf{u}^{(n-1)}\}$ for which we get the general form, $\mathbf{a} = \mathbf{A}$, $\mathbf{b} = \mathbf{u}^{(n)}$, $\mathbf{c} = \mathbf{u}^{(n-1)}$, such that

$$\begin{aligned}
\mathbf{a} \cdot \{\mathbf{b}\otimes\mathbf{c}\} &= \mathbf{a} \cdot \begin{bmatrix} b_0c_0 & b_0c_1 & \dots & b_0c_{N-1} \\ b_1c_0 & b_1c_1 & \dots & b_1c_{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N-1}c_0 & b_{N-1}c_1 & \dots & b_{N-1}c_{N-1} \end{bmatrix} \\
&= \begin{bmatrix} a_0b_0c_0 & + & a_1b_0c_1 & + & \dots & + & a_{N-1}b_0c_{N-1} \\ a_0b_1c_0 & + & a_1b_1c_1 & + & \dots & + & a_{N-1}b_1c_{N-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_0b_{N-1}c_0 & + & a_1b_{N-1}c_1 & + & \dots & + & a_{N-1}b_{N-1}c_{N-1} \end{bmatrix} \\
&= (\mathbf{a} \cdot \mathbf{c})\ \mathbf{b}
\end{aligned} \tag{2.8}$$

The usefulness of this will be made clearer in later sections.

In ChiTech this functionality is:

```
chi_mesh::Vector3 a;
chi_mesh::Vector3 b;
chi_mesh::Vector3 c;
//
//Some code
//
chi_mesh::Vector3 d = a.Dot(c)*b;
```

2.1.8 Divergence of a vector given its gradient, $\nabla\bullet\mathbf{a} = \mathbb{1} \bullet \textbf{diag}(\nabla\mathbf{a})$

Programatically the divergence of vector-quantity can easily be obtained by accessing the diagonal elements of the gradient tensor, $\nabla\mathbf{a}_{00}$, $\nabla\mathbf{a}_{11}$ and $\nabla\mathbf{a}_{22}$. Mathematically it can be written as

$$\boldsymbol{\nabla}\bullet\mathbf{a} = \mathbb{1} \bullet \text{diag}(\boldsymbol{\nabla}\mathbf{a}) \tag{2.9}$$

In ChiTech this functionality is:

```
chi_mesh::TensorRank2Dim3 grad_a;
double div_a = grad_a.DiagSum();
//
// or
//
chi_mesh::Vector3 b = grad_a.Diag();
double div_a1 = chi_mesh::Vector3(1.0,1.0,1.0).Dot(b);
```

2.2 Unstructured meshes and the treatment of certain quantities

This section is meant to clarify some items found in a useful paper by Sezai [1] which deals with unstructured meshes and the finite volume method. Figure 2.1 below contains a graphic depiction of unstructured 2D cells which we will use to define numerous concepts.

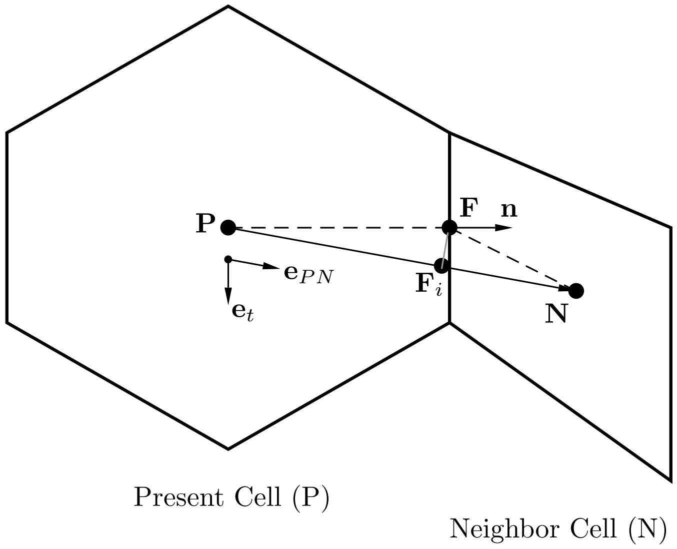

Figure 2.1: Reference vector layout for neighboring cells

The points **P**, **N** and **F** are the present-cell-, neighboring-cell- and adjoining-face-centroids, respectively. The vector **PN** is from **P** to **N**, and the projection of **PF** onto **PN** is at point $\mathbf{F}_i$. The vector $\mathbf{e}_{PN}$ is the unit vector along **PN** computed from $\mathbf{PN}/d_{PN}$, where $d_{PN} = ||\mathbf{PN}||_2$. The vector $\mathbf{e}_t$ is the unit tangential vector formed from $\mathbf{e}_t = \mathbf{e}_{PN} - \mathbf{n}$, with $\mathbf{n}$ being the face normal. The distance from **P** to $\mathbf{F}_i$, d_{PF_i}, is computed as $d_{PF_i} = \mathbf{PF} \cdot \mathbf{e}_{PN}$, and $\mathbf{F}_i = \mathbf{P} + d_{PF_i}\, \mathbf{e}_{PN}$.

2.2.1 Linear interpolation of a quantity ϕ to a face, option 1

A linear interpolation of any quantity ϕ, to find ϕ_{f_i}, can be determined by defining $r_P = \frac{d_{PF_i}}{d_{PN}}$ from which

$$\phi_{f_i} = (1-r_P)\phi_P+(r_P)\phi_N. \tag{2.10}$$

This is however not the desired value at the cell centroid, i.e. ϕ_f. To correct for this we can linearly interpolate the gradient of ϕ to $\mathbf{F}_i$ and extend towards $\mathbf{F}$ as

$$\boldsymbol{\nabla}\phi_{f_i} = (1-r_P)\boldsymbol{\nabla}\phi_P+(r_P)\boldsymbol{\nabla}\phi_N$$

using the vector from $\mathbf{F}_i$ to $\mathbf{F}$, $\mathbf{F_iF}$, to get

$$\phi_f \approx \phi_{f_i}+\boldsymbol{\nabla}\phi_{f_i}\boldsymbol{\cdot}\mathbf{F_iF} \tag{2.11}$$

which can be used for the interpolation of the pressure and/or temperature. Note however, that this form is not suitable for the interpolation of the velocity, which requires special treatment.

2.2.2 Linear interpolation of a quantity ϕ to a face, option 2

For quantities that are not very sensitive to variation in the direction of the face skewness, i.e., fluid viscosity, μ, fluid density, ρ, and thermal conductivity, k, one can apply a less accurate, but still suitable interpolation, by neglecting the gradients and using the normal distance to the face. This requires r_f defined as

$$r_f = \frac{\mathbf{PF}\boldsymbol{\cdot}\mathbf{n}}{\mathbf{PF}\boldsymbol{\cdot}\mathbf{n}+\mathbf{FN}\boldsymbol{\cdot}\mathbf{n}} \tag{2.12}$$

which can then be used to linearly interpolated quantities as

$$\phi_{f_i} = (1-r_f)\phi_P+(r_f)\phi_N. \tag{2.13}$$

2.2.3 Computing gradients

There are different methods for computing gradients of the form $\boldsymbol{\nabla}\phi$.

Green-Gauss:
This formulation is identical to “Option 3” of chapter 9.2 in the book by Moukalled et al. [2]. The first order Green-Gauss approximation reduces essentially to a summation over faces.

$$\boldsymbol{\nabla}\phi = \frac{1}{V}\sum_f \mathbf{A}_f\phi_f \tag{2.14}$$

where ϕ_f is taken as a linear interpolation between the adjoining face's cells. This formulation poses some challenges when undetermined faces fluxes are encountered, as is the case with boundaries and unstructured meshes, and therefore we require an algorithm to compensate. Therefore the following algorithm is applied:

1. Compute $\boldsymbol{\nabla}\phi^{(n)}$ over the entire domain using $\frac{1}{V}\sum_f \mathbf{A}_f\phi_f$ where ϕ_f is computed from (2.11) as

$$\phi_f = (1-r_P)\phi_P+(r_P)\phi_N+\left[(1-r_P)\boldsymbol{\nabla}\phi_P+(r_P)\boldsymbol{\nabla}\phi_N\right]^{(n-1)}\boldsymbol{\cdot}\mathbf{F_iF}$$

   and boundary faces are computed as

$$\phi_b = \phi_P+\boldsymbol{\nabla}\phi_P^{(n-1)}\boldsymbol{\cdot}\ \mathbf{PF}$$

2. Compute $||\boldsymbol{\nabla}\phi^{(n)}-\boldsymbol{\nabla}\phi^{(n-1)}||_2$. If $||\boldsymbol{\nabla}\phi^{(n)}-\boldsymbol{\nabla}\phi^{(n-1)}||_2 < \epsilon$ then terminate.

3. Swap $\boldsymbol{\nabla}\phi^{(n)}$ and $\boldsymbol{\nabla}\phi^{(n-1)}$.

4. Repeat 2 to 4.

In most cases the iteration process adds little to the iterative performance and it can be sufficient to just terminate after a single iteration.

Least squares:
Section 9.3 in [2] provides a very good description of how the weighted Least-Squares gradients are determined. In that formulation

$$\left\{\sum_f w_f\mathbf{PN}\otimes\mathbf{PN}\right\}\boldsymbol{\nabla}\phi_P = \sum_f\left[w_f\mathbf{PN}(\phi_N-\phi_P)\right]$$

where

$$w_f = \frac{1}{(||\mathbf{PN}||_2)^2}$$

On boundaries, $\mathbf{PN}$ and ϕ_N is replaced by $\mathbf{PF}$ and ϕ_b.

Note that this results in a system that needs to solved for each cell.

2.2.4 Flux of a gradient

A diffusion quantity that often arises in unstructured meshes is $\mathbf{A}_f\boldsymbol{\cdot}(\boldsymbol{\nabla}\phi)_f$, where $\mathbf{A}_f$ is the area vector formed by the face area A_f and its normal n as $\mathbf{A}_f = A_f\mathbf{n}$. When we deal with skewed faces, this diffusion term needs some treatment. In general, since the face is skewed, the flux at the cell face develops additional components other than the orthogonal component along the direction of $\mathbf{PN}$. These additional components are not intuitive to understand and the paper by Sezai [1] lacks explanation of some of the items in its graphical illustrations. Refer to Figure 2.2 for the following discussion.

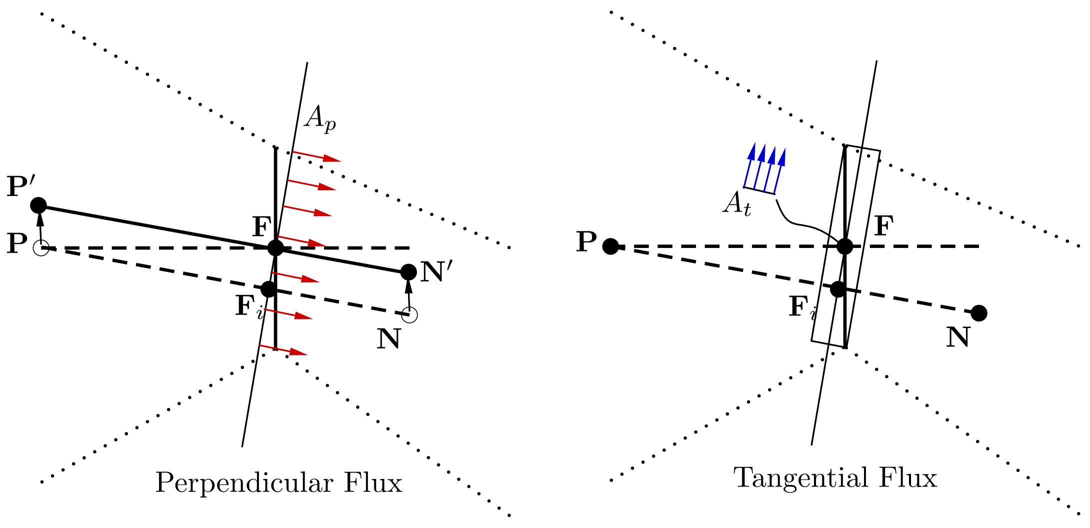

Figure 2.2: Decomposed areas for the interpolation of a gradient flux

Given that the correct way to approximate the flux across the face is to compute the gradient along its centroid, we require the gradient to be computed from $\phi_{P'}$ and $\phi_{N'}$ along the projected perpendicular area A_p. Additionally we need to compute the flux along the tangential area A_t. Including these terms within an implicit formulation is, however, cumbersome to say the least. Instead we include explicit corrections.

Firstly, we formulate the fluxes at $\mathbf{P}'$ and $\mathbf{N}'$ (see the left schematic in Figure 2.2) as

$$\begin{aligned}\phi_{P'} &= \phi_P + (\boldsymbol{\nabla}\phi)_P^{(n-1)} \cdot \mathbf{F_iF}\\ \phi_{N'} &= \phi_N + (\boldsymbol{\nabla}\phi)_N^{(n-1)} \cdot \mathbf{F_iF}\end{aligned}$$

where $\mathbf{F_iF}$ is the vector from $\mathbf{F}_i$ to $\mathbf{F}$ and the gradients are computed from the previous iteration's flux field. The perpendicular gradient term can then be computed from

$$\begin{aligned}\Big(\mathbf{A}_f\cdot(\boldsymbol{\nabla}\phi)_f\Big)_p &= \frac{A_p}{d_{PN}}\Big(\phi_{N'} - \phi_{P'}\Big)\\ &= \frac{A_p}{d_{PN}}\Big(\phi_N - \phi_P\Big) + \frac{A_p}{d_{PN}}\Big((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P\Big)^{(n-1)} \cdot \mathbf{F_iF}\end{aligned}$$

where $A_p = A_f/(\mathbf{e}_{PN}\cdot\mathbf{n})$. Secondly, the tangential component can be computed from (2.11).

$$\begin{aligned}\Big(\mathbf{A}_f\cdot(\boldsymbol{\nabla}\phi)_f\Big)_t &= (\boldsymbol{\nabla}\phi)_F\cdot\mathbf{A}_t\\ &\approx (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)}\cdot\mathbf{A}_t\end{aligned}$$

where $\mathbf{A}_t = \mathbf{A}_f - A_p\mathbf{e}_{PN}$ and the gradient at the face centroid is approximated by the gradient interpolated at $\mathbf{F}_i$ which is not an unreasonable approximation considering that cell skewness is normally minimized. Putting it all together we have

$$\mathbf{A}_f \bullet (\boldsymbol{\nabla}\phi)_f = \frac{A_p}{d_{PN}}\left(\phi_N - \phi_P\right) + \frac{A_p}{d_{PN}}\left((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P\right)^{(n-1)} \bullet \mathbf{F_i F} + (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)} \bullet \mathbf{A}_t \qquad (2.15)$$

This equation appears in the momentum equation as well as the pressure correction equation for the SIMPLE method.

3 Discretization of the momentum equation

We start our derivations with the conservation of momentum equation where the non-linear convective term, $\boldsymbol{\nabla}\boldsymbol{\cdot}(\rho\mathbf{u}\otimes\mathbf{u})$, has been linearized through the use of the velocity at a previous iteration $(n-1)$, and the time derivative has been taken as linearly changing from timestep $(t-1)$,

$$\rho\frac{\mathbf{u}-\mathbf{u}^{(t-1)}}{\Delta t}+\boldsymbol{\nabla}\boldsymbol{\cdot}(\rho\mathbf{u}^{(n-1)}\otimes\mathbf{u})-\boldsymbol{\nabla}\boldsymbol{\cdot}\{\mu\boldsymbol{\nabla}\mathbf{u}\}=-\boldsymbol{\nabla}p+\boldsymbol{\nabla}\boldsymbol{\cdot}\{\mu(\boldsymbol{\nabla}\mathbf{u})^{T}\}-\frac{2}{3}\boldsymbol{\nabla}\cdot(\mu\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}^{(n-1)})+\mathbf{f}.$$

Next we integrate this equation over a given cell's volume and apply Gauss' divergence theorem. We then also convert surface integrals to sum over areas of the faces. In the latter process we combine the area of a face, A_f, with its associated normal, $\mathbf{n}_f$, and introduce the area vector, $\mathbf{A}_f=A_f\mathbf{n}_f$,

$$\begin{aligned}\frac{\rho_P V_P}{\Delta t}(\mathbf{u}-\mathbf{u}^{(t-1)})+\int_S \mathbf{n}\boldsymbol{\cdot}(\rho\mathbf{u}\otimes\mathbf{u}^{(n-1)}).dA-\int_S \mathbf{n}\boldsymbol{\cdot}\{\mu\boldsymbol{\nabla}\mathbf{u}\}.dA&\\ =-V_P(\boldsymbol{\nabla}p)^{(n-1)}+\int_S \mathbf{n}\boldsymbol{\cdot}\{\mu(\boldsymbol{\nabla}\mathbf{u})^{T}\}^{(n-1)}.dA-\frac{2}{3}\int_S \mathbf{n}\cdot(\mu\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}^{(n-1)}).dA+V_P\mathbf{f}&\end{aligned}$$

and after application of Green's theorem,

$$\begin{aligned}\therefore\ \frac{\rho_P V_P}{\Delta t}\mathbf{u}+\sum_f \dot{m}_f^{(n-1)}\mathbf{u}_f-\sum_f \mathbf{A}_f\boldsymbol{\cdot}\{\mu\boldsymbol{\nabla}\mathbf{u}\}_f&\\ =-V_P(\boldsymbol{\nabla}p)^{(n-1)}+\sum_f \mathbf{A}_f\boldsymbol{\cdot}\{\mu(\boldsymbol{\nabla}\mathbf{u})^{T}\}_f^{(n-1)}-\frac{2}{3}\sum_f \mathbf{A}_f\cdot(\mu\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}^{(n-1)})_f+\frac{\rho_P V_P}{\Delta t}\mathbf{u}^{(t-1)}+V_P\mathbf{f}.&\end{aligned}\tag{3.1}$$

Note here that we used the property $\mathbf{a}\boldsymbol{\cdot}(\mathbf{b}\otimes\mathbf{c})=(\mathbf{a}\boldsymbol{\cdot}\mathbf{c})\mathbf{b}$, defined in the tools section, to get

$$\begin{aligned}\int_S \mathbf{n}\boldsymbol{\cdot}(\rho\mathbf{u}\otimes\mathbf{u}^{(n-1)}).dA&=\int_S\Big(\mathbf{n}\boldsymbol{\cdot}(\rho\mathbf{u}^{(n-1)})\Big)\mathbf{u}.dA\\ &=\sum_f\Big(\rho_f\mathbf{A}_f\boldsymbol{\cdot}\mathbf{u}_f^{(n-1)}\Big)\mathbf{u}_f\\ &=\sum_f \dot{m}_f^{(n-1)}\mathbf{u}_f\end{aligned}$$

3.1 Preface on the need for $\boldsymbol{\nabla}\mathbf{u}$

The gradient of the velocity cannot simply be computed, i.e., from the Green-Gauss method in (2.14), because we are using a collocated scheme for which it is known to create a phenomenon known as "checker-boarding". This is especially true since we require the face-velocities. Checkerboarding was first addressed in a paper by Rhie and Chow [3] by applying momentum interpolation and was later improved on by Majumdar [4]. In this project we use the Majumdar Momentum Interpolation Method (MMIM) **which we will detail later**.

Note: For now, assume that we can compute $\boldsymbol{\nabla}\mathbf{u}$ and have its previous iteration value, $\boldsymbol{\nabla}\mathbf{u}^{(n-1)}$, available.

We now require expressions for the face terms $\dot{m}_f^{(n-1)}\mathbf{u}_f$, $\{\mu\nabla\mathbf{u}\}_f$, $\{\mu(\nabla\mathbf{u})^T\}_f$, and $(\mu\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}^{(n-1)})_f$. In all the sub-sections that follow we will be working with a known mass flow rate $\dot{m}_f = \rho\mathbf{A}_f\boldsymbol{\cdot}\mathbf{u}_f^{(\ell-1)}$ which will be assumed zero at the start of the simulation but will be updated and stored later.

But first let us define the coefficient formulation.

3.2 Coefficient formulation and under-relaxation

When applying the discretization of the momentum equation above, we ultimately seek to arrange algebraic terms into the following general form:

$$\mathbf{a}^P\mathbf{u}^P + \sum_f \mathbf{a}^N\mathbf{u}^N = \mathbf{b}^P_{nur} + \sum_f -\mathbf{A}_f p_f$$

$$\therefore \mathbf{a}^P\mathbf{u}^P = -\sum_f \mathbf{a}^N\mathbf{u}^N + \mathbf{b}^P_{nur} + \sum_f -\mathbf{A}_f p_f$$

where the coefficients $\mathbf{a}^P$ and $\mathbf{a}^N$ are vector valued and $\mathbf{b}_{nur}$ signifies no under-relaxation. For iterative control we further introduce an under-relaxation with respect to the previous iteration velocity such that

$$\mathbf{a}^P\mathbf{u}^P = \alpha_u\left(-\sum_f \mathbf{a}^N\mathbf{u}^N + \mathbf{b}^P_{nur} + \sum_f -\mathbf{A}_f p_f\right) + (1-\alpha_u)a^P(\mathbf{u}^P)^{(n-1)}$$

$$\therefore \frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}^P = -\sum_f \mathbf{a}^N_f\mathbf{u}^N + \mathbf{b}^P_{nur} + \frac{1-\alpha_u}{\alpha_u}\mathbf{a}^P(\mathbf{u}^P)^{(n-1)} + \sum_f -\mathbf{A}_f p_f$$

Finally we combine the previous iteration under-relaxed velocity with $\mathbf{b}_{nur}$ as $\mathbf{b} = \mathbf{b}_{nur} + + \frac{1-\alpha_u}{\alpha_u}\mathbf{a}(\mathbf{u})^{(n-1)}$ to get the **momentum equation in coefficient form**:

$$\therefore \frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}^P = -\sum_f \mathbf{a}^N_f\mathbf{u}^N + \mathbf{b}^P + \sum_f -\mathbf{A}_f p_f \tag{3.2}$$

Note here that the **a** and **b** coefficients are vector valued.

3.3 Material properties at faces, μ_f and ρ_f

Material properties are generally not as rapidly varying as the pressure or velocity and therefore we linearly interpolate these values without the correction term specified in (2.11) and instead just use (2.13). Therefore,

$$\mu_f = (1-r_f)\mu_P + (r_f)\mu_N \tag{3.3}$$

$$\rho_f = (1-r_f)\rho_P + (r_f)\rho_N \tag{3.4}$$

3.4 Diffusion gradient terms, $\{\mu\nabla\mathbf{u}\}_f$

The gradient-term, $\mathbf{A}_f\bullet\{\mu\boldsymbol{\nabla}\mathbf{u}\}_f$ is treated as introduced in (2.15),

$$\begin{aligned}-\mathbf{A}_f\bullet(\mu\boldsymbol{\nabla}\mathbf{u})_f = -\mu_f\frac{A_p}{d_{PN}}\Big(\mathbf{u}_N-\mathbf{u}_P\Big)-\mu_f\frac{A_p}{d_{PN}}\Big\{\boldsymbol{\nabla}\mathbf{u}_N-\boldsymbol{\nabla}\mathbf{u}_P\Big\}^{(n-1)}\bullet\mathbf{F_iF}\\ -\mu_f\Big\{(1-r_P)\boldsymbol{\nabla}\mathbf{u}_P+(r_P)\boldsymbol{\nabla}\mathbf{u}_N\Big\}^{(n-1)}\bullet\mathbf{A}_t\end{aligned} \tag{3.5}$$

Note here that this formulation has both implicit terms and explicit terms. The coefficients get updated as

$$\begin{aligned}\mathbf{a}^N &= \mathbf{a}^N-\ \mu_f\frac{A_p}{d_{PN}}\cdot\mathbb{1}\\ \mathbf{a}^P &= \mathbf{a}^P+\ \mu_f\frac{A_p}{d_{PN}}\cdot\mathbb{1}\\ \mathbf{b}^P &= \mathbf{b}^P+\ \mu_f\frac{A_p}{d_{PN}}\Big\{\boldsymbol{\nabla}\mathbf{u}_N-\boldsymbol{\nabla}\mathbf{u}_P\Big\}^{(n-1)}\bullet\mathbf{F_iF}\ +\ \mu_f\Big\{(1-r_P)\boldsymbol{\nabla}\mathbf{u}_P+(r_P)\boldsymbol{\nabla}\mathbf{u}_N\Big\}^{(n-1)}\bullet\mathbf{A}_t\end{aligned}$$

3.5 Cross-diffusion gradient terms, $\{\mu(\nabla\mathbf{u})^T\}_f^{(n-1)}$

This term can be evaluated purely using explicitly values and by noting the term has a weak coupling to the momentum equation, we can apply the interpolation option given in (2.13)

$$\mathbf{A}_f\bullet\{\mu(\nabla\mathbf{u})^T\}_f^{(n-1)} = \mu_f\mathbf{A}_f\bullet\Big\{(1-r_f)\{\nabla\mathbf{u}\}_P^T+(r_f)\{\nabla\mathbf{u}\}_N^T\Big\}^{(n-1)} \tag{3.6}$$

The coefficients get updated as

$$\mathbf{b}^P = \mathbf{b}^P+\ \mu_f\mathbf{A}_f\bullet\Big\{(1-r_f)\{\nabla\mathbf{u}\}_P^T+(r_f)\{\nabla\mathbf{u}\}_N^T\Big\}^{(n-1)}$$

3.6 Divergence term, $(\mu\boldsymbol{\nabla}\bullet\mathbf{u}^{(n-1)})_f$

The divergence term is easy to obtain if we already computed the gradients at the faces. For this we use the notation defined in the tools section.

$$-\frac{2}{3}\mathbf{A}_f(\mu\boldsymbol{\nabla}\bullet\mathbf{u}^{(n-1)})_f = -\frac{2}{3}\mu_f\mathbf{A}_f\Big(\mathbb{1}\bullet\mathrm{diag}(\boldsymbol{\nabla}\mathbf{u})\Big)_f^{(n-1)} \tag{3.7}$$

The coefficients get updated as

$$\mathbf{b}^P = \mathbf{b}^P-\ \frac{2}{3}\mu_f\mathbf{A}_f\Big(\mathbb{1}\bullet\mathrm{diag}(\boldsymbol{\nabla}\mathbf{u})\Big)_f^{(n-1)}$$

3.7 Face convection term, $\dot{m}_f^{(n-1)}\mathbf{u}_f$

The face convective term is taken from a second-order upwind scheme. This is achieved by considering the upstream face velocity as $\mathbf{u}_f^U$ as pertaining to the associated upstream cell (either the present cell or the neighbor cell) as captured by

$$\begin{aligned}\dot{m}_f^{(n-1)}\mathbf{u}_f^U &= \min\left(\dot{m}_f^{(n-1)}, 0.0\right)\mathbf{u}^N + \max\left(\dot{m}_f^{(n-1)}, 0.0\right)\mathbf{u}^P \\ &+ \dot{m}_f^{(n-1)}\begin{cases}\nabla\mathbf{u}_N^{(n-1)}\boldsymbol{\cdot}\mathbf{NF}, & \text{if } \dot{m}_f^{(n-1)} < 0 \\ \nabla\mathbf{u}_P^{(n-1)}\boldsymbol{\cdot}\mathbf{PF}, & \text{if } \dot{m}_f^{(n-1)} > 0\end{cases}\end{aligned} \tag{3.8}$$

where again $\mathbf{PF}$ is the vector from the present cell centroid to the face centroid, and $\mathbf{NF}$ is the vector from the neighboring cell centroid to the face centroid. The neighbor N is the neighbor at face f.

The coefficients get updated as

$$\begin{aligned}\mathbf{a}^N &= \mathbf{a}^N + \min\left(\dot{m}_f^{(n-1)}, 0.0\right) \\ \mathbf{a}^P &= \mathbf{a}^P + \max\left(\dot{m}_f^{(n-1)}, 0.0\right) \\ \mathbf{b}^P &= \mathbf{b}^P - \dot{m}_f^{(n-1)}\begin{cases}\nabla\mathbf{u}_N^{(n-1)}\boldsymbol{\cdot}\mathbf{NF}, & \text{if } \dot{m}_f^{(n-1)} < 0 \\ \nabla\mathbf{u}_P^{(n-1)}\boldsymbol{\cdot}\mathbf{PF}, & \text{if } \dot{m}_f^{(n-1)} > 0\end{cases}\end{aligned}$$

3.8 Face pressure, p_f

The face pressure is simply taken as a geometrically averaging of the adjoining cell's pressures when the face is internal, as depicted in (2.11), and is interpolated to the boundary when the face is on a boundary. Therefore, with an internal face we have

$$p_{f_i} = (1-r_P)p_P + (r_P)p_N$$

$$\nabla p_{f_i}^{(n-1)} = (1-r_P)\nabla p_P^{(n-1)} + (r_P)\nabla p_N^{(n-1)}$$

and then

$$p_f = p_{f_i} + \nabla p_{f_i}^{(n-1)}\boldsymbol{\cdot}\mathbf{F_iF} \tag{3.9}$$

and for a boundary face we have

$$p_f = p_P + \nabla p_P^{(n-1)}\boldsymbol{\cdot}\mathbf{PF}.$$

Note again here that we need to store the gradient of the pressure.

4 Segregated solver (The SIMPLE method)

The momentum equation can be segregated from its implicit connection with the pressure by using the previous iteration's pressure field to solve an intermediate velocity, $\mathbf{u}_0$, and using explicit values for the pressure

$$\frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}_0^P = -\sum_f \mathbf{a}_f^N\mathbf{u}_0^N + \mathbf{b}^P + \sum_f -\mathbf{A}_f p_f^{(n-1)} \tag{4.1}$$

The formulation of a pressure correction follows from the fact that the velocity, computed from a guessed pressure field, does not satisfy the continuity equation, that is

$$\frac{\partial\rho}{\partial t} + \boldsymbol{\nabla}\cdot(\rho\mathbf{u}_0) \approx \frac{\rho^P - (\rho^P)^{(t-1)}}{\Delta t} + \frac{1}{V_P}\sum_f \rho\mathbf{A}_f\cdot\mathbf{u}_0 \neq 0$$
$$\therefore \frac{\rho^P - (\rho^P)^{(t-1)}}{\Delta t}V_P + \sum_f \dot{m}_{f0} \neq 0$$

We then postulate a mass flux correction, $\dot{m}'_f$, such that

$$\frac{\rho^P - (\rho^P)^{(t-1)}}{\Delta t}V_P + \sum_f \left(\dot{m}_{f0} + \dot{m}'_f\right) = 0 \tag{4.2}$$

and then apply a strategy to obtain a relation to pressure. Suppose we use the newly computed velocity field, $\mathbf{u}_0$, to isolate and compute a pressure gradient on a face. We could do this by reapplying (3.2), but instead of treating the pressure gradient as a known we make this the unknown. For the present cell P this takes the form

$$\frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}_0^P = -\sum_f \mathbf{a}_f^N\mathbf{u}_0^N + \mathbf{b}^P - V_P(\boldsymbol{\nabla}p)^P \tag{4.3}$$

for which we can arrange the pressure gradient to the left hand side

$$-V_P(\boldsymbol{\nabla}p)^P = \frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}_0^P + \sum_f \mathbf{a}_f^N\mathbf{u}_0^N - \mathbf{b}^P \tag{4.4}$$

From this we see that we can make a correction to the velocity of the form

$$-V_P(\boldsymbol{\nabla}p')_P = \frac{1}{\alpha_u}\mathbf{a}_P\ \mathbf{u}'_P$$
$$\therefore \mathbf{u}'_P = -\alpha_u V_P \mathbf{a}_P^{-1}(\boldsymbol{\nabla}p')_P$$

When we then plug this expression into (4.2) we get a Laplacian-like equation

$$\begin{aligned}\rho\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}_0+\rho\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}' &= 0\\ \rho\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}_0+\rho\boldsymbol{\nabla}\boldsymbol{\cdot}(-\alpha_u V\mathbf{a}^{-1}\boldsymbol{\nabla}p') &= 0\\ \therefore -\boldsymbol{\nabla}\boldsymbol{\cdot}(\rho\alpha_u V\mathbf{a}^{-1}\boldsymbol{\nabla}p') &= -\rho\boldsymbol{\nabla}\boldsymbol{\cdot}\mathbf{u}_0\end{aligned}$$

Discretizing this equation leads to

$$\sum_f -\mathbf{A}_f\boldsymbol{\cdot}(\rho\alpha_u V_f\mathbf{a}_f^{-1}\boldsymbol{\nabla}p'_f) = \sum_f -\rho\mathbf{A}_f\boldsymbol{\cdot}\mathbf{u}_{0f}.$$

Note here the appearance of $\mathbf{A}_f\boldsymbol{\cdot}\boldsymbol{\nabla}p'_f$ which technically needs to be treated as described in (2.15), however, since no explicit information for $\boldsymbol{\nabla}p'$ is available from previous iterations, we cannot make the skewness corrections. Fortunately, since p' tends to zero upon convergence, we can neglect these corrections.

$$\begin{aligned}\sum_f -\mathbf{A}_f\boldsymbol{\cdot}(\rho\alpha_u V_f\mathbf{a}_f^{-1}\boldsymbol{\nabla}p'_f) &= \sum_f -\rho\mathbf{A}_f\boldsymbol{\cdot}\mathbf{u}_{0f}\\ \sum_f -\rho\alpha_u V_f\mathbf{A}_f\boldsymbol{\cdot}\mathbf{a}_f^{-1}\mathbf{s}_\Delta^{-1}(p^N-p^P)') &= \sum_f -\dot{m}_{0f}\end{aligned}$$

which can be solved using a diffusion-type solver. After the solution is obtained we can correct certain quantities by noting

$$\dot{m}'_f = \rho\mathbf{A}_f\boldsymbol{\cdot}\mathbf{u}'_f = -\rho\alpha_u V_f\mathbf{A}_f\boldsymbol{\cdot}\mathbf{a}_f^{-1}\boldsymbol{\nabla}p'_f$$

and then applying the following:

$$\begin{aligned}\dot{m}_f &= \dot{m}_{0f}+\dot{m}'_f\\ \mathbf{u} &= \mathbf{u}_0-\alpha_u V_P\mathbf{a}_P^{-1}(\boldsymbol{\nabla}p')_P\\ p &= p^{(n-1)}+\alpha_p p'\end{aligned}$$

4.1 Computing face velocities

When using a collocated grid scheme the process of computing the face velocities, $\mathbf{u}_f$, which is required to determine $\dot{m}_f$ and $\boldsymbol{\nabla}u$, cannot simply be accomplished by applying the linear interpolation depicted in (2.11) because it results in the well known phenomenon of pressure"checkerboarding". A solution to this problem is to use the Majumdar Momemtum Interpolation Method (MIM) which is a modification of the Rhie-Chow MIM, and can be derived as follows:

We start with the momentum equation at the adjoining cells of a face in the same form as what we used to derive the pressure correction equation, recall (4.3), but now we write the equations for both the present cell, P, and the neighboring cell, N:

$$\frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}^P = -\sum_f \mathbf{a}_f^N\mathbf{u}^N+\mathbf{b}^P-V_P(\boldsymbol{\nabla}p)^P$$

$$\frac{1}{\alpha_u}\mathbf{a}^N\mathbf{u}^N = -\sum_f \mathbf{a}_f^N\mathbf{u}^N + \mathbf{b}^N - V_N(\boldsymbol{\nabla} p)^N$$

We can define the following to simplify these equations:

$$\begin{aligned} \mathbf{H}^P &= -\sum_f \mathbf{a}_f^N\mathbf{u}^N && \text{w.r.t. } P \\ \mathbf{H}^N &= -\sum_f \mathbf{a}_f^N\mathbf{u}^N && \text{w.r.t. } N \end{aligned} \tag{4.5}$$

with the simplified equations now

$$\frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}^P = \mathbf{H}^P + \mathbf{b}^P - V_P(\boldsymbol{\nabla} p)^P$$

$$\frac{1}{\alpha_u}\mathbf{a}^N\mathbf{u}^N = \mathbf{H}^N + \mathbf{b}^N - V_N(\boldsymbol{\nabla} p)^N$$

With yet another simplification we define $\mathbf{u}_{mim} = \mathbf{H} + \mathbf{b}$ as a **pseudo-velocity** that can be computed and stored for each cell. Our reduced equations are now

$$\frac{1}{\alpha_u}\mathbf{a}^P\mathbf{u}^P = \mathbf{u}_{mim}^P - V_P(\boldsymbol{\nabla} p)^P$$

$$\frac{1}{\alpha_u}\mathbf{a}^N\mathbf{u}^N = \mathbf{u}_{mim}^N - V_N(\boldsymbol{\nabla} p)^N$$

We then interpolate all the coefficient and terms but leave the interpolated $\mathbf{u}$ and $\boldsymbol{\nabla} p$ to get

$$\frac{1}{\alpha_u}\bigg((1-r_P)\mathbf{a}^P + (r_P)\mathbf{a}^N\bigg)\mathbf{u}_f = \bigg((1-r_P)\mathbf{u}_{mim}^P + (r_P)\mathbf{u}_{mim}^N\bigg) - \bigg((1-r_P)V_P + (r_P)V_N\bigg)(\boldsymbol{\nabla} p)_f \tag{4.6}$$

or more generally

$$\frac{1}{\alpha_u}\mathbf{a}_f\mathbf{u}_f = \mathbf{u}_{mim}^f - V_f(\boldsymbol{\nabla} p)_f \tag{4.7}$$

after which we get the **Momentum Interpolated velocity** as

$$\mathbf{u}_f = \alpha_u\mathbf{a}_f^{-1}\bigg(\mathbf{u}_{mim}^f - V_f(\boldsymbol{\nabla} p)_f.\bigg) \tag{4.8}$$

In this equation $(\boldsymbol{\nabla} p)_f$ can then be computed using (**??**).

References

[1] Sezai I., *Implementation of boundary conditions in pressure-based finite volume methods on unstructured grids*, Numerical Heat Transfer, Part B: Fundamentals, 2017

[2] Moukalled F., Mangani L., Darwish M., *The Finite Volume Method in Computational Fluid Dynamics - An Advanced Introduction with OpenFOAM and Matlab*, Springer, 2016.

[3] Rhie C.M., Chow W.L., *Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation*, AIAAJ Volume 21, Number 11, November 1983.

[4] Majumdar S., *Role of underrelaxation in momentum interpolation for calculation of flow with nonstaggered grids*, Numerical Heat Transfer 13, pages 125-132. 1998.