051176 - Computational Techniques for Thermochemical Propulsion
Master of Science in Aeronautical Engineering

My First Tutorial in OpenFOAM

Prof. **Federico Piscaglia**
Dept. of Aerospace Science and Technology (DAER)
Politecnico di Milano - Italy

federico.piscaglia@polimi.it

Learning outcome

You will learn ...

- how to run the icoFoam cavity tutorial
- how the cavity tutorial is set up, and how to modify the set-up
- how to search for examples of how to use the utilities.

Slides are based on OpenFOAM-7, released by the OpenFOAM Foundation.

Before starting...

1. Close all terminals and open a new one
2. Go to the 'run' folder: `run`
3. Make a working copy of the tutorial folder: `cp -r $FOAM_TUTORIALS`
4. Change dir: `cd incompressible/icoFoam/cavity/cavity`
5. Generate the mesh: `blockMesh`
6. Run the solver: `icoFoam`
7. Check the results: `paraFoam`

Required OpenFOAM environment

1) Check if OpenFOAM is available in your system. The command
   ```
   which icoFoam
   ```
   will provide you the path of the `icoFoam` executable.

2) Check if the folder `$WM_PROJECT_USER_DIR` exists. If not, create it:
   ```
   mkdir $WM_PROJECT_USER_DIR
   ```

3) Check whether the `$FOAM_RUN` folder exist; type
   ```
   mkdir -p $WM_PROJECT_USER_DIR/run
   ```
   otherwise.

Finding tutorials for the applications

- Use the pre-defined alias `tut` to go to the tutorials directory:

  `$WM_PROJECT_DIR/tutorials`

  where there are complete set-ups of cases for all the solvers.

- **Note**: it is strongly recommended to copy the tutorial you want to test/modify to the `$WM_PROJECT_USER_DIR/run` directory: the official installation must be kept clean!

- There are no specific tutorials for the utilities, but some of the solver tutorials also show how to use the utilities.

The icoFoam cavity tutorial

- We will use the icoFoam cavity tutorial as a general example of how to set up and run a case in OpenFOAM

- Copy the icoFoam cavity tutorial in the folder $FOAM_RUN

```
cp -r $FOAM_TUTORIALS/icoFoam/cavity $FOAM_RUN

cd $FOAM_RUN/cavity
```

Running the case

- The mesh is defined by a dictionary that is read by the `blockMesh` utility. Create the mesh by typing:

```
blockMesh
```

  You have now generated the mesh in OpenFOAM format.

- Check the mesh quality by

```
checkMesh
```

  You can check the mesh size, its quality and the geometrical features.

- The `Cavity`-case is set to run the `icoFoam` solver. Run the simulation in background using the settings in the case, and forward the output to the `log` file:

```
icoFoam > log.icoFoam 2>&1 &
```

Post-processing

- To visualize/post-process the results, type:

```
paraFoam
```

  - Click on “Accept”.
  - Go to the final time step
  - Choose which variable to color by with `Display` → `Color by`
  - Move, rotate and scale the visualization using the mouse

- Find more instructions on the use of paraFoam in the UserGuide:

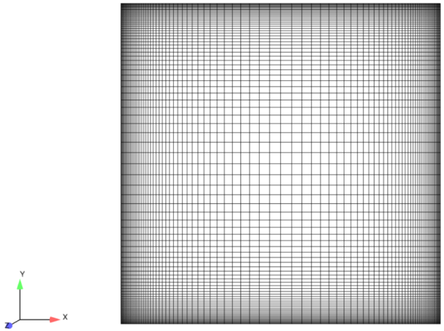
```
$WM_PROJECT_DIR/doc/Guides-a4/UserGuide.pdf
```

- Exit paraFoam: `File` → `Exit`

- Results may also be viewed using other thirdParty products (e.g. Visit)

Visualization of the mesh

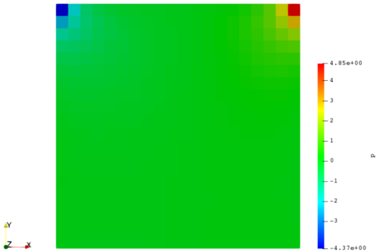
- `Representation` → Wireframe
- `Color by` → Solid color

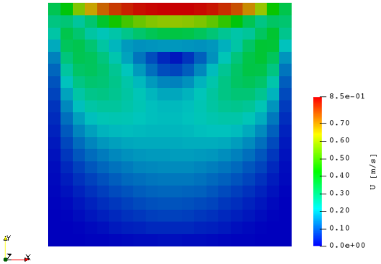


Visualization of the global fields

- Representation → Surface
- Color by → U
- Show scalar bar

- Representation → Surface
- Color by → U
- Show scalar bar





$FOAM_CASE: **organization**

We will look at the case files of the `cavity` tutorial.

a) First of all it should be noted that `icoFoam` is a Transient solver for incompressible, laminar flow of Newtonian fluids

b) The case directory originally contains the following sub-directories: `0`, `constant`, and `system`. After our run it also contains the output `0.1`, `0.2`, `0.3`, `0.4`, `0.5`, and `log`

- The `0*` directories contain the values of all the variables at those time steps. The `0` directory is thus the initial condition.
- The `constant` directory contains the mesh and dictionaries for thermophysical and turbulence models.
- The `system` directory contains settings for the run, discretization schemes and solution procedures.

The `icoFoam` solver reads the files in the case directory and runs the case according to those settings.

The constant directory

- The transportProperties file is a dictionary for the dimensioned scalar nu.
- The polyMesh directory originally contains the blockMeshDict dictionary for the blockMesh mesh generator, and now also the mesh in OpenFOAM format.
- We will now have a quick look at the blockMeshDict dictionary in order to understand what mesh we have used.

Intro

cavity

Fine mesh

Finding tutorials

blockMeshDict **dictionary**

The `blockMeshDict` dictionary first of all contains a number of vertices:

```
    convertToMeters 0.1;
    vertices
    (
        (0 0 0)
        (1 0 0)
        (1 1 0)
        (0 1 0)
        (0 0 0.1)
        (1 0 0.1)
        (1 1 0.1)
        (0 1 0.1)
    );
```

- There are eight vertices defining a 3D block. OpenFOAM always uses 3D meshes, even if the simulation is 2D.

- `convertToMeters 0.1;` multiplies the coordinates by 0.1.

blockMeshDict **dictionary**

The blockMeshDict dictionary secondly defines a block and the mesh from the vertices:

```
    blocks
    (
        hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
    );
```

- hex means that it is a structured hexahedral block.
- (0 1 2 3 4 5 6 7) is the vertices used to define the block. The order of these is important - they should form a right-hand system (read the User-Guide yourself).
- (20 20 1) is the number of mesh *cells* in each direction.
- simpleGrading (1 1 1) is the expansion ratio, in this case equidistant. The numbers are the ratios between the end cells along three edges. There are other grading schemes as well (read the UserGuide yourself).

blockMeshDict **dictionary**

The blockMeshDict dictionary finally defines three patches:

```
patches
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }

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

    ...
);
```

blockMeshDict **dictionary**

Each patch defines a type, a name, and a list of boundary faces:

```
wall fixedWalls
(
    (0 4 7 3)
    (2 6 5 1)
    (1 5 4 0)
)
```

- `wall` is the type of the boundary.
- `fixedWalls` is the name of the patch.
- The patch is defined by three sides of the block according to the list, which refers to the vertex numbers. The order of the vertex numbers is such that they are marched clock-wise when looking from inside the block. This is important, and unfortunately `checkMesh` will not find such problems!

blockMeshDict **dictionary**

- To sum up, the blockMeshDict dictionary generates a block with:

  x/y/z dimensions 0.1/0.1/0.01

  20×20×1 cells

  wall fixedWalls patch at three sides

  wall movingWall patch at one side

  empty frontAndBack patch at two sides

  The type empty means that this is a 2D case (the normal direction to the empty patch is neglected)

- Read more about blockMesh yourself in the UserGuide

- You can also convert mesh files from thirdParty products, see the UserGuide

blockMeshDict **dictionary**

blockMesh uses the blockMeshDict to generate some files in the constant/polyMesh folder:

```
boundary points faces owner neighbour
```

- boundary shows the definitions of the pathches, for instance:

```
movingWall
{
    type wall;
    nFaces 20;
    startFace 760;
}
```

- The other files define the points, faces, and the relations between the cells.

The system directory

The system directory consists of three set-up files:

```
controlDict fvSchemes fvSolution
```

- controlDict includes general instructions on how to run the case.
- fvSchemes includes instructions on which discretization schemes that should be used for different terms in the equations.
- fvSolution includes instructions on how to solve each discretized linear equation system. It also contains instructions for the PISO pressure-velocity coupling.

The controlDict dictionary

The controlDict dictionary consists of the following lines:

```
application       icoFoam;
startFrom         startTime;
startTime         0;
stopAt            endTime;
endTime           0.5;
deltaT            0.005;
writeControl      timeStep;
writeInterval     20;
purgeWrite        0;
writeFormat       ascii;
writePrecision    6;
writeCompression  uncompressed;
timeFormat        general;
timePrecision     6;
runTimeModifiable yes;
```

The controlDict dictionary

- application icoFoam; → set the application to run (used only by scripting!)

- The following lines tell icoFoam to start at startTime=0, and stop at endTime=0.5, with a time step deltaT=0.005:

```
    startFrom          startTime;
    startTime          0;
    stopAt             endTime;
    endTime            0.5;
    deltaT             0.005;
```

The controlDict dictionary

- The following lines tell icoFoam to write out results in separate directories (purgeWrite 0;) every 20 timeStep(s), and that they should be written in uncompressed ascii format with writePrecision 6. timeFormat and timePrecision are instructions for the names of the time directories.

```
    writeControl      timeStep;
    writeInterval     20;
    purgeWrite        0;
    writeFormat       ascii;
    writePrecision    6;
    writeCompression  uncompressed;
    timeFormat        general;
    timePrecision     6;
```

- runTimeModifiable yes; allows you to make modifications to the case while it is running.

A dictionary hint

- If you don't know which entries are available for a specific key word in a dictionary, just use a dummy and the solver will list the alternatives, for instance:

```
stopAt dummy;
```

- When running icoFoam you will get the message:

```
dummy is not in enumeration
4
(
    nextWrite
    writeNow
    noWriteNow
    endTime
)
```

- Note that "`startFrom dummy`" only gives a warning and the simulation will start from time 0.

```
    --> FOAM Warning :
    From function Time::setControls()
    in file db/Time/Time.C at line 132
    expected startTime, firstTime or latestTime \
    found 'dummy' in dictionary \texttt{controlDict}
    Setting time to 0
```

Dictionary advanced features

- C++ commenting:

```
// This is my comment
/* My comments, line 1
My comments, line 2 */
```

- `#include` directive:

```
#include "initialConditions"
```

- Macro expansion:

```
flowVelocity    (20 0 0);
/*
    ...
*/

internalField uniform $flowVelocity;
/*
    ...
*/
```

Dictionary advanced features: regexp

OpenFOAM dictionaries does accept also regular expressions syntax:

- instead of writing:

```
    leftWall
    {
      type    fixedValue;
      value   uniform (0 0 0);
    }
    rightWall
    {
      type    fixedValue;
      value   uniform (0 0 0);
    }

    topWall
    {
      type    fixedValue;
      value   uniform (0 0 0);
    }
```

- you could write:

```
    "(left|right|top)Wall"
    {
      type   fixedValue;
      value  uniform (0 0 0);
    }
```

- or:

```
    ".*Wall"
    {
      type   fixedValue;
      value  uniform (0 0 0);
    }
```

The fvSchemes dictionary

- The fvSchemes dictionary defines the discretization schemes, in particular the time marching scheme and the convections schemes:

```
ddtSchemes
{
    default         Euler;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}
```

- Here we use the Euler implicit temporal discretization, and the linear (central-difference) scheme for convection.

- default none; means that schemes must be explicitly specified.

- Find the available convection schemes using a 'dummy' dictionary entry. There are 50 alternatives, and the number of alternatives are increasing!

The fvSolution dictionary

The fvSolution dictionary defines the solution procedure. The solutions of the p linear equation systems is defined by:

```
p
{
    solver             PCG;
    preconditioner     DIC;
    tolerance          1e-06;
    relTol             0;
};
```

- The p linear equation system is solved using the Conjugate Gradient solver PCG, with the preconditioner DIC.
- The solution is considered converged when the residual has reached the tolerance, or if it has been reduced by relTol at each time step.
- relTol is here set to zero since we use the PISO algorithm. The PISO algorithm only solves each equation once per time step, and we should thus solve the equations to tolerance 1e-06 at each time step. relTol 0; disables relTol.

The `fvSolution` dictionary

The solutions of the U linear equation systems is defined by:

```
U
{
    solver              PBiCG;
    preconditioner      DILU;
    tolerance           1e-05;
    relTol              0;
};
```

- The U linear equation system in solved using the Conjugate Gradient solver `PBiCG`, with the preconditioner `DILU`.

- The solution is considered converged when the residual has reached the `tolerance 1e-05` for each time step.

The fvSolution dictionary

The settings for the PISO algorithm are specified in the PISO entry:

```
PISO
{
     nCorrectors                2;
     nNonOrthogonalCorrectors 0;
     pRefCell                   0;
     pRefValue                  0;
}
```

- nCorrectors is the number of PISO correctors. You can see this in the log file since the p equation is solved twice, and the pressure-velocity coupling is thus done twice.
- nNonOrthogonalCorrectors adds corrections for non-orthogonal meshes, which may sometimes influence the solution.
- The pressure is set to pRefValue 0 in cell number pRefCell 0. This is over-ridden if a constant pressure boundary condition is used for the pressure.

The 0 folder

The 0 directory includes the dimensions, and the initial and boundary conditions for all primary variables, in this case p and U. An exmaple of the file 0/U is:

```
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type                fixedValue;
        value               uniform (1 0 0);
    }

    fixedWalls
    {
        type                fixedValue;
        value               uniform (0 0 0);
    }

    frontAndBack
    {
        type                empty;
    }
}
```

The `0` folder

- `dimensions [0 1 -1 0 0 0 0];` states that the dimension of `U` is m/s.
- `internalField uniform (0 0 0);` sets `U` to zero internally.
- The boundary patches `movingWall` and `fixedWalls` are given the `type fixedValue;` `value uniform (1 0 0);` and `(0 0 0)` respectively, i.e. $U_x = 1$ m/s, and $U = 0$ m/s respectively.
- The `frontAndBack` patch is given `type empty;`, indicating that no solution is required in that direction since the case is 2D.
- You should now be able to understand `0/p` also.
- The resulting 0.* directories are similar but the `internalField` is now a `nonuniform List<scalar>` containing the results. There is also a `phi` file, containing the resulting face fluxes that are needed to yield a perfect restart. There is also some time information in `0.*/uniform/time`. The `0.*/uniform` directory can be used for uniform information in a parallel simulation.

The `log` file

If you followed the earlier instructions you should now have a `log` file. That file provide information about the convergence of the simulation and the Courant number.

```
Time = 0.09

Courant Number mean: 0.116099 max: 0.851428
PBiCG:  Solving for Ux, Initial residual = 0.000443324,
Final residual = 8.45728e-06, No Iterations 2
PBiCG:  Solving for Uy, Initial residual = 0.000964881,
Final residual = 4.30053e-06, No Iterations 3
PCG:  Solving for p, Initial residual = 0.000987921,
Final residual = 5.57037e-07, No Iterations 26
time step continuity errors : sum local = 4.60522e-09,
global = -4.21779e-19, cumulative = 2.97797e-18
PCG:  Solving for p, Initial residual = 0.000757589,
Final residual = 3.40873e-07, No Iterations 26
time step continuity errors : sum local = 2.81602e-09,
global = -2.29294e-19, cumulative = 2.74868e-18
ExecutionTime = 0.11 s  ClockTime = 1 s
```

The `log` file

```
    PBiCG:  Solving for Ux, Initial residual = 0.000443324,
    Final residual = 8.45728e-06, No Iterations 2
```

- We used the `PBiCG` solver
- The `Initial residual` is calculated before the linear equation system is solved, and the `Final residual` is calculated afterwards.
- We see that the `Final residual` is less than our `tolerance` in `fvSolution` (`tolerance 1e-05;`).
- The `PBiCG` solver used 2 iterations to reach convergence.
- We could also see in the log file that the pressure residuals and continuity errors were reported twice each time step. That is because we specified `nCorrectors 2;` for the `PISO` entry in `fvSolution`.
- The `ExecutionTime` is the elapsed CPU time, and the `ClockTime` is the elapsed wall clock time for the latest time step.

foamLog

- It is of interest to have a graphical representation of the residual development.
- The foamLog utility is basically a script using grep, awk and sed to extract values from a log file.
- foamLog uses a database (foamLog.db) to know what to extract. The foamLog.db database can be modified if you want to extract any other values that foamLog doesn't extract by default.
- foamLog is executed on the cavity case with log-file log by:

```
foamLog log
```

- A directory logs has now been generated, with extracted values in ascii format in two columns. The first column is the Time, and the second column is the value at that time.
- Type foamLog -h for more information.
- You can plot the residual using MATLAB, gnuplot or xmgrace:

```
xmgrace -log y Ux_0 p_0
```

Increasing the mesh resolution

In the `blockMeshDict` dictionary the user should now change the mesh refinement from (20 20 1) to (40 40 1) and save the file.

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (40 40 1) simpleGrading (1 1 1)
);
```

- The `mapFields` utility maps one or more fields relating to a given geometry onto the corresponding fields for another geometry.
- The field data that mapFields maps is read from the time directory specified by startFrom/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.
- In the `controlDict` dictionary `startTime` should be set to 0.5 s and `startFrom` should be set to `startTime`.

NOTE: Command-line options of `mapFields` can be found by typing:

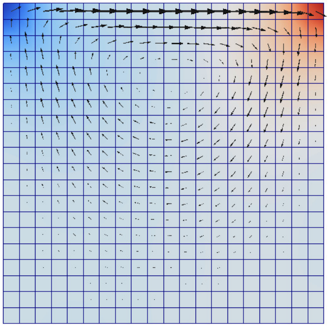
```
mapFields -help
```

mapFields

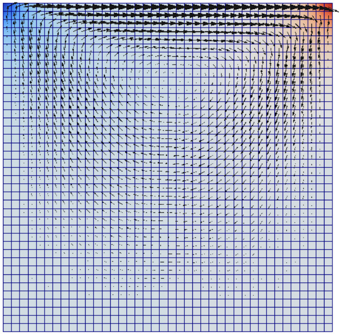
We are using the -consistent option of mapFields, which means that geometry and boundary conditions are the same:

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

Cavity



CavityFine



NOTE: if the 'consistent' options is not used, the dictionary mapFieldsDict must be supplied in the system folder. An example dictionary with comments can be found in source folder of mapFields.

Creating the graded mesh

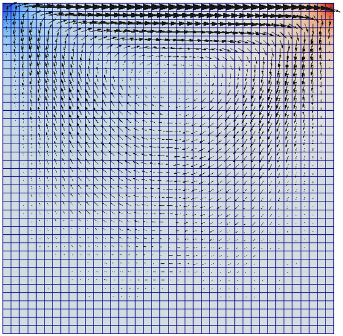
The error in any solution will be more pronounced in regions where the form of the true solution differ widely from the form assumed in the chosen numerical schemes. For example a numerical scheme based on linear variations of variables over cells can only generate an exact solution if the true solution is itself linear in form. The error is largest in regions where the true solution deviates greatest from linear form, i.e. where the change in gradient is largest. Error decreases with cell size.

The user can view the `blockMeshDict` file in the `constant/polyMesh` subdirectory of `cavityGrade`.

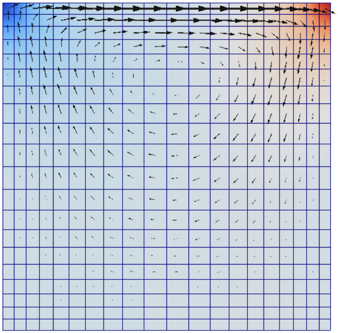
```
blocks
(
   hex (0 1 4 3 9 10 13 12) (10 10 1) simpleGrading (2 2 1)
   hex (1 2 5 4 10 11 14 13) (10 10 1) simpleGrading (0.5 2 1)
   hex (3 4 7 6 12 13 16 15) (10 10 1) simpleGrading (2 0.5 1)
   hex (4 5 8 7 13 14 17 16) (10 10 1) simpleGrading (0.5 0.5 1)
);
```

Graded mesh

CavityFine



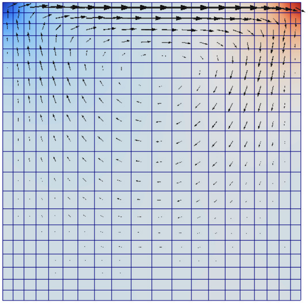
CavityGrade



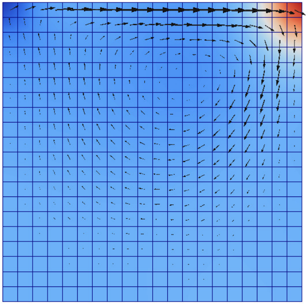
For the cavityGrade case the number of cells in each direction in a block is 10, the ratio between largest and smallest cells is 2 and the block height and width is 0.05 m. Therefore the smallest cell length is 3.45 mm. *The time step should be less than 3.45 ms to maintain a Courant of less than 1.* To ensure that results are written out at convenient time intervals, the time step deltaT should be reduced to 2.5 ms and the writeInterval set to 40 so that results are written out every 0.1 s. These settings can be viewed in the `cavityGrade/system/controlDict` file.

Increasing the Reynolds number

CavityGrade



CavityHighRe



We will now increase the Reynolds number to 50, at which point the solution takes a noticeably longer time to converge. The coarsest mesh in case cavity will be used initially. The user should make a copy of the cavity case and name it cavityHighRe by typing:

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam
cp -r cavity cavityHighRe
```

High Re flows: the `pisoFoam` solver

- The user can then increase the Reynolds number further by decreasing the viscosity and then rerun the case. The number of vortices increases so the mesh resolution around them will need to increase in order to resolve the more complicated flow patterns. In addition, as the Reynolds number increases the time to convergence increases.
- The need to increase spatial and temporal resolution then becomes impractical as the flow moves into the turbulent regime, where problems of solution stability may also occur.
- Reynolds-averaged stress (RAS) turbulence models may be used to solve for the mean flow behaviour and calculate the statistics of the fluctuations. The standard $k-\epsilon$ model with wall functions will be used in this tutorial to solve the lid-driven cavity case with a Reynolds number of 104.
- Two extra variables are solved for: k, the turbulent kinetic energy; and, ϵ, the turbulent dissipation rate. The additional equations and models for turbulent flow are implemented into an OpenFOAM solver called `pisoFoam`.

The pisoFoam solver: preprocessing

Change directory to the cavity case in the

```
$FOAM RUN/tutorials/incompressible/pisoFoam/ras/
```

Then:

- Generate the mesh by running `blockMesh`. Mesh grading towards the wall is not necessary when using the standard $k - \epsilon$ model with wall functions since the flow in the near wall cell is modelled, rather than having to be resolved.
- Select the wall function model to apply as boundary condition on individual patches through the turbulent viscosity field in the `0/nut` file.
- 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 `epsilonWall-Fun` boundary condition and a `kqRwallFunction` boundary condition is assigned to k.

The pisoFoam solver: preprocessing

From version 1.6, a range of wall function models may be applied as b.c. on individual patches. This enables different wall function models to be applied to different wall regions. The choice of wall function are specified through the turbulent viscosity field in the 0/nut file:

```
dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
  movingWall
  {
    type            nutWallFunction;
    value           uniform 0;
  }
  fixedWalls
  {
    type            nutWallFunction;
    value           uniform 0;
  }
  frontAndBack
  {
    type            empty;
  }
}
// ************************************************** //
```

Turbulence modeling

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:

```
    simulationType    RASModel;

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

The options for simulationType are `laminar`, `RASModel` and `LESModel`. With RASModel selected in this case, the choice of RAS modelling is specified in a RASProperties file, also in the constant directory. The turbulence model is selected by the RASModel entry from a long list of available models:

RAS turbulence models for compressible fluids – compressibleRASModels

laminar	Dummy turbulence model for laminar flow
kEpsilon	Standard k-ϵ model
kOmegaSST	k-ω-SST model
RNGkEpsilon	RNG k-ϵ model
LaunderSharmaKE	Launder-Sharma low-Re k-ϵ model
LRR	Launder-Reece-Rodi RSTM
LaunderGibsonRSTM	Launder-Gibson RSTM
realizableKE	Realizable k-ϵ model
SpalartAllmaras	Spalart-Allmaras 1-eqn mixing-length model

Finding tutorials for the utilities

- There are no tutorials for the utilities, but we can search for examples:

```
find $WM_PROJECT_DIR -name \*Dict | \
     grep -v blockMeshDict | grep -v controlDict
```

  You will get a list of example dictionaries for some of the utilities.
- Also, an example dictionary with comments can be found in each application source folder
- Most utilities take arguments and command-line options. Running the application with the option `-help`:

```
foamToVTK -help
```

  will display a help message.

Now you should be ready to go on exploring the applications by yourself!

Thank you for your attention!

contact: federico.piscaglia@polimi.it