CFD with OpenSource Software

A course at Chalmers University of Technology
Taught by Håkan Nillson

PANS turbulence model implementation

Developed for OpenFOAM-2.3.x

Author:
Guglielmo Minelli

Peer reviewed by:
Jelena Andric
Baris Bicer

January 8, 2015

Contents

1 Introduction

Computational Fluid Dynamics (CFD) simulations play a crucial role in both academic and industrial fluid mechanics research. Resolution and accuracy are the most important factors that lead to the choice of different approaches to solve the Navier-Stokes equations that describe the motion of the fluid. Direct numeric simulation, the so called DNS, is the most accurate approach to solve iteratively the complete N-S equations. DNS requires a very fine resolution of the flow domain in order to be able to have reliable results and having a proper DNS resolution became early a limitation, to solve ordinary but complex flow fields. For this reason, the DNS approach is mainly used in academic research for relatively simple flow fields. In order to avoid the mentioned limitation, a turbulence model approach is used. Different turbulence models are used, based on the mesh resolution required and the computer resources available. The higher the mesh resolution, the smaller structures of the flow are resolved, and more computer time is required. The modelling of turbulence involves sets of equations from Reynolds Averaged Navier-Stokes (RANS) equations, that resolve the average velocity and pressure of the flow, to Large Eddy Simulation (LES), that models only the dissipative part of the turbulence. In between RANS and LES approach, take place the so called Hybrid Methods. The aim of these methods is to find a good compromise between computational time and accuracy of the results. The mentioned methods switch solving equations from DNS or LES to RANS model. In particular Detached Eddy Simulation (DES), use a fixed zonal approach, where the flow close to surfaces is modeled by RANS and the flow far from the wall is resolved by LES. In the last years, there is an increasing interest in the so called Partially Averaged Navier Stokes (PANS) method. In PANS model the zonal approach is not fixed as for DES. However it uses a set of equations which includes a factor sensitive to the size of the cell and the level of turbulence, in terms of kinetic energy. This project work is concerned with the implementation of the PANS model for both low and high Reynolds number approach in OpenFoam, starting and modifying the RANS $k - \varepsilon$ turbulence model, already present in the software.

2 Background

RANS and DNS simulations give both limitations and advantages to the users. DNS resolves all the structures of the flow but requires an extremely fine grid to obtain a reliable results. On the other hand, RANS equations solve only the mean velocity field and require a smaller number of cells, making the simulation faster. Thus, it is of interest to find a balance between the solution and the computational time required. The PANS approach is able to switch between the mentioned methods, thanks to the blending factor f_k, and evaluating the flow field at each time step. The f_k parameter, which will be described in details in the next section, can be defined both as a fixed and a space and time varying value, between 0 and 1. The PANS equations system results in a more complex set of equations, which requires more time to be solved for each time step, but compared to DNS, a much coarser resolution can be used to achieve promising results as it has been proved, for example, in Ref. [1] and [2]. A first PANS turbulence model with a fixed f_k was already implemented in Ref. [3].

3 Theory and methodology

Two implemented turbulence models are discussed in this work. The first model uses a two equations RANS derived model governed by a varying f_k. The second uses a four equations system, the so called $k-\varepsilon-\zeta-f$ model, introduced in Ref. [4]. The description of the models is described and discussed in this section.

3.1 The standard OpenFOAM $k-\varepsilon$ model

The standard RANS $k-\varepsilon$ closure equations system on which is based this work reads

$$\frac{\partial k}{\partial t}+U_j\frac{\partial k}{\partial x_j}=P_k-\varepsilon+\frac{\partial}{\partial x_j}\left[\left(\nu+\frac{\nu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right] \tag{1}$$

$$\frac{\partial \varepsilon}{\partial t}+U_j\frac{\partial \varepsilon}{\partial x_j}=C_{1\varepsilon}\frac{\varepsilon}{k}P_k-C_{2\varepsilon}\frac{\varepsilon^2}{k}+\frac{\partial}{\partial x_j}\left[\left(\nu+\frac{\nu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_j}\right] \tag{2}$$

where k is the turbulent kinetic energy and ε is its dissipation. The constant in this model are

$$C_{1\varepsilon}=1.44; \quad C_{2\varepsilon}=1.92; \quad \sigma_k=1; \quad \sigma_\varepsilon=1.3; \tag{3}$$

This model is placed in the folder $/OpenFOAM/OpenFOAM-2.3.x/src/...$ $...turbulenceModels/incompressible/RAS/kEpsilon$. The model is implemented using two files *kEpsilon.C* and *kEpsilon.H*. The equations are implemented and presented in the *kEpsilon.C* file as follows

```
// Dissipation equation
tmp<fvScalarMatrix> epsEqn
(
    fvm::ddt(epsilon_)
  + fvm::div(phi_, epsilon_)
  - fvm::laplacian(DepsilonEff(), epsilon_)
 ==
    C1_*G*epsilon_/k_
  - fvm::Sp(C2_*epsilon_/k_, epsilon_)
);

epsEqn().relax();

epsEqn().boundaryManipulate(epsilon_.boundaryField());

solve(epsEqn);
bound(epsilon_, epsilonMin_);

// Turbulent kinetic energy equation
tmp<fvScalarMatrix> kEqn
(
    fvm::ddt(k_)
  + fvm::div(phi_, k_)
  - fvm::laplacian(DkEff(), k_)
 ==
    G
  - fvm::Sp(epsilon_/k_, k_)
);

kEqn().relax();
solve(kEqn);
bound(k_, kMin_);
```

And the effective diffusivity DkEff and DepsilonEff for k and ε respectively are computed in the *kEpsilon.H* as follow

```
//- Return the effective diffusivity for k
tmp<volScalarField> DkEff() const
{
    return tmp<volScalarField>
    (
        new volScalarField("DkEff", nut_ + nu())
    );
}

//- Return the effective diffusivity for epsilon
tmp<volScalarField> DepsilonEff() const
{
    return tmp<volScalarField>
    (
        new volScalarField("DepsilonEff", nut_/sigmaEps_ + nu())
    );
}
```

Where `nut_` is the modelled turbulent viscosity and nu is the laminar viscosity.

3.2 Two equations model

Considering the two equations model, a high Reynolds number approach is used. The main difference with respect the standard $k-\varepsilon$ model consists in the ε equation. In particular, the previous constant coefficient $C_{2\varepsilon}$ is substituted with the time and space varying $C^{\star}_{2\varepsilon}$. The new equations system reads,

$$\nu_u = C_\mu \frac{k_u^2}{\varepsilon_u}, \tag{4}$$

$$\frac{\partial k_u}{\partial t} + U_j \frac{\partial k_u}{\partial x_j} = P_u - \varepsilon_u + \frac{\partial}{\partial x_j}\left(\frac{\nu_u}{\sigma_{k_u}}\frac{\partial k_u}{\partial x_j}\right), \tag{5}$$

$$\frac{\partial \varepsilon_u}{\partial t} + U_j \frac{\partial \varepsilon_u}{\partial x_j} = C_{1\varepsilon_u}\frac{\varepsilon_u}{k_u}P_u - C^{\star}_{2\varepsilon}\frac{\varepsilon_u^2}{k_u} + \frac{\partial}{\partial x_j}\left(\frac{\nu_u}{\sigma_{\varepsilon_u}}\frac{\partial \varepsilon_u}{\partial x_j}\right), \tag{6}$$

Here

$$C^{\star}_{2\varepsilon} = C_{1\varepsilon} + f_k(C_{2\varepsilon} - C_{1\varepsilon}) \tag{7}$$

where the f_k parameter is given as

$$f_k = \frac{1}{\sqrt{c_\mu}}\left(\frac{\Delta}{\Lambda}\right)^{2/3}; \quad \Lambda = \frac{k^{3/2}}{\varepsilon}; \quad \Delta = (\Delta_x\Delta_y\Delta_z)^{1/3} \tag{8}$$

Here Δ is the characteristic cell size, and Λ is the Taylor scale calculated at each cell. The constants of the model are

$$C_{1\varepsilon} = 1.44; \;\; C_{2\varepsilon} = 1.92; \;\; C_\mu = 0.09; \;\; c_\mu = 0.22; \tag{9}$$

$$\sigma_{\varepsilon_u} = 1; \;\; \sigma_{k_u} = 1.3; \tag{10}$$

The parameters f_k and $C^\star_{2\varepsilon}$ are updated every timestep during the simulation and they directly influence the consecutive time step. They are calculated as a `scalarField` all over the domain. Based on the level of turbulence k and the size of the mesh Δ, they define the percentage of resolved and unresolved kinetic energy for each cell. The principle is to solve the flow using a DNS approach where it is allowed by f_k, grid and flow condition, or switch to a $k - \varepsilon$ model where the flow is undisturbed and the grid is allowed to have more stretched cells without affecting the results. In other words when the cell size is much lower than the Taylor scale, the model solve the flow using a DNS approach. When the Taylor scale become smaller then the cell size, instead, the model switch to RANS equations.

3.3 Four equations model $k - \varepsilon - \zeta - f$

In order to have a low Reynold number approach to solve the domain, a PANS four equations turbulence model is introduced, as presented in [4].

$$\nu_u = C_\mu \zeta \frac{k_u^2}{\varepsilon_u} \tag{11}$$

$$\frac{\partial k_u}{\partial t} + U_j \frac{\partial k_u}{\partial x_j} = P_u - \varepsilon_u + \frac{\partial}{\partial x_j}\left(\frac{\nu_u}{\sigma_{k_u}}\frac{\partial k_u}{\partial x_j}\right) \tag{12}$$

$$\frac{\partial \varepsilon_u}{\partial t} + U_j \frac{\partial \varepsilon_u}{\partial x_j} = C_{1\varepsilon_u}\frac{\varepsilon_u}{k_u}P_u - C^\star_{2\varepsilon}\frac{\varepsilon_u^2}{k_u} + \frac{\partial}{\partial x_j}\left(\frac{\nu_u}{\sigma_{\varepsilon_u}}\frac{\partial \varepsilon_u}{\partial x_j}\right) \tag{13}$$

$$C^\star_{2\varepsilon} = C_{1\varepsilon} + f_k(C_{2\varepsilon} - C_{1\varepsilon}) \tag{14}$$

$$\frac{\partial \zeta_u}{\partial t} + U_j \frac{\partial \zeta_u}{\partial x_j} = f_u - \frac{\zeta_u}{k_u}P_u + (1 - f_k)\frac{\zeta_u}{k_u}\varepsilon_u + \frac{\partial}{\partial x_j}\left(\frac{\nu_u}{\sigma_{\zeta_u}}\frac{\partial \zeta_u}{\partial x_j}\right) \tag{15}$$

$$L_u^2 \nabla^2 f_u - fu = \frac{1}{T_u}\left(c_1 + c_2\frac{P_u}{\varepsilon_u}\right)\left(\zeta_u - \frac{2}{3}\right) \tag{16}$$

and the constants of the model are

$$c_1 = 0.4; \;\; c_2 = 0.65 \tag{17}$$

Moreover, $C_{1\varepsilon}$ is now defined as

$$C_{1\varepsilon} = 1.4\left(1 + \frac{0.045}{\sqrt{\zeta_u}}\right) \tag{18}$$

Here L_u and T_u are the length and time scale respectively defined by using the unresolved kinetic energy.

$$T_u = max\left[\frac{k_u}{\varepsilon_u}, C_\tau\left(\frac{\nu}{\varepsilon}\right)^{1/2}\right]; \;\; L_u = C_L max\left[\frac{k_u^{3/2}}{\varepsilon_u}, C_\eta\left(\frac{\nu^3}{\varepsilon}\right)^{1/4}\right] \tag{19}$$

where

$$C_\tau = 6; \;\; C_L = 0.36; \;\; C_\eta = 85 \tag{20}$$

A transport equation for the ζ_u parameter and a Poisson equation for f_u are added to the two equations model. The f_k implementation is kept the same as for the previous model.

4 OpenFoam2.3x implementation

The OpenFoam already contains the well known, and previously presented, $k-\varepsilon$ turbulence model. This is a starting point for the modifications necessary to create the PANS model.

4.1 Two equations model

The procedure how to implement two equations PANS model according to varying f_k is explained here.

- Open a new terminal
- Load openFoam OF23x
- Copy kEpsilon turbulence model in your own directory
- Rename the folder, and its .C and .H files, for example to kEpsilonPANS
- Replace kEpsilon with kEpsilonPANS using the sed command

```
sed -i s/kEpsilon/kEpsilonPANS/g kEpsilonPANS*
```

- Modify `Make/files` file by substituting with the following, to create your own library.

```
kEpsilonPANS/kEpsilonPANS.C
LIB = $(FOAM_USER_LIBBIN)/libmyIncompressibleRASModels
```

Here, we start to intro duce a new varying f_K as distinct from fixed f_K implementation previously done by Asnaghi in Ref.[3].

4.1.1 kEpsilonPANS.C

It is necessary to introduce new volScalarField and constant. Before `Cmu_`, c_μ=`Cmi_` is introduced,

```
Cmi_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cmi",
        coeffDict_,
        0.22
    )
),
```

after sigmaEps introduce the following

```
fEpsilon_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "fEpsilon",
        coeffDict_,
        1.0
    )
),
```

Now one is ready to introduce the new parameter that defines the varying f_k adding the following, after the previously introduced `fEpsilon_`.

```
cellVolume
(
   IOobject
   (
       "cellVolume",
       runTime_.timeName(),
       mesh_,
       IOobject::NO_READ,
       IOobject::AUTO_WRITE
   ),
   mesh_,
   dimensionedScalar("zero",dimVolume,0.0)
),

fK_
(
    IOobject
    (
        "fK",
        runTime_.timeName(),
        mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_
),
C2U
(
    IOobject
    (
        "C2U",
        runTime_.timeName(),
        mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_,
   dimensionedScalar("zero",0.0)
),
```

after epsilon_ the variables, k_u and ε_u, of the unresolved field need to be introduced.

```
    kU_
    (
        IOobject
        (
            "kU",
            runTime_.timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        autoCreateK("kU", mesh_)
    ),

    epsilonU_
    (
        IOobject
        (
            "epsilonU",
            runTime_.timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        autoCreateEpsilon("epsilonU", mesh_)
    ),
```

Let us now initialize the variables by implementing the following in the curly brackets, before the member function section.

```
{
    cellVolume.internalField() = mesh_.V();
    C2U = C1_ + (fK_/fEpsilon_)*(C2_-C1_);
    ....
    ....
}
```

Now we need to change the solved equations. We want to solve for unresolved quantities.

```
    //Unresolved Dissipation equation
    tmp<fvScalarMatrix> epsUEqn
    (
        fvm::ddt(epsilonU_)
      + fvm::div(phi_, epsilonU_)
      - fvm::laplacian(DepsilonUEff(), epsilonU_)
     ==
        C1_*G*epsilonU_/kU_
      + fvm::Sp(C1_*(1-fK_)*(epsilonU_)/(kU_), epsilonU_)
      - fvm::Sp(C2U*epsilonU_/kU_, epsilonU_)
    );

    epsUEqn().relax();

    epsUEqn().boundaryManipulate(epsilonU_.boundaryField());

    solve(epsUEqn);
    bound(epsilonU_, fEpsilon_* epsilonMin_);


    //Unresolved Turbulent kinetic energy equation
    tmp<fvScalarMatrix> kUEqn
    (
        fvm::ddt(kU_)
      + fvm::div(phi_, kU_)
      - fvm::laplacian(DkUEff(), kU_)
     ==
        G
      - fvm::Sp(epsilonU_/kU_, kU_)
    );

    kUEqn().relax();
    solve(kUEqn);
    bound(kU_, min(fK_)*kMin_);
```

Finally, we introduce the recalculation of k epsilon viscosity, and the new parameters `fK_` and `C2U`, respectively.

```
    // Calculation of Turbulent kinetic energy and Dissipation rate
    k_=kU_/fK_;
    epsilon_=(epsilonU_)/(fEpsilon_);

    // Re-calculate viscosity
    nut_ = Cmi_*sqr(kU_)/epsilonU_; //changed to Cmu_ to Cmi_
    nut_.correctBoundaryConditions();

   // Re calculation of fK and C2U for PANS fK is calculated cell by
   // cell and upper and lower limited to avoid unfisical solution
    for (int i=0; i<fK_.size(); i++)
    {
      fK_[i]=(1/sqrt(Cmi_.value()))*(pow(pow(cellVolume[i],1.0/3.0)/
             (pow(k_[i],3.0/2.0)/epsilon_[i]),2.0/3.0));
      if (fK_[i]>1)   //upper limited
          fK_[i]=1;
      if (fK_[i]<0.1) //lower limited
          fK_[i]=0.1;
    }
    C2U = C1_+ (fK_/fEpsilon_)*(C2_-C1_);
```

4.1.2 kEpsilonPANS.H

The .H file must be modified to introduce the new declared variables. The `Cmi_` coefficient is introduced in the model coefficient section.

```
        // Model coefficients
            dimensionedScalar Cmi_;
            ...
            ...
```

In fields section the new fields are introduced in the following order,

```
        // Fields
            volScalarField cellVolume; //new
            volScalarField fK_;        //new
            volScalarField C2U;        //new
            volScalarField k_;
            volScalarField epsilon_;
            volScalarField kU_;        //new
            volScalarField epsilonU_;  //new
            volScalarField nut_;
```

It is necessary to calculate the effective diffusivity for k_u and ε_u by adding the following in the member function section.

```
        //- Return the effective diffusivity for unresolved k
        tmp<volScalarField> DkUEff() const
        {
            return tmp<volScalarField>
            (
                new volScalarField
                ("DkUEff", nut_/(fK_*fK_/fEpsilon_)
                 + nu())
            );
        }

        //- Return the effective diffusivity for unresolved epsilon
        tmp<volScalarField> DepsilonUEff() const
        {
            return tmp<volScalarField>
            (
                new volScalarField
                ("DepsilonUEff", nut_/(fK_*fK_*sigmaEps_/fEpsilon_)
                 + nu())
            );
        }
```

4.2 Result and discussion

In this section some interesting results and comparison are discussed in order to describe the behaviour of the implemented model. A comparison between the original $k-\varepsilon$, the fixed f_k and the varying f_k turbulence model is also reported. The test case is a 2D backstep geometry, created modifying the pitzDaily tutorial geometry, see Fig. 1.

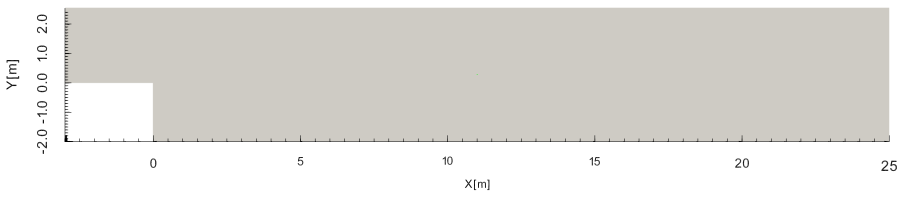

Figure 1: Backstep test geometry.

In order to simulate a test case, the inlet velocity is choden to be $U = 10m/s$ and all the simulation are carried out using the unsteady `pimpleFoam` solver. Instantaneous and averaged velocity of three different simulations are shown in Fig. 2 and Fig. 3, respectively. The shown results correspond to standard k_ε URANS, fixed f_k PANS, and varying f_k PANS.

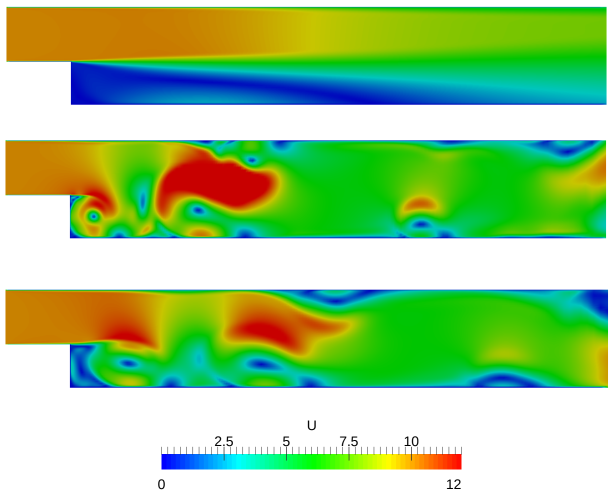

Figure 2: Instantaneous velocity field. From top to bottom: URANS $k-\varepsilon$, fixed f_k PANS, varying f_k PANS.

The Unsteady Raynolds Averaged Navier Stokes (URANS) simulation does not present any fluctuation of the instantaneous flow field. Usually URANS simulation are able to capture large oscillation of the flow, such as vortex shedding, that is not present in a 2D backstep case. The fixed f_k PANS simulation present the highest flow mixing, Fig. 2, the value of $f_k = 0.2$ is chosen and the solver is forced to solve the flow with an approach close to DNS, resulting in quite different averaged flow compared to averaged flow solved by the URANS, Fig. 3. On the other hand, the varying f_k PANS simulation switch to a DNS approach only in the region where is needed or where it is possible, resulting in an averaged field similar to the URANS simulation, Fig. 3.

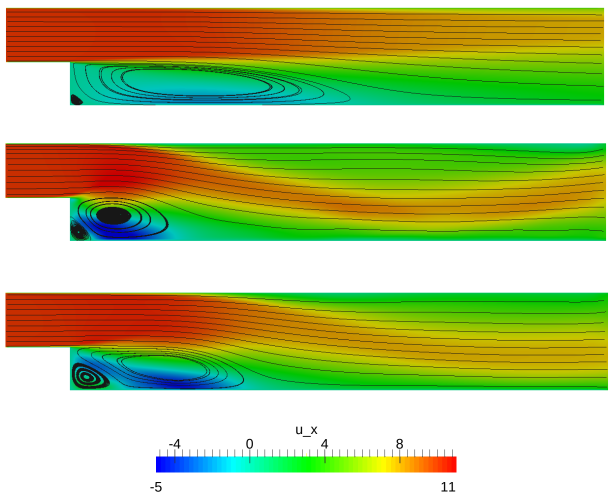

Figure 3: Averaged velocity field, streamwise component. From Top to bottom: URANS $k - \varepsilon$, fixed f_k PANS, varying f_k PANS.

An analysis performed on both meshes, coarse and fine grids, is needed to understand the behaviour of the solver. Figures 4 and 5 show the averaged and instantaneous f_k parameter, respectively.

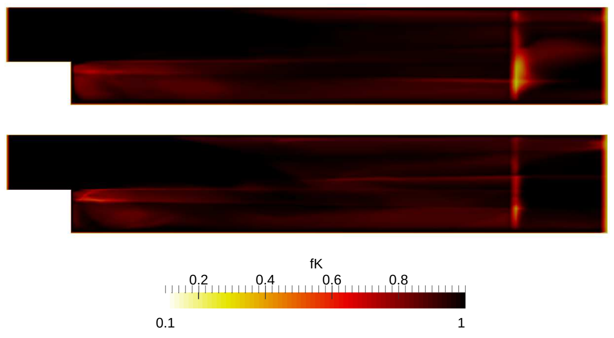

Figure 4: Averaged f_k for the PANS varying f_k simulation. From Top to bottom: coarse and fine mesh.

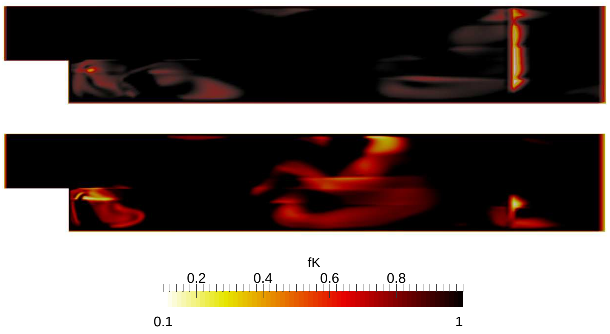

Figure 5: Instantaneous f_k for the PANS varying f_k simulation. From Top to bottom, coarse and fine mesh.

Refining the mesh, smaller Δ, Eq. 8, is obtained, therefore, the solver is allowed to solve more and smaller structures in the flow. The value of f_k decreases from coarse to fine grid, and the mixing of the flow increases, respectively, Fig. 6

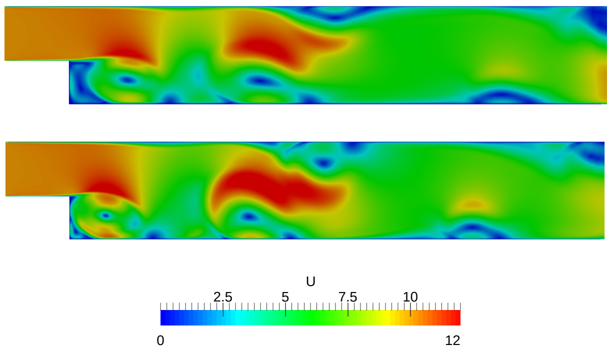

Figure 6: Averaged f_k for the PANS varying f_k simulation. From Top to bottom, coarse and fine mesh.

In Fig. 7, a comparison between fixed f_k simulation and varying f_k simulation is shown. The used mesh is very coarse compared to the previous example. The first simulation keeps the unsteadiness of the flow while the second simulation switches to a fully RANS simulation where no unsteadiness is seen in the flow.

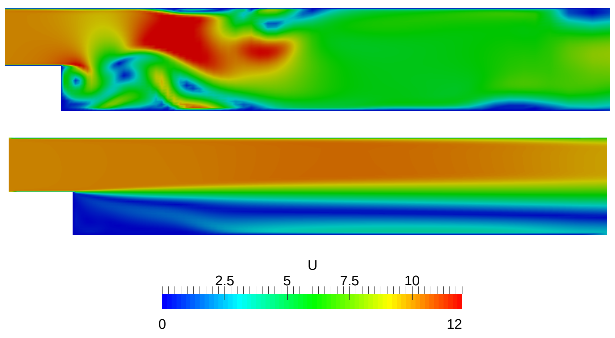

Figure 7: Instantaneous velocity for a very coarse mesh. From Top to bottom: fixed and varying f_k PANS simulaiton.

4.3 $k - \varepsilon - \zeta - f$ model

Starting from the previous implementation, some modifications are done to implement the four equations model. This model has to solve two more equations in order to describe the behaviour of the flow at the boundaries for a low Reynolds approach. First, the preliminary procedure that has been already described for the two equations model, must be done

- Open a new termminal
- Load openFoam OF23x
- Copy `kEpsilonPANS` turbulence model in a new directory naming it `kEpsZitaF`
- Rename the folder, and its .C and .H files, for example to `kEpsZitaF`
- Replace kEpsPANS with kEpsZitaF using the sed command

```
sed -i s/kEpsilonPANS/kEpsZitaF/g kEpsZitaF*
```

- Modify `Make/files` file substituting adding the following, to add the model to your library.

```
kEpsZitaF/kEpsZitaF.C
```

The modifications of the .C and .H file are described below.

4.3.1 kEpsZitaF.C

Two more parameters are added as `dimensioned<scalar>` after the constant `Cmu_`

```
C1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "C1",
        coeffDict_,
        0.4
    )
),
C2_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "C2",
        coeffDict_,
        0.65
    )
),
```

and three after the constant `fEpsilon_`

```
    CL_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "CL",
            coeffDict_,
            0.36
        )
    ),

    CTau_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "CTau",
            coeffDict_,
            6
        )
    ),
    Ceta_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "Ceta",
            coeffDict_,
            85
        )
    ),
```

and six more fields need to be created to add the new equations to the model. After the previously implemented C2U field, insert the following, paying attention that the constant `CEps1_` that now is a scalar field.

```
    ),
    CEps1_   //parameter in epsilon equation
```

```
(
    IOobject
    (
        "CEps1",
        runTime_.timeName(),
        mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_,
    dimensionedScalar("zero",0.0)
),

zitaU_   //parameter of zeta transport equation
(
    IOobject
    (
        "zitaU",
        runTime_.timeName(),
        mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_
),

    fU_   //parameter of poisson equation
(
    IOobject
    (
        "fU",
        runTime_.timeName(),
        mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_
),

    lU_   //Length scale parameter
(
    IOobject
    (
        "lU",
        runTime_.timeName(),
        mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_,
   dimensionedScalar("zero",dimLength,0.0)
),

    tU_   //Time scale parameter
(
    IOobject
    (
        "tU",
        runTime_.timeName(),
        mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_,
   dimensionedScalar("zero",dimTime,0.0)
),

   BfU_ //needed to bound fU
(
    IOobject
    (
        "BfU",
        runTime_.timeName(),
        mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
   mesh_,
   dimensionedScalar("zero",dimTime/(sqr(dimTime)),0.0001)

),
```

Now the definition of the introduced scale parameters and parameter `Ceps1_` are added before the member function section.

```
CEps1_=1.4*(1.0+0.045/(sqrt(zitaU_)));
cellVolume.internalField() = mesh_.V();
C2U = CEps1_ + (fK_/fEpsilon_)*(CEps2_-CEps1_);
tU_=max(kU_/epsilon_, CTau_*sqrt(nu())/sqrt(epsilon_));//time scale
lU_=CL_*max(pow(kU_,3.0/2.0)/epsilon_,
Ceta_*pow(nu()*nu()*nu()/epsilon_, 1.0/4.0)); //length scale
```

The epsilon equation is modified to

```
tmp<fvScalarMatrix> epsUEqn
(
    fvm::ddt(epsilonU_)
  + fvm::div(phi_, epsilonU_)
  - fvm::laplacian(DepsilonUEff(), epsilonU_)
 ==
    CEps1_*G*epsilonU_/kU_
  - fvm::Sp(C2U*epsilonU_/kU_, epsilonU_)
);
```

The two new equations are added to the model

```
    //Unresolved zita equation
    tmp<fvScalarMatrix> zitaUEqn
    (
        fvm::ddt(zitaU_)
      + fvm::div(phi_, zitaU_)
      - fvm::laplacian(DzitaUEff(), zitaU_)
     ==
      - fvm::Sp(G/kU_, zitaU_)
      + fU_
      + fvm::Sp(epsilonU_*(1.0-fK_)/kU_, zitaU_)
    );

    zitaUEqn().relax();
    zitaUEqn().boundaryManipulate(zitaU_.boundaryField());
    solve(zitaUEqn);
    bound(zitaU_, min(zitaU_)+0.0001); //bounding of zeta (dimensionless)
    Info << "zita done" << endl;//to check where the simulation stop

    //Unresolved fU equation
    tmp<fvScalarMatrix> fUEqn
    (
        (fvm::laplacian(fU_))
     -  fvm::Sp(1/sqr(lU_), fU_)
     -  1.0/(tU_*sqr(lU_))*(C1_+C2_*G/epsilonU_)*(zitaU_-2.0/3.0)
    );

    fUEqn().relax();
    fUEqn().boundaryManipulate(fU_.boundaryField());
    solve(fUEqn);
    bound(fU_, min(BfU_));//bounding of fU
    Info << "fU done" << endl; //to check where the simulation stop
```

and at each timestep, the new field parameters are updated after the already implemented f_k parameter.

```
    for (int i=0; i<fK_.size(); i++)
    {
      fK_[i]=(1/sqrt(Cmi_.value()))*(pow(pow(cellVolume[i],1.0/3.0)/
             (pow(k_[i],3.0/2.0)/epsilon_[i]),2.0/3.0));
      if (fK_[i]>1)
          fK_[i]=1;
      if (fK_[i]<0.1)
          fK_[i]=0.1;
    }
```

```
CEps1_=1.4*(1.0+0.045/(sqrt(zitaU_)));
C2U =CEps1_+ (fK_/fEpsilon_)*(CEps2_-CEps1_);

// updating scales
tU_=max(kU_/epsilon_, CTau_*sqrt(nu())/sqrt(epsilon_)); //time scale
lU_=CL_*max(pow(kU_,3.0/2.0)/epsilon_,
Ceta_*pow(nu()*nu()*nu()/epsilon_, 1.0/4.0));
Info << "Defining the new PANS k eps z f model" << endl;
```

4.3.2 kEpsZitaF.H

Here several modifications have to be done. Declaration of new constants and fields should be added in the protected data section.

```
// Model coefficients
    dimensionedScalar Cmi_;
    dimensionedScalar Cmu_;
    dimensionedScalar C1_; \\new
    dimensionedScalar C2_; \\new
    dimensionedScalar CEps2_;
    dimensionedScalar sigmaEps_;
    dimensionedScalar fEpsilon_;
    dimensionedScalar CL_; \\new
    dimensionedScalar CTau_; \\new
    dimensionedScalar Ceta_; \\new

// Fields
    volScalarField cellVolume;
    volScalarField fK_;
    volScalarField C2U;
    volScalarField CEps1_; \\new
    volScalarField zitaU_; \\new
    volScalarField fU_; \\new
    volScalarField lU_; \\new
    volScalarField tU_; \\new
    volScalarField BfU_; \\new
    volScalarField k_;
    volScalarField epsilon_;
    volScalarField kU_;
    volScalarField epsilonU_;
    volScalarField nut_;
```

After the effective diffusivity for ε, the effective diffusivity for ζ should be added. In this test case DzitaUEff is kept the same as the one for ε.

```
//- Return the effective diffusivity for unresolved zita
tmp<volScalarField> DzitaUEff() const
{
    return tmp<volScalarField>
    (
        new volScalarField("DzitaUEff",
        nut_/(fK_*fK_*sigmaEps_/fEpsilon_) + nu())
    );
}
```

To solve the equations with a low Reynolds number approach, proper boundary conditions for ζ and f should be set. In order to do this, new boundary conditions should be compiled and added to the dynamic library.

- Copy the folder *OpenFOAM−2.3.x/src/turbulenceModels/incompressible/RAS/...* *...derivedFvPatchFields/wallFunctions/epsilonWallFunctions* in your own RAS folder
- Change name of the folders and the files to *zitaUWallFunctions*
- Repeat the first two points for the fU equation boundary condition.
- Using the sed command substitute epsilon with zitaU or fU.

At this point we have four new boundary conditions, two for high and two for low Reynolds number approach. At this first stage, the boundary treatment for ζ is kept the same as for ε and the Low Reynolds boundary treatment for f is modified with a simple condition as described in Ref. [5].

```
        if (yPlus > yPlusLam_)
        {
            fU[cellI] = w*Cmu75*pow(k[cellI], 1.5)/(kappa_*y[faceI]);
        }
        else
        {
            fU[cellI] = -2.0*zitaU[cellI]*nuw[faceI]/sqr(y[faceI]);
        }
```

The previous implementation describes how f is calculated in the near wall region.

$$f_{wall} = \frac{-2\nu\zeta}{y^2} \tag{21}$$

4.4 Result and discussion

The same 2D backstep case is tested for the four equations model. A finer grid is required to test the low Reynolds number behaviour. In order to save time and have a first idea of how the model works, the Reynolds number of the simulation is reduced, by decreasing the inlet velocity by ten times. A fine and a coarse mesh are simulated, and a comparison of the two simulations is reported. In Fig. 8 and 9 the parameter f_k is presented.

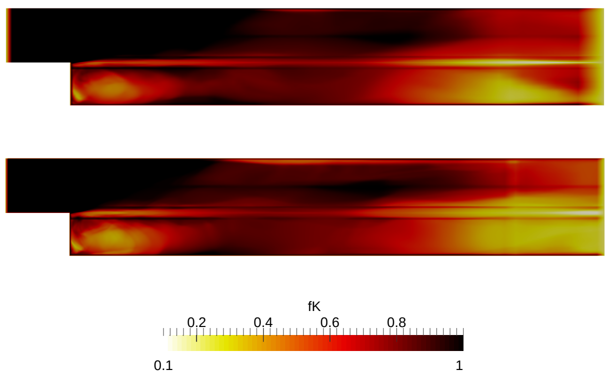

Figure 8: Averaged f_k for the $k - \varepsilon - \zeta - f$ model. From Top to bottom, coarse and fine mesh.

The coarse grid presents higher values of f_k especially at the end of the channel, where the cells are more stretched, and after the step where we have high turbulent kinetic energy. The refinement after the step allows lower value of f_k in order to solve more turbulence. All these consideration are in accordance with the definition of f_k in Eq. 8. By comparing this test case with the two equations model test case previously presented in Fig. 4, it is seen that more turbulence is resolved now, both because the grid is finer and the Reynolds number (connected with the level of kinetic energy) is lower. Fig. 9 shows a comparison between two simulation using different inlet velocity.

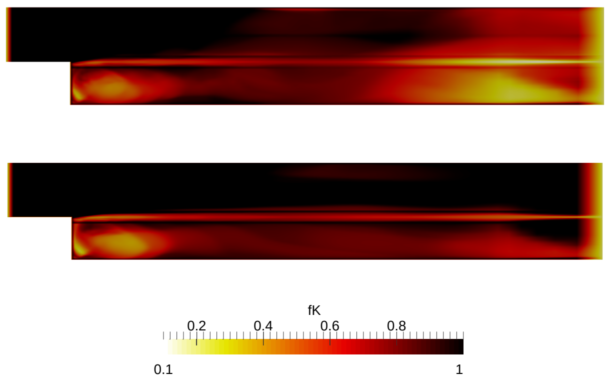

Figure 9: Averaged f_k for the $k - \varepsilon - \zeta - f$ model. From Top to bottom, low and high velocity. Coarse mesh.

Keeping the same grid, f_k increases by increasing the velocity at the inlet. Indeed, the mesh is not fine enough to resolve the scales, and the solver is more oriented to solve RANS equations. Over all f_k remains high because of the pour resolution of the case characterized by a high y^+ around the value of 20. Averaged and instantaneous flow fields are presented in Fig. 10 and 11, respectively.

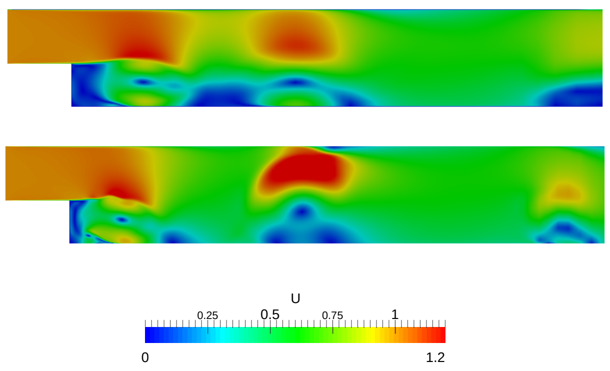

Figure 10: Instantaneous velocity for the $k - \varepsilon - \zeta - f$ model. From Top to bottom, coarse and fine mesh.

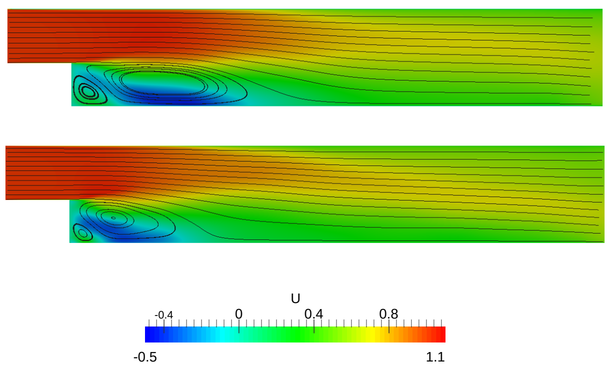

Figure 11: Averaged streamwise velocity for the $k - \varepsilon - \zeta - f$ model. From Top to bottom, coarse and fine mesh.

5 Post processing

In order to monitor a steady state simulation, it is useful to plot the residual, the iteration for each timestep and the maximum Courant number while the simulation is running. A gnuplot script `Residual` is printed below. The script can be executed using the command `gnuplot Residual` once it is placed in the case folder, and a log file of the simulation is recorded.

```
set term wxt 1
set logscale y
set title "Residuals"
set ylabel ’Residual’
set xlabel ’Timestep’
plot "< cat log | grep ’Solving for Ux’ | cut -d’ ’ -f13 | tr -d ’,’\\
" title ’Ux’ with lines,\
"< cat log | grep ’Solving for Uy’ | cut -d’ ’ -f13 | tr -d ’,’\\
" title ’Uy’ with lines,\
"< cat log | grep 'Solving for fU' | cut -d' ' -f13 | tr -d ','\\
" title 'fU' with lines,\
"< cat log | grep 'Solving for kU' | cut -d' ' -f13 | tr -d ','\\
"title 'kU' with lines,\
"< cat log | grep 'Solving for zitaU' | cut -d' ' -f13 | tr -d ','\\
"title 'Zita' with lines
pause 1

set term  wxt 2
set logscale y
set title "Iteration"
set ylabel 'Iteration'
set xlabel 'Timestep'
plot "< cat log | grep 'Solving for Ux' | cut -d' ' -f16 | tr -d ','"\\
 title 'Ux' with lines,\
"< cat log | grep 'Solving for Uy' | cut -d' ' -f16 | tr -d ','"\\
 title 'Uy' with lines,\
"< cat log | grep 'Solving for fU' | cut -d' ' -f16 | tr -d ','"\\
 title 'fU' with lines,\
"< cat log | grep 'Solving for kU' | cut -d' ' -f16 | tr -d ','"\\
 title 'kU' with lines,\
"< cat log | grep 'Solving for zitaU' | cut -d' ' -f16 | tr -d ','"\\
 title 'Zita' with lines
pause 1

set term  wxt 3
set logscale y
set title "Courant number"
set ylabel 'Courant Max'
set xlabel 'Timestep'
plot "< cat log | grep 'Courant' | cut -d' ' -f6" \\
title 'Max Courant number' with lines
pause 5

reread
```

The gnuplot script uses simple UNIX command to load the log file and extract what is required. In order to average quantities such as velocity, f_k or ζ, the following line should be added to the `system/controlDict` file.

```
functions
{
    fieldAve
    {
        type            fieldAverage;
        functionObjectLibs ("libfieldFunctionObjects.so");
        enabled         true;
        outputControl   outputTime;
        cleanRestart    False;
        timeStart       30;
        timeEnd         100;
        fields
        (
            U
            {
             mean       on;
             prime2Mean on;
             base       time;
            }
            fK
            {
             mean       on;
             prime2Mean on;
             base       time;
            }
            zitaU
            {
             mean       on;
             prime2Mean on;
             base       time;
            }
            fU
            {
             mean       on;
             prime2Mean on;
             base       time;
            }

        );

    }
}
```

One can post process the results by probing some lines in the domain, to easily extrapolate variable profiles with the following script called `sampleDict`. The script must be placed in `system` folder.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      sampleDict;
}
```

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


interpolationScheme cellPoint;
setFormat       raw;

// interpolationScheme.
interpolationScheme cellPointFace;
// Fields to sample.
fields
( U kU );
// Set sampling definition:
sets
(
        xOH
        {
                type midPoint;
                axis z;
                start (0 0 0);
                end (2 -2 2);
        }
        //...
        //More lines here//
        //...
);



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

6 Conclusion and future work

Two PANS turbulence models have been implemented and simulated using a 2D backstep test case. The PANS model reproduces the expected physic of the problem. In particular, the simulation is able to switch from RANS to DNS equations locally in the domain. Different configurations have been tested: coarse and fine meshes, and low and high inlet velocity. In all of them, the model resolves most of the turbulence where the grid is fine enough, and switches to RANS model where the grid is too coarse for the actual level of turbulence felt by a single cell. A low Reynolds number approach and a four equations model have also been simulated. The solver confirmed the same, already described, behaviour.

Considering the future work, a 3D case must be selected, in order to verify that the simulation reproduces the expected physical behaviour of the models. A validation of the model using experimental and LES data is needed to verify its features. Moreover, a deeper study of the low Reynolds number approach is necessary to implement the correct damping function for the new implemented variables. It is also interesting to calculate f_k by using the value of the kinetic energy of a previous RANS simulation. Thus, once f_k is calculated based on the RANS averaged kinetic energy, the model switches to the PANS equations calculating f_k with an always updated averaged value of k, as proposed in Ref. [6].

References

[1] S. Krajnović, R. Lárusson, and B. Basara, “Superiority of PANS compared to LES in predicting a rudimentary landing gear flow with affordable meshes,” *International Journal of Heat and Fluid Flow*, vol. 37, pp. 109–122, Oct. 2012.

[2] B. Basara, S. Krajnović, and S. Girimaji, “PANS methodology applied to elliptic-relaxation based eddy viscosity transport model,” *Turbulence and Interactions*, pp. 63–69, 2010.

[3] A. Asnaghi, O. Petit, and T. Lackmann, “interPhasChangeFoam Tutorial and PANS Turbulence Model.,” pp. 1–31, 2013.

[4] B. Basara, S. Krajnovic, S. Girimaji, and Z. Pavlovic, “Near-Wall Formulation of the Partially Averaged Navier Stokes Turbulence Model,” *AIAA Journal*, vol. 49, pp. 2627–2636, Dec. 2011.

[5] K. Hanjalić, M. Popovac, and M. Hadžiabdić, “A robust near-wall elliptic-relaxation eddy-viscosity turbulence model for CFD,” *International Journal of Heat and Fluid Flow*, vol. 25, pp. 1047–1051, Dec. 2004.

[6] S. Girimaji and K. Abdol-Hamid, “Partially Averaged NavierStokes Model for Turbulence: Implementation and Validation,” *AIAA paper*, no. January, 2005.