A low-Mach number solver for variable density flows

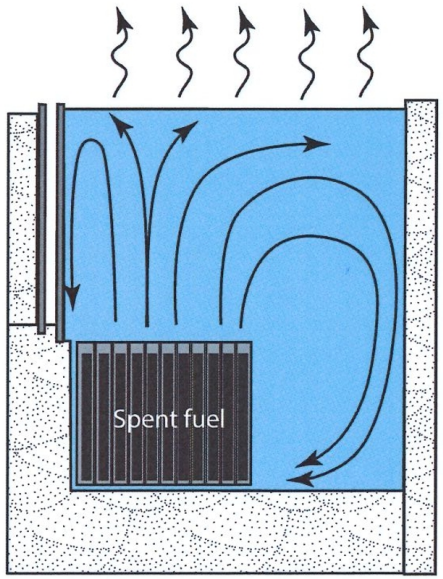
Will HAY

Thermodynamics and Fluid Mechanics,
Université catholique de Louvain
Louvain-la-Neuve, Belgium

2018-11-28

What is a low-Mach number variable density flow?

- Small pressure variations and large temperature (density) variations
- Appear in atmospheric, oceanographic and combustion flows



Shepherd (2016) [1]

[1] Fukushima Nuclear Accident (2016), DOI : 10.17226/21874

Governing equations before low-Mach number approximation

- Continuity

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

- Momentum

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u}\mathbf{u}) - \boldsymbol{\nabla} \cdot \left(\mu_{eff} \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) \right) = -\nabla p + \rho \mathbf{g} \tag{2}$$

- Energy

$$\frac{\partial (\rho h)}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u} h) = \boldsymbol{\nabla} \cdot \left(\frac{k}{C_p} \nabla h \right) + \frac{\mathrm{d}p}{\mathrm{d}t} \tag{3}$$

- Equation of state

$$p = \rho R T = \frac{\rho T}{\gamma M^2} \tag{4}$$

Low-Mach number approximation

- The low-Mach number equations are found by expanding the flow variables; ρ, **u**, T and p, as a power series around $\epsilon = \gamma M^2$.

$$\rho = \rho^0 + \epsilon\rho^1$$

$$\mathbf{u} = \mathbf{u}^0 + \epsilon\mathbf{u}^1$$

$$T = T^0 + \epsilon T^1$$

$$h = h^0 + \epsilon h^1$$

$$p = \frac{\rho T}{\gamma M^2} = \frac{1}{\epsilon}p^0 + p^1$$

- The thermodynamic pressure, p^0 and the dynamic pressure, p^1 hereon referred to as p_d are now decoupled

Governing equations after low-Mach number approximation

- Continuity

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u}) = 0 \tag{5}$$

- Momentum

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u}\mathbf{u}) - \boldsymbol{\nabla} \cdot \left(\mu_{eff} \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) \right) = \underbrace{-\nabla p'_d}_{\text{dynamic}} + \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{6}$$

- Energy

$$\frac{\partial (\rho h)}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u} h) = \boldsymbol{\nabla} \cdot (\alpha_{eff} \nabla h) + \underbrace{\frac{\partial p^0}{\partial t}}_{\text{thermodynamic}} \tag{7}$$

- Equation of state

$$\rho = \frac{p^0}{RT} \tag{8}$$

Why develop a low-Mach number solver?

- Existing solvers not sufficient
  - *buoyantBoussinesqPimpleFoam* valid only for small ΔT
  - *buoyantPimpleFoam* pressures are not decoupled and EEqn.H incorrect
- Useful for OpenFOAM community
  - **Solver:** similar solver (reactingLMFoam) on old version of OpenFOAM
  - **Tutorial:** reference required for the variable density PISO algorithm

What base solver can we use?

- the *buoyantPimplefoam* similar but must be updated to have
  - ∇p_d in PISO (i.e.uniquely dynamic pressure)
  - $\frac{\partial \rho}{\partial t}$ source term in PISO
  - $\frac{\mathrm{dp^0}}{\mathrm{dt}}$ term in EEqn.H (i.e. uniquely thermodynamic pressure)
- If open to atmosphere then $p^0 = p_{atm} =$ constant (in space *and* time)
- If closed domain p^0 can vary in time (see Lessani)

$$p^0 = \frac{M_0}{\int \frac{1}{RT} \mathrm{dV}} = \frac{M_0}{\int \psi \mathrm{dV}} \tag{9}$$

Variable Density Pressure Poisson Equation

- A semi-discretized version of the momentum equation

$$a_p U_p + \sum_N a_N U_N - \frac{U^0}{\Delta t} = -\nabla p'_d - \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{10}$$

- Get a relation for the velocity at point P

$$U_p = \frac{1}{a_p} H(U) - \frac{1}{a_p} \nabla p'_d - \frac{1}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{11}$$

- Multiplying (11) by ρ gives first term of continuity equation (5)

$$\rho U_p = \frac{\rho}{a_p} H(U) - \frac{\rho}{a_p} \nabla p'_d - \frac{\rho}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{12}$$

Low-Mach number approximation

- Taking the divergence of Equation (12)

$$\boldsymbol{\nabla} \cdot (\rho U_p) = \boldsymbol{\nabla} \cdot \left(\frac{\rho}{a_p} H(U) - \frac{\rho}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho \right) - \boldsymbol{\nabla} \cdot \left(\frac{\rho}{a_p} \nabla p'_d \right) \tag{13}$$

- Combining (13) and (5) gives the pressure Poisson equation

$$\boldsymbol{\nabla} \cdot \left(\frac{\rho}{a_p} \nabla p'_d \right) = \boldsymbol{\nabla} \cdot \left(\frac{\rho}{a_p} H(U) - \frac{\rho}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho \right) + \frac{\partial \rho}{\partial t} \tag{14}$$

$$\underbrace{\sum_f \left(\frac{\rho}{a_p} \right)_f \left(\nabla p'_d \right)_f \cdot \mathbf{S}_f}_{fvm::laplacian(rhorAUf,pd)} = \underbrace{\sum_f \left(\left(\frac{\rho}{a_p} \right)_f H(U)_f - \frac{\rho}{a_p} \left(\mathbf{g} \cdot \mathbf{h} \right)_f \left(\nabla \rho \right)_f \right) \cdot \mathbf{S}_f}_{fvc::div(phiHbyA)} + \underbrace{\frac{\partial \psi}{\partial t} p^0 \cdot V}_{fvc::ddt(psi)*p} \tag{15}$$

Transient Variable Density PISO Algorithm

Starting with known ρ, T, **u**, p and p_d at time n the full algorithm is :

1. Using ρ^n and p_d^n solve for $\mathbf{u}^*$ (momentum predictor step)
2. Using ρ^n and $\mathbf{u}^*$ solve for h^* from energy equation (get T^*)
3. Using T^* solve for ψ^* from equation of state
4. Using integral of ψ^* find p^{0*} from (9)
5. Using p^{0*} and T^* find ρ^* from equation of state
6. Using $\mathbf{u}^*$ and ρ^* formulate the $H(U)^*$ operator
7. Using $H(U)^*$ and ρ^* formulate p_d Poisson equation (15)
8. Solve for p_d^* (pressure solution step)
9. Solve for $\mathbf{u}^{**}$ from (11) (explicit velocity correction)

Copying and Renaming

```
1  foam
2  cp -r --parents applications/solvers/heatTransfer/
       buoyantPimpleFoam/ $WM_PROJECT_USER_DIR
3  ufoam
4  cd applications/solvers/heatTransfer/
5  mv buoyantPimpleFoam/ lowMachBuoyantPimpleFoam
6  cd lowMachBuoyantPimpleFoam/
7  mv buoyantPimpleFoam.C lowMachBuoyantPimpleFoam.C
8  sed -i s/buoyantPimpleFoam/lowMachBuoyantPimpleFoam/g Make/files
9  wclean
10 wmake
```

Create readRayleighBenardNusselt.H

```
Info<< "Reading flowProperties\n" << endl;
IOdictionary flowProperties
(
    IOobject
    (
        "flowProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);
dimensionedScalar TRef
(
    "TRef",
    dimTemperature,
    flowProperties
);
dimensionedScalar deltaTRef
(
    "deltaTRef",
    dimTemperature,
    flowProperties
);
```

Create readRayleighBenardNusselt.H

```
dimensionedScalar rhoRef
(
    "rhoRef",
    dimDensity,
    flowProperties
);
dimensionedScalar URef
(
    "URef",
    dimVelocity,
    flowProperties
);
dimensionedScalar RaRef
(
    "RaRef",
    dimless,
    flowProperties
);
dimensionedScalar PrRef
(
    "PrRef",
    dimless,
    flowProperties
);
```

Create readRayleighBenardNusselt.H

```
Info<< "Creating field nonDimT\n" << endl;
volScalarField nonDimT
(
    IOobject
    (
        "nonDimT",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
Info<< "Creating field nonDimRho\n" << endl;
volScalarField nonDimRho
(
    IOobject
    (
        "nonDimRho",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

Create readRayleighBenardNusselt.H

```
Info<< "\nReading field nonDimU\n" << endl;
volVectorField nonDimU
(
    IOobject
    (
        "nonDimU",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
Info<< "Creating field nusseltOne\n" << endl;
volScalarField nusseltOne
(
    IOobject
    (
        "nusseltOne",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

Create readRayleighBenardNusselt.H

```
Info<< "Creating field nusseltTwo\n" << endl;
volScalarField nusseltTwo
(
    IOobject
    (
        "nusseltTwo",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
Info<< "Creating field nusseltTotal\n" << endl;
volScalarField nusseltTotal
(
    IOobject
    (
        "nusseltTotal",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

Create calculateRayleighBenardNusselt.H

```
nonDimU = (U/URef);
nonDimRho = (rho/rhoRef);
nonDimT = (T-TRef)/(deltaTRef);
volVectorField gradNonDimT(fvc::grad(nonDimT));
volScalarField gradNonDimTy
(
    IOobject
    (
        "gradNonDimTy",
        runTime.timeName(),
        mesh
    ),
    gradNonDimT.component(vector::Y)
);
volScalarField nonDimUy
(
    IOobject
    (
        "nonDimUy",
        runTime.timeName(),
        mesh
    ),
    nonDimU.component(vector::Y)
);
nusseltOne = sqrt(RaRef*PrRef)*nonDimUy*nonDimRho*nonDimT;
nusseltTwo = hRef*gradNonDimTy; // hRef is to nonDimensionalize
nusseltTotal = nusseltOne - nusseltTwo;
```

$$Nu = \underbrace{\sqrt{RaPr}\,\langle \rho v T \rangle_t}_{component\,1} - \underbrace{\frac{\partial \langle T \rangle_t}{\partial y}}_{component\,2}$$

Modify createFields.H

```
Info<< "Reading thermophysical properties\n" << endl;
autoPtr<rhoThermo> pThermo(rhoThermo::New(mesh));
rhoThermo& thermo = pThermo();
thermo.validate(args.executable(), "h");
volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    thermo.rho()
);
dimensionedScalar totalMass0 = fvc::domainIntegrate(rho);
volScalarField psi
(
    IOobject
    (
        "psi",
        runTime.timeName(),
        mesh
    ),
    thermo.psi()
);
psi.oldTime(); // need to store for ddt term
```

Modify createFields.H

```
volScalarField& p = thermo.p();
p = totalMass0/fvc::domainIntegrate(psi);
Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
#include "compressibleCreatePhi.H"
Info<< "Creating field pd\n" << endl;
volScalarField pd
(
    IOobject
    (
        "pd",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
```

Modify createFields.H

```
Info << "Creating turbulence model.\n" << nl;
autoPtr<compressible::turbulenceModel> turbulence
(
    compressible::turbulenceModel::New
    (
        rho,
        U,
        phi,
        thermo
    )
);
#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"
Info<< "Creating field dpdt\n" << endl;
volScalarField dpdt
(
    IOobject
    (
        "dpdt",
        runTime.timeName(),
        mesh
    ),
    mesh,
    dimensionedScalar("dpdt", p.dimensions()/dimTime, 0)
);
mesh.setFluxRequired(pd.name());
#include "createFvOptions.H"
```

Modify temperature correction in PISO

```
    // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
        #include "UEqn.H"
        #include "EEqn.H"

        // --- Pressure corrector loop
        while (pimple.correct())
        {
            #include "pEqn.H"
        }

        if (pimple.turbCorr())
        {
            turbulence->correct();
        }
    }
    #include "EEqn.H" // temp correction
    psi = thermo.psi();
    p = totalMass0/fvc::domainIntegrate(psi);
    rho = thermo.rho();

    runTime.write();

    runTime.printExecutionTime(Info);
}
Info<< "End\n" << endl;
```

Modify UEqn.H

The pressure gradient term is the only change required in the UEqn.H file.

```
fvVectorMatrix UEqn
(
    fvm::ddt(rho, U)
  + fvm::div(phi, 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(pd)
            )*mesh.magSf()
        )
    );
```

Modify EEqn.H

```
{
    volScalarField& h = thermo.he();
    fvScalarMatrix EEqn
    (
        fvm::ddt(rho, h) + fvm::div(phi, h)
      - dpdt
      - fvm::laplacian(turbulence->alphaEff(), h)
     ==
      fvOptions(rho, h)
    );

    EEqn.relax();
    fvOptions.constrain(EEqn);
    EEqn.solve();
    fvOptions.correct(h);
    thermo.correct();
    Info<< "min/max(T) = "
        << min(T).value() << ", " << max(T).value() << endl;
}
```

Modify pEqn.H

```
psi = thermo.psi();
p = totalMass0/fvc::domainIntegrate(psi);
rho = thermo.rho();
dimensionedScalar totalMass = fvc::domainIntegrate(rho);
scalar constantMass = (totalMass0/totalMass).value();
Info<< "Should be constant in time " << constantMass << endl;

volScalarField rAU(1.0/UEqn.A());
surfaceScalarField rhorAUf("rhorAUf", fvc::interpolate(rho*rAU));
volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
surfaceScalarField phig("phig", -rhorAUf*ghf*fvc::snGrad(rho)*mesh.magSf());
surfaceScalarField phiHbyA
(
    "phiHbyA",
    (
        fvc::flux(rho*HbyA)
    )
    + phig
);

constrainPressure(pd, rho, U, phiHbyA, rhorAUf);
```

Modify pEqn.H

```
while (pimple.correctNonOrthogonal())
{
    fvScalarMatrix pdEqn
    (
        fvm::ddt(psi, pd) // implicit addition of pd
      - fvc::ddt(psi, pd) // explicit subtraction of pd
      + fvc::ddt(psi)*p
      + fvc::div(phiHbyA)
      - fvm::laplacian(rhorAUf, pd)
     ==
        fvOptions(psi, pd, rho.name())
    );
    pdEqn.solve(mesh.solver(pd.select(pimple.finalInnerIter())));
    if (pimple.finalNonOrthogonalIter())
    {
        phi = phiHbyA + pdEqn.flux();
        U = HbyA + rAU*fvc::reconstruct((pdEqn.flux() + phig)/rhorAUf);
        U.correctBoundaryConditions();
        fvOptions.correct(U);
    }
}
#include "rhoEqn.H"
#include "compressibleContinuityErrs.H"
if (thermo.dpdt())
{
    dpdt = fvc::ddt(p);
}
#include "calculateRayleighBenardNusselt.H"
```

Compiling the Solver

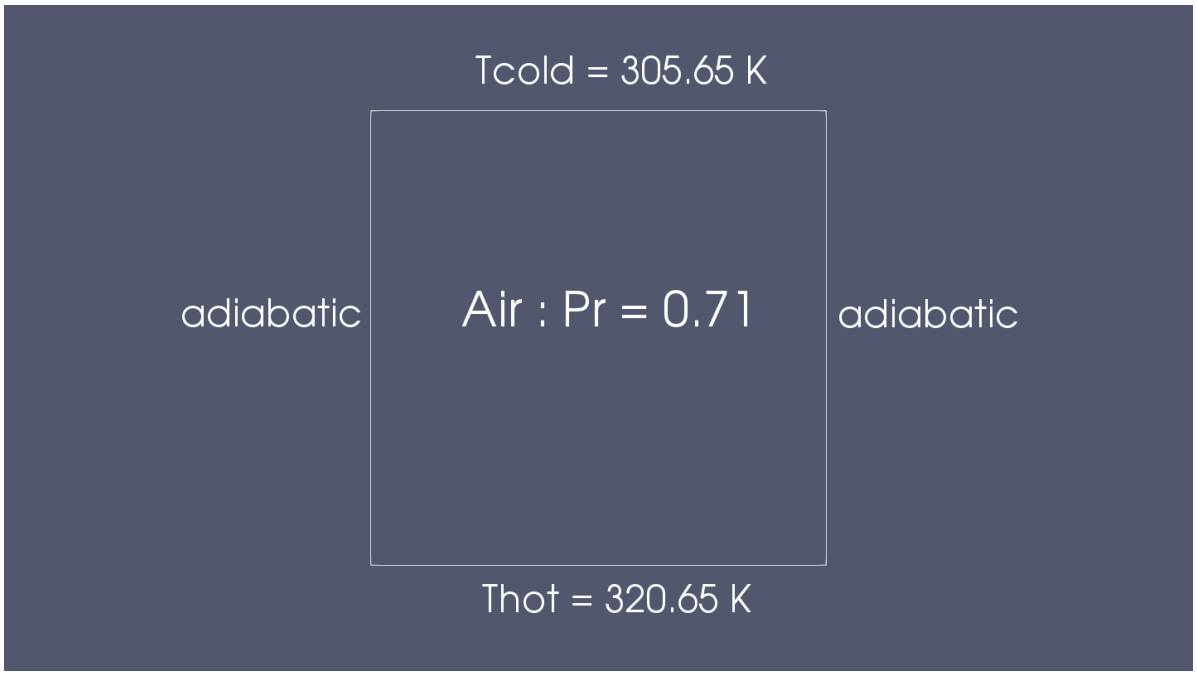
No extra libraries are required and the code can be compiled

```
1 ufoam
2 cd applications/solvers/heatTransfer/lowMachBuoyantPimpleFoam
3 wclean
4 wmake
```

Rayleigh-Benard Convection

- Numerical set-up
  - 2D square with adiabatic side-walls
  - Isothermal top and bottom
  - $Pr = 0.71$ and $Ra = 10^5$

$$Ra = \frac{g\beta\Delta T L^3}{\nu\alpha} \qquad Pr = \frac{\nu}{\alpha}$$



Copying and Renaming Tutorial

- The hotRoom tutorial provides a case template

```
1 run
2 cp -r $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/hotRoom ./
      rayleighBenard2DTestCase
3 cd rayleighBenard2DTestCase
4 mv 0.orig 0
5 sed -i s/cleanCase0/cleanCase/g Allclean
6 cp $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/
      thermocoupleTestCase/Allrun .
```

Initial and Boundary Conditions

- Turbulence fields are removed and p_d field is created

```
1 cd 0
2 rm alphat epsilon k nut
3 mv p_rgh pd
4 sed -i s/"internalField uniform 1e5"/"internalField uniform 0"/g
      pd
5 sed -i s/"uniform 1e5"/"\$internalField"/g pd
```

0/T

```
/*--------------------------------*- C++ -*----------------------------------*\

dimensions [0 0 0 1 0 0 0];

internalField uniform 313.15;

boundaryField
{
    floor
    {
        type fixedValue;
        value uniform 320.65;
    }
    ceiling
    {
        type fixedValue;
        value uniform 305.65;
    }
    fixedWalls
    {
        type zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}
```

Create dummy variables for post-processing

```
1  cp 0/U 0/nonDimU
2  cp 0/nonDimU 0/nusseltOne
3  cp 0/nonDimU 0/nusseltTwo
4  cp 0/nonDimU 0/nusseltTotal
5  sed -i s/"uniform (0 0 0)"/"uniform 0"/g 0/nusselt*
6  sed -i s/"noSlip"/"zeroGradient"/g 0/nusselt*
7  sed -i s/volVector/volScalar/g 0/n*
8  sed -i s/volScalar/volVector/g 0/nonDimU
9  cp 0/nusseltOne 0/nonDimRho
10 cp 0/T 0/nonDimT
11 sed -i s/"uniform 320.65"/"uniform 0.5"/g 0/nonDimT
12 sed -i s/"uniform 305.65"/"uniform -0.5"/g 0/nonDimT
13 sed -i s/"uniform 313.15"/"uniform 0"/g 0/nonDimT
14 sed -i s/"dimensions \[0 1 -1 0 0 0 0\]"/"dimensions \[0 0 0 0 0
       0 0\]"/g 0/n*
15 sed -i s/"dimensions \[0 0 0 1 0 0 0\]"/"dimensions \[0 0 0 0 0 0
        0\]"/g 0/nonDimT
```

Thermophysical And Flow Properties

- No turbulence model in a DNS, simulationType is **laminar**
- Not an indication of a laminar flow, but an indication of the absence of subgrid scale modelling.

```
sed -i s/"simulationType RAS"/"simulationType laminar"/g constant
    /turbulenceProperties
```

- The equation of state is perfectGas not incompressiblePerfectGas irrespective of p^0 definition

$$\rho = \frac{p^0}{RT}$$

Create constant/flowProperties

```
/*---------------------------------*- C++ -*----------------------------------*\
// Reference temperature
TRef 313.15;

// Reference temperature
deltaTRef 15;

// Reference density
rhoRef  1.1272;

// Reference freefall velocity (beta*deltaT*g*H)^0.5
URef  0.1441;

// Reference Rayleigh
RaRef  1e5;

// Reference Pr
PrRef  0.712;

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

Create hRef

- hRef file is copied and edited from a fireFoam tutorial

```
1 cp $FOAM_TUTORIALS/combustion/fireFoam/LES/smallPoolFire3D/
      constant/hRef constant/
2 sed -i s/"value 1"/"value 0.0442"/g constant/hRef
```

Modify system/blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*/
scale 0.0221;

vertices
(
    (-1 -1 -0.001)
    (1 -1 -0.001)
    (1 1 -0.001)
    (-1 1 -0.001)
    (-1 -1 0.001)
    (1 -1 0.001)
    (1 1 0.001)
    (-1 1 0.001)
);
blocks
(
hex (0 1 2 3 4 5 6 7) (32 32 1) simpleGrading (1 1 1)
);
boundary
(
    floor
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
```

Modify system/blockMeshDict

```
    ceiling
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

Modify system/fv* files

- Modify fvSchemes: 2nd-order time integration and div schemes

```
1 sed -i s/" default Euler"/" default backward"/g system/fvSchemes
2 sed -i s/" div(phi,U) Gauss upwind"/" div(phi,U) Gauss linear"/g
      system/fvSchemes
3 sed -i s/" div(phi,h) Gauss upwind"/" div(phi,h) Gauss linear"/g
      system/fvSchemes
```

- Modify fvSolution 3 corrector steps in PISO

```
1 sed -i '/ pRefCell 0/d' system/fvSolution
2 sed -i '/ pRefValue 1e5/d' system/fvSolution
3 sed -i s/p_rgh/pd/g system/fvSolution
4 sed -i s/" nCorrectors 2"/" nCorrectors 3"/g system/fvSolution
```

Modify system/controlDict

```
application lowMachBuoyantPimpleFoam;

startFrom latestTime;

startTime 0;

stopAt endTime;

endTime 60;

deltaT 1e-2;

writeControl adjustableRunTime;

writeInterval 5;

purgeWrite 0;

writeFormat ascii;

writePrecision 6;
```

Modify system/controlDict

```
writeCompression off;

timeFormat general;

timePrecision 6;

runTimeModifiable true;

adjustTimeStep yes;

maxCo 0.1;

#include "sampleControls"

functions
{
    fieldAverage1
    {
        type fieldAverage;
        libs ("libfieldFunctionObjects.so");
        timeStart 30;
        timeEnd 60;
        writeControl writeTime;
```

Modify system/controlDict

```
fields
(
    nonDimU
    {
        mean on;
        prime2Mean on; //RMS
        base time;
    }

    nusseltTotal
    {
        mean on;
        prime2Mean off;
        base time;
    }
    nusseltOne
    {
        mean on;
        prime2Mean off;
        base time;
    }
    nusseltTwo
    {
        mean on;
        prime2Mean off;
        base time;
    }
);
```

Modify system/controlDict

```
    volAverageNusseltField
    {
        type volFieldValue;
        libs ("libfieldFunctionObjects.so");
        log true;
        timeStart 30;
        timeEnd 60;
        writeControl writeTime;
        writeFields true;
        regionType all;
        operation volAverage;
        fields
        (
            nusseltOneMean
            nusseltTwoMean
            nusseltTotalMean
        );
    }

// #include "cuttingPlaneNusseltWalls"
    #include "fieldTransfer"
}
```

Create system/fieldTransfer

```
// ************************************************************************* //
// Transcribe volume fields to surfaces.
fieldTransfer
{
    type surfMeshes;
    libs ("libsampling.so");
    log true;
    timeStart 30;
    timeEnd 60;
    writeControl none;
    createOnRead true;
    executeControl timeStep;
    executeInterval 1;
    fields (nusseltTotalMean nusseltOneMean nusseltTwoMean);
    surfaces
    (
        middle
        {
            type plane;
            source cells;
            planeType pointAndNormal;
            pointAndNormalDict
            {
                point (0 0 0);
                normal (0 1 0);
            }
            interpolate true;
        }
```

Create system/fieldTransfer

```
top
{
    type plane;
    source cells;
    planeType pointAndNormal;
    pointAndNormalDict
    {
        point (0 0.0116485 0); //2 X boundary layer
        normal (0 1 0);
    }
    interpolate true;
}
bottom
{
    type plane;
    source cells;
    planeType pointAndNormal;
    pointAndNormalDict
    {
        point (0 -0.0116485 0);
        normal (0 1 0);
    }
    interpolate true;
}
```

Create system/fieldTransfer

```
        veryTop
        {
            type plane;
            source cells;
            planeType pointAndNormal;
            pointAndNormalDict
            {
                point (0 0.0168742 0); // 1 X boundary layer
                normal (0 1 0);
            }
            interpolate true;
        }
        veryBottom
        {
            type plane;
            source cells;
            planeType pointAndNormal;
            pointAndNormalDict
            {
                point (0 -0.0168742 0);
                normal (0 1 0);
            }
            interpolate true;
        }
    );
}
```

Create system/fieldTransfer

```
areaAveragePlaneMiddle
{
    ${__surfaceFieldValue}
    regionType surface;
    name middle;
    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
areaAveragePlaneTop
{
    ${__surfaceFieldValue}
    regionType surface;
    name top;
    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
areaAveragePlaneBottom
{
    ${__surfaceFieldValue}
    regionType surface;
    name bottom;
    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
```

Create system/fieldTransfer

```
areaAveragePlaneVeryTop
{
    ${__surfaceFieldValue}
    regionType surface;
    name veryTop;
    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
areaAveragePlaneVeryBottom
{
    ${__surfaceFieldValue}
    regionType surface;
    name veryBottom;
    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
```

Running the case

- The sampleControl file is copied from another tutorial. It should be edited to have a timeStart and timeEnd

```
cp $FOAM_TUTORIALS/incompressible/simpleFoam/squareBend/system/
    sampleControls system/
```

- Run from inside the case directory the Allrun script can be run.

```
./Allrun
```

Heat transfer in RBC

- System response to the imposed Ra and Pr measured in terms of Nu
- Nu given either by a time and **area** averaged slice taken at slice

$$Nu_y = \underbrace{\sqrt{RaPr}\,\langle \rho v T \rangle_{xzt}}_{component\,1} - \underbrace{\frac{\partial \langle T \rangle_{xzt}}{\partial y}}_{component\,2}$$

- or alternatively by a time and **volume** average over the entire domain

$$Nu = \sqrt{RaPr}\,\langle \rho v T \rangle_{xyzt} + 1$$

- Nu constant but component contribution varies between bulk and BL

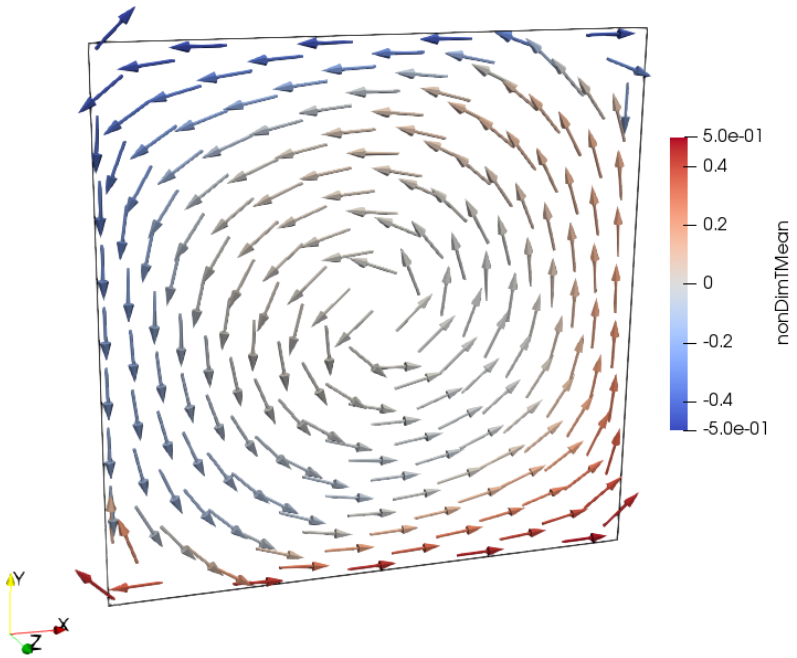
Heat transfer in RBC

Slice	$\mathrm{Nu}_{\mathrm{comp1}}$	$\mathrm{Nu}_{\mathrm{comp2}}$	$\mathrm{Nu}_{\mathrm{tot}}$
Very top	1.87	-2.06	3.93
Top	3.61	-0.23	3.84
Middle	3.85	0.03	3.82
Bottom	3.59	-0.24	3.84
Very Bottom	1.83	-2.09	3.92
Global	2.86	-1	3.86

Table: Nu summary

Large Scale Circulation in RBC

- Time averaged LSC velocity takes up the volume of the container



To do...

- easy
  - switch needed for open and closed domains (definition of p^0)
  - temperature correction step in PISO algorithm
- less easy
  - implement low-Mach variable density PISO into interIsoFoam
  - then implement heat and mass transfer into lowMachInterIsoFoam