Summer School Notes

Shenan Grossberg

October 5, 2015

Contents

Chapter 1

Running Cases

1.1 fvSolution

1.1.1 PIMPLE loop

- The tolerance determines the number of times the matrix is solved.
- nCorrectors (innerCorrectors) is the number of times pressure is solved again (usually 2-4).
- nOuterCorrectors (not for SIMPLE) is the number of times all variables are solved again.

Tolerance should be tighter for pressure, because it is tougher to solve for pressure, and, when it is solved, we are done.

1.1.2 General Guidance

- The solvers in fvSolution improve the speed.
- The schemes in fvSchemes improve the accuracy.
  - Using upwind will work 90% of the time.
  - Improve the schemes after you improve the solvers.

1.2 fvSchemes - Discretisation Best Practices

1.2.1 gradSchemes

Use Least Squares for tetrahedral meshes:

- It is less accurate but more robust.
- Use limited to make it bounded.

1.2.2 divSchemes

Linear differencing (first order) is less accurate but gives realistic values (bounded)

- Use upwind

Central differencing (second order) is more accurate but gives unrealistic values (oscillates about bounds).

- Change to VanLeer if want second order.
- For momentum, use linear upwind. It is unbounded but that is okay, because velocity is not bounded.

1.2.3 laplacianSchemes

- Gauss linear corrected chops off the corrected component of the face normal vector if the mesh is highly skewed.
- Gauss linear limited chops the gradient so that the value is bounded by the space defined by the points.

1.3 Steady State

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

If you want to make it steady state, enforce $\nabla \cdot (\rho \mathbf{u}) = 0$.
There is more work to type up here.

1.4 T-Junction

1.4.1 Introduction

This section documents my experience of running the following solvers on the T-Junction test case:

- steadySettlingFoam
  - on a primed solution
  - on a not-primed solution
- LTSSettlingFoam

Underflow

The following BC should be used at the underflow:

pressure	velocity
zeroGradient	fixedValue

1.4.2 steadySettlingFoam

Before the Summer School, steadySettlingFoam crashed and Alpha did not exit the overflow. During the Summer School, I made the following changes to the case set up and achieved the following results:

On a Primed solution

run	changes	results
3	decreased p_rgh relTol from 1e-2 to 1e-4	crashed
4	decreased p_rgh relTol to 1e-6	crashed
5	**decreased relaxationFactors from 0.7 to: 0.25 (Alpha) 0.5 (U) 0.3 (other)**	ran
6	Vanja's solvers in fvSolution	ran slower
5b	increased p_rgh relaxationFactors to 1 (no p_rgh.relax() in pEqn.H)	ran same
5c	replaced (fvc::grad(U) & fvc::grad(muEff)) with fvc::div(muEff*dev2(T(fvc::grad(U)))) (see Section 2.3.4) and divScheme div((muEff*dev2(T(grad(U))))) Gauss linear;	ran same
5d	**added p_rgh.relax() after p_rghEqn.solve() in pEqn.H** (relax before solve in UEqn.H and alphaEqn.H) **decreased p_rgh relaxationFactors to 0.3**	velocity residuals lower
5e	increased relaxationFactors to 0.7 (U) 0.5 (other)	spikier residuals
5f	increased nCorrectors from 0 to 1 (no nCorrector loop in pEqn.H)	ran same

Note: decreasing the relaxation factors shortens the runtime but not necessarily the time to a converged solution; it simply means the time from one time step to the next is less.

Summary

- Decrease p_rgh relTol
- Simulation ran and Alpha exited at overflow by decreasing the relaxation factors
  - add relaxation to pEqn.H
- nCorrectors has no effect as no nCorrector loop in pEqn.H

On a not-primed solution

Alpha was not updated despite making the following changes:

- include ddt term in alphaEqn.H
- #include rhoEqn.H
- maxAlphaCo 10 in controlDict
- Alpha relaxation 0.7
- minIterations 1 in fvSolution
- ddt(rho,Alpha) and ddt(rho) Euler in fvSchemes

1.4.3 LTSSettlingFoam

- Alpha is updated
- Residuals oscillating but small (as with settlingFoam)
- Results consistent with settlingFoam at t=38

1.4.4 Conclusions

- Use either:
  - steadySettlingFoam on a primed solution or
  - LTSSettlingFoam
- Need to find out which method is faster.

Chapter 2

Coding in OpenFOAM

2.1 Scalar Transport Equation

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\mathbf{u}\phi) - \nabla \cdot (\gamma \nabla \phi) = \frac{\phi^* - \phi}{\tau} \tag{2.1.1}$$

We want to discretise Eqn. (2.1.1) into the form:

$$A\phi = b$$

2.1.1 Convection Term

The convection term is equivalent to a coordinate transformation. It can be written in 2 ways:

$$\begin{aligned} \nabla \cdot (\mathbf{u}\phi) &= \mathbf{u} \cdot \nabla \phi + \phi \nabla \cdot \mathbf{u} && (2.1.2) \\ &= \mathbf{u} \cdot \nabla \phi && (2.1.3) \end{aligned}$$

if $\nabla \cdot \mathbf{u} = 0$, i.e. it is conservative. Hence, writing it in the form given in Eqn. (2.1.3) ensures that there are no new extrema.

2.1.2 Diffusion Term

- The diffusion term is a gradient transport, i.e. it smooths the cat.
- It does not go above or below certain bounds. If a cell's neighbour goes up, the cell goes up. If a cell's goes down, the cell goes down.
- The diffusion transport is defined by the diffusion coefficient.

2.1.3 Source and Sink Terms

The RHS of Eqn. (2.1.1) is a relaxation term, i.e. if there are no convection or diffusion terms and you wait long enough $\phi \to \phi^*$.

- If there is a sink term (negative term on the RHS of Eqn. (2.1.1)), put it into matrix A to ensure diagonal dominance; use Sp().
- Conversely, if there is a source term (positive term on the RHS of Eqn. (2.1.1)), put it into vector b; you can use Su() but it is not necessary.

- If you don't know whether it will be a source or a sink term, use SuSp - it will put sink terms into A and source terms into b.

2.1.4 Ampersand operator

The & operator has been stolen from C++ to make the dot product, because binary 'and' does not make sense for vectors. However, binary 'and' has a lower priority in operator precedence than the dot product, so put brackets around the dot product operation.

2.2 Git

Git is a version control tool. The following are some useful Git commands and their function:

git init	initialises git
git add .	adds this directory to git
git status	shows which files are modified
git commit -m "My first commit"	-m flag to add a comment
git log	
git add	this is a staging post before git commit
gitg	gives graphical representation of branches
qgit	use instead of gitg
git gui	use instead of gitg
git checkout #id	opens branch #id
git checkout master	opens master branch
git branch xyz	
git branch	lists branches
git merge abc	merges current branch with branch abc
git diff HEAD^	
git difftool	uses meld
git mergetool	fires up meld
git branch -d xyz	deletes branch xyz
git push origin master	pushes master to remote server
git pull	
git checkout -b abc	opens branch abc from remote server
git push	updates server
git rebase	to clean up
git disentangle commits	not sure if this is a command

2.3 settlingFoam

2.3.1 Why is Alpha (not alpha) coded in settlingFoam

In OF, the divergence operator takes 2 arguments: a surface scalar patch field (flux) and a volume field. In Brennan's drift equation, the arguments in the divergence terms do not satisfy this requirement. Hence, the dispersed-phase volume fraction (α) is replaced with $\mathrm{A} = \frac{\rho_d \alpha}{\rho}$, the dispersed-phase mass fraction, and the drift equation is multiplied by ρ_d, which is constant, to create the required arguments.

I did the same in Section 2.4.2, to create the required arguments for the divergence terms in betaEqn.H.

2.3.2 Derivation of Diffusion Stress term in UEqn.H

Alpha (A) is defined as:

$$\begin{aligned}
\mathrm{A} &= \frac{\rho_d \alpha}{\rho} \\
\implies \rho_d \alpha &= \rho \mathrm{A} \\
&= (\alpha \rho_d + (1-\alpha)\rho_c)\mathrm{A} \\
\implies \rho_d \alpha - \alpha \rho_d \mathrm{A} + \alpha \rho_c \mathrm{A} &= \rho_c \mathrm{A} \\
\implies \alpha &= \frac{\rho_c \mathrm{A}}{\rho_d - \rho_d \mathrm{A} + \rho_c \mathrm{A}} \\
\implies 1 - \alpha &= \frac{\rho_d - \rho_d \mathrm{A}}{\rho_d - \rho_d \mathrm{A} + \rho_c \mathrm{A}} \\
\implies \frac{\alpha}{1-\alpha} &= \frac{\rho_c \mathrm{A}}{\rho_d - \rho_d \mathrm{A}} \\
&= \frac{\rho_c}{\rho_d} \frac{\mathrm{A}}{1 - \mathrm{A}}
\end{aligned}$$

This substitution is made in Brennan's Mixture Momentum Equation to create the diffusion stress term in UEqn.H.

2.3.3 Matrix notation

Typically, in matrix calculus, where:

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \text{ and } \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{u}}{\partial x_1} & \frac{\partial \mathbf{u}}{\partial x_2} & \frac{\partial \mathbf{u}}{\partial x_3} \end{pmatrix} = \begin{pmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_1}{\partial x_2} & \frac{\partial u_1}{\partial x_3} \\ \frac{\partial u_2}{\partial x_1} & \frac{\partial u_2}{\partial x_2} & \frac{\partial u_2}{\partial x_3} \\ \frac{\partial u_3}{\partial x_1} & \frac{\partial u_3}{\partial x_2} & \frac{\partial u_3}{\partial x_3} \end{pmatrix} = \frac{\partial u_i}{\partial x_j}$$

i.e. count u_i through the rows and ∂x_j through the columns.

In OpenFOAM, this is reversed:

$$\nabla \mathbf{u} = \begin{pmatrix} \dfrac{\partial}{\partial x_1} \\ \dfrac{\partial}{\partial x_2} \\ \dfrac{\partial}{\partial x_3} \end{pmatrix} \begin{pmatrix} u_1 & u_2 & u_3 \end{pmatrix} = \begin{pmatrix} \dfrac{\partial u_1}{\partial x_1} & \dfrac{\partial u_2}{\partial x_1} & \dfrac{\partial u_3}{\partial x_1} \\ \dfrac{\partial u_1}{\partial x_2} & \dfrac{\partial u_2}{\partial x_2} & \dfrac{\partial u_3}{\partial x_2} \\ \dfrac{\partial u_1}{\partial x_3} & \dfrac{\partial u_2}{\partial x_3} & \dfrac{\partial u_3}{\partial x_3} \end{pmatrix} = \frac{\partial u_j}{\partial x_i} = \partial_i u_j$$

i.e. count ∂x_i through the rows and u_j through the columns.

2.3.4 Derivation of Viscous and Turbulent Stress terms in UEqn.H

In settlingFoam, the viscous and turbulent stress terms are coded as:

```
fvm::laplacian(muEff, U) + (fvc::grad(U) & fvc::grad(muEff))
```

Mathematically, this is equivalent to:

$$\nabla \cdot (\mu \nabla \mathrm{U}) + \nabla \mathrm{U} \cdot \nabla \mu$$

In driftFluxFoam, the viscous and turbulent stress terms are coded as:

```
fvm::laplacian(muEff, U) + fvc::div(muEff*dev2(T(fvc::grad(U))))
```

Mathematically, this is equivalent to:

$$\nabla \cdot (\mu \nabla \mathrm{U}) + \nabla \cdot \left(\mu \left((\nabla \mathrm{U})^T - \frac{2}{3} \mathrm{tr}(\nabla \mathrm{U})^T \mathrm{I} \right) \right)$$

In Einstein notation:

$$\nabla \cdot (\mu \nabla \mathrm{U}) = \frac{\partial}{\partial x_i} \left(\mu \frac{\partial u_j}{\partial x_i} \right)$$

According to Ubbink (1997), the mixture can be treated as incompressible. Hence $\nabla \cdot \mathrm{U} = 0$.

Hence,

$$\begin{aligned} \nabla \cdot \left(\mu (\nabla \mathrm{U})^T \right) &= \frac{\partial}{\partial x_i} \left(\mu \frac{\partial u_i}{\partial x_j} \right) \\ &= \frac{\partial \mu}{\partial x_i} \cdot \frac{\partial u_i}{\partial x_j} + \mu \frac{\partial^2 u_i}{\partial x_i \partial x_j} \\ &= \frac{\partial u_j}{\partial x_i} \cdot \frac{\partial \mu}{\partial x_i} + \mu \frac{\partial}{\partial x_j} \left(\frac{\partial u_i}{\partial x_i} \right) \\ &= \nabla \mathrm{U} \cdot \nabla \mu \end{aligned}$$

As $\mathrm{tr}(\nabla \mathrm{U})^T = \mathrm{tr}(\nabla \mathrm{U}) = \nabla \cdot \mathrm{U} = 0$,

$$\nabla \cdot \left(\mu \left((\nabla \mathrm{U})^T - \frac{2}{3} \mathrm{tr}(\nabla \mathrm{U})^T \mathrm{I} \right) \right) = \nabla \mathrm{U} \cdot \nabla \mu$$

Hence in settlingFoam and driftFluxFoam, the viscous and turbulent stress terms are equal.

2.3.5 Examining the divergence terms in the Drift equation

The steady state drift equation is:

$$\nabla \cdot (\alpha \mathbf{v}) = -\nabla \cdot \left(\frac{\alpha \rho_c}{\rho} \mathbf{v}_d \right) + \nabla \cdot K \nabla \alpha$$

As OpenFOAM uses a linear system to solve equations, the arguments of the divergence operator must be linear. Note that:

$$\begin{aligned}
\rho &= \alpha \rho_d + (1-\alpha)\rho_c \\
&= (1-\alpha)\rho_c \left(\frac{\alpha \rho_d}{(1-\alpha)\rho_c} + 1 \right) \\
&= (1-\alpha)\rho_c \left(1 + \frac{\alpha}{1-\alpha} \frac{\rho_d}{\rho_c} \right)
\end{aligned}$$

$$\begin{aligned}
\implies F &= \frac{\alpha \rho_c}{\rho} \mathbf{v}_d \\
&= \alpha \rho_c \frac{(1-\alpha)^{-1}}{\rho_c} \left(1 + \frac{\alpha}{1-\alpha} \frac{\rho_d}{\rho_c} \right)^{-1} \mathbf{v}_d \\
&= \alpha (1 + \alpha + \ldots) \left(1 - \frac{\alpha}{1-\alpha} \frac{\rho_d}{\rho_c} + \ldots \right) \mathbf{v}_d
\end{aligned}$$

Ignoring squared and higher terms in the expansions, because $\frac{\rho_d}{\rho_c} \approx 2$ and α is small (0.002) $\implies \alpha^2$ and $\left(\frac{\alpha}{1-\alpha} \right)^2$ are very small:

$$F \approx \alpha v_d$$

where:

$$\begin{aligned}
v_d &= v_0 10^{-k\alpha} \\
&= v_0 e^{-k'\alpha}
\end{aligned}$$

and $k' = k \ln(10)$
Now,

$$\begin{aligned}
\frac{\partial F}{\partial \alpha} &= \frac{\partial (\alpha v_d)}{\partial \alpha} \\
&= \alpha \frac{\partial v_d}{\partial \alpha} + v_d \\
&= -k' \alpha v_d + v_d \\
&= v_0 (1 - k'\alpha) e^{-k'\alpha}
\end{aligned}$$

As $k \approx 285$, $k'\alpha \approx 1.3 \implies \frac{\partial F}{\partial \alpha}$ is not constant for the values of α used in the model.

However, OpenFOAM treats the two arguments of the divergence operator together, as follows:

$$\nabla \cdot F' = \nabla \cdot \left(\alpha \mathbf{v} + \frac{\alpha \rho_c}{\rho} \mathbf{v}_d \right)$$

Though a plot of F against α is not linear for the values used in the model:

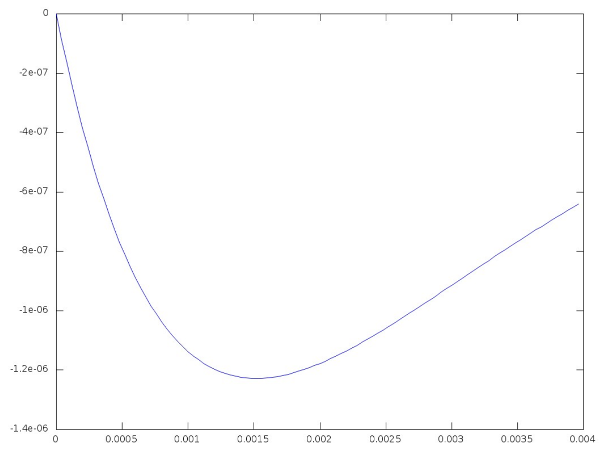

Figure 2.1: plot of F against α

a plot of F' against α is linear, because the first term in F' dominates the second term:

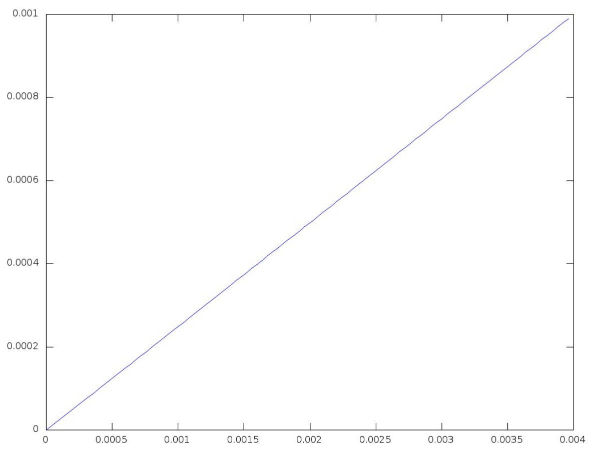

Figure 2.2: plot of F' against α

2.4 adjointSettlingFoam

2.4.1 The Adjoint Momentum Equation

$$-(\rho\mathbf{v}\cdot\nabla)\mathbf{u} - \nabla\mathbf{u}\cdot\rho\mathbf{v} = -\rho\nabla q + \nabla\cdot(\mu\nabla\mathbf{u}) + \nabla\cdot\left(\mu(\nabla\mathbf{u})^T\right) + \alpha\nabla\beta - \aleph\mathbf{u} \tag{2.4.1}$$

is coded in OpenFOAM as follows:

```
volVectorField adjointTransposeConvection((fvc::grad(Ua) & (U*rho)));

tmp<fvVectorMatrix> UaEqn
(
    fvm::div(-phi, Ua)
  - adjointTransposeConvection
  - fvm::laplacian(muEff, Ua, "laplacian(muEff,U)")
  //  - (fvc::grad(Ua) & fvc::grad(muEff))
  - fvc::grad(beta) * alpha
  + fvm::Sp(aleph, Ua)
);

UaEqn().relax();

solve
(
    UaEqn
  ==
    fvc::reconstruct
    (
        (
          - fvc::snGrad(pa) * fvc::interpolate(rho)
        )*mesh.magSf()
    )
);
```

Notes:

- `adjointTransposeConvection` is solved explicitly
- `(fvc::grad(Ua) & fvc::grad(muEff))` is commented out until everything else works for stability reasons
- `fvc::grad(beta) * alpha` calculates the values at the cell centres
- `fvm::Sp(aleph, Ua)` adds sink terms to the diagonal of matrix A
- `fvc::interpolate(rho)` calculates `rho` at the face centre
  - This is because `pa` is calculated at the face centre, i.e. the pressure equation is assembled with the face fluxes.
  - **I am actually struggling to see this.**
- `fvc::reconstruct` calculates `fvc::snGrad(pa) * fvc::interpolate(rho) * mesh.magSf()` at the cell centre
  - Note that in tetrahedral meshes, the `fvc::reconstruct` operator is not very accurate!
  - This is because you are creating a vector out of a scalar.

2.4.2 The Adjoint Drift Equation – aka the Beta Equation

$$-\big(\mathbf{v}\cdot\nabla-(\alpha k'-1)(\mathbf{v}_{dj}\cdot\nabla)\big)\beta=\nabla\cdot K\nabla\beta+(\rho_d-\rho_c)\big((\mathbf{v}\cdot\nabla)(\mathbf{u}\cdot\mathbf{v}-q)+\mathbf{u}\cdot\mathbf{g}\big) \quad (2.4.2)$$

To create a flux, $\beta=\dfrac{\rho \mathrm{B}}{\rho_d}$ is substituted into Eqn. (2.4.2):

$$-\big(\rho\mathbf{v}\cdot\nabla-\rho(\alpha k'-1)(\mathbf{v}_{dj}\cdot\nabla)\big)\mathrm{B}=\nabla\cdot\mu_t\nabla\mathrm{B}+(\rho_d-\rho_c)\big((\rho_d\mathbf{v}\cdot\nabla)(\mathbf{u}\cdot\mathbf{v}-q)+\rho_d\mathbf{u}\cdot\mathbf{g}\big) \quad (2.4.3)$$

and is coded in OpenFOAM as follows:

```
surfaceScalarField phiBeta
(
    ...
    phi
  - (
        fvc::interpolate(alpha)*kp
      - scalar(1)
    )*(mesh.Sf() & fvc::interpolate(Vdj))*fvc::interpolate(rho)
);

fvScalarMatrix betaEqn
(
      fvm::div(-phiBeta, Beta)
    - fvm::laplacian(mut, Beta)
);

surfaceScalarField phid
(
    ...
    rhod*phi/linearInterpolate(rho)
);

solve
(
    betaEqn
  ==
    (
        fvc::div(phid,Ua & U)
      - fvc::div(phid,pa)
      + rhod*(Ua & g)
    )*(rhod-rhoc)
);
```

Notes:

- `fvc::interpolate(alpha)`, `fvc::interpolate(Vdj)` and `fvc::interpolate(rho)` all calculate values at the face centres
- `mesh.Sf() & fvc::interpolate(Vdj)` is the dot product of $\mathbf{v_d}$ at the face centre with the face normal vector, whose magnitude is the face area, i.e. it gives the drift flux at the face centre.
- `solve (betaEqn == (fvc::div(phid,Ua & U) - fvc::div(phid,pa) + ...)` does not require `fvc::reconstruct` because all the values are calculated at the cell centres.

Chapter 3

Boundary Conditions

BC have value and behaviour: fixedValue, zeroGradient, etc.

3.1 FV Discretisation

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\mathbf{u}\phi) - \nabla \cdot (\gamma \nabla \phi) = \mathrm{S}_u - \mathrm{S}_p \phi \tag{3.1.1}$$

We want to discretise Eqn. (2.1.1) into the form:

$$A\phi = b$$

3.1.1 ddt Term

$$\int_{\mathrm{dV}} \frac{\partial \phi}{\partial t}\,\mathrm{dV} = \mathrm{V}_p \frac{\partial \phi_p}{\partial t} = \mathrm{V}_p \frac{\phi_p^n - \phi_p^o}{\Delta t} = \text{ Euler} \tag{3.1.2}$$

No BC are needed, because the terms are on the diagonal and ϕ_p^o goes into b.

3.1.2 Source Term

$$\int_{\mathrm{dV}} \mathrm{S}_u \,\mathrm{dV} = \mathrm{S}_{u_p} \mathrm{V}_p \tag{3.1.3}$$

No BC are needed, because the terms are on the diagonal and they go into b.

3.1.3 Sink Term

$$\int_{\mathrm{dV}} \mathrm{S}_p\, \phi \,\mathrm{dV} = \mathrm{S}_{p_p} \mathrm{V}_p\, \phi_p^n \tag{3.1.4}$$

No BC are needed, because the terms are on the diagonal and they go into A.

3.1.4 Convection Term

$$\int_{\mathrm{dV}} \nabla \cdot (\mathbf{u}\phi)\,\mathrm{dV} = \int_{\partial\mathrm{dV}} \mathrm{d}\mathbf{S} \cdot \mathbf{u}\phi = \sum_f \mathbf{S}_f \cdot \mathbf{u}_f \phi_f = \sum_f \mathrm{F}_f \phi_f = \sum_f \mathrm{F}_f (w\phi_P^n - (1-w)\phi_N^n) \tag{3.1.5}$$

We need $\mathrm{F}_b \phi_b$ at the boundary.

Dirichlet BC

In this case, ϕ_b is specified and $-\mathrm{F}_b\phi_b$ is a source term.

von Neumann BC

In this case, $\nabla\phi_b \cdot \mathbf{n} = \mathrm{g}_b$ is specified and:

$$\phi_b = \phi_P + |\mathbf{d}_b|\mathrm{g}_b \tag{3.1.6}$$

where $\mathbf{d}_b$ is the vector from the cell centre to the boundary centre.

$$\mathrm{F}_b\phi_b = \mathrm{F}_b\phi_P + \mathrm{F}_b|\mathbf{d}_b|\mathrm{g}_b \tag{3.1.7}$$

Hence, $\mathrm{F}_b\phi_P$ is a diagonal term and $\mathrm{F}_b|\mathbf{d}_b|\mathrm{g}_b$ is a source term.

3.1.5 Diffusion Term

$$\int_{\mathrm{dV}} \nabla \cdot (\gamma\nabla\phi)\,\mathrm{dV} = \int_{\partial\mathrm{dV}} \gamma \mathrm{d}\mathbf{S} \cdot \nabla\phi = \sum_f \gamma_f |\mathrm{S}_P| \frac{\phi_N - \phi_P}{|\mathbf{d}_f|} \tag{3.1.8}$$

$$\sum_f \gamma_f |\mathrm{S}_P| \frac{\phi_N - \phi_P}{|\mathbf{d}_f|} \to \gamma_b|\mathrm{S}_b|\frac{\phi_b - \phi_P}{|\mathbf{d}_b|} = \gamma_b|\mathrm{S}_b|\mathrm{g}_b \text{ at the boundary} \tag{3.1.9}$$

Note: $|\mathrm{S}_b|$ is magSf in OpenFOAM.

Dirichlet BC

In this case, ϕ_b is specified. Hence, $-\frac{\gamma_b|\mathrm{S}_b|}{|\mathbf{d}_b|}$ is a diagonal term and $-\frac{\gamma_b|\mathrm{S}_b|\phi_b}{|\mathbf{d}_b|}$ is a source term.

von Neumann BC

In this case, g_b is specified and $-\gamma_b|\mathrm{S}_b|\mathrm{g}_b$ is a source term.

3.1.6 Summary BC

		diagonal	source
Convection	Dirichlet		$-\mathrm{F}_b\phi_b$
	von Neumann	$\mathrm{F}_b\phi_P$	$\mathrm{F}_b\|\mathbf{d}_b\|\mathrm{g}_b$
Diffusion	Dirichlet	$-\frac{\gamma_b\|\mathrm{S}_b\|}{\|\mathbf{d}_b\|}$	$-\frac{\gamma_b\|\mathrm{S}_b\|\phi_b}{\|\mathbf{d}_b\|}$
	von Neumann		$-\gamma_b\|\mathrm{S}_b\|\mathrm{g}_b$

3.1.7 fvPatchField

	diagonal	source
Dirichlet	valueInternalCoeffs	valueBoundaryCoeffs
von Neumann	gradientInternalCoeffs	gradientBoundaryCoeffs

3.2 Mixed Boundary Condition

3.2.1 General Theory

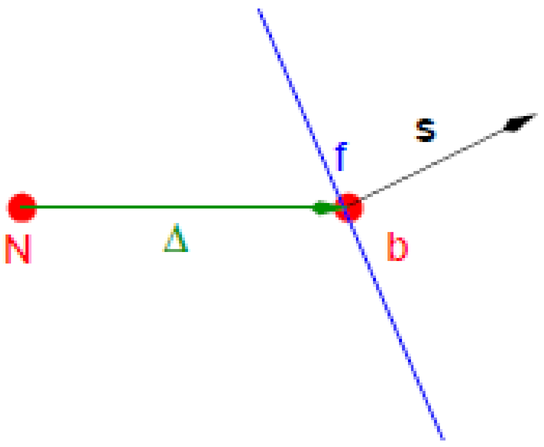

Figure 3.1: Boundary face: notation.

Here, f denotes the boundary face with face normal vector **s** and computational point b. The internal cell next to the face is denoted as N and the vector between N and b as $\mathbf{\Delta}$.

Stability is an issue when solving the adjoint system and it is important to set the BC to minimise stability issues. Therefore, where there is a mixed BC, we will use the Robin condition. The Robin condition for variable, ϕ, can be written in the form:

$$\phi_b = \omega\phi_V + (1-\omega)\phi_G \tag{3.2.1}$$

where:

- ϕ_b is the value at the boundary face centre
- ϕ_V is the contribution from the fixed value
- ϕ_G is the contribution from the gradient
- ω is the weighting assigned to the contribution from the fixed value $0 < \omega < 1$

Now,

$$\begin{aligned}\phi_G &= \phi_N + \mathbf{\Delta}\cdot\mathbf{g}_b \\ &= \phi_N + |\mathbf{\Delta}|g\end{aligned}$$

where:

- ϕ_N is the value at N
- $\mathbf{g}_b$ is the surface normal gradient at b
- $g = \mathbf{n}\cdot\mathbf{g}_b$ is the normal component of $\mathbf{g}_b$

Therefore, Eqn. (3.2.1) becomes:

$$\phi_b = \omega\phi_V + (1-\omega)(\phi_N + |\mathbf{\Delta}|g) \tag{3.2.2}$$

3.2.2 Example

The advection equation:

$$\frac{\partial \phi}{\partial t} + \mathbf{u}_w \cdot \nabla \phi = 0$$

can be discretised into:

$$\frac{\phi_b - \phi_b^o}{\delta t} + \mathbf{u}_w \frac{\phi_b - \phi_N}{|\mathbf{\Delta}|} = 0 \tag{3.2.3}$$

where ϕ_b^o is the old-time value of ϕ_b and δt is the time-step size.

Eqn. (3.2.3) can be rewritten as:

$$\phi_b \left(1 + \frac{\mathbf{u}_w \delta t}{|\mathbf{\Delta}|}\right) = \phi_b^o + \frac{\mathbf{u}_w \delta t}{|\mathbf{\Delta}|} \phi_N$$

Formalising the notation:

$$\phi_b (1 + A) = \phi_b^o + A \phi_N \tag{3.2.4}$$

and comparing Eqn. (3.2.4) with Eqn. (3.2.2), we have:

$$\begin{aligned} \phi_V &= \phi_b^o \\ g &= 0 \\ \omega &= \frac{1}{1+A} \end{aligned}$$

3.2.3 Adjoint Drift Flux System

In the adjoint Drift Flux system, there are 3 cases of the Robin condition, associated with the variable, β, at the wall, underflow and overflow, respectively:

$$v_{djn}(\alpha k' - 1)\beta - K(\mathbf{n} \cdot \nabla)\beta = 0 \tag{3.2.5a}$$

$$v_{djn}(\alpha k' - 1)\beta - K(\mathbf{n} \cdot \nabla)\beta = \rho_d \big(v_n - v_{djn}(\alpha k' - 1)\big) \tag{3.2.5b}$$

$$-\big(v_n - v_{djn}(\alpha k' - 1)\big)\beta - K(\mathbf{n} \cdot \nabla)\beta = v_n(\rho_d - \rho_c)(\mathbf{u} \cdot \mathbf{v} - q) - \rho_d u_n \mathbf{v}_{dj} \cdot \mathbf{v}_{dj}(2\alpha k' - 1) \tag{3.2.5c}$$

where:

- α is the dispersed-phase volume-fraction
- β is the adjoint dispersed-phase volume-fraction
- ρ_c is the continuum density
- ρ_d is the dispersed-phase density
- k' is a constant associated with the drift velocity
- K is the eddy diffusivity
- q is the adjoint pressure

- $u_n = \mathbf{u}.\mathbf{n}$ is the normal component of the adjoint velocity
- $v_{djn} = \mathbf{v}_{dj}.\mathbf{n}$ is the normal component of the drift velocity
- $v_n = \mathbf{v}.\mathbf{n}$ is the normal component of the primary velocity

Eqns. (3.2.5) can be formalised as follows:

$$A\beta_b + B(\mathbf{n} \cdot \nabla)\beta_b = C \tag{3.2.6}$$

where A, B, and C are known.

Discretising Eqn. (3.2.6), we have:

$$A\beta_b + B\frac{\beta_b - \beta_N}{\mathbf{\Delta} \cdot \mathbf{s}} = C \tag{3.2.7}$$

Rearranging Eqn. (3.2.7), we have:

$$\left(\frac{A}{B}\mathbf{\Delta} \cdot \mathbf{s} + 1\right)\beta_b = \beta_N + \frac{C}{B}\mathbf{\Delta} \cdot \mathbf{s} \tag{3.2.8}$$

Comparing Eqn. (3.2.8) with Eqn. (3.2.2), we have:

$$\beta_V = 0 \tag{3.2.9a}$$

$$g = \frac{C}{B}\frac{\mathbf{\Delta} \cdot \mathbf{s}}{|\mathbf{\Delta}|} \tag{3.2.9b}$$

$$\omega = \frac{1}{1 + \dfrac{B}{A\mathbf{\Delta} \cdot \mathbf{s}}} \tag{3.2.9c}$$

I am not sure if this is correct. I have asked Hrv to check it.