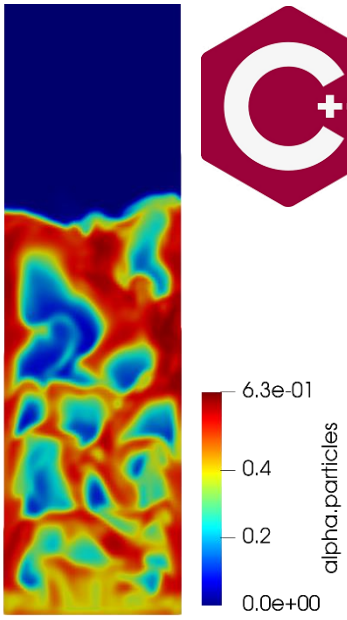
CEMF.ir
Center of Engineering Multiscale Modeling Fluid Flow

Mastering twoPhaseEulerFoam

Two: Extending the solver



How to add new physical models to the standard solver:

Drag model & Turbulence model

Compatible with	OpenFOAM® 7, OpenFOAM® 6 OpenFOAM® v1912

Author

Hamidreza Norouzi

Amirkabir University of Technology

Center of Engineering and Multiscale Modeling of Fluid Flow

Extra consideration:

- This document is developed to teach how to use OpenFOAM® software. The document has gone under several reviews to reduce any possible errors, though it may still have some. We will be glad to receive your comments on the content and error reports through this address: h.norouzi@aut.ac.ir

Document history

Revision	Description	Date
Rev1.1	Tutorial was rechecked and run by OpenFOAM 6 and some parts were added, Contributor: Mohammad Zareie.	May 20, 2020
Rev1	Errors corrected and codes compiled and tested with OpenFoam v1912.	April 30, 2020
Rev0	The first draft was prepared and run with OpenFOAM 7.	April 19, 2020

Table of Contents

Prerequisites

You need to be familiar with:

- twoPhaseEulerFoam solver and basics of OpenFoam® to start this tutorial. You are encouraged to read the tutorial One (One: Fluidized bed) of this series.
- C++ programming, specially the concepts of class, class templates, inheritance, and macros to understand code lines presented here.

How to get simulation setup files?

You will have access to the C++ codes and simulation test setups for this tutorial. You can download these files (a compressed file) from www.cemf.ir alongside this tutorial file.

1. What is the motivation

The solver twoPhaseEulerFoam solves a complete set of compressible balance equations including continuity, momentum and energy equations for the two phase system: with one phase as dispersed phase. The model equations are closed by using closure relations for drag force, heat transfer coefficients, turbulence models, and other sub-models.

A set of conventional sub-models are already implemented in the solver (see other tutorials on this series). You always face times when we want to test the solver outputs with new sub-models. Here, you are going to learn how to add these new sub-models to the existing ones: a new drag model in section 2 and a turbulence model in section 3.

2. How to add new drag force model to twoPhaseEulerFoam

As mentioned in part One of this series, a set of drag force models are already implemented in the solver: `Gibilaro`, `WenYu`, `Ergun`, `GidospowWenYu` and etc. Here, you add the correlation of Di-Felice (which is suitable for fluid-solid systems) to this solver [Di Felice, 1994. International Journal of Multiphase Flow, 20, 153-159.]:

$$K = 0.75\ \frac{\mu_c}{d_d^2}(1-\alpha_c)\,.CdRe$$

$$CdRe = Re.\,\alpha_c C_D \alpha_c^{-\chi}$$

$$\chi = 3.7 - 0.65\ e^{-0.5\left(1.5-log10(Re)\right)^2}$$

$$C_D = (0.63 + 4.8\ Re^{-0.5})^2$$

$$Re = \frac{\alpha_c.\rho_c\,|u_c-u_d|\,d_d}{\mu_c}$$

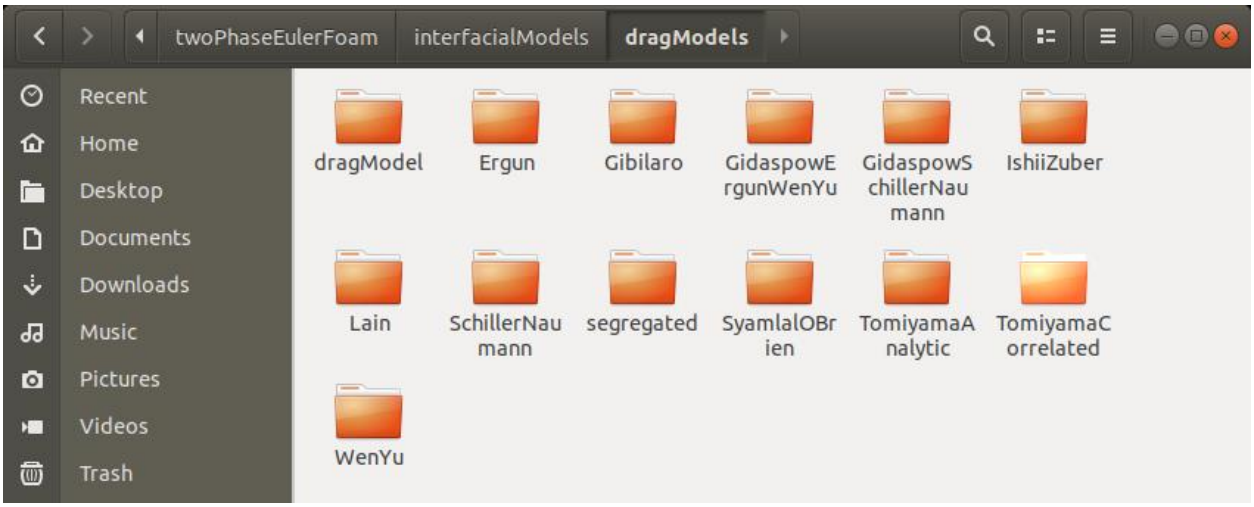
where *K* is the interphase momentum transfer coefficient, and subscripts *c* and *d* refer to continuous and discrete phases, respectively. For any new drag model, you only need to provide a function for *CdRe* in the code.

2.1. Preparing folders and files

Before you start implementing the new drag force, you need to have a look at the code to see how the current drag force models are implemented. All the interfacial models (including drag force, lift force, heat transfer and etc.) are implemented in folder:

`$FOAM_SOLVERS/multiphase/twoPhaseEulerFoam/interfacialModels`

where `$FOAM_SOLVERS` is the address of OpenFOAM® solvers folder on your computer. In sub-folder dragModels you will find all the implemented models.



To implement a new drag model, start with a model which is the most similar to the new model. Here, Wen & Yu model is very similar to Di-Felice model. Suppose that you want to implement new this new model, and possibly other models, in the user directory of OpenFOAM, which is created during the normal installation of the software. Note that any other location/folder on your computer with write-access can be selected. In the terminal, execute the following commands:

```
> cd $WM_PROJECT_USER_DIR
> mkdir twoPhaseEulerFoamModels
```

These two commands direct you to user directory of OpenFOAM® and creates a new folder with name twoPhaseEulerFoamModels. With the following command, the folder WenYu is copied to this folder and is renamed to DiFelice.

```
>    cp    $FOAM_SOLVERS/multiphase/twoPhaseEulerFoam/interfacialModels/dragModels/WenYu
$WM_PROJECT_USER_DIR/twoPhaseEulerFoamModels/DiFelice -r
```

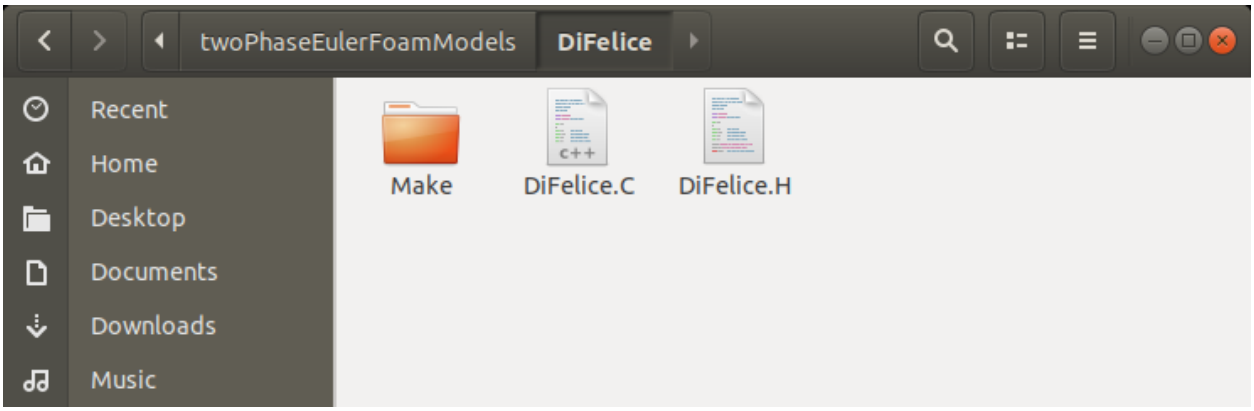
Go to sub-folder DiFelice and rename the existing files with the following commands:

```
> cd DiFelice
> mv WenYu.C DiFelice.C -v
> mv WenYu.H DiFelice.H -v
```

In this sub-folder you need to create another sub-folder with name “Make” that contains the required information for compiling and making the new library.

```
> mkdir Make
```

DiFelice folder now should look like this:



In this stage, you are ready to make the required changes to the files to prepare everything for compilation.

2.2. Changes to source files

The simplest way, and of course the safest way, is to start with the existing code and make the required modifications. Open header file DiFelice.H with any text editor on your computer and replace every occurrence of the word WenYu with DiFelice. Your header file and class declaration should look like this after this replacement:

twoPhaseEulerFoamModels/DiFelice/DiFelice.H

```
#ifndef DiFelice_H
#define DiFelice_H

#include "dragModel.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

class phasePair;

namespace dragModels
{

/*---------------------------------------------------------------------------*\
                           Class DiFelice Declaration
\*---------------------------------------------------------------------------*/

class DiFelice
:
    public dragModel
{
    // Private Data

        //- Residual Reynolds Number
        const dimensionedScalar residualRe_;


public:

    //- Runtime type information
    TypeName("DiFelice");


    // Constructors

        //- Construct from a dictionary and a phase pair
        DiFelice
        (
            const dictionary& dict,
            const phasePair& pair,
            const bool registerObject
        );


    //- Destructor
    virtual ~DiFelice();


    // Member Functions

        //- Drag coefficient
        virtual tmp<volScalarField> CdRe() const;
};
```

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

} // End namespace dragModels
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif
```

All you need to do are: 1) defining any model-specific variables in the private data section, which is residualRe_ in this case; 2) providing a definition for the class constructor in the source file DiFelice.C; and 3) defining (overriding) `CdRe()` method in the source file DiFelice.C, which calculates the drag coefficient field.

Open DiFelice.C and make the same word replacement (WenYu -> DiFelice) and modify the body of `CdRe()` method similar to what you see in the followings:

twoPhaseEulerFoamModels/DiFelice/DiFelice.C

```
#include "DiFelice.H"
#include "phasePair.H"
#include "addToRunTimeSelectionTable.H"

// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
namespace dragModels
{
    defineTypeNameAndDebug(DiFelice, 0);
    addToRunTimeSelectionTable(dragModel, DiFelice, dictionary);
}
}


// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::dragModels::DiFelice::DiFelice
(
    const dictionary& dict,
    const phasePair& pair,
    const bool registerObject
)
:
    dragModel(dict, pair, registerObject),
    residualRe_("residualRe", dimless, dict)
{}


// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * //
```

```
Foam::dragModels::DiFelice::~DiFelice()
{}


// * * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField> Foam::dragModels::DiFelice::CdRe() const
{
    volScalarField alpha2
    (
        max(scalar(1) - pair_.dispersed(), pair_.continuous().residualAlpha())
    );

    volScalarField Res(max(alpha2*pair_.Re(), residualRe_));

    volScalarField Cds
    (
        pow( 0.63 + 4.8 * pow(Res, -0.5) , 2)

    );

    return
        Res*Cds
       *pow(alpha2, -(3.7-0.65*exp(-0.5*pow(1.5-log10(Res),2))) )
       *max(pair_.continuous(), pair_.continuous().residualAlpha());
}
```

`defineTypeNameAndDebug` and `addToRunTimeSelectionTable` are some C++ macros that are defined elsewhere. `defineTypeNameAndDebug` defines run-time type information to be used by OpenFOAM run-time selectors. Macro `addToRunTimeSelectionTable`, adds this new class, `DiFelice`, to the list of selection table of the base class `dragModel`, so OpenFOAM can find and create the newly added drag force model (here, DiFelice) in the solver when you select this model in your simulation. A detailed description on virtual constructors and run-time selection tables can be found on openFoamWiki website by David Gaden.

Note

A nice description of run-time selection mechanism can be found in the following address:

https://openfoamwiki.net/index.php/OpenFOAM_guide/runTimeSelection_mechanism

Constructor `DiFelice::DiFelice` reads the keyword "residualRe" from the drag dictionary that is provided by user in the phaseProperties file. Method `DiFelice::CdRe` calculates the drag coefficient based on the Di-Felice model. It first calculates the volume fraction of dispersed phase and then calculates the Reynolds number. In this model, phase Reynolds number is obtained by multiplying local Reynolds number, $u_{continuous}d_{disperssed}/v_{continuous}$, by the fluid volume fraction. Next steps are calculating single particle drag force, `Cds`; calculating dispersed phase drag force; and returning the result.

2.3. Make folder

You must have two files in Make folder: "files" and "options". Create a new document in your text editor and copy the following lines into it and save it as "files" in the Make folder.

twoPhaseEulerFoamModels/DiFelice/Make/files

```
DiFelice.C

LIB = $(FOAM_USER_LIBBIN)/libDiFeliceDragForceModel
```

The first line gives the name of source files to be compiled to the compiler. The last line, gives the name of library that you create, `libDiFeliceDragForceModel`. The compiler creates a library with this name, if the compilation and linking were successful.

Create a new document in your text editor and copy the following lines into it and save it as "options" in the Make folder.

twoPhaseEulerFoamModels/DiFelice/Make/options

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(FOAM_SOLVERS)/multiphase/twoPhaseEulerFoam/interfacialModels/lnInclude \
    -I$(FOAM_SOLVERS)/multiphase/twoPhaseEulerFoam/twoPhaseSystem/lnInclude  \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude

LIB_LIBS = \
    -lfiniteVolume \
    -lcompressibleEulerianInterfacialModels \
    -lcompressibleTwoPhaseSystem
```

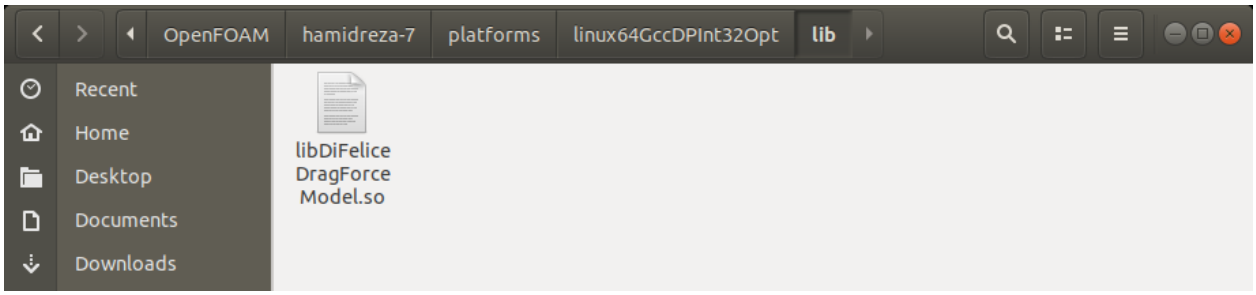
In `EXE_INC` line, you define the paths to folders containing header files that are required for compilation and in `LIB_LIBS` line, you define the name of libraries that are required for the linking and building steps.

2.4. Build the library for the new drag model

Execute the following command from twoPhaseEulerFoamModels/DiFelice to build your library.

```
> wmake
```

If everything is done correctly and the compilation returns no error, wmake creates your library in the lib sub-folder in the user folder of OpenFOAM®.



2.5. Simulating the fluidized bed with the new drag model

You can use this new model in the fluidized bed simulation that you performed in part One (find tutorial mastering twoPhaseEulerFoam, One: Fluidized bed, on www.cemf.ir). Alternatively, you can just use the new model on one of the standard tutorial cases of OpenFOAM. For example, you can find one in `$FOAM_TUTORIALS/multiphase/twoPhaseEulerFoam/RAS/fluidisedBed`.

Two modifications are required: including new library into the simulation and selecting new drag model in setup files.

Open system/controlDict and add the following lines to it and save it:

```
libs
(
    "libDiFeliceDragForceModel.so"
);
```

Open constant/phaseProperties and in drag dictionary just select `DiFelice` drag model as follows and save it.

```
drag
(
    (particles in air)
    {
        type            DiFelice;
        residualRe      1e-3;
        swarmCorrection
        {
            type        none;
        }
    }
);
```

Now you can normally perform simulation with the new drag model:

```
> twoPhaseEulerFoam
```

3. How to add new turbulence model to twoPhaseEulerFoam

As you learnt in part One of this series, there are a number of turbulence models implemented for `twoPhaseEulerFoam` solver. These models are RAS models including `kEpsilon`, `kOmegaSST`, `kOmegaSSTSato`, `mixtureKEpsilon`, `laheyKEpsilon`, and `ContinuousGasKEpsilon`; and LES models including `SmagorinskyZhang`, `NicenoKEqn`, and `continuousGasKEqn`.

Suppose that, for any technical or scientific reason, you want to add one of the already implemented basic compressible turbulence models in the OpenFOAM® to `twoPhaseEulerFoam` solver. A complete list of them can be found in the following folders:

```
$ FOAM_SRC/TurbulenceModels/turbulenceModels/RAS
$ FOAM_SRC/TurbulenceModels/turbulenceModels/LES
```

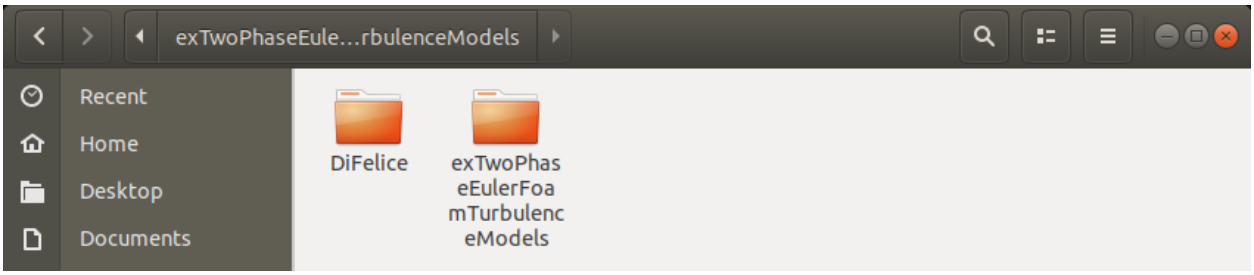
Among all the available options, you are going to add `RNGkEpsilon` turbulence model, which is categorized in RAS-type turbulence models.

3.1. Preparing folders and files

In the first step, you must prepare the folder and files for the new library. Suppose that you are going to implement the new turbulent model in the similar folder that you implemented the new drag model in the previous section. Execute the following commands to create a new sub-folder for the new library:

```
> cd $WM_PROJECT_USER_DIR/twoPhaseEulerFoamModels
> mkdir exTwoPhaseEulerFoamTurbulenceModels
```

The twoPhaseEulerFoamModels folder then should look like this:



In the same terminal, execute the following commands:

```
> cd exTwoPhaseEulerFoamTurbulenceModels
> mkdir Make
```

And copy and paste the following lines in a text editor and save the file as “options” in the sub-folder Make.

exTwoPhaseEulerFoamTurbulenceModels/Make/options

```
EXE_INC = \
    -I$(FOAM_SOLVERS)/multiphase/twoPhaseEulerFoam/twoPhaseSystem/lnInclude \
    -I$(FOAM_SOLVERS)/multiphase/twoPhaseEulerFoam/interfacialModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/phaseCompressible/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude

LIB_LIBS = \
    -lfiniteVolume
```

and then copy and paste the following lines in a new file and save it as “files” in the sub-folder Make.

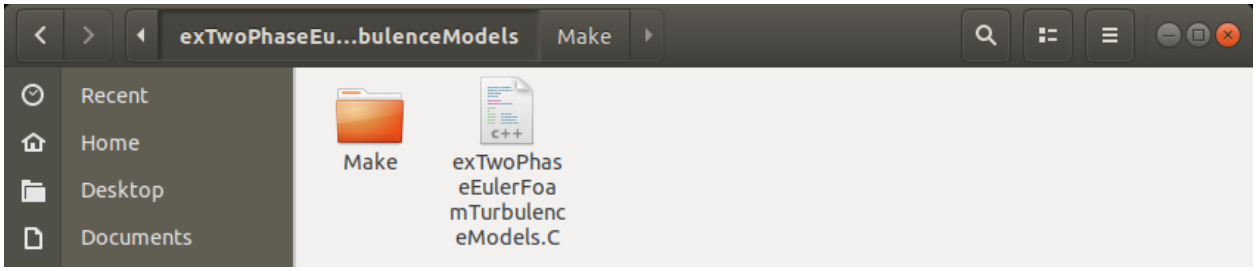
exTwoPhaseEulerFoamTurbulenceModels/Make/files

```
exTwoPhaseEulerFoamTurbulenceModels.C

LIB = $(FOAM_USER_LIBBIN)/libexTwoPhaseEulerFoamTurbulenceModels
```

As can be seen, the name of the new library is `libexTwoPhaseEulerFoamTurbulenceModels`.

In folder exTwoPhaseEulerFoamTurbulenceModels, create a source code file with name exTwoPhaseEulerFoamTurbulenceModels.C. Later the code lines will be added to this source code. The content of this folder should look like this:



3.2. Understanding class hierarchy of turbulence model in twoPhaseEulerFoam

`phaseModel` class is declared to hold the properties and methods related to phase properties in the two-phase System. In the declaration of `phaseModel` class (as you see in code lines below), `PhaseCompressibleTurbulenceModel<phaseModel>` is declared as the base class for all turbulence models in this solver. So, all the implemented turbulence models should be derived from this class. Here, `phaseModel` is a class that provides methods for evaluating transport properties of the phase [1].

$FOAM_SOLVERS/multiphase/twoPhaseEulerFoam/twoPhaseSystem/phaseModel/phaseModel.H

```
// some code lines ...

class phaseModel
:
    public volScalarField,
    public transportModel
{
    // Private Data

        // some code lines ...

        //- Turbulence model
        autoPtr<PhaseCompressibleTurbulenceModel<phaseModel>> turbulence_;

public:
```

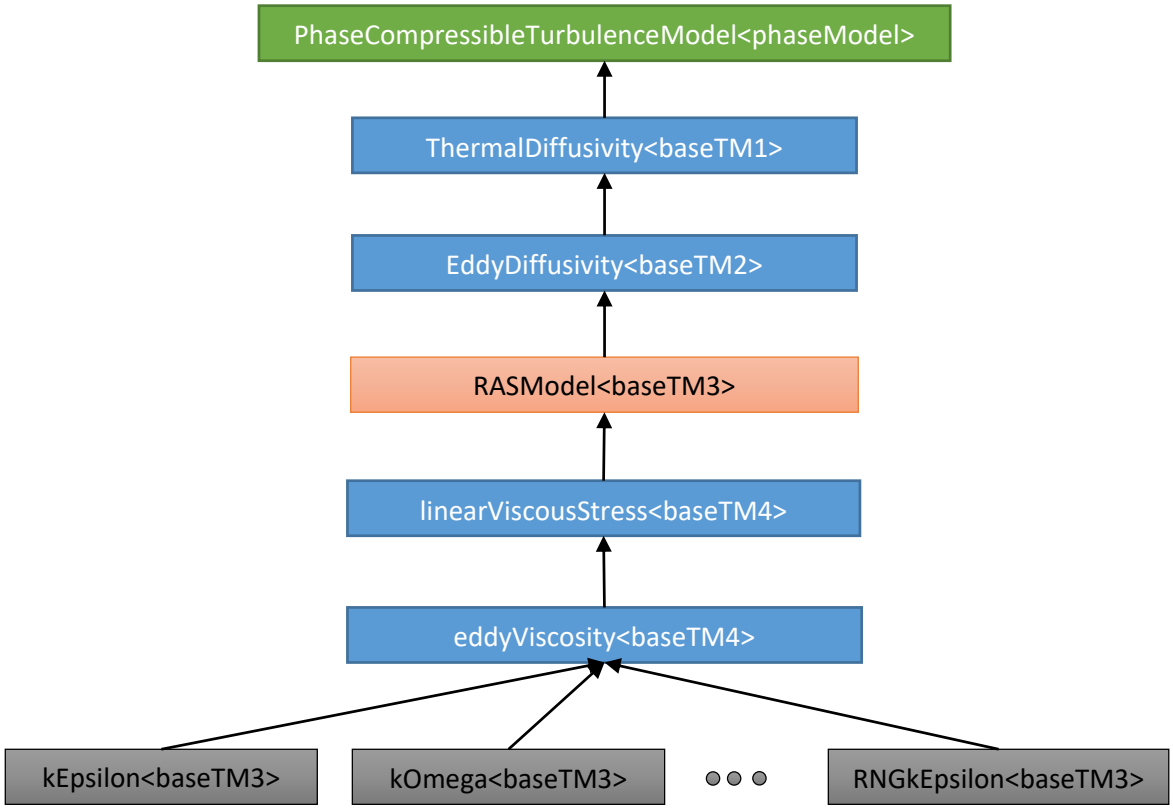
RAS-type turbulence models are derived from (built based on) `RASModel` class template. Therefore, in the class hierarchy, `RASModel` class template is the base class for all the RAS-type turbulences. The definition of `RASModel` template class can be found in:

`$FOAM_SRC/TurbulenceModels/turbulenceModels/RAS/RASModel`

Now, we need to find the relation between `RASModel` class template and `PhaseCompressibelTurbulenceModel` class template. In the graph below (Figure 1), the

[1] `phaseModel` is a wrapper around `heRhoThermo` class. So this solver only works with this thermophysical type.

graphical representation of the relation between these two classes is shown. In this class hierarchy, `RASModel` class template is derived from `PhaseCompressibelTurbulenceModel` and `RNGkEpsilon` class template is derived from `RASModel`. Therefore, the `RNGkEpsilon` class template can be created with `PhaseCompressibleTurbulenceModel<phaseModel>` as its base class (`RNGkEpsilon` class can be down-casted to this base class).




```
baseTM1 = PhaseCompressibleTurbulenceModel<phaseModel>
baseTM2 = ThermalDiffusivity<baseTM1>
baseTM3 = EddyDiffusivity<baseTM2>
baseTM4 = RASModel<baseTM3>
```

Figure 1: Class relation for RAS turbulence models

3.3. Defining runtime name and selection table

Copy and paste the following code lines into file exTwoPhaseEulerFoamTurbulenceModel.C and then save it.

exTwoPhaseEulerFoamTurbulenceModels/exTwoPhaseEulerFoamTurbulenceModels.C

```
#include "PhaseCompressibleTurbulenceModel.H"
#include "phaseModel.H"
#include "twoPhaseSystem.H"
#include "addToRunTimeSelectionTable.H"

#include "ThermalDiffusivity.H"
#include "EddyDiffusivity.H"

#include "RASModel.H"
#include "RNGkEpsilon.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

    typedef ThermalDiffusivity
            <
            PhaseCompressibleTurbulenceModel
                <
                phaseModel
                >
            >   phaseModelPhaseCompressibleTurbulenceModel;


    typedef RASModel
            <
            EddyDiffusivity
                <
                phaseModelPhaseCompressibleTurbulenceModel
                >
            > RASphaseModelPhaseCompressibleTurbulenceModel;


/*defineNamedTemplateTypeNameAndDebug(RASphaseModelPhaseCompressibleTurbulenceModel,
0);*/

    /*defineTemplateRunTimeSelectionTable
    (RASphaseModelPhaseCompressibleTurbulenceModel, dictionary);*/


    typedef
RASModels::RNGkEpsilon<EddyDiffusivity<phaseModelPhaseCompressibleTurbulenceModel>>
            RNGkEpsilonRASphaseModelPhaseCompressibleTurbulenceModel;

    defineNamedTemplateTypeNameAndDebug
    (
        RNGkEpsilonRASphaseModelPhaseCompressibleTurbulenceModel,
        0
    );
```

```
    namespace RASModels
    {
        addToRunTimeSelectionTable
        (
            RASphaseModelPhaseCompressibleTurbulenceModel,
            RNGkEpsilonRASphaseModelPhaseCompressibleTurbulenceModel,
            dictionary
        );
    }
}
```

As you saw in previous section, all RAS turbulence models should be constructed based on the `RASModel` class template. The `RASModel` class template is constructed by instantiating `EddyDiffusivity` class template as its template parameter (and also as its base class). `EddyDiffusivity` template class is constructed by instantiating `ThermalDiffusivity` class template with `PhaseCompressibleTurbulenceModel<phaseModel>` as its template parameter (and as its base class). The first two `typedef` statements perform these instantiations.

The following code lines

```
defineNamedTemplateTypeNameAndDebug(RASphaseModelPhaseCompressibleTurbulenceModel, 0);


defineTemplateRunTimeSelectionTable
    (RASphaseModelPhaseCompressibleTurbulenceModel, dictionary);
```

define the run-time type information and initializes the hash table pointer (for use in the selection table) for the new base class, `RASphaseModelPhaseCompressibleTurbulenceModel`[2]. Since this type name and its run-time selection table has been previously defined elsewhere – when implementing the RAS-type turbulence models for this solver – they turned into comments.

The class of the new turbulence model, `RNGkEpsilon`, is instantiated based on the `EddyDiffusivity<phaseModelPhaseCompressibleTurbulenceModel>`. Note that the template parameter of `RNGkEpsilon` and `RASModel` class are the same (See Figure 1):

[2] This last line takes effect if the macro `declareRunTimeSelectionTable` with proper arguments is defined in the definition of `RASModel` class. Looking at the definition of this class in the source code, you can see that this macro has been defined there.

```
typedef
RASModels::RNGkEpsilon<EddyDiffusivity<phaseModelPhaseCompressibleTurbulenceModel>>
            RNGkEpsilonRASphaseModelPhaseCompressibleTurbulenceModel;
```

This macro defines the run-time type information of the newly instantiated class:

```
defineNamedTemplateTypeNameAndDebug
(
    RNGkEpsilonRASphaseModelPhaseCompressibleTurbulenceModel,
    0
);
```

and the macro below tells the compiler that RNGkEpsilonRASphaseModelPhaseCompressibleTurbulenceModel is derived from `RASphaseModelPhaseCompressibleTurbulenceModel` and when the user selects "RNGkEpsilon" as the `RASModel` in turbulenceProperties dictionary, allocate `RNGkEpsilon` class as the turbulence model.

```
addToRunTimeSelectionTable
(
    RASphaseModelPhaseCompressibleTurbulenceModel,
    RNGkEpsilonRASphaseModelPhaseCompressibleTurbulenceModel,
    dictionary
);
```

Note

LES-type turbulence models can also be added to this solver with a similar method. You only need to learn the class hierarchy between LES models and PhaseCompressibleTurbulenceModel, as we did in this tutorial for RAS-type model.

Note

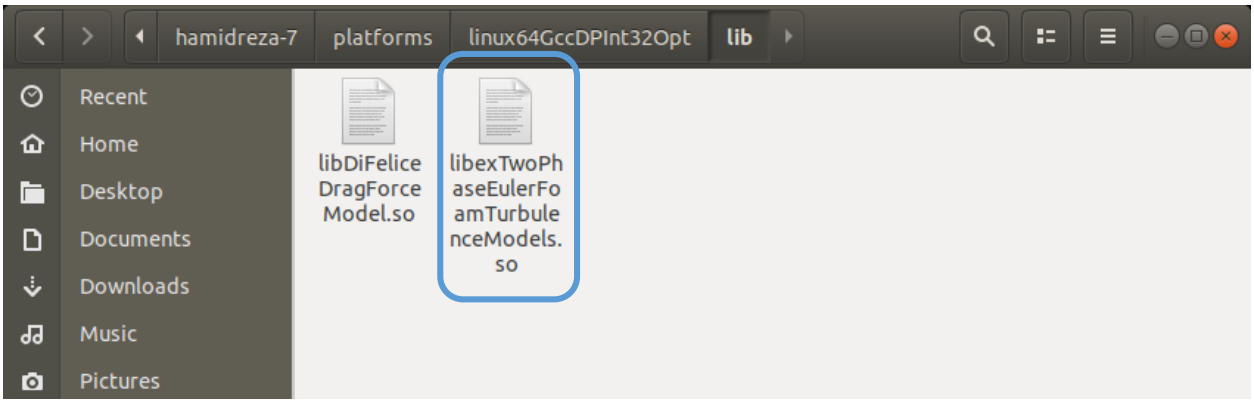
Class template definition in OpenFOAM v1912 is implemented a bit different from OpenFOAM 7. But, RASModel is still a child of PhaseCompressibleTurbulenceModel and RNGkEpsilon is a child of RASModel. Therefore, this implementation works with OpenFOAM v1912.

3.4. Compiling

Execute the following command from twoPhaseEulerFoamModels/ exTwoPhaseEulerFoamTurbulenceModels to build your library:

```
> wmake
```

If everything is done correctly and the compilation returns no error, `wmake` creates your library in the lib sub-folder in the user folder of OpenFOAM.



3.5. Running the simulation

You can use this new turbulence model in the fluidized bed simulation that you performed in “Mastering twoPhaseEulerFoam, One: Fluidized bed” (find it on www.cemf.ir). Alternatively, you can just use the new model on one of the standard tutorial cases of OpenFOAM. For example, you can find one in `$FOAM_TUTORIALS/multiphase/twoPhaseEulerFoam/RAS/fluidisedBed`.

You first need to add this new library to run-time libraries. Open system/controlDict and add the following lines to it and save it:

```
libs
(
    "libexTwoPhaseEulerFoamTurbulenceModels.so"
);
```

Second, select the new turbulence model for your simulation. Open constant/turbulenceProperties.air and change the RAS dictionary as follows:

```
simulationType  RAS;

RAS
{
    RASModel RNGkEpsilon;

    turbulence      on;
    printCoeffs     on;
}
```

Now you can normally perform simulation with the new turbulence model. Just execute the following command from simulation root case:

> twoPhaseEulerFoam