Chapter 1

Introduction

This guide accompanies the release of version 7 of the Open Source Field Operation and Manipulation (OpenFOAM) C++ libraries. It provides a description of the basic operation of OpenFOAM, first through a set of tutorial exercises in chapter 2 and later by a more detailed description of the individual components that make up OpenFOAM.

OpenFOAM is a framework for developing *application* executables that use packaged functionality contained within a collection of approximately 100 *C+ libraries*. OpenFOAM is shipped with approximately 250 pre-built applications that fall into two categories: *solvers*, that are each designed to solve a specific problem in fluid (or continuum) mechanics; and *utilities*, that are designed to perform tasks that involve data manipulation. The solvers in OpenFOAM cover a wide range of problems in fluid dynamics, as described in chapter 3.

Users can extend the collection of solvers, utilities and libraries in OpenFOAM, using some pre-requisite knowledge of the underlying method, physics and programming techniques involved.

OpenFOAM is supplied with pre- and post-processing environments. The interface to the pre- and post-processing are themselves OpenFOAM utilities, thereby ensuring consistent data handling across all environments. The overall structure of OpenFOAM is shown in Figure 1.1. The pre-processing and running of OpenFOAM cases is described in chapter 4.

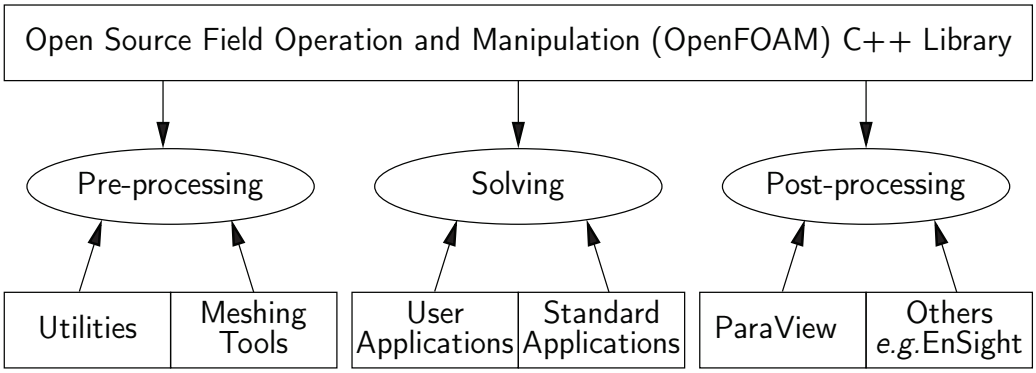

Figure 1.1: Overview of OpenFOAM structure.

In chapter 5, we cover both the generation of meshes using the mesh generator supplied with OpenFOAM and conversion of mesh data generated by third-party products. Post-processing is described in chapter 6 and some aspects of physical modelling, *e.g.* transport and thermophysical modelling, are described in in chapter 7.

The pressure field should appear as shown in Figure 2.5, with a region of low pressure at the top left of the cavity and one of high pressure at the top right of the cavity.

With the point icon (p) the pressure field is interpolated across each cell to give a continuous appearance. Instead if the user selects the cell icon, p, from the `Coloring` menu, a single value for pressure will be attributed to each cell so that each cell will be denoted by a single colour with no grading.

A colour legend can be added by either by clicking the Toggle Color Legend Visibility button in the `Active Variable Controls` toolbar or the Show button in the Coloring section of the Display panel. The legend can be located in the image window by drag and drop with the mouse. The Edit button, either in the `Active Variable Controls` toolbar or in the Coloring panel of the Display panel, opens the Color Map Editor window, as shown in Figure 2.6, where the user can set a range of attributes of the colour scale and the color bar. In particular, ParaView defaults to using a colour scale of blue to white to red rather than the more common blue to green to red (rainbow). Therefore *the first time* that the user executes ParaView, they may wish to change the colour scale. This can be done by selecting the Choose Preset button (with the heart icon) in the Color Scale Editor and selecting Blue to Red Rainbow. After clicking the OK confirmation button, the user can click the Save as Default button at the bottom of the panel (disk drive symbol) so that ParaView will always adopt this type of colour bar.

The user can also edit the color legend properties, such as text size, font selection and numbering format for the scale, by clicking the Edit Color Legend Properties to the far right of the search bar, as shown in Figure 2.6.

2.1.4.2 Cutting plane (slice)

If the user rotates the image, by holding down the left mouse button in the image window and moving the cursor, they can see that they have now coloured the complete geometry surface by the pressure. In order to produce a genuine 2-dimensional contour plot the user should first create a cutting plane, or 'slice'. With the `cavity.OpenFOAM` module highlighted in the Pipeline Browser, the user should select the `Slice` filter from the `Filters` menu in the top menu of ParaView (accessible at the top of the screen on some systems). The `Slice` filter can be initially found in the `Common` sub-menu, but once selected, it **moves** to the `Recent` sub-menu, **disappearing** from the the `Common` sub-menu. The cutting plane should be centred at $(0.05, 0.05, 0.005)$ and its normal should be set to $(0, 0, 1)$ (click the Z Normal button).

2.1.4.3 Contours

Having generated the cutting plane, contours can be created using by applying the `Contour` filter. With the `Slice` module highlighted in the Pipeline Browser, the user should select the `Contour` filter. In the Properties panel, the user should select pressure from the `Contour By` menu. Under `Isosurfaces`, the user could delete the default value with the minus button, then add a range of 10 values. The contours can be displayed with a `Wireframe` representation if the `Coloring` is solid or by a field, *e.g.* pressure.

2.1.4.4 Vector plots

Before we start to plot the vectors of the flow velocity, it may be useful to remove other modules that have been created, *e.g.* using the `Slice` and `Contour` filters described above.

The cavityFine case can be created by making a new case directory and copying the relevant directories from the cavity case.

```
mkdir cavityFine
cp -r cavity/constant cavityFine
cp -r cavity/system cavityFine
```

The user can then prepare to run the new case by changing into the case directory.

```
cd cavityFine
```

2.1.5.2 Creating the finer mesh

We now wish to increase the number of cells in the mesh by using blockMesh. The user should open the *blockMeshDict* file in the *system* directory in an editor and edit the block specification. The blocks are specified in a list under the `blocks` keyword. The syntax of the block definitions is described fully in section 5.3.1.3; at this stage it is sufficient to know that following `hex` is first the list of vertices in the block, then a list (or vector) of numbers of cells in each direction. This was originally set to `(20 20 1)` for the cavity case. The user should now change this to `(40 40 1)` and save the file. The new refined mesh should then be created by running blockMesh as before.

2.1.5.3 Mapping the coarse mesh results onto the fine mesh

The mapFields utility maps one or more fields relating to a given geometry onto the corresponding fields for another geometry. In our example, the fields are deemed 'consistent' because the geometry and the boundary types, or conditions, of both source and target fields are identical. We use the `-consistent` command line option when executing mapFields in this example.

The field data that mapFields maps is read from the time directory specified by `startFrom` and `startTime` in the *controlDict* of the target case, *i.e.* those **into which** the results are being mapped. In this example, we wish to map the final results of the coarser mesh from case cavity onto the finer mesh of case cavityFine. Therefore, since these results are stored in the *0.5* directory of cavity, the `startTime` should be set to 0.5 s in the *controlDict* dictionary and `startFrom` should be set to `startTime`.

The case is ready to run mapFields. Typing `mapFields -help` quickly shows that mapFields requires the source case directory as an argument. We are using the `-consistent` option, so the utility is executed from withing the *cavityFine* directory by

```
mapFields ../cavity -consistent
```

The utility should run with output to the terminal including:

```
Source: ".." "cavity"
Target: "." "cavityFine"

Create databases as time

Source time: 0.5
Target time: 0.5
```

```
Create meshes

Source mesh size: 400   Target mesh size: 1600

Consistently creating and mapping fields for time 0.5

    interpolating p
    interpolating U

End
```

2.1.5.4 Control adjustments

To maintain a Courant number of less that 1, as discussed in section 2.1.1.4, the time step must now be halved since the size of all cells has halved. Therefore `deltaT` should be set to to 0.0025 s in the *controlDict* dictionary. Field data is currently written out at an interval of a fixed number of time steps. Here we demonstrate how to specify data output at fixed intervals of time. Under the `writeControl` keyword in *controlDict*, instead of requesting output by a fixed number of time steps with the `timeStep` entry, a fixed amount of run time can be specified between the writing of results using the `runTime` entry. In this case the user should specify output every 0.1 and therefore should set `writeInterval` to `0.1` and `writeControl` to `runTime`. Finally, since the case is starting with a the solution obtained on the coarse mesh we only need to run it for a short period to achieve reasonable convergence to steady-state. Therefore the `endTime` should be set to 0.7 s. Make sure these settings are correct and then save the file.

2.1.5.5 Running the code as a background process

The user should experience running icoFoam as a background process, redirecting the terminal output to a *log* file that can be viewed later. From the *cavityFine* directory, the user should execute:

```
icoFoam > log &
cat log
```

2.1.5.6 Vector plot with the refined mesh

The user can open multiple cases simultaneously in ParaView; essentially because each new case is simply another module that appears in the Pipeline Browser. There is an inconvenience when opening a new OpenFOAM case in ParaView because it expects that case data is stored in a single file which has a file extension that enables it to establish the format. However, OpenFOAM stores case data in multiple files without an extension in the name, within a specific directory structure. The ParaView reader module works on the basis that, when opening case data in OpenFOAM format, it is passed a dummy (empty) file with the *.OpenFOAM* extension that resides in the case directory. The paraFoam script automatically creates this file — hence, the cavity case module is called `cavity.OpenFOAM`.

If the user wishes to open a second case directly from within ParaView, they need to create such a dummy file. They can do this 'by hand' or, more simply, use the paraFoam script with the option `-touch`. For the cavityFine example, that involves executing from the case directory:

```
35      {
36          type            empty;
37      }
38  }
39
40
41  // ************************************************************************* //
```

This case uses standard wall functions, specified by the `nutWallFunction` type on the `movingWall` and `fixedWalls` patches. Other wall function models include the rough wall functions, specified though the `nutRoughWallFunction` keyword.

The user should now open the field files for k and ε (*0/k* and *0/epsilon*) and examine their boundary conditions. For a wall boundary condition, ε is assigned a epsilonWallFunction boundary condition and a kqRwallFunction boundary condition is assigned to k. The latter is a generic boundary condition that can be applied to any field that are of a turbulent kinetic energy type, *e.g.* k, q or Reynolds Stress R. The initial values for k and ε are set using an estimated fluctuating component of velocity $\mathbf{U}'$ and a turbulent length scale, l. k and ε are defined in terms of these parameters as follows:

$$k = \frac{1}{2}\overline{\mathbf{U}' \bullet \mathbf{U}'} \tag{2.8}$$

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} \tag{2.9}$$

where C_μ is a constant of the $k-\varepsilon$ model equal to 0.09. For a Cartesian coordinate system, k is given by:

$$k = \frac{1}{2}({U'_x}^2 + {U'_y}^2 + {U'_z}^2) \tag{2.10}$$

where ${U'_x}^2$, ${U'_y}^2$ and ${U'_z}^2$ are the fluctuating components of velocity in the x, y and z directions respectively. Let us assume the initial turbulence is isotropic, *i.e.* ${U'_x}^2 = {U'_y}^2 = {U'_z}^2$, and equal to 5% of the lid velocity and that l, is equal to 5% of the box width, 0.1 m, then k and ε are given by:

$$U'_x = U'_y = U'_z = \frac{5}{100} 1 \text{ m s}^{-1} \tag{2.11}$$

$$\Rightarrow k = \frac{3}{2}\left(\frac{5}{100}\right)^2 \text{ m}^2\,\text{s}^{-2} = 3.75 \times 10^{-3} \text{ m}^2\,\text{s}^{-2} \tag{2.12}$$

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{l} \approx 7.54 \times 10^{-3} \text{ m}^2\text{s}^{-3} \tag{2.13}$$

These form the initial conditions for k and ε. The initial conditions for $\mathbf{U}$ and p are $(0, 0, 0)$ and 0 respectively as before.

Turbulence modelling includes a range of methods, *e.g.* RAS or large-eddy simulation (LES), that are provided in OpenFOAM. The choice of turbulence modelling method is selectable at run-time through the `simulationType` keyword in *turbulenceProperties* dictionary. The user can view this file in the *constant* directory:

```
17
18  simulationType  RAS;
19
20  RAS
21  {
22      RASModel        kEpsilon;
23
```

```
24      turbulence      on;
25
26      printCoeffs     on;
27  }
28
29  // ************************************************************************* //
```

The options for `simulationType` are `laminar`, `RAS` and `LES`. With `RAS` selected in this case, the choice of RAS modelling is specified in a `RAS` subdictionary. The turbulence model is selected by the `RASModel` entry from a long list of available models that are listed in Section 7.2.1.1. The `kEpsilon` model should be selected which is is the standard $k-\varepsilon$ model; the user should also ensure that `turbulence` calculation is switched `on`.

The coefficients for each turbulence model are stored within the respective code with a set of default values. Setting the optional switch called `printCoeffs` to `on` will make the default values be printed to standard output, *i.e.* the terminal, when the model is called at run time. The coefficients are printed out as a sub-dictionary whose name is that of the model name with the word `Coeffs` appended, *e.g.* `kEpsilonCoeffs` in the case of the `kEpsilon` model. The coefficients of the model, *e.g.* `kEpsilon`, can be modified by optionally including (copying and pasting) that sub-dictionary within the `RAS` sub-dictionary and adjusting values accordingly.

The user should next set the laminar kinematic viscosity in the *transportProperties* dictionary. To achieve a Reynolds number of 10^4, a kinematic viscosity of 10^{-5} m is required based on the Reynolds number definition given in Equation 2.1.

Finally the user should set the `startTime`, `stopTime`, `deltaT` and the `writeInterval` in the *controlDict*. Set `deltaT` to 0.005 s to satisfy the Courant number restriction and the `endTime` to 10 s.

2.1.8.2 Running the code

Execute pisoFoam by entering the case directory and typing "`pisoFoam`" in a terminal. In this case, where the viscosity is low, the boundary layer next to the moving lid is very thin and the cells next to the lid are comparatively large so the velocity at their centres are much less than the lid velocity. In fact, after $\approx$ 100 time steps it becomes apparent that the velocity in the cells adjacent to the lid reaches an upper limit of around 0.2 $\mathrm{m\,s^{-1}}$ hence the maximum Courant number does not rise much above 0.2. It is sensible to increase the solution time by increasing the time step to a level where the Courant number is much closer to 1. Therefore reset `deltaT` to 0.02 s and, on this occasion, set `startFrom` to `latestTime`. This instructs pisoFoam to read the start data from the latest time directory, *i.e.10.0*. The `endTime` should be set to 20 s since the run converges a lot slower than the laminar case. Restart the run as before and monitor the convergence of the solution. View the results at consecutive time steps as the solution progresses to see if the solution converges to a steady-state or perhaps reaches some periodically oscillating state. In the latter case, convergence may never occur but this does not mean the results are inaccurate.

2.1.9 Changing the case geometry

A user may wish to make changes to the geometry of a case and perform a new simulation. It may be useful to retain some or all of the original solution as the starting conditions for the new simulation. This is a little complex because the fields of the original solution are not consistent with the fields of the new case. However the mapFields utility can map fields that are inconsistent, either in terms of geometry or boundary types or both.

This essentially switches off the time derivative terms. Not all solvers, especially in fluid dynamics, work for both steady-state and transient problems but solidDisplacementFoam does work, since the base algorithm is the same for both types of simulation.

The momentum equation in linear-elastic stress analysis includes several explicit terms containing the gradient of displacement. The calculations benefit from accurate and smooth evaluation of the gradient. Normally, in the finite volume method the discretisation is based on Gauss's theorem The Gauss method is sufficiently accurate for most purposes but, in this case, the least squares method will be used. The user should therefore open the `fvSchemes` dictionary in the *system* directory and ensure the `leastSquares` method is selected for the `grad(U)` gradient discretisation scheme in the `gradSchemes` sub-dictionary:

```
17
18  d2dt2Schemes
19  {
20      default         steadyState;
21  }
22
23  ddtSchemes
24  {
25      default         Euler;
26  }
27
28  gradSchemes
29  {
30      default         leastSquares;
31      grad(D)         leastSquares;
32      grad(T)         leastSquares;
33  }
34
35  divSchemes
36  {
37      default         none;
38      div(sigmaD)     Gauss linear;
39  }
40
41  laplacianSchemes
42  {
43      default         none;
44      laplacian(DD,D) Gauss linear corrected;
45      laplacian(DT,T) Gauss linear corrected;
46  }
47
48  interpolationSchemes
49  {
50      default         linear;
51  }
52
53  snGradSchemes
54  {
55      default         none;
56  }
57
58  // ************************************************************************* //
```

The *fvSolution* dictionary in the *system* directory controls the linear equation solvers and algorithms used in the solution. The user should first look at the *solvers* sub-dictionary and notice that the choice of `solver` for `D` is `GAMG`. The solver `tolerance` should be set to 10^{-6} for this problem. The solver relative tolerance, denoted by `relTol`, sets the required reduction in the residuals within each iteration. It is uneconomical to set a tight (low) relative tolerance within each iteration since a lot of terms in each equation are explicit and are updated as part of the segregated iterative procedure. Therefore a reasonable value for the relative tolerance is 0.01, or possibly even higher, say 0.1, or in some cases even 0.9 (as in this case).

```
17
18  solvers
```

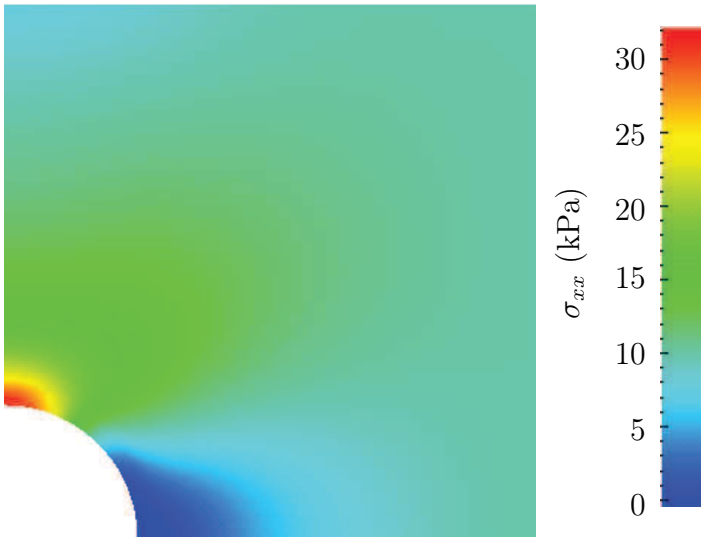

Figure 2.19: σ_{xx} stress field in the plate with hole.

Components named `sigmaxx`, `sigmaxy` *etc.* are written to time directories of the case. The σ_{xx} stresses can be viewed in paraFoam as shown in Figure 2.19.

We would like to compare the analytical solution of Equation 2.14 to our solution. We therefore must output a set of data of σ_{xx} along the left edge symmetry plane of our domain. The user may generate the required graph data using the postProcess utility with the `singleGraph` function. Unlike earlier examples of postProcess where no configuration is required, this example includes a *singleGraph* file pre-configured in the *system* directory. The sample line is set between $(0.0, 0.5, 0.25)$ and $(0.0, 2.0, 0.25)$, and the fields are specified in the `fields` list:

```
singleGraph
{
    start   (0 0.5 0.25);
    end     (0   2 0.25);
    fields  (sigmaxx);

    #includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"

    setConfig
    {
        axis    y;
    }

    // Must be last entry
    #includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
}

// ************************************************************************* //
```

The user should execute postProcessing with the `singleGraph` function:

```
postProcess -func "singleGraph"
```

Data is written is `raw` 2 column format into files within time subdirectories of a *postProcessing/singleGraph* directory, *e.g.* the data at $t = 100$ s is found within the file *singleGraph/100/line_sigmaxx.xy*. If the user has GnuPlot installed they launch it (by typing `gnuplot`) and then plot both the numerical data and analytical solution as follows:

```
plot [0.5:2] [0:] "postProcessing/singleGraph/100/line _sigmaxx.xy",
```

3.5.11 Stress analysis of solids

solidDisplacementFoam Transient segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses.

solidEquilibriumDisplacementFoam Steady-state segregated finite-volume solver of linear-elastic, small-strain deformation of a solid body, with optional thermal diffusion and thermal stresses.

3.5.12 Finance

financialFoam Solves the Black-Scholes equation to price commodities.

3.6 Standard utilities

The utilities with the OpenFOAM distribution are in the *$FOAM_UTILITIES* directory. The names are reasonably descriptive, *e.g.* ideasToFoam converts mesh data from the format written by I-DEAS to the OpenFOAM format. The descriptions of current utilities distributed with OpenFOAM are given in the following Sections.

3.6.1 Pre-processing

applyBoundaryLayer Apply a simplified boundary-layer model to the velocity and turbulence fields based on the 1/7th power-law.

boxTurb Makes a box of turbulence which conforms to a given energy spectrum and is divergence free.

changeDictionary Utility to change dictionary entries, e.g. can be used to change the patch type in the field and `polyMesh`/boundary files.

createExternalCoupledPatchGeometry Application to generate the patch geometry (points and faces) for use with the `externalCoupled` boundary condition.

dsmcInitialise Initialise a case for `dsmcFoam` by reading the initialisation dictionary system/-`dsmcInitialise`.

engineSwirl Generates a swirling flow for engine calulations.

faceAgglomerate Agglomerate boundary faces using the pairPatchAgglomeration algorithm. It writes a map from the fine to coarse grid.

foamSetupCHT Sets up a multi-region case using template files for material properties, field and system files.

foamUpgradeCyclics Tool to upgrade mesh and fields for split cyclics.

mapFields Maps volume fields from one mesh to another, reading and interpolating all fields present in the time directory of both cases. Parallel and non-parallel cases are handled without the need to reconstruct them first. belongs to mapFieldsPar

- **Nonuniform field** each field element is assigned a unique value from a list, taking the following form where the token identifier form of list is recommended:

```
internalField nonuniform <List>;
```

The `boundaryField` is a dictionary containing a set of entries whose names correspond to each of the names of the boundary patches listed in the *boundary* file in the *polyMesh* directory. Each patch entry is itself a dictionary containing a list of keyword entries. The mandatory entry, `type`, describes the patch field condition specified for the field. The remaining entries correspond to the type of patch field condition selected and can typically include field data specifying initial conditions on patch faces. A selection of patch field conditions available in OpenFOAM are listed in section 5.2.1, section 5.2.2 and section 5.2.3, with a description and the data that must be specified with it. Example field dictionary entries for velocity `U` are shown below:

```
17  dimensions      [0 1 -1 0 0 0 0];
18
19  internalField   uniform (0 0 0);
20
21  boundaryField
22  {
23      movingWall
24      {
25          type            fixedValue;
26          value           uniform (1 0 0);
27      }
28
29      fixedWalls
30      {
31          type            noSlip;
32      }
33
34      frontAndBack
35      {
36          type            empty;
37      }
38  }
39
40  // ************************************************************************* //
```

4.2.9 Macro expansion

OpenFOAM dictionary files include a macro syntax to allow convenient configuration of case files. The syntax uses the the dollar (`$`) symbol in front of a keyword to expand the data associated with the keyword. For example the value set for keyword `a` below, 10, is expanded in the following line, so that the value of `b` is also 10.

```
a 10;
b $a;
```

Variables can be accessed within different levels of sub-dictionaries, or scope. Scoping is performed using a '`.`' (dot) syntax, illustrated by the following example, where `b` is set to the value of `a`, specified in a sub-dictionary called `subdict`.

```
subdict
{
    a 10;
}
b $subdict.a;
```

```
div(phiSt,b)    Gauss limitedLinear01 1;
```

The underlying scheme is `limitedLinear`, specialised for stronger bounding between 0 and 1 by adding `01` to the name of the scheme.

The `multivariateSelection` mechanism also exists for grouping multiple equation terms together, and applying the same limiters on all terms, using the strongest limiter calculated for all terms. A good example of this is in a set of mass transport equations for fluid species, where it is good practice to apply the same discretisation to all equations for consistency. The example below comes from the smallPoolFire3D tutorial in *$FOAM_TUTORIALS/combustion/fireFoam/les*, in which the equation for enthalpy h is included with the specie mass transport equations in the calculation of a single limiter.

```
div(phi,Yi_h)   Gauss multivariateSelection
{
    O2 limitedLinear01 1;
    CH4 limitedLinear01 1;
    N2 limitedLinear01 1;
    H2O limitedLinear01 1;
    CO2 limitedLinear01 1;
    h limitedLinear 1 ;
}
```

4.5.4 Surface normal gradient schemes

It is worth explaining the *snGradSchemes* sub-dictionary that contains surface normal gradient terms, before discussion of *laplacianSchemes*, because they are required to evaluate a Laplacian term using Gaussian integration. A surface normal gradient is evaluated at a cell face; it is the component, normal to the face, of the gradient of values at the centres of the 2 cells that the face connects.

A search for the `default` scheme for *snGradSchemes* reveals the following entries.

```
default         corrected;
default         limited corrected 0.33;
default         limited corrected 0.5;
default         orthogonal;
default         uncorrected;
```

The basis of the gradient calculation at a face is to subtract the value at the cell centre on one side of the face from the value in the centre on the other side and divide by the distance. The calculation is second-order accurate for the gradient *normal to the face* if the vector connecting the cell centres is orthogonal to the face, *i.e.* they are at right-angles. This is the `orthogonal` scheme.

Orthogonality requires a regular mesh, typically aligned with the Catersian co-ordinate system, which does not normally occur in meshes for real world, engineering geometries. Therefore, to maintain second-order accuracy, an explicit non-orthogonal correction can be added to the orthogonal component, known as the `corrected` scheme. The correction

increases in size as the non-orthonality, the angle α between the cell-cell vector and face normal vector, increases.

As α tends towards 90°, *e.g.* beyond 70°, the explicit correction can be so large to cause a solution to go unstable. The solution can be stabilised by applying the `limited` scheme to the correction which requires a coefficient $\psi, 0 \leq \psi \leq 1$ where

$$\psi = \begin{cases} 0 & \text{corresponds to } \texttt{uncorrected}, \\ 0.333 & \text{non-orthogonal correction} \leq 0.5 \times \text{orthogonal part}, \\ 0.5 & \text{non-orthogonal correction} \leq \text{orthogonal part}, \\ 1 & \text{corresponds to } \texttt{corrected}. \end{cases} \tag{4.2}$$

Typically, *psi* is chosen to be 0.33 or 0.5, where 0.33 offers greater stability and 0.5 greater accuracy.

The corrected scheme applies under-relaxation in which the implicit orthogonal calculation is increased by $cos^{-1}\alpha$, with an equivalent boost within the non-orthogonal correction. The `uncorrected` scheme is equivalent to the `corrected` scheme, without the non-orthogonal correction, so includes is like `orthogonal` but with the $cos^{-1}\alpha$ under-relaxation.

Generally the `uncorrected` and `orthogonal` schemes are only recommended for meshes with very low non-orthogonality (*e.g.* maximum 5°). The `corrected` scheme is generally recommended, but for maximum non-orthogonality above 70°, `limited` may be required. At non-orthogonality above 80°, convergence is generally hard to achieve.

4.5.5 Laplacian schemes

The *laplacianSchemes* sub-dictionary contains Laplacian terms. A typical Laplacian term is $\nabla \bullet (\nu \nabla \mathbf{U})$, the diffusion term in the momentum equations, which corresponds to the keyword `laplacian(nu,U)` in *laplacianSchemes*. The `Gauss` scheme is the only choice of discretisation and requires a selection of both an interpolation scheme for the diffusion coefficient, *i.e.* ν in our example, and a surface normal gradient scheme, *i.e.* $\nabla \mathbf{U}$. To summarise, the entries required are:

```
Gauss <interpolationScheme> <snGradScheme>
```

The user can search for the `default` scheme for *laplacianSchemes* in all the cases in the *$FOAM_TUTORIALS* directory.

```
foamSearch $FOAM_TUTORIALS fvSchemes laplacianSchemes. default
```

It reveals the following entries.

```
default         Gauss linear corrected;
default         Gauss linear limited corrected 0.33;
default         Gauss linear limited corrected 0.5;
default         Gauss linear orthogonal;
default         Gauss linear uncorrected;
```

In all cases, the `linear` interpolation scheme is used for interpolation of the diffusivity. The cases uses the same array of `snGradSchemes` based on level on non-orthogonality, as described in section 4.5.4.

- `mergeLevels`: keyword controls the speed at which coarsening or refinement is performed; the default is 1, which is safest, but for simple meshes, the solution speed can be increased by coarsening/refining 2 levels at a time, *i.e.* setting `mergeLevels 2`.

Smoothing is specified by the `smoother` as described in section 4.6.1.3. The number of sweeps used by the smoother at different levels of mesh density are specified by the following optional entries.

- `nPreSweeps`: number of sweeps as the algorithm is coarsening (default 0).
- `preSweepsLevelMultiplier`: multiplier for the the number of sweeps between each coarsening level (default 1).
- `maxPreSweeps`: maximum number of sweeps as the algorithm is coarsening (default 4).
- `nPostSweeps`: number of sweeps as the algorithm is refining (default 2).
- `postSweepsLevelMultiplier`: multiplier for the the number of sweeps between each refinement level (default 1).
- `maxPostSweeps`: maximum number of sweeps as the algorithm is refining (default 4).
- `nFinestSweeps`: number of sweeps at finest level (default 2).

4.6.2 Solution under-relaxation

A second sub-dictionary of *fvSolution* that is often used in OpenFOAM is *relaxationFactors* which controls under-relaxation, a technique used for improving stability of a computation, particularly in solving steady-state problems. Under-relaxation works by limiting the amount which a variable changes from one iteration to the next, either by modifying the solution matrix and source prior to solving for a field or by modifying the field directly. An under-relaxation factor $\alpha, 0 < \alpha \leq 1$ specifies the amount of under-relaxation, as described below.

- No specified α: no under-relaxation.
- $\alpha = 1$: guaranteed matrix diagonal equality/dominance.
- α decreases, under-relaxation increases.
- $\alpha = 0$: solution does not change with successive iterations.

An optimum choice of α is one that is small enough to ensure stable computation but large enough to move the iterative process forward quickly; values of α as high as 0.9 can ensure stability in some cases and anything much below, say, 0.2 are prohibitively restrictive in slowing the iterative process.

Relaxation factors for under-relaxation of fields are specified within a *field* sub-dictionary; relaxation factors for equation under-relaxation are within a *equations* sub-dictionary. An example is shown below from tutorial example of simpleFoam, showing typical settings for an incompressible steady-state solver. The factors are specified for pressure `p`, pressure `U`, and turbulent fields grouped using a regular expression.

- mixed: mixed fixedValue/ fixedGradient condition depending on `valueFraction` ($0 \leq$ `valueFraction` ≤ 1) where

$$\texttt{valueFraction} = \begin{cases} 1 & \text{corresponds to } \mathbf{Q} = \texttt{refValue}, \\ 0 & \text{corresponds to } \partial\mathbf{Q}/\partial n = \texttt{refGradient}. \end{cases} \tag{5.1}$$

- directionMixed: mixed condition with tensorial `valueFraction`, to allow different conditions in normal and tangential directions of a vector patch field, *e.g.* fixedValue in the tangential direction, zeroGradient in the normal direction.

5.2.3 Derived types

There are numerous more complex boundary conditions derived from the basic conditions. For example, many complex conditions are derived from fixedValue, where the value is calculated by a function of other patch fields, time, geometric information, *etc.* Some other conditions derived from mixed/directionMixed switch between fixedValue and fixedGradient (usually zeroGradient).

There are a number of ways the user can list the available boundary conditions in OpenFOAM, with the -listScalarBCs and -listVectorBCs utility being the quickest. The boundary conditions for scalar fields and vector fields, respectively, can be listed for a given solver, *e.g.* simpleFoam, as follows.

```
simpleFoam -listScalarBCs -listVectorBCs
```

These produce longs lists which the user can scan through. If the user wants more information of a particular condition, they can run the foamInfo script which provides a description of the boundary condition and lists example cases where it is used. For example, for the totalPressure boundary condition, run the following.

```
foamInfo totalPressure
```

In the following sections we will highlight some particular important, commonly used boundary conditions.

5.2.3.1 The inlet/outlet condition

The inletOutlet condition is one derived from mixed, which switches between zeroGradient when the fluid flows out of the domain at a patch face, and fixedValue, when the fluid is flowing into the domain. For inflow, the inlet value is specified by an `inletValue` entry. A good example of its use can be seen in the damBreak tutorial, where it is applied to the phase fraction on the upper `atmosphere` boundary. Where there is outflow, the condition is well posed, where there is inflow, the phase fraction is fixed with a value of 0, corresponding to 100% air.

```
17  dimensions      [0 0 0 0 0 0 0];
18
19  internalField   uniform 0;
20
21  boundaryField
22  {
23      leftWall
```

5.5.2.7 Converting the mesh to OpenFOAM format

The translator utility starToFoam can now be run to create the boundaries, cells and points files necessary for a OpenFOAM run:

```
starToFoam <meshFilePrefix>
```

where <meshFilePrefix> is the name of the the prefix of the mesh files, including the full or relative path. After the utility has finished running, OpenFOAM boundary types should be specified by editing the *boundary* file by hand.

5.5.3 gambitToFoam

GAMBIT writes mesh data to a single file with a *.neu* extension. The procedure of converting a GAMBIT*.neu* file is first to create a new OpenFOAM case, then at a command prompt, the user should execute:

```
gambitToFoam <meshFile>
```

where <meshFile> is the name of the *.neu* file, including the full or relative path.

The GAMBIT file format does not provide information about type of the boundary patch, *e.g.* wall, symmetry plane, cyclic. Therefore all the patches have been created as type patch. Please reset after mesh conversion as necessary.

5.5.4 ideasToFoam

OpenFOAM can convert a mesh generated by I-DEAS but written out in ANSYS format as a *.ans* file. The procedure of converting the *.ans* file is first to create a new OpenFOAM case, then at a command prompt, the user should execute:

```
ideasToFoam <meshFile>
```

where <meshFile> is the name of the *.ans* file, including the full or relative path.

5.5.5 cfx4ToFoam

CFX writes mesh data to a single file with a *.geo* extension. The mesh format in CFX is block-structured, *i.e.* the mesh is specified as a set of blocks with glueing information and the vertex locations. OpenFOAM will convert the mesh and capture the CFX boundary condition as best as possible. The 3 dimensional 'patch' definition in CFX, containing information about the porous, solid regions *etc.* is ignored with all regions being converted into a single OpenFOAM mesh. CFX supports the concept of a 'default' patch, where each external face without a defined boundary condition is treated as a wall. These faces are collected by the converter and put into a defaultFaces patch in the OpenFOAM mesh and given the type wall; of course, the patch type can be subsequently changed.

Like, OpenFOAM 2 dimensional geometries in CFX are created as 3 dimensional meshes of 1 cell thickness. If a user wishes to run a 2 dimensional case on a mesh created by CFX, the boundary condition on the front and back planes should be set to empty; the user should

6.1.2 The Parameters panel

The Properties window for the case module includes the Paramters panel that contains the settings for mesh, fields and global controls. The controls are described in Figure 6.2. The

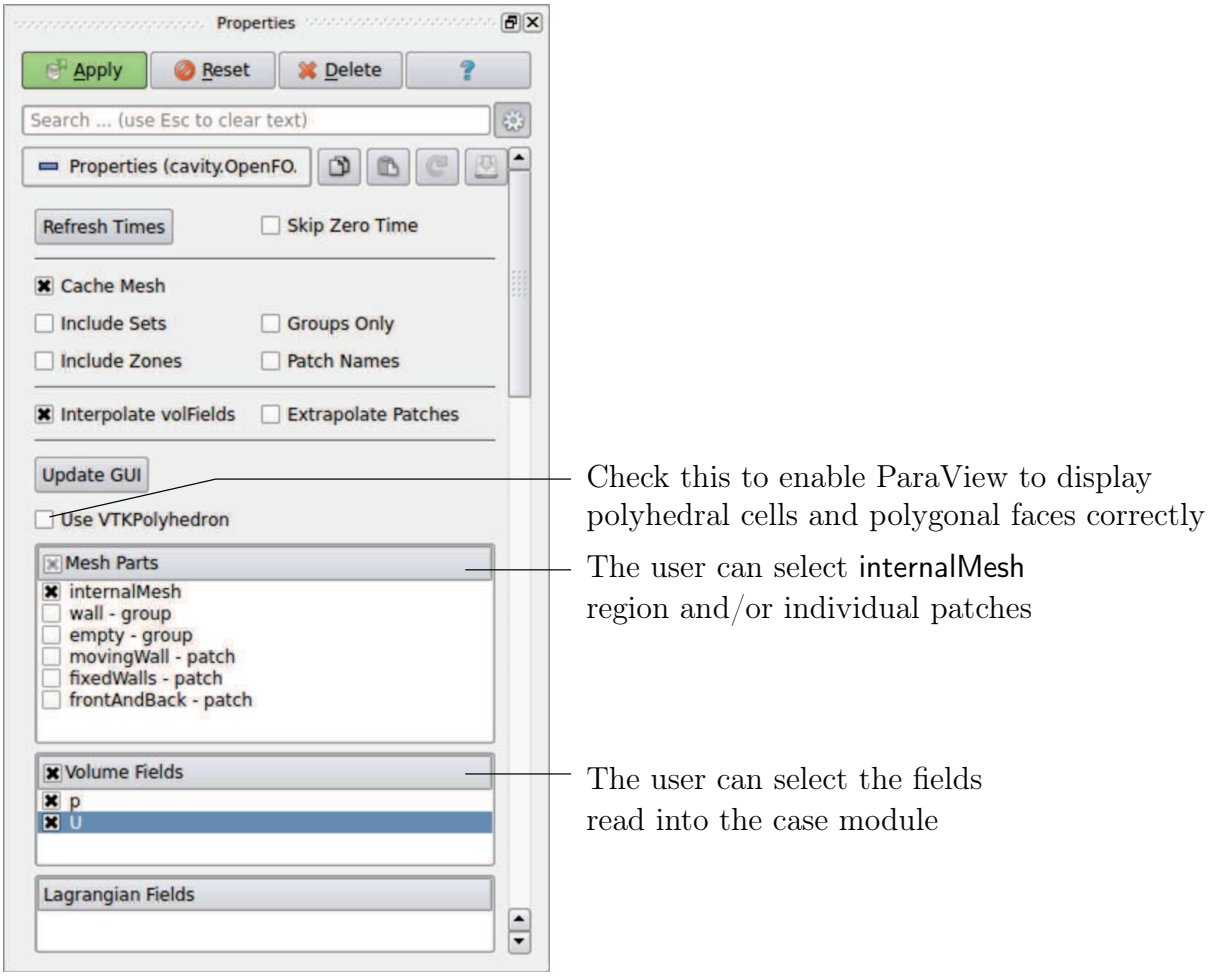

Figure 6.2: The Properties panel for the case module

user can select mesh and field data which is loaded for all time directories into ParaView. The buttons in the `Current Time Controls` and `VCR Controls` toolbars then select the time data to be displayed, as shown is section 6.1.4.

As with any operation in paraFoam, the user must click Apply after making any changes to any selections. The Apply button is highlighted in green to alert the user if changes have been made but not accepted. This method of operation has the advantage of allowing the user to make a number of selections before accepting them, which is particularly useful in large cases where data processing is best kept to a minimum.

If new data is written to time directories while the user is running ParaView, the user must load the additional time directories by checking the Refresh Times button. Where there are occasions when the case data changes on file and ParaView needs to load the changes, the user can also check the Update GUI button in the Parameters panel and apply the changes.

The Lights button opens detailed lighting controls within the Light Kit panel. A separate Headlight panel controls the direct lighting of the image. Checking the Headlight button with white light colour of strength 1 seems to help produce images with strong bright colours, e.g. with an isosurface.

The Camera Parallel Projection is is the usual choice for CFD, especially for 2D cases, and so should generally be checked. Other settings include Cube Axes which displays axes on the selected object to show the its orientation and geometric dimensions.

6.1.5.2 General settings

The general `Settings` are selected from the `Edit` menu, which opens a general Options window with General, Colors, Animations, Charts and Render View menu items.

The General panel controls some default behaviour of ParaView. In particular, there is an Auto Accept button that enables ParaView to accept changes automatically without clicking the green Apply button in the Properties window. For larger cases, this option is generally not recommended: the user does not generally want the image to be re-rendered between each of a number of changes he/she selects, but be able to apply a number of changes to be re-rendered in their entirety once.

The Render View panel contains 3 sub-items: General, Camera and Server. The General panel includes the level of detail (LOD) which controls the rendering of the image while it is being manipulated, *e.g.* translated, resized, rotated; lowering the levels set by the sliders, allows cases with large numbers of cells to be re-rendered quickly during manipulation.

The Camera panel includes control settings for 3D and 2D movements. This presents the user with a map of rotation, translate and zoom controls using the mouse in combination with Shift- and Control-keys. The map can be edited to suit by the user.

6.1.6 Contour plots

A contour plot is created by selecting `Contour` from the `Filter` menu at the top menu bar. The filter acts on a given module so that, if the module is the 3D case module itself, the contours will be a set of 2D surfaces that represent a constant value, *i.e.* isosurfaces. The Properties panel for contours contains an Isosurfaces list that the user can edit, most conveniently by the New Range window. The chosen scalar field is selected from a pull down menu.

6.1.6.1 Introducing a cutting plane

Very often a user will wish to create a contour plot across a plane rather than producing isosurfaces. To do so, the user must first use the `Slice` filter to create the cutting plane, on which the contours can be plotted. The `Slice` filter allows the user to specify a cutting `Plane`, `Box` or `Sphere` in the `Slice Type` menu by a center and normal/radius respectively. The user can manipulate the cutting plane like any other using the mouse.

The user can then run the `Contour` filter on the cut plane to generate contour lines.

6.1.7 Vector plots

Vector plots are created using the `Glyph` filter. The filter reads the field selected in `Vectors` and offers a range of `Glyph Types` for which the `Arrow` provides a clear vector plot images.

Each glyph has a selection of graphical controls in a panel which the user can manipulate to best effect.

The remainder of the Properties panel contains mainly the `Scale Mode` menu for the glyphs. The most common options are `Scale Mode` are: `Vector`, where the glyph length is proportional to the vector magnitude; and, `Off` where each glyph is the same length. The Set Scale Factor parameter controls the base length of the glyphs.

6.1.7.1 Plotting at cell centres

Vectors are by default plotted on cell vertices but, very often, we wish to plot data at cell centres. This is done by first applying the `Cell Centers` filter to the case module, and then applying the `Glyph` filter to the resulting cell centre data.

6.1.8 Streamlines

Streamlines are created by first creating tracer lines using the `Stream Tracer` filter. The tracer Seed panel specifies a distribution of tracer points over a `Line Source` or `Point Cloud`. The user can view the tracer source, *e.g.* the line, but it is displayed in white, so they may need to change the background colour in order to see it.

The distance the tracer travels and the length of steps the tracer takes are specified in the text boxes in the main Stream Tracer panel. The process of achieving desired tracer lines is largely one of trial and error in which the tracer lines obviously appear smoother as the step length is reduced but with the penalty of a longer calculation time.

Once the tracer lines have been created, the `Tubes` filter can be applied to the *Tracer* module to produce high quality images. The tubes follow each tracer line and are not strictly cylindrical but have a fixed number of sides and given radius. When the number of sides is set above, say, 10, the tubes do however appear cylindrical, but again this adds a computational cost.

6.1.9 Image output

The simplest way to output an image to file from ParaView is to select `Save Screenshot` from the `File` menu. On selection, a window appears in which the user can select the resolution for the image to save. There is a button that, when clicked, locks the aspect ratio, so if the user changes the resolution in one direction, the resolution is adjusted in the other direction automatically. After selecting the pixel resolution, the image can be saved. To achieve high quality output, the user might try setting the pixel resolution to 1000 or more in the x-direction so that when the image is scaled to a typical size of a figure in an A4 or US letter document, perhaps in a PDF document, the resolution is sharp.

6.1.10 Animation output

To create an animation, the user should first select `Save Animation` from the `File` menu. A dialogue window appears in which the user can specify a number of things including the image resolution. The user should specify the resolution as required. The other noteworthy setting is number of frames per timestep. While this would intuitively be set to 1, it can be set to a larger number in order to introduce more frames into the animation artificially.

The list represents the underlying post-processing functionality. Almost all the functionality is packaged into a set of configured tools that are conveniently integrated within the post-processing CLI. Those tools are located in *$FOAM_ETC/caseDicts/postProcessing* and are listed by running postProcess with the `-list` option.

```
postProcess -list
```

This produces a list of tools that are described in the following sections.

6.2.1.1 Field calculation

CourantNo Calculates the Courant Number field from the flux field.

Lambda2 Calculates and writes the second largest eigenvalue of the sum of the square of the symmetrical and anti-symmetrical parts of the velocity gradient tensor.

MachNo Calculates the Mach Number field from the velocity field.

PecletNo Calculates the Peclet Number field from the flux field.

Q Calculates the second invariant of the velocity gradient tensor.

R Calculates the Reynolds stress tensor field and stores it on the database.

XiReactionRate Writes the turbulent flame-speed and reaction-rate `volScalarFields` for the Xi-based combustion models.

add Add a list of fields.

components Writes the component scalar fields (*e.g.* Ux, Uy, Uz) of a field (*e.g.* U).

ddt Calculates the Eulerian time derivative of a field.

div Calculates the divergence of a field.

enstrophy Calculates the enstrophy of the velocity field.

flowType Calculates and writes the `flowType` of velocity field where: -1 = rotational flow; 0 = simple shear flow; +1 = planar extensional flow.

grad Calculates the gradient of a field.

mag Calculates the magnitude of a field.

magSqr Calculates the magnitude-squared of a field.

randomise Adds a random component to a field, with a specified perturbation magnitude.

scale Multiplies a field by a scale factor

streamFunction Writes the steam-function `pointScalarField`, calculated from the specified flux `surfaceScalarField`.

subtract From the first field, subtracts the remaining fields in the list.

The function can be included as normal from the by adding the `#includeFunc` directive to `functions` in the the *controlDict* file. Alternatively, the user could test running the function using the solver post-processing by the following command.

```
simpleFoam -postProcess -func surfaces
```

This produces VTK format files of the cutting plane with pressure and velocity data in time directories in the the *postProcessing/surfaces* directory. The user can display the cutting plane by opening ParaView (type `paraview`), then doing `File->Open` and selecting one of the files, *e.g.* *postProcessing/surfaces/296/U_zNormal.vtk* as shown in Figure 6.7.

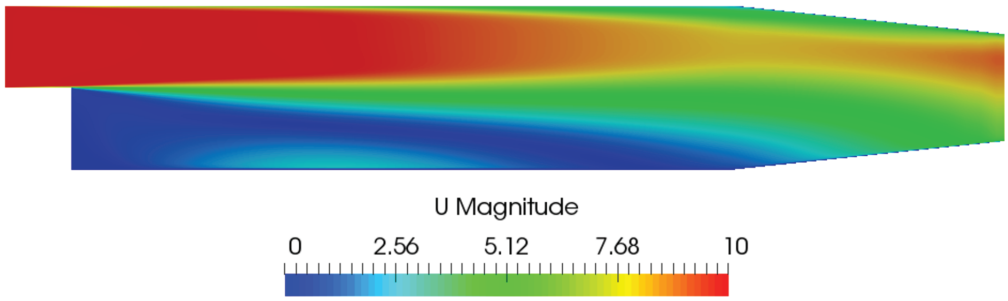

Figure 6.7: Cutting plane with velocity

6.3.4 Live monitoring of data

Functions like probes produce a singe file of time-value data, suitable for graph plotting. When the function is executed during a simulation, the user may wish to monitor the data live on screen. The foamMonitor script enables this; to discover its functionality, the user run it with the `-help` option. The help option includes an example of monitoring residuals that we can demonstrate in this section.

Firstly, include the residuals function in the *controlDict* file.

```
functions
{
#includeFunc  residuals
... other function objects here ...
}
```

The default fields whose residuals are captured are *p* and *U*. Should the user wish to configure other fields, they should make copy the *residuals* file in their *system* and edit the `fields` entry accordingly. All functions files are within the *$FOAM_ETC/caseDicts* directory. The *residuals* file can be located using foamInfo:

```
foamInfo residuals
```

It can then be copied into the *system* directory conveniently using foamGet:

```
foamGet residuals
```

SpalartAllmarasIDDES SpalartAllmaras IDDES turbulence model for compressible flows

WALE The Wall-adapting local eddy-viscosity (WALE) SGS model.

dynamicKEqn Dynamic one equation eddy-viscosity model

dynamicLagrangian Dynamic SGS model with Lagrangian averaging

kEqn One equation eddy-viscosity model

kOmegaSSTDES Implementation of the k-omega-SST-DES turbulence model for compressible flows.

7.2.3 Model coefficients

The coefficients for the RAS turbulence models are given default values in their respective source code. If the user wishes to override these default values, then they can do so by adding a sub-dictionary entry to the `RAS` sub-dictionary file, whose keyword name is that of the model with `Coeffs` appended, *e.g.* `kEpsilonCoeffs` for the `kEpsilon` model. If the `printCoeffs` switch is `on` in the `RAS` sub-dictionary, an example of the relevant `...Coeffs` dictionary is printed to standard output when the model is created at the beginning of a run. The user can simply copy this into the `RAS` sub-dictionary file and edit the entries as required.

7.2.4 Wall functions

A range of wall function models is available in OpenFOAM that are applied as boundary conditions on individual patches. This enables different wall function models to be applied to different wall regions. The choice of wall function model is specified through the turbulent viscosity field ν_t in the *0/nut* file. For example, a *0/nut* file:

```
17
18  dimensions      [0 2 -1 0 0 0 0];
19
20  internalField   uniform 0;
21
22  boundaryField
23  {
24      movingWall
25      {
26          type            nutkWallFunction;
27          value           uniform 0;
28      }
29      fixedWalls
30      {
31          type            nutkWallFunction;
32          value           uniform 0;
33      }
34      frontAndBack
35      {
36          type            empty;
37      }
38  }
39
40
41  // ************************************************************************* //
```

There are a number of wall function models available in the release, *e.g.* `nutWallFunction`, `nutRoughWallFunction`, `nutUSpaldingWallFunction`, `nutkWallFunction` and `nutkAtmWallFunction`. The user can get the fill list of wall function models using foamInfo:

```
foamInfo wallFunctions wallFunction
```

Within each wall function boundary condition the user can over-ride default settings for E, κ and C_μ through optional `E`, `kappa` and `Cmu` keyword entries.

Having selected the particular wall functions on various patches in the *nut*/*mut* file, the user should select epsilonWallFunction on corresponding patches in the *epsilon* field and kqRwallFunction on corresponding patches in the turbulent fields *k*, *q* and *R*.

7.3 Transport/rheology models

In OpenFOAM, solvers that do not include energy/heat, include a library of models for viscosity ν. The models typically relate viscosity to strain rate $\dot{\gamma}$ and are specified by the user in the *transportProperties* dictionary. The available models are listed in the following sections.

7.3.1 Newtonian model

The Newtonian model assumes ν is constant. Viscosity is specified by a dimensionedScalar `nu` in *transportProperties*, *e.g.*

```
transportModel Newtonian;

nu              [ 0 2 -1 0 0 0 0 ] 1.5e-05;
```

Note the units for kinematic viscosity are L^2/T.

7.3.2 Bird-Carreau model

The Bird-Carreau model is:

$$\nu = \nu_\infty + (\nu_0 - \nu_\infty)\left[1 + (k\dot{\gamma})^a\right]^{(n-1)/a} \tag{7.16}$$

where the coefficient a has a default value of 2. An example specification of the model in *transportProperties* is:

```
transportModel BirdCarreau;
BirdCarreauCoeffs
{
    nu0      [ 0 2 -1 0 0 0 0 ] 1e-03;
    nuInf    [ 0 2 -1 0 0 0 0 ] 1e-05;
    k        [ 0 0  1 0 0 0 0 ] 1;
    n        [ 0 0  0 0 0 0 0 ] 0.5;
}
```