



# 2D combustor case - incompressible flow test

Federico Piscaglia \*

Dept. of Aerospace Science and Technology (DAER), Politecnico di Milano, Italy

## Abstract.

Lab handout for study of incompressible flow inside a combustor-like geometry. Topics covered: setup of incompressible simulation of internal flow with different algorithms: SIMPLE, PISO, PIMPLE), analysis of residuals, simple function objects.

## 1 Learning outcome

The software used is the open-source CFD software OpenFOAM®-7.0 by the OpenFOAM Foundation. In this Lab you will learn how to:

- Set up an incompressible flow study by a steady solver;
- Set up an incompressible flow study by an unsteady solver;
- Set up simple function objects for residuals extraction, pressure average, aerodynamic drag;
- Critically analyze the results.

## 2 Prepare the mesh

The geometry to be simulated is represented in fig. 1. Flow will be simulated first using a steady-state incompressible solver (simpleFoam), then using its unsteady version (pimpleFoam).

Air is blown from the inlet at 10 m/s. In the first run, fuel inlets are closed (i.e. they are solid walls). All solid walls are modeled as no-slip.

Reference tutorial is ‘pitzDaily’ from the ‘incompressible/simpleFoam’ section.

```
user@host:run$ cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily combustorSimple  
user@host:run$ cd combustorSimple
```

### 2.1 Construction

1. Download the `blockMeshDict` file from the Beep web page of the course

```
user@host:combustorSimple$ cp $HOME/Downloads/blockMeshDict system/.
```

\*Tel. (+39) 02 2399 8620, E-mail: [federico.piscaglia@polimi.it](mailto:federico.piscaglia@polimi.it)

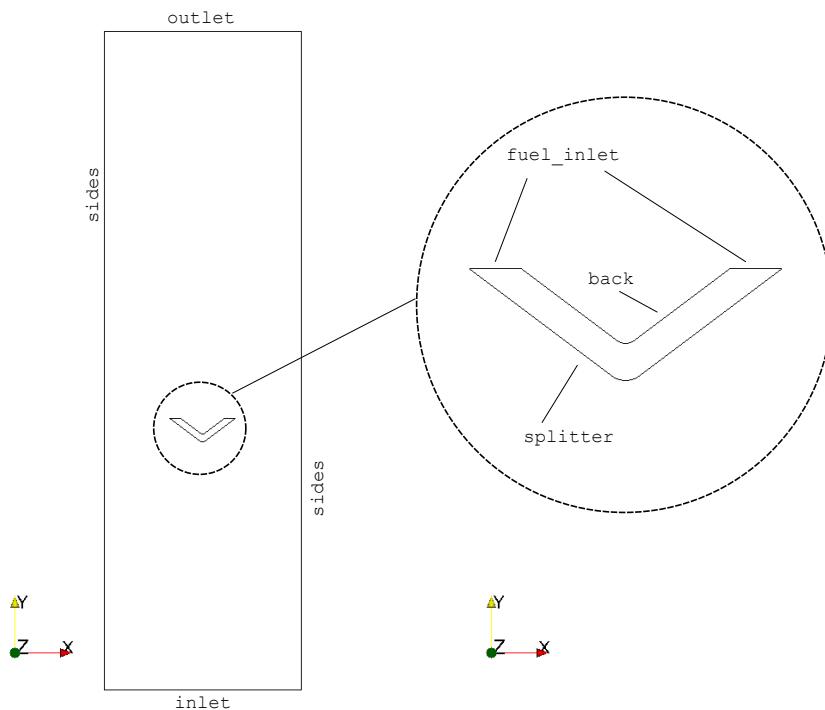


Figure 1: geometry

- Check the new `blockMeshDict`; run `blockMesh`

```
user@host:combustorSimple$ blockMesh
```

- Check the mesh; see if mesh generated as expected and no error are present

```
user@host:combustorSimple$ checkMesh
```

Note: the folder ‘0’ contains initial and boundary conditions not yet defined for this mesh. Trying to open the initial fields with ParaView will crash with an error. Two workaround are possible:

- rename ‘0’ to ‘0\_’
- Disable all fields when opening ParaView.

## 2.2 Adjust patch types

- Open ‘constant/polyMesh/boundary’

- Check patch types. They must be:

- ‘wall’ for all solid walls (sides, splitter, back, fuel\_inlet)
- ‘patch’ for all fluid patches (inlet, outlet)
- ‘empty’ for frontAndBack

- Remove group info line from fluid and empty patches. Save and close.

## 2.3 Adjust physical BCs

- Delete unnecessary files in ‘0’: v2, f, nuTilda, omega.

2. Open all files in ‘0’.
3. Patch can be grouped: you can use the group name instead of the patch name to set the same BC on all patch belonging to a group (e.g. ‘wall’).
4. Velocity:
  - ‘noSlip’ on solid walls
  - fixed value, 10 m/s aligned with  $+y$  on inlet;
  - ‘zeroGradient’ on outlet
  - ‘empty’ for frontAndBack
5. Pressure:
  - ‘zeroGradient’ on solid walls and inlet
  - fixed value, 0  $\text{m}^2/\text{s}^2$  on the outlet<sup>1</sup>
  - ‘empty’ for frontAndBack
6. Turbulent Kinetic Energy (see Sec. A.1)
  - In the internal field  $k = 0.375 \text{ m}^2/\text{s}^2$
  - TKE wall function on walls:
 

```
type          kqRWallFunction;
value         uniform 0.375;
```
  - corresponding to turbulence intensity 5% on the inlet
 

```
type          turbulentIntensityKineticEnergyInlet;
intensity     0.05;
value         uniform 0.375;
```
  - ‘zeroGradient’ on outlet
  - ‘empty’ for frontAndBack
7. Turbulent dissipation rate (see Sec. A.2):
  - In the internal field  $\varepsilon = 2.0 \text{ m}^2/\text{s}^3$
  - Epsilon wall function on walls:
 

```
type          epsilonWallFunction;
value         uniform 2.0;
```
  - corresponding to mixing length of 0.02 m on the inlet:
 

```
type          turbulentMixingLengthDissipationRateInlet;
mixingLength  2e-2;
value         uniform 2.0;
```
  - ‘zeroGradient’ on outlet
  - ‘empty’ for frontAndBack
8. Turbulent viscosity (see Sec. A.3)
  - Standard wall function on walls
 

```
type          nutkWallFunction;
value         uniform 0;
```
  - ‘calculated’ on all boundaries but empty
 

```
type          calculated;
value         uniform 0;
```
  - ‘empty’ for frontAndBack

<sup>1</sup>Incompressible N-S equations are solved for  $(p - p_{\text{ref}})/\rho$

## 2.4 Renumber the mesh

```
user@host:combustor2D$ renumberMesh -overwrite
```

## 3 Set up numerics

1. Open system/fvSolution
2. Tighten residual controls in SIMPLE subdictionaries (use  $10^{-3}$  for pressure and  $10^{-4}$  for velocity)
3. Delete residual control on turbulent quantities
4. Increment under-relaxation (i.e. decrement coefficients):

```
relaxationFactors
{
    fields
    {
        p          0.3;
    }

    equations
    {
        U          0.7;
        "(k|epsilon)" 0.5;
    }
}
```

## 4 Set up post-processing

The following quantities are to be extracted:

1. Residuals of  $p$ ,  $U$ ,  $k$ ,  $\varepsilon$
2. Pressure on the inlet patch
3. Drag force on the splitter

All extracted quantities are written in the subfolder ‘postProcessing’. Data can be plotted run-time with the command (-l means log-scaling)

```
user@host:combustor2D$ foamMonitor [-l] <path-to-dat-file>
```

### 4.1 Residuals

Delete any line in subdictionary ‘functions’, file ‘controlDict’.

Add the following lines to the controlDict, subdictionary ‘functions’:

```
#includeFunc residuals(p, U, k, epsilon)
```

that substitutes the old syntax (still active) of OpenFOAM-7 and below:

```

residuals
{
    type    residuals;
    enabled yes;
    fields (p U k epsilon);
    libs ("libutilityFunctionObjects.so");
}

```

## 4.2 Pressure drop

```

pInlet
{
    type    surfaceFieldValue;
    enabled yes;
    log true;
    writeControl timeStep;
    writeInterval 1;

    regionType patch;
    writeFields no;
    name inlet;
    operation average;
    fields (p);

    libs ("libfieldFunctionObjects.so");
}

```

## 4.3 Drag

```

force
{
    type forces;
    enabled yes;
    libs ("libforces.so");
    log      yes;
    patches  (splitter back fuel_inlet);
    rho rhoInf;
    rhoInf 1.18;

    writeControl timeStep;
    writeInterval 1;

    liftDir      (1 0 0 );
    dragDir      (0 1 0 );
    CofR         (0 0 0 );
}

```

## 5 Run the case (steady-state)

Launch the solver `simpleFoam`:

```
user@host:combustor2D$ simpleFoam > log.simpleFoam
```

Look at the results, both with ParaView and the extracted data. What conclusion can be drawn?

## 6 Unsteady solver

Now run the case using an unsteady solver. Copy the first case and delete all results:

```
user@host:combustorSimple$ cd ..
user@host:combustorSimple$ cp -r combustorSimple combustorPimple
user@host:combustorSimple$ cd combustorPimple
user@host:combustorPimple$ rm -r *00 log* postProcessing*
```

Modify the setup as in the followings.

### 6.1 Run control

- end time: 0.5 s
- initial  $\Delta t = 10^{-4}$  s
- write controls:

```
writeControl      adjustableRunTime;
writeInterval    0.01;
```

- Automatic timestep selection (see Appendix B). Set a maximum  $Co = 20$ :

```
adjustTimeStep yes;
maxCo 20;
```

### 6.2 Numerical schemes

- Use first-order Euler for time derivatives:

```
ddtSchemes
{
    default      Euler;
}
```

- Limit the gradient of velocity:

```
gradSchemes
{
    default      Gauss linear;
    grad(U) cellLimited Gauss linear 1;
}
```

- Do not use ‘bounded’ convection schemes (remove the keyword)

### 6.3 Algorithm control

- Add ‘final’ iteration linear solvers:

```

pFinal
{
    $p;
    relTol 0;
}

"(U|k|epsilon|omega|f|v2)Final"
{
    solver          smoothSolver;
    smoother        symGaussSeidel;
    tolerance       1e-05;
    relTol 0;
}

```

Add the PIMPLE subdictionary:

```

PIMPLE
{
    nCorrectors 1;
    nOuterCorrectors 100;//20;
    turbOnFinalIterOnly no;
    nNonOrthogonalCorrectors 0;
    consistent      no;

    outerCorrectorResidualControl
    {
        "(U|k|epsilon|p)"
        {
            relTol      0;
            tolerance   1e-4;
        }
    }
}

```

- Add under-relaxation for ‘final’ quantities:

```

relaxationFactors
{
    fields
    {
        "p.*"           0.3;
    }
    equations
    {
        "U.*"           0.7;
        "(k|omega|epsilon).*" 0.5;
    }
}

```

Note: if you use `maxCo=20`; you are enlarging the timestep significantly. It is recommended to update the turbulent field at any PIMPLE outer iteration. This can be done by setting

```
turbOnFinalIterOnly no
```

in the `fvSolution.PIMPLE` dictionary.

## 6.4 Run the solver

```
user@host:combustor2D$ pimpleFoam > log.pimpleFoam 2>&1 &
```

Look at the results and analyze them. What considerations can be made with respect to the steady-state solver run?

# 7 Hands-on

## 7.1 Use PISO instead of PIMPLE

1. set nCorrectors to 2 and nOuterCorrectors to 1
2. Comment out the relaxation factors (or set them to 1)
3. Set CFL to less than 1
4. Run and compare the results (**If crash occurs, did you check the  $\Delta t$ !?**)

## 7.2 Simulate the fuel inlet

Repeat all analysis done so far with the fuel inlet activated. Inlet velocity is 15 m/s aligned with  $y$ .

**Hint #1** Copy the cases. Do not overwrite the old ones nor start from scratch.

**Hint #2** You have to modify both the patch type and the physical BC.

# A RANS turbulence model setup

## A.1 Turbulent kinetic energy $k$

- On the inlet,  $k$  is calculated as:

$$k = \frac{3}{2} \overline{u'_i u'_i} \quad (1)$$

where  $u'$  is defined as a fraction of the mean velocity  $U(x, y)$ ; Ratio between  $u'$  and  $U$  is the turbulence intensity  $I$

- As internal field for  $k$ , use the same value that is set at the inlet, from Eq. (1).
- On the walls, the use of a standard wall function requires  $\partial k / \partial n = 0$  (i.e. ‘zeroGradient’). However, to stress the fact that a wall-function is used, OpenFOAM requires the use of ‘kqrWallFunctions’ on solid walls when using RANS wall functions.

## A.2 Kinetic energy dissipation rate $\varepsilon$

- On wall cells, the dissipation rate has to be calculated according to:

$$\varepsilon_P = \frac{C_\mu^{3/4} k_P^{3/2}}{\kappa y_P} \quad (2)$$

where  $\kappa$  is the von Karman constant ( $\kappa = 0.41$ ),  $C_\mu = 0.09$  and  $P$  represents the center of wall cell. To use the above relation, specify a dissipation wall function:

```
"wall-.*"
{
    type          epsilonWallFunction;
    value         uniform 0;
}
```

- On the inlet, the dissipation can be set using the mixing-length relation:

$$\varepsilon = \frac{C_\mu^{3/4} k^{3/2}}{\ell} \quad (3)$$

where  $\ell$  is the mixing length, which can be considered as  $\ell = 0.07L$  where  $L$  is a characteristic length of the geometry (inlet diameter).

- As internal field for  $\varepsilon$ , please use the same value that it is set at the inlet, coming from Eq. (3).

### A.3 Turbulent viscosity $\nu_T$

It is not mandatory to specify boundary conditions for eddy viscosity  $\nu_T$ , but it is advisable because it allows for choosing the appropriate wall treatment for solid boundaries.

The following types of wall treatment are available in OpenFOAM:

- Standard (log-law) WF for smooth surfaces;

```
nutkWallFunction
nutUWallFunction
```

The difference between the two models is in the way  $y^+$  is computed, with either  $k$  or  $\nabla U$ . Both of them need  $y^+ > 30$

- Standard WF for rough walls:

```
nutkRoughWallFunction
nutURoughWallFunction
```

- Universal Spalding's profile: no constraints on  $y^+$

```
nutUSpaldingWallFunction
```

- Low-Re wall treatment; actual it does nothing but setting  $\nu_T = 0$  on walls; it requires  $y^+ < 1$

```
nutLowReWallFunction
```

- Tabulated WF: reads  $u^+(y^+)$  from a file.

```
nutUTabulatedWallFunction
```

If boundary conditions on  $\nu_T$  are not explicitly set (i.e.: no file `nut` is present), all solid walls are given a standard wall function (for smooth walls). For  $k - \varepsilon$  models, a standard WF has to be employed. On other boundaries the BC to be imposed must be of 'calculated' type.

## B Automatic timestep selection

OpenFOAM allows the automatic selection of  $\Delta t$  to fulfill the CFL condition specified in the controlDict. The CFL (or Courant Number) is defined as:

$$Co = \frac{U \Delta t}{\Delta x} \quad (4)$$

Hence, the timestep must be adapted to the maximum local ratio  $U/\Delta x$ .

Implicit solvers like those implemented in OpenFOAM do not strictly require  $Co \leq 1$  for stability. However, in practice some constraints hold:

- In PISO-type solvers  $Co < 1$  is required for accuracy of the pressure-velocity coupling rather than stability of the solver. However, a poor accuracy (i.e. running with  $Co > 1$ ) will often lead to divergence and solution crash.
- In PIMPLE-type solvers the  $p - U$  coupling is iterated until convergence, hence one can set  $Co > 1$ . The higher the CFL, the more the solution will tend to a steady-state one. Too high a  $Co$  might lead to divergence too, because unphysical solutions are computed. Moreover, sometimes the solution is stable but far from the physical reality because phenomena with high characteristic velocity are not captured.

Moreover, some practical aspects must be taken into account when using automatic timestep selection:

- $Co$  is computed at the beginning of each time step with previous-time velocities. This can lead to a CFL that is slightly different from the pre-set one.
- At the first timestep, CFL is computed using the initial velocity conditions.
- $\Delta t$  variation is limited around 15% between subsequent timesteps, to avoid instability problems. This can have unexpected consequences:
  - i. if the initial  $\Delta t$  is too high, OpenFOAM cannot adjust it to fulfill the pre-set  $Co$ , and the simulation might crash at first timesteps
  - ii. if the initial  $\Delta t$  is too low, it can take a long time to reach the selected  $Co$

Finally, when computing compressible flows (wave propagation), an accurate description of the phenomena requires the following condition:

$$Co_{\text{acoustic}} = \frac{(U + c)\Delta t}{\Delta x} \quad (5)$$

where  $c$  is the local speed of sound.