Tutorial 6: Modifying OpenFOAM Solvers

Learning Outcomes:

Compilation of OpenFOAM codes; heat transfer and buoyancy; Boussinesq modelling.

VI.1 Compiling in OpenFOAM

One common task with OpenFOAM is modifying and recompiling code. In OpenFOAM, each individual program is stored in its own separate directory, which can contain .C (code) and .H (header) files which are compiled and joined together to create an executable program. OpenFOAM provides a very convenient mechanism for compiling its programs; just go to the specific code directory and type

```
wmake
```

and the program will be compiled. Then type

```
rehash
```

(this just informs the computer than something has been updated); then that particular code will be accessible to run in the usual way.

`wmake` is quite a powerful compilation tool. Within the program directory is a subdirectory `Make` which stores information to control the compilation process. Specifically, inside this subdirectory are two files; `files` and `options`. `options` specifies various flags to the compiler (for example where to find additional libraries) and need not concern us at this stage. `files` specifies which files need to be compiled, and importantly, where the resulting executable is to be placed. If you look in the `files` file for `icoFoam` you will find the line

```
EXE = $(FOAM_APPBIN)/icoFoam
```

which specifies that the executable is to be called `icoFoam` and that it is to go in a particular directory (specified by the environment variable `FOAM_APPBIN`). Very often we want to copy programs from the base installation and modify them, and this line will need to be changed in two ways :

1. The directory specified will be `FOAM_APPBIN` – this would need to be changed to `FOAM_USER_APPBIN` as we do not have the correct privilege to write to the main installation (nor is it a good idea to be overwriting the main installation even if it is possible).

2. The name of the executable will probably need to be changed, to avoid confusion. For example we would probably want to have a program called `myIcoFoam` to avoid confusion with `icoFoam` itself! We could rename the files as well, but this is seldom necessary.

VI.2 Buoyancy and heat transfer

This tutorial models buoyancy effects for a stream of hot air. To do this we must modify `icoFoam` to take account of heat transfer and buoyancy. Heat transfer involves solving the standard heat conduction equation;

$$\frac{\partial \theta}{\partial t} + \nabla.(\underline{u}\theta) = \frac{\kappa}{\rho_0 C_V}\nabla^2\theta$$

Changes of temperature create changes of density in the fluid and this generates different gravitational forces, leading to convective motion. A full solution would require solving for a compressible flow, and would need to be done very accurately because the buoyancy effects are driven by only very small variations in density. However, under certain circumstances the Boussinesq approximation can be used.[1] In this, we can neglect density differences in the fluid and treat it as an incompressible fluid, but with a body force proportional to the temperature

$$\frac{\partial \underline{u}}{\partial t} + \nabla.\underline{u}\,\underline{u} = -\nabla p + \nu\nabla^2\underline{u} - \beta\underline{g}(\theta_0 - \theta)$$

where β is the coefficient of thermal expansion of the fluid, $\underline{g}$ the gravitational force vector and θ_0 a reference temperature.

A copy of `icoFoam` is contained in the tutorial file, renamed as `boussinesqFoam`, and we will modify it to introduce these additional effects. We need to read in the various coefficients; $\kappa, \rho_0, C_V, \theta_0$ and β. The `transportProperties` dictionary is already opened in the file `createFields.H`, so it makes sense to read the additional properties from here. Open `createFields.H` in emacs and add lines

```
dimensionedScalar kappa
(
      transportProperties.lookup("kappa")
);
```

and similar lines for `rho0, Cv, theta0` and `beta`. It is also worth introducing a variable `hCoeff` :

```
dimensionedScalar hCoeff = kappa/(rho0*Cv);
```

[1] This is the Boussinesq approximation in buoyancy, which is different from the Boussinesq approximation for turbulence modelling.

We need to introduce the gravitational accelleration $\underline{g}$; this can be read in from the same dictionary, but of course is a `dimensionedVector` rather than a `dimensionedScalar`.

`createFields.H` also creates the dependent variable fields; we need to create a temperature field `theta` as a `volScalarField` and read it in. This is very similar to the pressure field, so make a copy of the lines starting

```
Info<< "Reading field p" << endl;
volScalarField p
⋮
```

and change them to create a field `theta` instead.

In the file `icoFoam.C` we need to modify the momentum equation to include the extra term, and to create and solve the temperature equation. The momentum equation is `UEqn` ; add the additional term

$$-\beta \underline{g}(\theta_0 - \theta)$$

to this. Add the line

```
        + beta*g*(theta0-theta)
```

At the end of the PISO loop we need to create and solve the temperature equation :

```
fvScalarMatrix tempEqn
(
     fvm::ddt(theta)
   + fvm::div(phi,theta)
   - fvm::laplacian(hCoeff,theta)
);

tempEqn.solve();
```

Having done this, type `wmake` to compile the code.

Also in `tutorial6` is a case, `bendHeat`, which consists of a duct with cooling water flowing along it. Run `blockMesh` to generate the mesh, and take a look at the details of the `blockMeshDict`; this illustrates how to create curved edges in `blockMesh`.

We need to modify this case to function with `boussinesqFoam`. This requires the following ;

1. Create a `theta` file in the `0` timestep directory. This is best done by creating a copy of `U` and editing it. The inlet conditions for the temperature are $\theta = 300$ K, and the wall temperatures are $\theta = 350$K. Don't forget to change the dimensions of `theta` as well.

2. Introduce the physical parameters. `boussinesqFoam` looks for the thermophysical constants in `transportProperties`; check that these are in there and that the values are correct.

3. The differencing schemes need to be specified for the `theta` equation. These are in `fvSchemes`; check that they are appropriate.

4. Finally, `solvers` in `fvSolution` needs an entry for the `theta` equation. Again, this has been provided, but you should check that it is correct.

Then run `boussinesqFoam` on the case, and plot the results for two of the resulting timesteps. Note that if there are errors in the input steps (1-3 above) the code will not run; but the resulting error messages are quite informative.

[Q.VI.1] Evaluate the Reynolds, Prandtl and Nusselt numbers for this case.
[Q.VI.2] Produce plots of temperature and velocity for two later timesteps. Comment on the results.
[Q.VI.3] Plot the temperature profile across the outlet for these two timesteps. Estimate the outlet temperature (you might like to average over several timesteps for this).
[Q.VI.4] Rewrite `boussinesqFoam` to include an averaged temperature field `thetaAv`; for each timestep update this using the current temperature field using the running average formula :

$$\theta_{av}^{n+1} = \frac{n}{n+1}\theta_{av}^{n} + \frac{1}{n+1}\theta^{n+1}$$

Plot this and the temperature profile at the outlet.