CFD with OpenSource software

Project work:

$\gamma - Re_\theta$ transitional turbulence model tutorial

Developed for OpenFOAM-2.0.x

Author:
Ayyoob Zarmehri

Peer reviewed by:
Nina Gall Jørgensen
Jelena Andric

November 19, 2012

Contents

Chapter 1

Introduction

This work describes the implementation of a newly published turbulent model into OpenFoam. The main advantage of this model is its capability to effectively take into account the important phenomenon "Laminar-Turbulent" transition in the boundary layer[1]. From physical point of view, initially a laminar boundary layer developes which due to increased instabilities of the flow, looses its stability and the transition to a turbulent boundary layer occurs. Ordinary RANS modelling cannot capture this phenomenon and they usually result in completely turbulent boundary layer. In this model, two extra transport equations are solved and coupled to the $k-\omega SST$ model and control the production term of turbulent kinetic energy in the k equation. In the literature, this new model is referred as $\gamma - Re_\theta$ model. In order to demonstrate the behavior of this model a flat plate test case is simulated by both the $k-\omega SST$ model and the $\gamma - Re_\theta$ model.

1.1 Flat plate test case

The geometry of the case is shown in Fig.1.1. The flat-plate test case used is one of those of European Research Community on Flow Turbulence and Combustion (ERCOFTAC) T3 series of flat-plate experiments [2] and [3]. This test case has zero pressure gradient in the free stream with specified free stream turbulent intensity (FSTI). The inlet conditions at 0.04 m upstream of the leading edge of the plate are summarized in table 1.1.

Table 1.1: Inlet conditions for flat-plate test case

Case	U inlet [m/s]	FSTI	$\frac{\mu_t}{\mu}$	ρ	μ
T3A	5.4	3.3	12.0	1.2	1.8 e-5

The computational domain of the simulation is shown in Fig.1.1. The inlet is located 0.04 m upstream of the leading edge of the plate. The boundary condition at the outlet is set to Neuman boundary condition, which corresponds to fully developed flow condition. This is not a realistic boundary condition for this setup, since the boundary layer continuously grows over the plate and there will be no fully developed flow. This approximated boundary condition is justified by the fact that the approximated information at this boundary can only be transported upstream by diffusion terms which are very weak compared to convective forces and hence affect the solution of a small number of upstream cells. For the top boundary a Neuman boundary condition with zero normal flux is specified. This condition is also an approximation of the reality. As the boundary layer develops on the plate, the fluid particles in this region have lack of momentum compared to fluid particles in the free stream. Since all cross sections normal to the plate are equal thus the continuity equation requires that free stream particles gain higher velocities compared to inlet conditions. In

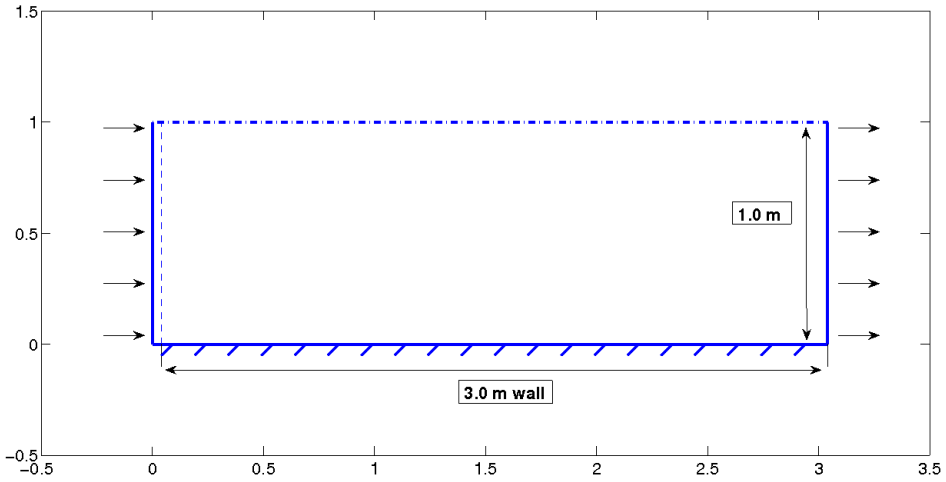

Figure 1.1: Computational domain for flat plate simulation

reality, however, this doesn't happen and streamlines go out of the box to make a larger cross section and thus there is outflow at the top boundary. Since the boundary layer is very thin, this error can be minimized by locating the top boundary at a large distance to the plate.

Chapter 2

SST simulation

This section explains how to make the case for the flat plate simulation with $k-\omega$ SST turbulence model, including mesh generation, creating appropriate fields and setting correct configuration. The starting point is to copy a similar tutorial to the `run` directory and make appropriate changes.

Copy the pitzDaily tutorial to the run directory.

```
run
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily .
mv pitzDaily flatplate
cd flatplate
```

The original pitzDaily tutorial uses $k-\epsilon$ turbulent model with wall function, while $\gamma - Re_\theta$ model doesn't use wall function for capturing the laminar and transitional boundary layers correctly and a y^+ of 1 is needed for the first node off the wall. The first task is to create the mesh.

2.1 Mesh generation

Block Mesh utility is used to create the mesh for the simulation. Modify the `blocMeshDict` by copy and pasting the following lines into `blocMeshDict` file:

```
convertToMeters 1;

vertices
(
    (0 0 0)
    (0.04 0 0)
    (0.08 0 0)
    (1.14 0 0)
    (3.04 0 0)
    (0 1 0)
    (0.04 1 0)
    (0.08 1 0)
    (1.14 1 0)
    (3.04 1 0)
    (0 0 0.1)
    (0.04 0 0.1)
    (0.08 0 0.1)
    (1.14 0 0.1)
    (3.04 0 0.1)
    (0 1 0.1)
    (0.04 1 0.1)
    (0.08 1 0.1)
    (1.14 1 0.1)
    (3.04 1 0.1)
);

blocks
(
    hex (0 1 6 5 10 11 16 15) (40 80 1) simpleGrading (0.0222 700 1)
    hex (1 2 7 6 11 12 17 16) (40 80 1) simpleGrading (45 700 1)
    hex (2 3 8 7 12 13 18 17) (160 80 1) simpleGrading (1 700 1)
    hex (3 4 9 8 13 14 19 18) (60 80 1) simpleGrading (13 700 1)
);

edges
(
);

boundary
(
    fixedWall
    {
        type wall;
        faces
        (
            (1 2 12 11)
            (2 3 13 12)
            (3 4 14 13)
        );
    }
    above
    {
        type patch;
        faces
```

```
        (
            (0 1 11 10)
        );
    }
    top
    {
        type patch;
        faces
        (
            (5 15 16 6)
            (6 16 17 7)
            (7 17 18 8)
            (8 18 19 9)
        );
    }

    inlet
    {
        type patch;
        faces
        (
            (0 10 15 5)
        );
     }
    outlet
    {
        type patch;
        faces
        (
            (4 14 19 9)
        );
     }

);

mergePatchPairs
(
);
```

where the boundary named `above` stands for the small `0.04 m` part of the domain upstream of the flat plate. The remaining boundaries are clear through their names.

Use `blocMesh` command in the root directory of flatplate to generate the mesh and then checkMesh utility to check the quality of the mesh:

```
blockMesh
checkMesh
```

2.2 Case setup

After generating the mesh, one can continue with creating the required fields related to the new mesh and boundary names. Since the names and types of boundaries of the new mesh are different from the original pitzDaily case, the easiest way to create the new field is to modify an existing field and then copy and rename it to suit for the remaining fields. Go to the `/0` directory and open the `U` field and replace the old lines with the following ones:

```
internalField   uniform (5.4 0 0);

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (5.4 0 0);
    }

    outlet
    {
        type            zeroGradient;
    }

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

    above
    {
        type            zeroGradient;
    }

    top
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

While staying in the `/0` directory, copy `/0/U` file to make the `/0/k` field :

```
cp U k
```

The `U` field is a `volVectorField` while the other fields are of the type `volScalarField`. The inlet boundary condition and for `k` needs to be set to `0.0475` which corresponds to a free stream turbulent intensity (FSTI) of `0.033` as in table 1.1.

```
sed -i s/volVectorField/volScalarField/g k
sed -i s/U/k/g k
sed -i s/"0 1 -1 0 0 0 0"/"0 2 -2 0 0 0 0"/g k
sed -i s/"(5.4 0 0)"/0.0475/g k
sed -i s/"(0 0 0)"/0/g k
```

Create the `omega` field from the `U` field and modify it by:

```
sed -i s/volVectorField/volScalarField/g omega
sed -i s/U/omega/g omega
sed -i s/"0 1 -1 0 0 0 0"/"0 0 -1 0 0 0 0"/g omega
sed -i s/"(5.4 0 0)"/264.6/g omega
sed -i s/"(0 0 0)"/9.26e05/g omega
```

The last line sets the boundary condition at the wall. Since no wall function will be used, one needs to set the Dirichlet boundary condition for `omega` at the wall which is dependent on the distance of first node off the wall.

The `nut` file should also be corrected. Replace the following lines with the old ones:

```
boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }
    outlet
    {
        type            calculated;
        value           uniform 0;
    }
    fixedWall
    {
        type            nutkWallFunction;
        value           uniform 0;
    }
    above
    {
        type            calculated;
        value           uniform 0;

    }
    top
    {
        type            calculated;
        value           uniform 0;

    }
    frontAndBack
    {
        type            empty;
    }
}
```

Modify the pressure field according to the new boundary names and set the type of all boundaries to `zeroGradient` except for the outlet which should be set to `fixedValue` of `uniform 0`.

Remove the unnecessary fields `epsilon` and `nuTilda` :

```
rm epsilon nuTilda
```

In `constant/transportProperties` set the value of `nu` to `1.5e-5` and in `constant/RASProperties` set the `RASModel` to `kOmegaSST`.

You also need to modify `system/fvSchemes` to specify what discretization scheme will be used for `omega` equation. Under `divSchemes` add:

```
    div(phi,omega) Gauss upwind;
```

and for `laplacianSchemes`:

```
    laplacian(DomegaEff,omega) Gauss linear corrected;
```

In `system/fvSolution` add the following under the solver :

```
    omega
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0.1;
    }
```

and under `relaxationFactors` add:

```
    omega           0.7;
```

now the case is ready to simulate. Run the case by typing :

```
simpleFoam
```

After some 90 iterations the solution converges. Wall shear stress is of primary concern in this simulation. That's where the difference between a laminar and a turbulent boundary layer becomes apparent. In order to compute the wall shear stress use the standard utility `wallShearStress`.

```
wallShearStress
```

You can see the results in `paraFoam` by :

```
paraFoam
```

Especially look at the turbulent kinetic energy profile and wall shear stress. In order to plot wall shear stress at the wall use the filter `Plot Data` and select `fixedWall` from `patches` under the `Display` tab. Fig. 2.1 shows the wall shear stress predicted by $k-\omega$ SST model.

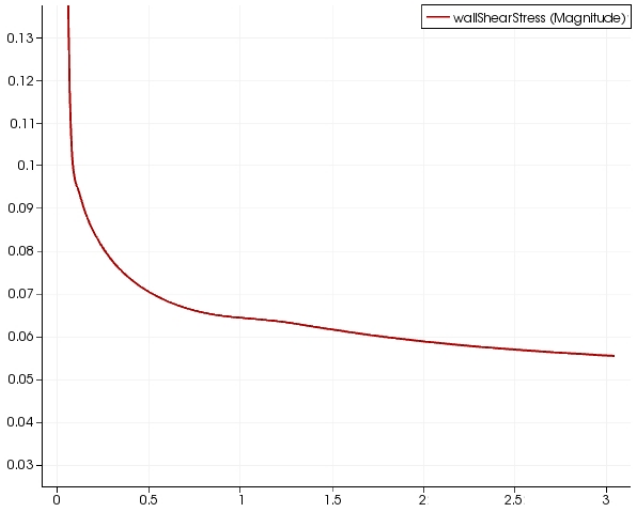

Figure 2.1: Wall shear stress for flat plate simulation

Chapter 3

Transition modelling

In this section the new transitional turbulence model will be described. However before describing the way to implement this new model, one issue needs to be addressed. In this model, for each cell a nonlinear algebraic equation needs to be solved. Thus we are facing the root finding problem. One aim of this project work is to describe the current root finding facilities available in OpenFoam and to add mored advanced methods to it.

3.1 Root finding problem

In OpenFoam 1.6.ext there are two classes for root finding problem, namely `BisectionRoot` and `RiddersRoot`. They are located in `src/ODE/findRoot` directory. Here the class `BisectionRoot` will be described.

This class has three data members which are :

- f_ : A refrence to the function class which needs to be solved.
- eps_ : The required accuracy of the solution.
- maxIter : The maximum number of iterations for finding the root

and one member function and a constructor.

The constructor of `BisectionRoot` accepts 2 argument, i.e. the reference function and the required accuracy to create an object of this class. Once an object of this class is created, the root of the object can be found by calling the member function `root` which requires two inputs, the two ends of the span which contains the root. In order to use these classes, the function class needs to be introduced. For `BisectionRoot` class, the function class only needs to have a method to return the value of the function when the input is given.

The `BisectionRoot` class, however is not an efficient way for finding the root,especially because one need to know the span in which the root lies. Also if there are multiple roots within that span, then it is not evident which root will be found. One of the objects of these work is to develop the root finding methods based on advanced Newton-Raphson method. The Newton-Raphson method requires both the value of the function to be solved and also the value of the differential of that function. Thus the function class needs to have appropriate member functions to return these values. The type of this function can be arbitrary. In this work the return value of the function is the one needed for the transitional-turbulent model.

In order to use these classes for general root finding, they need to be introduced via dynamic libraries. Copy the folder `findroot`, which comes with this work to the `run` directory (http://www.tfd.chalmers.se/ hani/kurser/OS_CFD/). Once inside this folder, type :

```
wmake libso
```

Now this library is ready to use for any root finding applications.

3.2 Transition model implementation

As it was previously mentioned, this model, solves two additional transport equations for determining the *intermittency* factor which varies between 0 and 1 where 0 stands for laminar region and 1 represents the turbulent region and a value between 0 and 1 determines the transition region. The transport equation for intermittency γ reads as [1] :

$$\frac{\partial \rho\gamma}{\partial t} + \frac{\partial \rho \upsilon_j \gamma}{\partial x_j} = P_\gamma - E_\gamma + \frac{\partial}{\partial x_i}[(\mu + \mu_t \sigma_f)\frac{\partial \gamma}{\partial x_j}] \tag{3.1}$$

The transition sources are defined as :

$$P_\gamma = 2F_{length}\rho S(\gamma F_{onset})^{0.5}(1-\gamma) \tag{3.2}$$

and the destruction source :

$$E_\gamma = 0.06\rho\Omega\gamma F_{turb}(50\gamma - 1) \tag{3.3}$$

where the two dimensionless functions F_{length} and F_{onset} control the length of the transition region and the location of the transition onset respectively. The transition onset is determined by the following equations:

$$F_{onset1} = \frac{Re_v}{2.193 Re_{\theta c}} \tag{3.4}$$

$$F_{onset2} = min(max(F_{onset1}, F_{onset1}^4), 2) \tag{3.5}$$

$$F_{onset3} = max(1 - (\frac{R_T}{2.5})^3, 0) \tag{3.6}$$

$$F_{onset} = max(F_{onset2} - F_{onset3}, 0) \tag{3.7}$$

In the above equations R_T is defined as :

$$R_T = \frac{\rho k}{\mu\omega} \tag{3.8}$$

where $Re_{\theta c}$ is the critical Reynolds number where the intermittency starts to increase in the boundary layer. Based on empirical studies, the following correlation functions are obtained that relates F_{length} and $Re_{\theta c}$ to transition Reynolds number $\tilde{R}e_{\theta t}$:

$$F_{length} = \begin{cases} 398.189 \cdot 10^{-1} + (-119.270 \cdot 10^{-4})\tilde{R}e_{\theta t} + (-132.567 \cdot 10^{-6})\tilde{R}e_{\theta t}^2, & \tilde{R}e_{\theta t} < 400 \\ 263.404 + (-123.939 \cdot 10^{-2})\tilde{R}e_{\theta t} + (-194.548 \cdot 10^{-5})\tilde{R}e_{\theta t}^2 + (-101.695 \cdot 10^{-8})\tilde{R}e_{\theta t}^3, & 400 \leq \tilde{R}e_{\theta t} < 596 \\ 0.5 - (\tilde{R}e_{\theta t} - 596.0) \cdot 3.0 \cdot 10^{-4}, & 596 \leq \tilde{R}e_{\theta t} < 1200 \\ 0.3188, & 1200 \leq \tilde{R}e_{\theta t} \end{cases}$$

$$Re_{\theta c} = \begin{cases} \tilde{R}e_{\theta t} - (-396.035 \cdot 10^{-2} - 120.656 \cdot 10^{-4}\tilde{R}e_{\theta t} + 868.230 \cdot 10^{-6}\tilde{R}e_{\theta t}^2 \\ \quad -696.506 \cdot 10^{-9}\tilde{R}e_{\theta t}^3 + 174.105 \cdot 10^{-12}\tilde{R}e_{\theta t}^4), & \tilde{R}e_{\theta t} \leq 1870 \\ \tilde{R}e_{\theta t} - (593.11 + (\tilde{R}e_{\theta t} - 187.0) \cdot 0.482), & \tilde{R}e_{\theta t} > 1200 \end{cases}$$

In order to correct the behavior of F_{length} for transition at high Reynolds number flows the following modifications are introduced:

$$F_{length} = F_{length}(1 - F_{sublayer}) + 40 \cdot F_{sublayer} \tag{3.9}$$

$$F_{sublayer} = e^{-(\frac{R_\omega}{0.4})^2} \tag{3.10}$$

$$R_\omega = \frac{\rho y^2 \omega}{500\mu} \tag{3.11}$$

For predicting separation induced transition the following modification is given:

$$\gamma_{sep} = min(2.0max(0, \frac{Re_v}{3.235Re_{\theta c}} - 1)F_{reattach}, 2)F_{\theta t} \tag{3.12}$$

$$F_{reattach} = e^{-(\frac{R_T}{20})^4} \tag{3.13}$$

The effective value of γ is thus obtained by the following:

$$\gamma_{effective} = max(\gamma, \gamma_{sep}) \tag{3.14}$$

The other equation of this model is a transport equation for transition momentum Reynolds number $\tilde{R}e_{\theta t}$ which reads as:

$$\frac{\partial \rho \tilde{R}e_{\theta t}}{\partial t} + \frac{\partial \rho \upsilon_j \tilde{R}e_{\theta t}}{\partial x_j} = P_{\theta t1} + \frac{\partial}{\partial x_i}[2.0(\mu + \mu_t)\frac{\partial \tilde{R}e_{\theta t}}{\partial x_j}] \tag{3.15}$$

This is a simple convection-diffusion equation with only one source term. The source term is intended to force $\tilde{R}e_{\theta t}$ to match the value of $Re_{\theta t}$ outside the boundary layer and is turned off inside the boundary layer, letting $\tilde{R}e_{\theta t}$ be simply diffused in from the free stream. The source term $P_{\theta t}$ is defined by the local difference of $\tilde{R}e_{\theta t}$ and $Re_{\theta t}$ and a blending function which reads as:

$$P_{\theta t} = 0.03\frac{\rho}{t}(Re_{\theta t} - \tilde{R}e_{\theta t})(1.0 - F_{\theta t}) \tag{3.16}$$

where t is a time scale defined by :

$$t = \frac{500\mu}{\rho U^2} \tag{3.17}$$

The blending function $F_{\theta t}$ reads as :

$$F_{\theta t} = min(max(F_{wake} \cdot e^{-(\frac{y}{\delta})^4}, 1.0 - (\frac{\gamma - 1/50}{1.0 - 1/50})^2), 1.0) \tag{3.18}$$

with the following parameters:

$$\delta = \frac{50\Omega y}{U} \cdot \delta_{BL} \qquad \delta_{BL} = \frac{15}{2}\theta_{BL} \qquad \theta_{BL} = \frac{\tilde{R}e_{\theta t}\mu}{\rho U} \tag{3.19}$$

$$F_{wake} = e^{-(\frac{Re_\omega}{1E+5})^2} \qquad Re_\omega = \frac{\rho \omega y^2}{\mu} \tag{3.20}$$

The function F_{wake} is added to make sure that the blending function is not active in the wake region.

The empirical correlation used in this model is based on the pressure gradient parameter and turbulent intensity defined as :

$$\lambda_\theta = \frac{\rho\theta^2}{\mu}\frac{dU}{ds} \tag{3.21}$$

$$Tu = 100\frac{\sqrt{2k/3}}{U} \tag{3.22}$$

with Re_θ defined as :

$$Re_{\theta t} = \frac{\rho \theta_t U_0}{\mu} \tag{3.23}$$

the following correlation equations are defined:

$$Re_{\theta t} = (1173.51 - 589.428Tu + \frac{0.2196}{Tu^2})F(\lambda_\theta) \qquad Tu \leq 1.3 \tag{3.24}$$

$$Re_{\theta t} = 331.50(Tu - 0.5658)^{-0.671}F(\lambda_\theta) \qquad Tu > 1.3 \tag{3.25}$$

$$F(\lambda_\theta) = 1 - (-12.986\lambda_\theta - 123.66\lambda_\theta^2 - 405.689\lambda_\theta^3)e^{-(\frac{Tu}{1.5})^{1.5}} \qquad \lambda_\theta \leq 0 \tag{3.26}$$

$$F(\lambda_\theta) = 1 + 0.275(1 - e^{-35.0\lambda_\theta})e^{-\frac{Tu}{0.5}} \qquad \lambda_\theta > 0 \tag{3.27}$$

For numerical robustness the following relations are imposed:

$$-0.1 \leq \lambda_\theta \leq 0.1 \qquad Tu \geq 0.027 \qquad Re_{\theta t} \geq 20 \tag{3.28}$$

The empirical correlation equations need to be solved iteratively since the momentum-thickness θ_t appears on both sides of the equations. The final output of this transition model is the γ_{eff} defined in Eq.(3.14), which controls the production and destruction term of k equation in the original $k-\omega SST$ model through the following equations:

$$\tilde{P_k} = \gamma_{eff}P_k \qquad \tilde{D_k} = min(max(\gamma_{eff}, 0.1), 1.0)D_k \tag{3.29}$$

where the P_k and D_k are the production and destruction term of the original $k-\omega$SST model and $\tilde{P_k}$ and $\tilde{D_k}$ are the corrected values. Some modifications is also made to the blending function F_1 as bellow:

$$F_1 = max(F_{1original}, F_3) \qquad F_3 = e^{-(\frac{R_y}{120})^8} \qquad R_y = \frac{\rho y\sqrt{k}}{\mu} \tag{3.30}$$

Since this model is coupled to $k-\omega SST$ model, the easiest way is to make modifications to the SST model by first copying the source of SST model and renaming :

```
cd $WM_PROJECT_DIR
cp -r --parents src/turbulenceModels/incompressible/RAS/kOmegaSST $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/turbulenceModels/incompressible/RAS
mv kOmegaSST SSTtransition
cd SSTtransition
```

rename the files by one of the following commands :

```
rename s/kOmegaSST/SSTtransition/ *
rename kOmegaSST SSTtransition *
```

and inside these files rename all the occurances of `kOmegaSST` to `SSTtransition` by :

```
sed -i s/kOmegaSST/SSTtransition/g SSTtransition.C
sed -i s/kOmegaSST/SSTtransition/g SSTtransition.H
```

We also need to make the `Make` directory:

```
mkdir Make
```

and then create `Make/files` :

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

and also `Make/options` :

```
EXE_INC = \
-I$(LIB_SRC)/turbulenceModels \
-I$(LIB_SRC)/transportModels \
-I$(LIB_SRC)/finiteVolume/lnInclude \
-I$(LIB_SRC)/meshTools/lnInclude \
-I$(LIB_SRC)/turbulenceModels/incompressible/RAS/lnInclude \
-I$(FOAM_RUN)/findroot
LIB_LIBS = \
    -L$(FOAM_USER_LIBBIN)\
    -lfindroot
```

note that we have added `-I$(FOAM_RUN)/findroot` directory where the file `findRoot.H` is located and also the path and the name of our previously created library `findroot`.

Now we need to add the extra equations and variables to `SSTtransition.H` and `SSTtransition.C` in a step by step manner. In the declaration file `SSTtransition.H` do the following:

first include the `findRoot.H` by :

```
#include "findRoot.H"
```

after `#include "wallDist.H"`. And then declare the new variables `im_`, for intermittency factor, and `Ret_` for transition Reynolds number and `Reo_`in the protected data member of SSTtransition class after declaration of `nut_` :

```
volScalarField im_;
volScalarField Ret_;
volScalarField Reo_;
```

In order to avoid division by zero in some of the functions needed by this model, the following dimensioned scalar values are introduced, which assume small numbers. In the protected data members after `dimensionedScalar c1_;` declare :

```
dimensionedScalar minvel;
dimensionedScalar mindis;
dimensionedScalar mintim;
```

This model needs the free stream value of the velocity and the easiest way for defining the free stream velocity value is through a defined dictionary. In this project a dictionary named `transitionProperties` which is located in the `/constant` directory is read for the free stream value. For now this dictionary contains only the free stream value, but in the future other constants might be added. Thus you need to create `transitionProperties` dictionary under `/constant` directory of each case you want to run with this model with following format (In our flat plate simulation it takes the value of `5.4`):

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.0.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transitionProperties;
}
```

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

freeStreamU              freeStreamU [ 0 1 -1 0 0 0 0 ] 5.4;
```

Define the dictionary and the free stream variable after `dimensionedScalar mintim;` :

```
    IOdictionary transitionProperties;
    dimensionedScalar freeStreamU;
```

In `SSTtransition.H` add the following member functions for returning the effective diffusitivity of these 2 new equations. After the member function `DomegaEff(const volScalarField& F1) const` add :

```
    //- Return the effective diffusivity for Ret
    tmp<volScalarField> DRetEff() const
    {
        return tmp<volScalarField>
        (
            new volScalarField("DRetEff", 2.0*(nut_+nu()))
        );
    }
    //- Return the effective diffusivity for intermittency
    tmp<volScalarField> DimEff() const
    {
        return tmp<volScalarField>
        (
            new volScalarField("DimEff", nut_+nu())
        );
    }
```

There are other member functions for calculating the source terms and some other parameters which are provided in the external file `additional.H`. You need to copy this file (which comes with this tutorial) to the directory of `SSTtransition.H` and include this file by adding the following line at the end of member functions declaration:

```
#include "additional.H"
```

That is all the changes we needed to do in `SSTtransition.H`. Now in `SSTtransition.C` define the variables declared previously in `SSTtransition.H`. Thus in the member-initializer part of the class constructor in `SSTtransition.C` after c1_ add :

```
minvel
(
    dimensionedScalar
    (
         "minvel",
         dimensionSet( 0, 1, -1, 0, 0, 0, 0),
         0.000001
    )
),
mindis
(
    dimensionedScalar
    (
          "mindis",
          dimensionSet( 0, 1, 0, 0, 0, 0, 0),
          0.000001
```

```
        )
    ),
    mintim
    (
        dimensionedScalar
        (
              "mintim",
              dimensionSet( 0, 0, -1, 0, 0, 0, 0),
              0.000001
        )
    ),
    transitionProperties
     (IOdictionary
         (
         IOobject
         (
             "transitionProperties",
             runTime_.constant(),
             mesh_,
             IOobject::MUST_READ_IF_MODIFIED,
             IOobject::NO_WRITE
         )
      )
     ),
     freeStreamU
     (dimensionedScalar
       (
        transitionProperties.lookup("freeStreamU")
       )
     ),
```

also add the following in the member-initializer part after `nut_` in `SSTtransition.C`. In order to do so, add a comma after `nut_` defenition and add:

```
   im_
    (
        IOobject
        (
            "im",
            runTime_.timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_
    ),
    Ret_
    (
        IOobject
        (
            "Ret",
            runTime_.timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
```

```
        ),
        mesh_
    ),
    Reo_
    (
        Ret_
    )
```

Before solving the transport equations, the transition onset momentum-thickness Reynolds number (based on freestream conditions) from Eq.s 3.23-3.24, which in the code is denoted by `Reo_`, needs to be solved by the root finding methods developed earlier. Thus in `SSTtransition.C`, in the member function `SSTtransition::correct()` after `nut_.correctBoundaryConditions()`(which re-calculates viscosity) add the following lines :

```
//   transition model
     volScalarField nufield=nu();
     volScalarField Duds=findGrad(U_); //velocity gradient along a streamline
     scalar Umag; // absolute value of the velocity
     scalar Tu;   // Turbulent intensity
     rootFunction tf(1,2,3,4);  // creating the function object
        forAll(k_, cellI)
        {
         Umag=max(mag(U_[cellI]),0.00000001); // avoiding division by zero
         Duds[cellI]=max(min(Duds[cellI],0.5),-0.5); // bounding DUds for robustness
         Tu=100.0*sqrt(2.0/3.0*k_[cellI])/Umag;
         tf.modify(Tu,Duds[cellI],nufield[cellI],freeStreamU.value()); // modifying the object
         Reo_[cellI]=NewtonRoot<rootFunction>(tf, 1e-5).root(0.0); // solving the nonlinear equation
         Reo_[cellI]=Reo_[cellI]*freeStreamU.value()/nufield[cellI];
        }
     Reo_=max(Reo_,20.0); //limiting Reo_ for robustness
```

Now the transport equation for transition onset momentum-thickness can be solved. Simply add the following after previous lines :

```
    // Re_theta  equation
    tmp<fvScalarMatrix> ReEqn
    (
        fvm::ddt(Ret_)
      + fvm::div(phi_, Ret_)
      - fvm::Sp(fvc::div(phi_), Ret_)
      - fvm::laplacian(DRetEff(), Ret_)
     ==
        Ptheta()*Reo_
      - fvm::Sp(Ptheta(), Ret_)
    );
    ReEqn().relax();
    solve(ReEqn);
```

where the source term of this equation is calculated by calling the member function `Ptheta()`. To implement the correlation formulas for `F_length` and `Re_critical`, add the following lines after previous lines:

```
volScalarField Re_crit(Ret_);
volScalarField F_length(Ret_);
```

```
forAll(Ret_, cellI)
{
  if (Ret_[cellI]<=1870.0)
     { Re_crit[cellI]=Ret_[cellI]-396.035*1.0e-02+120.656*1.0e-04*Ret_[cellI]
                     -868.230*1.0e-06*Foam::pow(Ret_[cellI],2.0)
                     +696.506*1.0e-09*Foam::pow(Ret_[cellI],3.0)
                     -174.105*1.0e-12*Foam::pow(Ret_[cellI],4.0);
     }
  else
     { Re_crit[cellI]=Ret_[cellI]-593.11-0.482*(Ret_[cellI]-1870.0);}

  if (Ret_[cellI]<400.0)
    {F_length[cellI]=398.189*1.0e-01-119.270*1.0e-04*Ret_[cellI]
                    -132.567*1.0e-06*Foam::pow(Ret_[cellI],2.0);}
 else if (Ret_[cellI]>=400.0 && Ret_[cellI]<596.0)
    {F_length[cellI]=263.404-123.939*1.0e-02*Ret_[cellI]
                    +194.548*1.0e-05*Foam::pow(Ret_[cellI],2.0)
                    -101.695*1.0e-08*Foam::pow(Ret_[cellI],3.0);}
 else if (Ret_[cellI]>=596.0 && Ret_[cellI]<1200.0)
    {F_length[cellI]=0.5-(Ret_[cellI]-596.0)*3.0*1.0e-04;    }
 else
    {F_length[cellI]=0.3188;}
}
```

and then add the following lines for solving the intermittency equation `im_` :

```
    volScalarField F_sub=(Foam::exp(-Foam::pow(sqr(y_)*omega_/(500.0*nu()*0.4),2.0)));
    F_length=F_length*(1.0-F_sub)+40.0*F_sub;
    const volScalarField vor(sqrt(2*magSqr(skew(fvc::grad(U_)))));
    volScalarField Pr1(2.0*F_length*sqrt(im_*F_onset(Re_crit))*sqrt(S2));
    volScalarField Pr2(0.06*vor*F_turb()*im_);
    // intermittency equation
    tmp<fvScalarMatrix> imEqn
    (
        fvm::ddt(im_)
      + fvm::div(phi_, im_)
      - fvm::Sp(fvc::div(phi_), im_)
      - fvm::laplacian(DimEff(), im_)
     ==

        Pr1
       +Pr2
      - fvm::Sp(Pr1,im_)
      - fvm::Sp(Pr2*50.0,im_)

    );
    imEqn().relax();
    solve(imEqn);
    volScalarField sep_im=separation_im(Re_crit);
    im_=max(im_,sep_im);//correction for separation induced transition
    im_=min(max(im_,0.00001),1.0); //bounding intermittency
```

There are still some modifications that need to be introduced. The production and destruction term of turbulent kinetic energy equation should be modified by replacing the following lines of the `k` equation:

```
        min(G, c1_*betaStar_*k_*omega_)
      - fvm::Sp(betaStar_*omega_, k_)
```

with :

```
        min(G, c1_*betaStar_*k_*omega_)*im_
      - fvm::Sp(min(max(im_,0.1),1.0)*betaStar_*omega_, k_)
```

The blending function `SSTtransition::F1(const volScalarField& CDkOmega) const` should also be modified by replacing the original return value of this function by :

```
volScalarField R_y=y_*sqrt(k_)/nu();
volScalarField F3_y=Foam::exp(-Foam::pow(R_y/120.0,8.0));
return max(tanh(pow4(arg1)),F3_y);
```

Compile the new model by :

```
wmake libso
```

3.3 Flat plate simulation

In this section it will be shown how to use this model for the `flatplate` case which was already solved by $k-\omega SST$ model. In order to use this model the two new fields, `im` and `Ret`, needs to be created in `0/` directory. In `0/` directory of the flat plate case do the followings :

```
cp k Ret
sed -i s/k/Ret/g Ret
sed -i s/"0 2 -2 0 0 0 0"/"0 0 0 0 0 0 0"/g Ret
cp k im
sed -i s/k/im/g im
sed -i s/"0 2 -2 0 0 0 0"/"0 0 0 0 0 0 0"/g im
```

The inlet boundary condition for `Ret` should be calculated by Eq.s 3.24-3.28 based on zero velocity gradient and free stream turbulent velocity which in this case becomes 171. The boundary condition at the wall is `zeroGradient` . Open the file in a text editor and make appropriate changes with internal field set to a number near the inlet boundary condition.

For intermittency equation, the boundary condition at the inlet is equal to 1 and `zeroGradient` at the wall. Open `0/im` and make appropriate changes. Set the internal field to 1.

In `system/controlDict` make sure that the libraries `findroot` and `myIncompressibleRASModels` are included by adding this line :

```
libs ("libmyIncompressibleRASModels.so" "libfindroot.so");
```

to the dictionary. Since this model is sensitive to the choice of discretization scheme, the following schemes are used for `divSchemes` in `system/fvSchemes`:

```
    div(phi,U)      Gauss vanLeerV;
    div(phi,k)      Gauss upwind;
    div(phi,k)      Gauss MUSCL;
    div(phi,omega) Gauss MUSCL;
    div(phi,Ret) Gauss  linear;
    div(phi,im) Gauss linear;
```

You also need to add the following for `laplacianSchemes`:

```
    laplacian(DRetEff,Ret) Gauss linear corrected;
    laplacian(DimEff,im) Gauss linear corrected;
```

In `system/fvSolution` add the following under the solver :

```
    Ret
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0.1;
    }
    im
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-05;
        relTol          0.1;
    }
```

and under `relaxationFactors` add:

```
    im          0.7;
    Ret         0.7;
```

In `constant/RASProperties` make sure you have selected `SSTtransition` as the RASModel.

Run the case by typing `simpleFoam`.

Post-process the results in paraFoam, but first compute the wall shear stress by :

```
wallShearStress
paraFoam
```

Now look at the turbulent kinetic energy profile and wall shear stress. Figure 2.1 shows the wall shear stress along the plate. Compare Fig. 3.1 with Fig. 2.1. The wall shear stress clearly shows that at first a laminar boundary layer is developed on the plate and then through the transition process (which shows itself by the increase in wall shear stress at the interval x=[0.36 0.65]), the boundary layer becomes turbulent. The following figure demonstrates this :

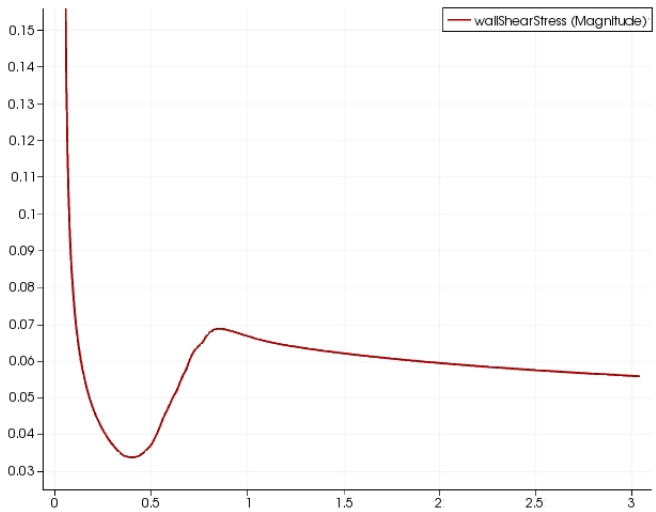

Figure 3.1: Wall shear stress for flat plate simulation

Bibliography

[1] F.R. Menter. Correlation-based transition modeling for unstructured parallelized computational fluid dynamics codes,*AIAA Journal*,47(12), 2009.

[2] Savill, A. M., Some Recent Progress in the Turbulence Modeling of By-Pass Transition*Near-Wall Turbulent Flows* edited by R. M. C. So, C. G. Speziale, and B. E. Launder, Elsevier, NewYork, 1993, p. 829.

[3] Savill, A. M., One-Point Closures Applied to Transition, Turbulence and Transition Modeling, edited by M. Hallbäck, Kluwer, Dordrecht,The Netherlands, 1996, pp. 233−268.