buoyantPimpleFoam and buoyantSimpleFoam in OpenFOAM

In this blog post, I will try to give a description of the governing equations of the following solvers in OpenFOAM for **buoyant**, turbulent flow of compressible fluids:

- ***buoyantPimpleFoam*** (**Transient** solver)
- ***buoyantSimpleFoam*** (**Steady-state** solver)

The solvers in OpenFOAM are named properly and comprehensibly so that the users can guess the meanings from their names ☺

Mass Conservation

The mass conservation (continuity) equation of ***buoyantPimpleFoam*** is given by the following equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{1}$$

where $\boldsymbol{u}$ is the velocity field and ρ is the density field. For the steady-state solver, the time derivative term is omitted.

Momentum Conservation

The momentum conservation equation of ***buoyantPimpleFoam*** is given by the following equation:

$$\frac{\partial(\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) = -\nabla p + \rho \boldsymbol{g} + \nabla \cdot (2\mu_{eff} D(\boldsymbol{u})) - \nabla \left(\frac{2}{3} \mu_{eff} (\nabla \cdot \boldsymbol{u}) \right), \quad (2)$$

where p is the static pressure field and $\boldsymbol{g}$ is the gravitational acceleration. The effective viscosity μ_{eff} is the sum of the molecular and turbulent viscosity and the rate of strain (deformation) tensor $D(\boldsymbol{u})$ is defined as $D(\boldsymbol{u}) = \frac{1}{2}\left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right)$. As is the case in the continuity equation, the time derivative term is omitted in ***buoyantSimpleFoam***.

In terms of the implementation in OpenFOAM, the pressure gradient and gravity force terms are rearranged in the following form:

$$-\nabla p + \rho \boldsymbol{g} = -\nabla (p_{rgh} + \rho \boldsymbol{g} \cdot \boldsymbol{r}) + \rho \boldsymbol{g} \quad (3a)$$
$$= -\nabla p_{rgh} - (\boldsymbol{g} \cdot \boldsymbol{r}) \nabla \rho - \rho \boldsymbol{g} + \rho \boldsymbol{g} \quad (3b)$$
$$= -\nabla p_{rgh} - (\boldsymbol{g} \cdot \boldsymbol{r}) \nabla \rho, \quad (3c)$$

where $p_{rgh} = p - \rho \boldsymbol{g} \cdot \boldsymbol{r}$ and $\boldsymbol{r}$ is the position vector.

buoyantPimpleFoam/UEqn.H C++

```
// Solve the Momentum equation

MRF.correctBoundaryVelocity(U);

fvVectorMatrix UEqn
(
    fvm::ddt(rho, U) + fvm::div(phi, U)
  + MRF.DDt(rho, U)
  + turbulence->divDevRhoReff(U)
 ==
    fvOptions(rho, U)
);

UEqn.relax();

fvOptions.constrain(UEqn);

if (pimple.momentumPredictor())
{
    solve
    (
        UEqn
     ==
        fvc::reconstruct
        (
            (
              - ghf*fvc::snGrad(rho)
              - fvc::snGrad(p_rgh)
            )*mesh.magSf()
```

```
30              )
31          );
32
33          fvOptions.correct(U);
34          K = 0.5*magSqr(U);
35      }
```

In the above code, ***MRF*** stands for [Multiple Reference Frame](), which is one method for solving the problems including the rotating parts with the static mesh. This option is activated if ***constant/MRFProperties*** file exists. The line 11(***fvOptions(rho, U)***) is to add the user-specific source terms using ***fvOptions*** functionality.

Energy Conservation

We can choose either **internal energy** e or **enthalpy** h as the energy solution variable [3]. This selection is made according to ***energy*** keyword in ***thermophysicalProperties*** file.

```
constant/thermophysicalProperties                                    C++
thermoType
{
    type            heRhoThermo;
    ...
    energy          sensibleEnthalpy or sensibleInternalEnergy;
}
```

For each variable, the energy conservation equations of ***buoyantPimpleFoam*** are given by the following equations:

- Enthalpy (***sensibleEnthalpy***)

$$\frac{\partial(\rho h)}{\partial t} + \nabla\cdot(\rho \boldsymbol{u} h) + \frac{\partial(\rho K)}{\partial t} + \nabla\cdot(\rho \boldsymbol{u} K) - \frac{\partial p}{\partial t} = \nabla\cdot(\alpha_{eff}\nabla h) + \rho \boldsymbol{u}\cdot\boldsymbol{g} \tag{4}$$

- Internal energy (***sensibleInternalEnergy***)

$$\frac{\partial(\rho e)}{\partial t} + \nabla\cdot(\rho \boldsymbol{u} e) + \frac{\partial(\rho K)}{\partial t} + \nabla\cdot(\rho \boldsymbol{u} K) + \nabla\cdot(p\boldsymbol{u}) = \nabla\cdot(\alpha_{eff}\nabla e) + \rho \boldsymbol{u}\cdot\boldsymbol{g} \tag{5}$$

where $K \equiv |\boldsymbol{u}|^2/2$ is kinetic energy per unit mass and the enthalpy per unit mass h is the sum of the internal energy per unit mass e and the kinematic pressure $h \equiv e + p/\rho$. We can

get the following relations from this definition:

$$\frac{\partial(\rho e)}{\partial t} = \frac{\partial(\rho h)}{\partial t} - \frac{\partial p}{\partial t}, \tag{6a}$$

$$\nabla \cdot (\rho \boldsymbol{u} e) = \nabla \cdot (\rho \boldsymbol{u} h) - \nabla \cdot (p \boldsymbol{u}). \tag{6b}$$

The effective thermal diffusivity α_{eff}/ρ is the sum of laminar and turbulent thermal diffusivities:

$$\alpha_{eff} = \frac{\rho \nu_t}{Pr_t} + \frac{\mu}{Pr} = \frac{\rho \nu_t}{Pr_t} + \frac{k}{c_p}, \tag{7}$$

where k is the thermal conductivity, c_p is the specific heat at constant pressure, μ is the dynamic viscosity, ν_t is the turbulent (kinematic) viscosity, Pr is the Prandtl number and Pr_t is the turbulent Prandtl number. The expression of the laminar thermal diffusivity changes depending on the selected thermodynamics package. As is the case in the momentum equation, the time derivative terms are omitted in ***buoyantSimpleFoam***.

buoyantPimpleFoam/EEqn.H C++

```
{
    volScalarField& he = thermo.he();

    fvScalarMatrix EEqn
    (
        fvm::ddt(rho, he) + fvm::div(phi, he)
      + fvc::ddt(rho, K) + fvc::div(phi, K)
      + (
            he.name() == "e"
          ? fvc::div
            (
                fvc::absolute(phi/fvc::interpolate(rho), U),
                p,
                "div(phiv,p)"
            )
          : -dpdt
        )
      - fvm::laplacian(turbulence->alphaEff(), he)
     ==
        rho*(U&g)
      + radiation->Sh(thermo)
      + fvOptions(rho, he)
    );

    EEqn.relax();

    fvOptions.constrain(EEqn);

    EEqn.solve();

    fvOptions.correct(he);

    thermo.correct();
```

```
34      radiation->correct();
35  }
```

The line 21 represents the contributions from the radiative heat transfer. The effective thermal diffusivity α_{eff} (7) is calculated in ***heThermo.C***:

src/thermophysicalModels/basic/heThermo/heThermo.C C++

```
784 template<class BasicThermo, class MixtureType>
785 Foam::tmp<Foam::volScalarField>
786 Foam::heThermo<BasicThermo, MixtureType>::alphaEff
787 (
788     const volScalarField& alphat
789 ) const
790 {
791     tmp<Foam::volScalarField> alphaEff(this->CpByCpv()*(this->alpha_ + alphat));
792     alphaEff.ref().rename("alphaEff");
793     return alphaEff;
794 }
```

When solving for enthalpy h, the pressure-work term dp/dt can be excluded by setting the ***dpdt*** option to ***no*** in ***thermophysicalProperties*** file [3]. The ***volScalarField*** dp/dt is defined and uniformly initialized to zero in ***createFields.H*** and updated in ***pEqn.H*** if this option is activated.

buoyantPimpleFoam/createFields.H C++

```
78  Info<< "Creating field dpdt\n" << endl;
79  volScalarField dpdt
80  (
81      IOobject
82      (
83          "dpdt",
84          runTime.timeName(),
85          mesh
86      ),
87      mesh,
88      dimensionedScalar("dpdt", p.dimensions()/dimTime, 0)
89  );
```

buoyantPimpleFoam/pEqn.H C++

```
69      if (thermo.dpdt())
70      {
71          dpdt = fvc::ddt(p);
72      }
```

When the mesh is static, the ***fvc::absolute*** function does nothing as shown below:

src/finiteVolume/finiteVolume/fvc/fvcMeshPhi.C C++

```
187 Foam::tmp<Foam::surfaceScalarField> Foam::fvc::absolute
188 (
189     const tmp<surfaceScalarField>& tphi,
190     const volVectorField& U
191 )
192 {
193     if (tphi().mesh().moving())
194     {
195         return tphi + fvc::meshPhi(U);
196     }
197     else
198     {
```

```
199         return tmp<surfaceScalarField>(tphi, true);
200     }
201 }
```

References

[1] Energy Equation in OpenFOAM | CFD Direct

[2] OpenFOAM User Guide: 7.1 Thermophysical models | CFD Direct

[3] OpenFOAM 2.2.0: Thermophysical Modelling

More from my site

- Temperature calculation from energy variables in OpenFOAM
- Introduction to laplacianFoam and simple validation calculation
- Conduction, convection and radiation in OpenFOAM (Under construction)
- Solvers for heat transfer problems in OpenFOAM – buoyantBoussinesqPimpleFoam
- Boundary Layer Mesh Calculator
- Reynolds-Averaged Navier-Stokes (RANS)

Author: fumiya

CFD engineer in Japan View all posts by fumiya

fumiya / May 6, 2016 / Solvers, OpenFOAM / Heat Transfer, buoyantPimpleFoam, buoyantSimpleFoam

3 thoughts on “buoyantPimpleFoam and buoyantSimpleFoam in OpenFOAM”

varsey

May 7, 2016 at 6:17 PM

It’s nice blog you have here. Can I subscribe to new post via email somehow here?

fumiya

May 8, 2016 at 12:04 AM

Hi varsey,

Thank you for your interest in my blog.
I added the subscribe button in the upper right-hand part of this page.
I will be glad if you be the first subscriber!

Joao

August 1, 2017 at 12:39 AM

wow! thank you for this great explanation =)