Cite as: Barba Piña, J.L.A.: Implementation of a mass flux term with thermodiffusion mass transport into the species transport equation in a compressible solver. In Proceedings of CFD with OpenSource Software, 2019, Edited by Nilsson. H., `http://dx.doi.org/10.17196/OS_CFD#YEAR_2019`

CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Implementation of a mass flux term with thermodiffusion mass transport into the species transport equation in a compressible solver

Developed for OpenFOAM-v1906

Author:
J. Lorenzo Alejandro
BARBA-PIÑA
University of Leeds
pmjlab@leeds.ac.uk

Peer reviewed by:
PRADIP ARYAL
XIAOAN MAO
MOHAMMAD H.
ARABNEJAD-KHANOUKI

January 16, 2020

Learning outcomes

The reader will learn:
How to use it:

- How thermophysical properties are structured through the thermophysicalModels library and how the thermophysicalProperties dictionary works for compressible solvers.
- How to implement a species transport equation for binary gas mixtures into a compressible solver.

The theory of it:

- A description of the thermophysical properties in OpenFOAM context and how thermophysical models are structured under the *thermophysicalProperties* dictionary entries.
- The theory of thermo-diffusion separation phenomena in fluid mixtures (also known as Ludwig-Soret effect).
- The definition of constant mass transport coefficients and how they are defined and accessed through the *thermophysicalProperties* dictionary.

How it is implemented:

- The implementation of new transport coefficients by modifying the *consTransport* class which is part of the *specie thermophysicalModels* sub-library.
- The addition of the new transport model and the new added transport coefficients into a main thermophysical model, based on *rhoThermo* model contained in the *basic thermophysicalModels* sub-library.
- The addition of the species transport equation for binary gas mixtures for a compressible solver (*rhoSimpleFoam*) which will work with the newly added transport coefficients.

How to modify it:

- Through the modification of the *thermophysicalModels* library so it can include the mass diffusion and thermodiffusion coefficients.
- The new transport coefficients should be accessed trhough the thermophysicalProperties library so they can be added into the species transport equation for binary gas mixtures.
- A new compressible solver capable of solving the species transport equation should be tested by a simple tutorial case, so mass transfer by thermodiffusion can be visualized.

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- Basic knowledge on heat and mass transfer processes (check reference [2]).
- A general knowledge of OpenFOAM's code organization, as well as some knowledge of C++ object-oriented programming.
- It is recomended that the reader gets a general overview on thermodiffusion phenomena in gas mixtures. References [1, 4, 5, 8, 9] can be consulted for this purpose.
- As a suggestion, if there is an extra interest in how to implement or change OpenFOAM's *thermophysicalModels* library for different applications, the reader can consult the works from Hummel, D. [6] and Choquet, I. [3].

Contents

1 Theoretical background **4**
- 1.1 Introduction . . . 4
- 1.2 Mass diffusion caused by temperature gradients . . . 4
- 1.3 Thermophysical models in OpenFOAM . . . 6
  - 1.3.1 *thermophysicalProperties* dictionary . . . 8

2 Implementing constant transport coefficients through thermophysicalProperties library **10**
- 2.1 Creating the *thermophysicalModels* user library . . . 10
  - 2.1.1 *specie* and *basic* libraries . . . 10
- 2.2 Creating the new transport model: *thermoDiffconstTransport* . . . 12
  - 2.2.1 Creating the new thermophysical model: *thermoDiffRhoThermo* . . . 17

3 Creating a new compressible solver for binary gas-mixtures **24**
- 3.1 Copying the compressible solver: *thermoDiffRhoSimpleFoam* . . . 24
- 3.2 The CaEqn.H file . . . 25
- 3.3 The createFields.H file . . . 26
- 3.4 Final implementations in thermoDiffusionFoam.C file and solver compilation . . . 27

4 Tutorial set up **29**
- 4.1 The *thermophysicalproperties* and *initialMassFraction* dictionaries . . . 30
- 4.2 Simulation results post-processing . . . 31

Appendix: Simulation case files **33**
- *system* folder . . . 33
- *constant* folder . . . 37
- *0* folder . . . 38

Study Questions **41**

Bibliography **42**

Chapter 1

Theoretical background

1.1 Introduction

In separation processes, a general approach for getting purified products most efficiently is to take advantage of some transport properties of the mixture system. In the specific case of homogeneous fluid mixtures, there is an interesting phenomenon that can be used for the purification of mixture components, that is, by inducing temperature gradients in the flow field which combined with other molecular transport and diffusive processes can produce a total mass flux of each component. This process is known as thermophoresis, thermodiffusion, or the Ludwig-Soret effect for liquid mixtures [9]. There are several natural and industrial processes where thermophoresis takes part as the main driving force for mass diffusivity, such phenomena includes the thermohaline circulation which is of highly importance in large scale ocean circulation [4], and thermogravitational columns used for separation of gaseous mixtures, crude oil and mixtures of liquid polymers [4, 8, 9].

It would be interesting to analize this kind of processes in the context of CFD since the involvement of energy and momentum transport affects the behavior of mixture separation, which makes the problem complex enough to be tackled by numerical analysis. Luckily, OpenFOAM already counts with a structured library that allows the simulation of cases where energy transfer can be modeled through the coupling of thermodynamic, transport, and specie interaction models.

The purpose of this tutorial is to contribute with a basic numerical model that can be used for the study of the thermodiffusion separation in binary gas mixtures, showing a method for code modification and addition in OpenFOAM's libraries. The writer hopes that this method can help the interested reader to be used for similar implementations.

1.2 Mass diffusion caused by temperature gradients

Temperature gradients in fluid homogeneous mixtures results in a relative concentration difference depending on the temperature field values, at the same time, regular mass diffusion is caused by these concentration gradients. Terefore, it can be said that in a temperature steady-state system, the total mass flux equals to a balance between the ordinary mass diffusion and thermodiffusion transport. For gas mixtures, a theoretical description of this mechanism was made by Chapman and Enskog based on the kinetic theory of gases [1]. The mathematical expression for the net mass flux for a reference component a of a binary gas mixture, with constant pressure and without the effect of external forces, can be expressed as in Rahman et al. [9] by the following equation,

$$J_a(x, y, z, C_a, T, \rho) = \rho \left[D_{ab} \nabla C_a + D_T C_{a0}(1 - C_{a0}) \nabla T \right] \tag{1.1}$$

Where $J_a(x, y, z, C_a, T, \rho)$ is the net mass flux of component a which is a function of temperature, concentration, density, and the direction of the flow (in this case, represented by the cartesian space coordinates of the system: x, y, z). Here, $\rho = f(p, T)$ is the density of the mixture, D_{ab} is the binary mass diffusion coefficient $[\frac{m}{s^2}]$, C_a is the mass fraction of the reference component a, C_{a0} is the initial mass fraction of component a, and D_T is the thermodiffusion coefficient $[\frac{m}{K \cdot s^2}]$. In a steady state system, the mass flux is at equilibrium, so that $J_a = 0$ and the concentration gradient can be set on terms of the temperature gradient and the ratio of the transport coefficients

$$\nabla C_a = -\frac{D_T}{D_{ab}} C_{a0}(1 - C_{a0}) \nabla T \tag{1.2}$$

Dimensionless quantities can help to obtain the values of the transport coefficients needed in the calculation of transport equations such as in equation (1.1). This feature can be used for getting unknown physical properties from other known ones, which is actually done by OpenFOAM's *constTransport* class for calculating the thermal conductivity for simulation cases that use the *thermophysicalProperties* library (Something that will be used for this tutorial).

The thermodiffusion coefficient D_T can be obtained with a dimensionless number known as thermodiffusion ratio K_T [5]. This dimensionless quantity represents the relative effects between thermodiffusion and ordinary mass diffusion at average temperature

$$K_T = T \frac{D_T}{D_{ab}} \tag{1.3}$$

Like the thermodiffusion and mass diffusivity transport coefficients, the value of K_T depends on several molecular parameters, such as molecular masses, the molecular size, the mixture composition, temperature and intermolecular interactions [1]. Regardless of the complexity of all of these variable interactions, a simple interpretation of the value of this coefficient can be given, when K_T is positive, the heaviest specie molecules tend to move toward a colder region, and when it is negative, the heaviest molecules move toward a warmer region [1]. Equation (1.1) can be rearranged in terms of K_T

$$J_a = \rho D_{ab} \left(\nabla C_a + K_T \frac{C_{a0}(1 - C_{a0})}{T} \nabla T \right) \tag{1.4}$$

If the temperature and concentration gradients are assumed to depend on one dimension only (let's say x), for a steady state process, the equation (1.2) can be expressed as follows

$$\frac{\partial C_a}{\partial x} = -K_T \frac{C_{a0}(1 - C_{a0})}{T} \frac{\partial T}{\partial x} \tag{1.5}$$

If thermodiffusion ratio K_T is taken as a constant value independent of the composition of the mixture, an integration can be carried out between temperatures T_c and T_h (lower to higher temperature), giving the following expression

$$\Delta C_a = -K_T C_{a0}(1 - C_{a0}) \ln \left(\frac{T_h}{T_c} \right) \tag{1.6}$$

This equation, can be used for validating the thermodiffusion simulation case that will be used to test the code implementations shown in the following chapters.

1.3 Thermophysical models in OpenFOAM

When simulating flow problems where energy and mass transfer occur in compressible systems it is necessary to implement a model that couples the requirement of a state equation, thermodynamic fluid properties, transport properties, and energy calculation models. For this kind of problems, OpenFOAM already counts with a broad library that allows retrieving thermophysical properties as constant values or functions of temperature, pressure, and composition, with an additional set of equations of state that all together can be used to obtain thermal energy calculations in terms of enthalpy or internal energy ([3]).

The *thermophysicalModels* library is constituted as a set of models that allows calculating several fluid properties that depend on temperature, pressure, and composition of a fluid (or solid) mixture. The structure of this library is organized in such a way that the main thermophysical models depend on other submodels for mixture properties, which in turn depend on transport, thermodynamic properties, and state equations submodels. Table 1.1 shows which thermophysical variables correspond to each submodel, this dependence forms a complex built library that allows the simulation of different types of problems such as compressible flows, heat transfer, multiphase flows, combustion, etc.

Thermophysical Model Structure	
Submodel	**Variables and fluid properties**
Mixture models	T, p, X_a, C_a, etc.
Transport models	μ , κ, α, etc.
Thermodynamic properties	C_p, C_v, H_f, e, etc.
Equations of state	ρ calculations

Table 1.1: Basic structure of the thermophysical modelling in OpenFOAM.

OpenFOAM counts with submodels sets that are combined to form a main thermophysical model forming the first *layer* of the model to be used, then, a second layer is necessary to define the mixture type, which is formed with the *specie*, *thermodynamics* and *transport* numerical value entries. As a reference, the table 1.2 contains a list of each set of the thermophysical submodels ([7]).

1-Equation of State — equationOfState	Description
icoPolynomial	Incompressible polynomial equation of state, e.g. for liquids
perfectGas	Perfect gas equation of state
2-Basic thermophysical properties — thermo	Description
eConstThermo	Constant specific heat cp model with evaluation of internal energy e and entropy s
hConstThermo	Constant specific heat cp model with evaluation of enthalpy h and entropy s
hPolynomialThermo	cp evaluated by a function with coefficients from polynomi- als, from which h, s are evaluated
janafThermo	C_p evaluated by a function with coefficients from JANAF thermodynamic tables, from which h, s are evaluated
3-Derived thermophysical properties — specieThermo	Description
specieThermo	Thermophysical properties of species, derived from C_p , h and/or s
4-Transport properties — transport	Description
constTransport	Constant transport properties
polynomialTransport	Polynomial based temperature-dependent transport properties
sutherlandTransport	Sutherland's formula for temperature-dependent transport properties
5-Mixture properties — mixture	Description
pureMixture	General thermophysical model calculation for passive gas mixtures
homogeneousMixture	Combustion mixture based on normalised fuel mass fraction b
inhomogeneousMixture	Combustion mixture based on b and total fuel mass fraction f_t
veryInhomogeneousMixture	Combustion mixture based on b, ft and unburnt fuel mass fraction f_u
dieselMixture	Combustion mixture based on f_t and f_u
basicMultiComponentMixture	Basic mixture based on multiple components
multiComponentMixture	Derived mixture based on multiple components
reactingMixture	Combustion mixture using thermodynamics and re- action schemes
egrMixture	Exhaust gas recirc ulation mixture
6-Thermophysical model — thermoModel	Description
hePsiThermo	General thermophysical model calculation based on enthalpy h or internal energy e, and compressibility Φ
heRhoThermo	General thermophysical model calculation based on enthalpy h or internal energy e, and density ρ
hePsiMixtureThermo	Calculates enthalpy for combustion mixture based on enthalpy h or internal energy e, and Φ
heRhoMixtureThermo	Calculates enthalpy for combustion mixture based on enthalpy h or internal energy e, and ρ
heheuMixtureThermo	Calculates enthalpy h or internal energy e for un- burnt u gas and combustion mixture

Table 1.2: Submodel sets of the *thermophysicalModels* library, (retrieved from OpenFOAM User Guide [7])

1.3.1 *thermophysicalProperties* dictionary

The *thermophysicalProperties* dictionary is the file where the user can specify the entry values for any solver that uses the *thermophysicalModels* library. In a simulation case, this file can be found in the *constant* folder. The structure of this file begins with the chosen thermophysical model, formed by a combination of each of the thermophysical properties submodels that are specified in the two entry *layers* mentioned in the previous section. The following example of a *thermophysicalProperties* dictionary gives a brief explanation of how this dictionary is constituted.

```
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
location    "constant";
object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//- First layer of the thermophysical model, submodels from table 1.2
// are specified here
thermoType
{
     type            hePsiThermo;
     mixture         pureMixture;
     transport       const;
     thermo          eConst;
     equationOfState perfectGas;
     specie          specie;
     energy          sensibleInternalEnergy;
}
//- Second layer of the thermophysical model, the model is constructed with the entry
// values specified here, starting with the "mixture" keyword that involves
// the "specie" , "thermodynamics", and "transport" entries, corresponding with the
// submodel settings in the first layer.
mixture
{
specie
{
   molWeight   28.9;
}
thermodynamics
{
   Cv          712;
   Hf          0;
}
transport
{
   mu          1.8e-05;
   Pr          0.7;
}
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

It is important to mention that not all the submodel combinations are allowed. Checking the possible combinations can be done by copying a tutorial case that uses thermophysical modelling, (e.g. *$WM_PROJECT_DIR/tutorials/compressible/rhoPimpleFoam/laminar/sineWaveDamping*) and then try to change one of the entries in the *thermoPhysicalProperties* dictionary (e.g. change *const* for *banana*). Now, when running the case, the output will show an error message complaning that the combination is not appropriate and will show a list of the possible *thermophysicalProperties* combinations. An example of this error message can be seen as follows,

```
Create time
Create mesh for time = 68
SIMPLE: convergence criteria
field p      tolerance 0.001
field U      tolerance 0.0001
field e      tolerance 0.001
Reading thermophysical properties
Selecting thermodynamics package
{
type            heRhoThermo;
mixture         pureMixture;
transport       banana;
thermo          hConst;
equationOfState perfectGas;
specie          specie;
energy          sensibleEnthalpy;
}
--> FOAM FATAL ERROR:
Unknown fluidThermo type
thermoType
{
type            heRhoThermo;
mixture         pureMixture;
transport       banana;
thermo          hConst;
equationOfState perfectGas;
specie          specie;
energy          sensibleEnthalpy;
}
Valid fluidThermo types are:
hePsiThermo   homogeneousMixture     const       hConst   perfectGas    specie  sensibleEnthalpy
hePsiThermo   homogeneousMixture     sutherland  hConst   perfectGas    specie  sensibleEnthalpy
hePsiThermo   homogeneousMixture     sutherland  janaf    perfectGas    specie  sensibleEnthalpy
hePsiThermo   inhomogeneousMixture   const       hConst   perfectGas    specie  sensibleEnthalpy
hePsiThermo   inhomogeneousMixture   sutherland  hConst   perfectGas    specie  sensibleEnthalpy
hePsiThermo   inhomogeneousMixture   sutherland  janaf    perfectGas    specie  sensibleEnthalpy
hePsiThermo   multiComponentMixture  const       eConst   perfectGas    specie  sensibleInternalEnergy
... ...
heRhoThermo  homogeneousMixture    const     hConst  incompressiblePerfectGas  specie  sensibleEnthalpy
heRhoThermo  homogeneousMixture    const     hConst  perfectGas         specie  sensibleEnthalpy
heRhoThermo  homogeneousMixture    sutherland   janaf  incompressiblePerfectGas  specie  sensibleEnthalpy
heRhoThermo  homogeneousMixture    sutherland   janaf  perfectGas       specie  sensibleEnthalpy
heRhoThermo  inhomogeneousMixture  const     hConst  incompressiblePerfectGas    specie  sensibleEnthalpy
heRhoThermo  inhomogeneousMixture  const     hConst  perfectGas         specie  sensibleEnthalpy
heRhoThermo  inhomogeneousMixture  sutherland   janaf  incompressiblePerfectGas  specie  sensibleEnthalpy
heRhoThermo  inhomogeneousMixture  sutherland   janaf  perfectGas       specie  sensibleEnthalpy
heRhoThermo  multiComponentMixture const     eConst  adiabaticPerfectFluid     specie  sensibleInternalEnergy
heRhoThermo  multiComponentMixture const     eConst  incompressiblePerfectGas  specie  sensibleInternalEnergy
heRhoThermo  multiComponentMixture const     eConst  perfectFluid     specie  sensibleInternalEnergy
... ...
```

Chapter 2

Implementing constant transport coefficients through thermophysicalProperties library

2.1 Creating the *thermophysicalModels* user library

Before doing any new implementation, the first thing to do is to make sure that the original code will not be affected, this is done by copying the required library folders and substituting the original compilation path with the users one. The following instructions are meant to show how to do this for the *specie* and *basic* sub-libraries which are part of the thermophysicalModels library.

2.1.1 *specie* and *basic* libraries

The *specie* library contains models for transport properties, state equations, thermodynamics and chemical reactions. In a terminal window with the *OpenFOAM-v1906* enviriment active, copy the following lines to get your own version of the *specie* library.

```
mkdir $WM_PROJECT_USER_DIR/src/thermophysicalModels
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels
cp -r $FOAM_SRC/thermophysicalModels/specie .
sed -i s/"FOAM_LIBBIN"/"FOAM_USER_LIBBIN"/g specie/Make/files
```

The thermophysical base models are defined in the *basic* library, the type of calculations depends on the base classes that are contained here (e.g. rhoThermo or psiThermo). Copy the following lines to get your own version of the *basic* library. Notice that the name of the *specie* and *basic* libraries are not changed, it helps to avoid renaming all the file dependencies besides the new *thermophysicalModel* library coexist in parallel with the original one. Additionally, the line *-L$(FOAM_ USER_ LIBBIN)* gives priority to the user's library when the compiler looks for the linked files [3].

```
cp -r $FOAM_SRC/thermophysicalModels/basic $WM_PROJECT_USER_DIR/src/thermophysicalModels
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/basic
sed -i s/"FOAM_LIBBIN"/"FOAM_USER_LIBBIN"/g ./Make/files
sed -i '/EXE_INC = \\/a -I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/specie/lnInclude \\' ./Make/options
sed -i '/LIB_LIBS = \\/a -L$(FOAM_USER_LIBBIN) \\' ./Make/options
```

Now, let's check that the *sed* commands have worked properly for the *files* and *options* files in both libraries. We open the *specie* library *files* file just to check that the library code will be compiled in the user's library directory:

```
...
    atomicWeights/atomicWeights.C
    specie/specie.C
    reaction/reactions/makeReactions.C
    reaction/reactions/makeLangmuirHinshelwoodReactions.C

    LIB = $(FOAM_USER_LIBBIN)/libspecie
...
```

Now, we continue with the *specie* library *options* file and we verify that no executable or library paths are written in this file:

```
...
    EXE_INC =

    LIB_LIBS =
...
```

Then, we check that the *files* file of the *basic* library looks as follows:

```
...
    basicThermo/basicThermo.C
    fluidThermo/fluidThermo.C

    psiThermo/psiThermo.C
    psiThermo/psiThermos.C

    rhoThermo/rhoThermo.C
    rhoThermo/rhoThermos.C
    rhoThermo/liquidThermo.C

    derivedFvPatchFields/fixedEnergy/fixedEnergyFvPatchScalarField.C
    derivedFvPatchFields/gradientEnergy/gradientEnergyFvPatchScalarField.C
    derivedFvPatchFields/mixedEnergy/mixedEnergyFvPatchScalarField.C

    derivedFvPatchFields/energyJump/energyJump/energyJumpFvPatchScalarField.C
    derivedFvPatchFields/energyJump/energyJumpAMI/energyJumpAMIFvPatchScalarField.C

    LIB = $(FOAM_USER_LIBBIN)/libfluidThermophysicalModels
...
```

And finally, we check on the *options* file of the *basic* library:

```
...
    EXE_INC = \
    -I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalProperties/lnInclude \

    LIB_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lfiniteVolume \
    -lmeshTools \
    -lcompressibleTransportModels \
    -lspecie \
    -lthermophysicalProperties
...
```

Now, we can proceed with the compilation of the new libraries, beginning with the *specie* library, and then continuing with the *basic* library. We start with the *specie* library compilation by writing the next command lines in the terminal window.

```
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels
wclean lib specie
wmake libso specie
```

If the compilation process for the *specie* library finishes without errors, we can continue with the compilation of the *basic* library.

```
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels
wclean lib basic
wmake libso basic
```

2.2 Creating the new transport model: *thermoDiffconstTransport*

At the user's *./thermophysicalModels/specie* directory, one can find the transport models folder where the *const* transport code model for constant transport properties is located. In this model, thermal conductivity and thermal diffusivity are calculated from *thermophysicalProperties* dictionary entries *mu* and *Pr* (dynamic viscosity and Prandtl number respectively).

The idea is to use the *const* transport model as a base for our new *thermoDiffusionTransport* model, where we will define two entries *Dab* and *KT*, which are the mass diffusivity and the thermodifussion coefficient that are going to be used for calculating the thermodiffusion coefficient DT.

The first step is to copy the *constTransport* and rename the copied folder and the files inside it. Notice that it is necessary to change the class name *constTransport* with *thermoDiffconstTransport* inside the renamed files (this is done with *sed* command).

```
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/specie/transport
cp -r const thermoDiffconst
cd thermoDiffconst
mv constTransport.C thermoDiffconstTransport.C
mv constTransport.H thermoDiffconstTransport.H
mv constTransportI.H thermoDiffconstTransportI.H
sed -i s/constTransport/thermoDiffconstTransport/g thermoDiffconstTransport*
sed -i s/"const<"/"thermoDiffconst"/g thermoDiffconstTransport.H
cd ../..
cd include
```

Now, we have to include the new code in the thermophysical models types, returning to the *specie* directory folder and access to the *./specie/include* folder where the file *thermoPhysicsTypes.H* is located. Open the file *thermoPhysicsTypes.H* file with the text editor of your preference, and include the *thermoDiffconstTransport.H* header file between *constTransport.H* and *icoPolynomial.H* like in the following example:

```
...
#include "sutherlandTransport.H"
#include "constTransport.H"
#include "thermoDiffconstTransport.H" //- New Transport Model.

#include "icoPolynomial.H"
...
```

Then, inside the *namespace Foam* declaration, the following lines are added to create a thermophysics type that contains our new transport model. For *thermo-physics* types based on sensibleEnthalpy:

```
...
namespace Foam
{
  // thermo physics types based on sensibleEnthalpy
    typedef
    thermoDiffconstTransport //- New transport model
    <
       species::thermo
       <
          hConstThermo
          <
             perfectGas<specie>
          >,
          sensibleEnthalpy
       >
    > thermoDiffconstGasHThermophysics;  //- New thermophysical model name

 typedef
 constTransport
 ...
```

As well as the next lines for *thermo-physics* types based on *sensibleInternalEnergy*:

```
...
 // thermo physics types based on sensibleInternalEnergy
 typedef
 thermoDiffconstTransport //- New transport model
 <
    species::thermo
    <
       eConstThermo
       <
          perfectGas<specie>
       >,
       sensibleInternalEnergy
    >
 > thermoDiffconstGasEThermoPhysics; //- New thermophysical model name

 typedef
```

```
    217| constTransport
...
```

It is important to notice that each line of the new code has to correspond with the marked line-numbers in the file (this can be compared with the accompanying files). Now, return to the folder *thermoDiffconst* (*cd ../transport/thermoDiffconst/*) and open the file *thermoDiffconstTransport.H*. At the *thermoDiffconstTransport* class declaration add the following lines that correspond to D_{ab} and K_T as new private data members:

```
...
    87 | // Private data
    88 |
    89 | //- Mass diffusivity [m^2/s]
    90 | scalar Dab_; //- New  data member Dab
    91 |
    92 | //- Thermodiffusion ratio []
    93 | scalar KT_; //- New data member KT
    94 |
    95 | //- Constant dynamic viscosity [Pa.s]
...
```

And at the *private member function* declaration add the new data members D_{ab} and KT inside the constructor declaration:

```
...
    103| //- Construct from components
    104| inline thermoDiffconstTransport
    105| (
    106|     const Thermo& t,
    107|     const scalar Dab, //- New data member  Dab
    108|     const scalar KT,  //- New  data member  KT
    109|     const scalar mu,
    110|     const scalar Pr
    111| );
...
```

Finally, at the *member functions section*, add the next function declarations (between *typename()* and dynamic viscosity functions):

```
...
    137| return "thermoDiffconst<" + Thermo::typeName() + '>';
    138| }
    139|
    140| //- Mass diffusivity [m^2/s]
    141| inline scalar Dab(const scalar p, const scalar T) const; //- New data member  Dab
    142|
    143| //- thermodiffusion coefficient [m^2/s]
    144| inline scalar DT(const scalar p, const scalar T) const; //- New  data member  DT
    145|
    146| //- Dynamic viscosity [kg/ms]
...
```

Save the file and then open the *thermoDiffconstI.H* file where the inline functions of the class *thermoDiffconstTransport* are defined. Write the next code lines at the specified line numbers in the *constructors declaration* section:

```
...
    31 | template<class Thermo>
    32 | inline Foam::thermoDiffconstTransport<Thermo>::thermoDiffconstTransport
    33 | (
    34 | const Thermo& t,
    35 |    const scalar Dab, //- New data member Dab
    36 |    const scalar KT,  //- New data member KT
    37 |    const scalar mu,
    38 |    const scalar Pr
    39 | )
    40 | :
    41 |    Thermo(t),
    42 |    Dab_(Dab), //- New data member Dab
    43 |    KT_(KT),   //- New data member KT
    44 |    mu_(mu),
    45 |    rPr_(1.0/Pr)
    46 | {}
...
    49 | template<class Thermo>
    50 | inline Foam::thermoDiffconstTransport<Thermo>::thermoDiffconstTransport
    51 | (
    52 | const word& name,
    53 |    const thermoDiffconstTransport& ct
    54 | )
    55 | :
    56 |    Thermo(name, ct),
    57 |    Dab_(ct.Dab_), //- New data member Dab
    58 |    KT_(ct.KT_),   //- New data member KT
    59 |    mu_(ct.mu_),
    60 |    rPr_(ct.rPr_)
    61 | {}
...
```

After this addition, two inline functions have to be defined in the member functions section, the first one returns the constant value of D_{ab} (specified as a dictionary entry) and the second one returns the value of D_T calculated as a function of D_{ab}, K_T and T, this addition is done at the beginning of the *member functions* section:

```
...
    83 |// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
    84 |
    85 | template<class Thermo>
    86 | inline Foam::scalar Foam::thermoDiffconstTransport<Thermo>::Dab
    87 | (
    88 |     const scalar p,
    89 |     const scalar T
    90 | ) const
    91 | {
    92 |     return Dab_;  //- Constant mass diffusivity for binary mixtures.
    93 | }
    94 |
    95 |
    96 | template<class Thermo>
    97 | inline Foam::scalar Foam::thermoDiffconstTransport<Thermo>::DT
    98 | (
    99 |   const scalar p,
    100|   const scalar T
    101| ) const
    102| {
    103|     return (KT_*Dab(p, T))/T;  //- Thermodiffusion coefficient.
    104| }
...
```

Finally at the *member operators* and the *friend operators* sections, the two new data members have to be added as dictionary entries, for each operator function. Add the corresponding code lines for D_{ab} and KT data members, at the specified line numbers:

```
...
    149| Dab_ = ct.Dab_; //- New data member Da
    150| KT_ = ct.KT_;   //- New data member KT
    151| mu_ = ct.mu_;
    152| rPr_ = ct.rPr_;
...
    171| Dab_ = Y1*Dab_ + Y2*st.Dab_; //- New data member Dab
    172| KT_ = Y1*KT_ + Y2*st.KT_;    //- New data member KT
    173| mu_ = Y1*mu_ + Y2*st.mu_;
    174| rPr_ = 1.0/(Y1/rPr_ + Y2/st.rPr_);
...
    207| t,
    208| 0,
    209| ct1.Dab_, //- New data member Dab
    210| ct1.KT_,  //- New data member KT
    211| ct1.rPr_
...
    236| t,
    237| Y1*ct1.Dab_ + Y2*ct2.Dab_,  //- New data member Dab
    238| Y1*ct1.KT_ + Y2*ct2.KT_,    //- New data member KT
    239| Y1*ct1.mu_ + Y2*ct2.mu_,
    240| 1.0/(Y1/ct1.rPr_ + Y2/ct2.rPr_)
...
    240| s*static_cast<const Thermo&>(ct),
    241| ct.Dab_,   //- New data member Dab
    242| ct.KT_,    //- New data member KT
    243| ct.mu_,
...
```

Save the file and close it. For the final step, open the file *thermoDiffconstTransport.C* and add the next lines as part of the dictionary reading constructor and the dictionary reading member function declaration respectively:

```
...
    37 | Thermo(dict),
    38 | Dab_(dict.subDict("transport").get<scalar>("Dab")), //- New data member Dab
    39 | KT_(dict.subDict("transport").get<scalar>("KT")),   //- New data member KT
    40 | mu_(dict.subDict("transport").get<scalar>("mu")),
...
    56 | os.beginBlock("transport");
    57 | os.writeEntry("Dab", Dab_);  //- New data member Dab
    58 | os.writeEntry("KT", KT_);    //- New data member KT
    59 | os.writeEntry("mu", mu_);
...
```

Save the file and close it. Now the *specie* library can be recompiled. Return to the main folder *thermophysicalModels* using the terminal and write the following commands:

```
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/
wclean lib specie
wmake libso specie
```

2.2.1 Creating the new thermophysical model: *thermoDiffRhoThermo*

In order to execute the functions defined in the past section with the *thermophysicalProperties* dictionary, it is necessary to declare and define the new data and function members in a base thermophysical model. At the *basic* directory, one can find a folder called *rhoThermo*, this model constructs a basic thermophysical model based on density, hence it can be used for compressible flow modelling which can be used for binary gas mixture separation modelling.

The first step is to copy the *rhoThermo* folder and rename it as *thermoDiffrhoThermo*, then the *liquidThermo.H* and *liquidThermo.C* will be removed as those files correspond to a liquid properties selector function that is not necessary for our implementation.

```
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels/basic/
cp -r rhoThermo thermoDiffrhoThermo
cd thermoDiffrhoThermo
rm liquidThermo*
mv rhoThermo.H thermoDiffrhoThermo.H
mv rhoThermo.C thermoDiffrhoThermo.C
mv rhoThermos.C thermoDiffrhoThermos.C
mv heRhoThermo.H thermoDiffheRhoThermo.H
mv heRhoThermo.C thermoDiffheRhoThermo.C
sed -i s/rhoThermo/thermoDiffrhoThermo/g thermoDiffrhoThermo*
sed -i s/rhoThermo/thermoDiffrhoThermo/g thermoDiffheRhoThermo*
sed -i s/heRhoThermo/thermoDiffheRhoThermo/g thermoDiffheRhoThermo*
sed -i s/heRhoThermo/thermoDiffheRhoThermo/g thermoDiffrhoThermos.C
```

Now, all the files have been modified to contain the new class names that correspond to the name of the new thermophysical model (e.g. *rhoThermo* changed to *thermoDiffrhoThermo*). We continue with the new additions by opening the first file *thermoDiffrhoThermo.H* and we add the next lines in the *thermoDiffrhoThermo* class declaration at the *protected data* section and the *member functions* for accessing transport variables section, following the corresponding line counter numbers:

```
...
    65 | //- Density field [kg/m^3]
    66 | //  Named 'thermoDiffrhoThermo' to avoid (potential) conflict with solver density
    67 | volScalarField rho_;
    68 |
    69 | //- Mass diffusivity [m^2/s]
    70 | volScalarField Dab_; //New data member Dab
    71 |
    72 | //- Thermodiffusion coefficient [m^2/s*K]
    73 | volScalarField DT_; //New data member DT
    74 |
    75 | //- Compressibility [s^2/m^2]
```

```
...
//- Dynamic viscosity of mixture for patch [kg/m/s]
virtual tmp<scalarField> mu(const label patchi) const;

//- Mass diffusivity [m^2/s]
virtual tmp<volScalarField> Dab() const; //- New data member Dab

//- Mass diffusivity for patch [m^2/s]
virtual tmp<scalarField> Dab(const label patchi) const; //- New data member Dab

//- Thermodiffusion coefficient [m^2/s*K]
virtual tmp<volScalarField> DT() const; //- New data member DT

//- Thermodiffusion coefficient for patch [m^2/s*K]
virtual tmp<scalarField> DT(const label patchi) const; //- New data member DT
};
...
```

Save and close the file. Now, open the thermoDiffrhoThermo.C file and add the D_{ab} and D_T function object definitions in the constructors section to construct the new transport coefficients as dimensioned scalars, then at the memeber functions section, add the member function declarations for both D_{ab} and D_T:

```
...
Dab_ //- New data member Dab
(
   IOobject
   (
      phasePropertyName("thermo:Dab"),
      mesh.time().timeName(),
      mesh,
      IOobject::NO_READ,
      IOobject::NO_WRITE
    ),
    mesh,
    dimensionSet(0, 2, -1, 0, 0)
),

DT_ //- New data member DT
(
   IOobject
   (
      phasePropertyName("thermo:DT"),
      mesh.time().timeName(),
      mesh,
      IOobject::NO_READ,
      IOobject::NO_WRITE
   ),
   mesh,
   dimensionSet(0, 2, -1, -1, 0)
),
...
Dab_ //- New data member Dab
(
   IOobject
   (
      phasePropertyName("thermo:Dab"),
      mesh.time().timeName(),
      mesh,
      IOobject::NO_READ,
      IOobject::NO_WRITE
    ),
    mesh,
    dimensionSet(0, 2, -1, 0, 0)
),

DT_ //- New data member DT
(
   IOobject
   (
      phasePropertyName("thermo:DT"),
      mesh.time().timeName(),
      mesh,
      IOobject::NO_READ,
      IOobject::NO_WRITE
   ),
   mesh,
   dimensionSet(0, 2, -1, -1, 0)
),
...
Dab_ //- New data member Dab
(
   IOobject
   (
      phasePropertyName("thermo:Dab"),
      mesh.time().timeName(),
      mesh,
      IOobject::NO_READ,
      IOobject::NO_WRITE
    ),
    mesh,
    dimensionSet(0, 2, -1, 0, 0)
),

DT_ //- New data member DT
(
   IOobject
   (
      phasePropertyName("thermo:DT"),
      mesh.time().timeName(),
      mesh,
      IOobject::NO_READ,
      IOobject::NO_WRITE
   ),
   mesh,
   dimensionSet(0, 2, -1, -1, 0)
),
...
Foam::tmp<Foam::volScalarField> Foam::thermoDiffrhoThermo::Dab() const
```

```
{
   return Dab_; //- New member function Dab
}


Foam::tmp<Foam::scalarField> Foam::thermoDiffrhoThermo::Dab(const label patchi) const
{
   return Dab_.boundaryField()[patchi]; //- New member function Dab
}


Foam::tmp<Foam::volScalarField> Foam::thermoDiffrhoThermo::DT() const
{
   return DT_; //- New member function DT
}


Foam::tmp<Foam::scalarField> Foam::thermoDiffrhoThermo::DT(const label patchi) const
{
   return DT_.boundaryField()[patchi]; //- New member function DT
}
...
```

Now, open the next file *thermoDiffheRhoThermo.H* and add the following code lines in the class *thermoDiffheRhoThermo* declaration:

```
...
volScalarField& alpha,
volScalarField& Dab, //- New member function Dab
volScalarField& DT,  //- New member function DT
const bool doOldTimes
...
```

Then, we follow with the *thermoDiffheRhoThermo.C* where the new transport coefficients are calculated and updated adding the next lines (Make sure that the previous declarations have a comma at the end, e.g. volScalarField& mu,):

```
...
volScalarField& alpha,
volScalarField& Dab, //- New data member Dab
volScalarField& DT,  //- New data member DT
const bool doOldTimes
...
alpha.oldTime(),
Dab.oldTime(), //- New data member Dab
DT.oldTime(),  //- New data member DT
true
...
scalarField& alphaCells = alpha.primitiveFieldRef();
scalarField& DabCells = Dab.primitiveFieldRef(); //- New data member Dab
scalarField& DTCells = DT.primitiveFieldRef();   //- New data member DT

forAll(TCells, celli)
...
alphaCells[celli] = mixture_.alphah(pCells[celli], TCells[celli]);
DabCells[celli] = mixture_.Dab(pCells[celli], TCells[celli]); //- New data member Dab
```

```
DTCells[celli]  =  mixture_.DT(pCells[celli], TCells[celli]); //- New data member DT
}
...
volScalarField::Boundary& alphaBf = alpha.boundaryFieldRef();
volScalarField::Boundary& DabBf = Dab.boundaryFieldRef(); //- New data member Dab
volScalarField::Boundary& DTBf = DT.boundaryFieldRef();   //- New data member DT

forAll(pBf, patchi)
...
fvPatchScalarField& palpha = alphaBf[patchi];
fvPatchScalarField& pDab = DabBf[patchi]; //- New data member Dab
fvPatchScalarField& pDT = DTBf[patchi];   //- New data member DT

if (pT.fixesValue())
...
palpha[facei] = mixture_.alphah(pp[facei], pT[facei]);
pDab[facei] =  mixture_.Dab(pp[facei], pT[facei]); //- New data member Dab
pDT[facei]  =  mixture_.DT(pp[facei], pT[facei]);  //- New data member DT
}
...
palpha[facei] = mixture_.alphah(pp[facei], pT[facei]);
pDab[facei] = mixture_.Dab(pp[facei], pT[facei]); //- New data member Dab
pDT[facei] =  mixture_.DT(pp[facei], pT[facei]);  //- New data member DT
}
...
this->alpha_,
this->Dab_,  //- New data member Dab
this->DT_,   //- New data member DT
true                    // Create old time fields
...
this->alpha_,
this->Dab_,  //- New data member Dab
this->DT_,   //- New data member DT
true                    // Create old time fields
...
this->alpha_,
this->Dab_,  //- New data member Dab
this->DT_,   //- New data member DT
false           // No need to update old times
...
```

Save the file and close it. Finally, we proceed to modify the *thermoDiffrhoThermos.C* file, where the new model is constructed. First, all the unnecessary headers are commented as shown in the following example. Then, inside the Foam namespace, two new thermophysical models will be added and all the current models inside the file can be commented or erased. As the comment lines show in the example, each new model are based on constant enthalpy and a constant internal energy respectively, and each one includes the created thermodiffusion transport model, and the thermodiffusion energy model based on density (*thermoDiffrhoThermo* and *thermoDiffheRhoThermo* respectively), the *thermoDiffrhoThermos.C* file should looks as follows:

```
\*---------------------------------------------------------------------------*/

#include "thermoDiffrhoThermo.H"
#include "makeThermo.H"

```

```
#include "specie.H"
#include "perfectGas.H"
//#include "incompressiblePerfectGas.H"
//#include "Boussinesq.H"
//#include "rhoConst.H"
//#include "perfectFluid.H"
//#include "PengRobinsonGas.H"
//#include "adiabaticPerfectFluid.H"

#include "hConstThermo.H"
#include "eConstThermo.H"
//#include "janafThermo.H"
#include "sensibleEnthalpy.H"
#include "sensibleInternalEnergy.H"
#include "thermo.H"

//#include "constTransport.H"
//#include "sutherlandTransport.H"
//#include "WLFTransport.H"
#include "thermoDiffconstTransport.H" //- New transport model for thermodiffusion

//#include "icoPolynomial.H"
//#include "hPolynomialThermo.H"
//#include "polynomialTransport.H"

#include "thermoDiffheRhoThermo.H"
#include "pureMixture.H"

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

namespace Foam
{

/* * * * * * * * * * * * * * * * private static data * * * * * * * * * * * * * */

makeThermos
(
   thermoDiffrhoThermo,      //- Thermophysical model with thermodiffusion
   thermoDiffheRhoThermo,
   pureMixture,
   thermoDiffconstTransport, //- Constant transport model with thermodiffusion
   sensibleEnthalpy,
   hConstThermo,
   perfectGas,
   specie
);

makeThermos
(
   thermoDiffrhoThermo,      //- Thermophysical model with thermodiffusion
   thermoDiffheRhoThermo,
   pureMixture,
   thermoDiffconstTransport, //- Constant transport model with thermodiffusion
   sensibleInternalEnergy,
    eConstThermo,
    perfectGas,
    specie
);

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

} // End namespace Foam

// ************************************************************************* //
...
```

Save this file and then proceed with the recompilation of *basic* library, adding the new thermophysical model *thermoDiffheRhoThermo* to the *files* which is the file that contains the main *.C* source files list of the library. Open the file *$WM_PROJECT_USER_DIR/src/thermophysicalModels/basic/Make/files* with the text editor of your choice and make sure that it looks like in the following example:

```
...
    basicThermo/basicThermo.C
    fluidThermo/fluidThermo.C

    psiThermo/psiThermo.C
    psiThermo/psiThermos.C

    rhoThermo/rhoThermo.C
    rhoThermo/rhoThermos.C
    rhoThermo/liquidThermo.C

    thermoDiffrhoThermo/thermoDiffrhoThermo.C
    thermoDiffrhoThermo/thermoDiffrhoThermos.C

    derivedFvPatchFields/fixedEnergy/fixedEnergyFvPatchScalarField.C
    derivedFvPatchFields/gradientEnergy/gradientEnergyFvPatchScalarField.C
    derivedFvPatchFields/mixedEnergy/mixedEnergyFvPatchScalarField.C

    derivedFvPatchFields/energyJump/energyJump/energyJumpFvPatchScalarField.C
    derivedFvPatchFields/energyJump/energyJumpAMI/energyJumpAMIFvPatchScalarField.C

    LIB = $(FOAM_USER_LIBBIN)/libfluidThermophysicalModels
...
```

Finally, at the terminal window, write the following command lines to recompile the new *basic* library.

```
cd $WM_PROJECT_USER_DIR/src/thermophysicalModels
wclean lib basic
wmake libso basic
```

Chapter 3

Creating a new compressible solver for binary gas-mixtures

The implementation of the species transport equation for binary gas mixtures will be made into one of the current OpenFOAM's compressible solvers, which are one of the set of solvers that allows the utilization of the *thermophysicalProperties* dictionary. The chosen solver is the *rhoSimpleFoam* solver, which is a steady state compressible solver that is used for solving fluid problems with variable density.

3.1 Copying the compressible solver: *thermoDiffRhoSimpleFoam*

The first thing to do, is avoiding compilation errors that can cause important issues to OpenFOAM's correct performance. One of the most common errors happens when the compilation paths are duplicated or wrongly targeted, causing bugs and interference with OpenFOAM's main code structure. The usual way to do this is to copy the solver code files that will work as the base code for the new implementations, then the main .C file should be renamed, making the necessary changes inside this file and the *Make* folder files, where not only the name of the new solver should be added, the executable address should be changed as well.

In a terminal window, set the OpenFOAM environment (e.g. by typing of-v1906) and add the next lines in order to set the base code files for the new solver.

```
mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/compressible/thermoDiffRhoSimpleFoam
cd $FOAM_SOLVERS/compressible/rhoSimpleFoam
cp -r .  $WM_PROJECT_USER_DIR/applications/solvers/compressible/thermoDiffRhoSimpleFoam
cd $WM_PROJECT_USER_DIR/applications/solvers/compressible/thermoDiffRhoSimpleFoam
rm -r overRhoSimpleFoam
rm -r rhoPorousSimpleFoam
mv rhoSimpleFoam.C thermoDiffRhoSimpleFoam.C
sed -i s/"rhoSimpleFoam"/"thermoDiffRhoSimpleFoam"/g thermoDiffRhoSimpleFoam.C
sed -i s/"rhoSimpleFoam"/"thermoDiffRhoSimpleFoam"/g Make/files
sed -i s/"FOAM_APPBIN"/"FOAM_USER_APPBIN"/g Make/files
wclean
wmake
```

Then, check for any error messages at the compilation output in the terminal window, After this, it is a good practice to check that the new solver works with one of the available tutorial cases, this can be done by following the next lines.

```
cd $WM_PROJECT_USER_DIR/run
cp -r $FOAM_TUTORIALS/compressible/rhoSimpleFoam/squareBendLiq ./thermoDiffSquareBendLiq
cd thermoDiffSquareBendLiq
cd thermoDiffSquareBendLiq
sed -i s/"rhoSimpleFoam"/"thermoDiffRhoSimpleFoam"/g system/controlDict
blockMesh
thermoDiffRhoSimpleFoam >& log &
```

Finally, we can have a look at the log file just to see if the new solver is running properly.

3.2 The CaEqn.H file

The species transport equation will be written in terms of the mass fraction of the reference component C_a, considering a variable density, this equation can be represented as follows,

$$\frac{\partial(\rho C_a)}{\partial t} + \nabla \cdot (\Phi C_a) = \nabla \cdot (\rho D_{ab} \nabla C_a + \rho D_T C_{a0}(1 - C_{a0}) \nabla T) \tag{3.1}$$

Where Φ represents the total mass flux of the mixture ($\Phi = \rho \vec{v}$). The expression can be rearranged to let all the terms that depend on C_a at the left hand side of the equation so that the thermodiffusion term can be solved explicitly by the solver.

$$\frac{\partial(\rho C_a)}{\partial t} + \nabla \cdot (\Phi C_a) - \nabla \cdot (\rho D_{ab} \nabla C_a) = \nabla \cdot (\rho D_T C_{a0}(1 - C_{a0}) \nabla T) \tag{3.2}$$

Since the chosen solver which will be used for the new implementation is a steady state solver, the time derivative term will be removed from the last equation, so that, the expression to be used for the new solver looks like the following one,

$$\nabla \cdot (\Phi C_a) - \nabla \cdot (\rho D_{ab} \nabla C_a) = \nabla \cdot (\rho D_T C_{a0}(1 - C_{a0}) \nabla T) \tag{3.3}$$

Now, continue by opening the text editor of your chice, create a new file called *CaEqn.H* and add the following lines,

```
...
    //Species transport equation in terms of the reference component 'Ca' mass fraction.
    fvScalarMatrix CaEqn
    (
        fvm::div(phi, Ca)
      - fvm::laplacian(thermo.Dab()*rho, Ca)
    );

    solve(CaEqn == fvc::laplacian(thermo.DT()*Ca0*(1-Ca0)*rho, thermo.T()));
    CaEqn.relax();
...
```

Save this file inside the same folder that contains the *thermoDiffRhoSimpleFoam* solver.

3.3 The createFields.H file

The first thing to do is to modify the first lines of the *createFields.H* file to make that the new solver can work with the new thermophysical model. With the text editor of your preference, open the *createFields.H* file and then change the text line *fluidThermo* by *thermoDiffrhoThermo*, it should looks like in the next example:

```
...
    Info<< "Reading thermophysical properties\n" << endl;

    autoPtr<thermoDiffrhoThermo> pThermo
    (
        thermoDiffrhoThermo::New(mesh)
    );
    thermoDiffrhoThermo& thermo = pThermo();
    thermo.validate(args.executable(), "h", "e");
...
```

The mass fraction of the two components are created inside the createFields.H file, this is done by adding the following lines after the definition of the velocity field, notice that the second component is defined depending on the value of the reference component a.

```
...
    //Reference component mass fraction
    volScalarField Ca
    (
    IOobject
    (
    "Ca",
    runTime.timeName(),
    mesh,
    IOobject::READ_IF_PRESENT,
    IOobject::AUTO_WRITE
    ),
    mesh
    );

    //Second component mass fraction
    volScalarField Cb
    (
    IOobject
    (
        "Cb",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    1-Ca
    );
...
```

The initial mass fraction of the reference component can be defined as a constant entry in a customized dictionary called *initialMassFraction*, the value of the initial mass fraction can be easily changed by creating a dictionary file called (*initialMassFraction*) inside the *constant* folder of any simulation case that works with the *thermoDiffRhoSimpleFoam* solver. For this purpose, add the next code lines after the turbulence model definition, in the *createFields.H* file:

```
...
    Info<< "Reading initialMassFraction of Ca\n" << endl;

    IOdictionary initialMassFraction
    (
       IOobject
       (
         "initialMassFraction",
          runTime.constant(),
          mesh,
          IOobject::MUST_READ,
          IOobject::NO_WRITE
       )
    );

    Info<< "Reading initial mass fraction of component 'a'\n" << endl;
    dimensionedScalar Ca0
    (
       "Ca0",  dimensionSet(0, 0, 0, 0, 0, 0 ,0), initialMassFraction
    );
...
```

We finally save the *createFields.H* file with the new implementations and proceed with the next modifications.

3.4 Final implementations in thermoDiffusionFoam.C file and solver compilation

In this step, the thermoDiffusionFoam.C file should be opened. The header file of the new thermophysical model is included at the begining of the file replacing the *"fluidThermo.H"* header, and the CaEqn.H file has to be added before the end of the time-loop, as solution values of temperature and velocity will be needed for the species transport equation calculations. the boundary conditions are corrected as well as the thermophysical variables.

```
...
    #include "fvCFD.H"
    //#include "fluidThermo.H" (fluidThermo.H is commented or erased)
    #include "turbulentFluidThermoModel.H"
    #include "simpleControl.H"
    #include "pressureControl.H"
    #include "fvOptions.H"
    #include "thermoDiffrhoThermo.H" //- New thermophysical model

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

    int main(int argc, char *argv[])
...
    turbulence->correct();

    //New additions
    U.correctBoundaryConditions();
    thermo.T().correctBoundaryConditions();
    p.correctBoundaryConditions();

    thermo.correct();

    #include "CaEqn.H" //Include the species transport equation
    Cb = 1 - Ca;       //Calculate the mass fraction of the second component b

    Ca.correctBoundaryConditions(); //Correct boundary conditions after calculations
    //End of new additions

    runTime.write();

    runTime.printExecutionTime(Info);
...
```

Now, let's open the file *Make/options* and replace the link path to the basic library with the user directory path (Change the line: *-I$(LIB_SRC)/thermophysicalModels/basic/lnInclude * with *-I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/basic/lnInclude *), then, after the line "*EXE_LIBS = *" add the link path to the user's library "*-L$(FOAM_USER_LIBBIN) *". Make sure that the *options* file looks like in the following example:

```
    EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/cfdTools \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/sampling/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(WM_PROJECT_USER_DIR)/src/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \

    EXE_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling \
    -lcompressibleTransportModels \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -latmosphericModels
```

Save the file and proceed with the compilation of the new solver by entering the following command lines in the terminal window.

```
cd $WM_PROJECT_USER_DIR/applications/solvers/compressible
wclean thermoDiffRhoSimpleFoam
wmake thermoDiffRhoSimpleFoam
```

Chapter 4

Tutorial set up

To test the new implementations made in the *thermophysicalModels* library, a basic test case will be set up. The case consist on a simple 2-D simulation of a *thermodiffusion* cell with a $50\% - 50\%$ mass fraction mixture of He and N_2 entering to the domain at $4 \cdot 10^{-06} \frac{kg}{s}$. The domain resembles a parallel plate flow system where a temperature gradient is induced by setting the higher temperature in the upper wall and the lower temperature in the bottom wall. Table 4.1 shows the thermophysical constant values that are necessary as initial conditions for this test case.

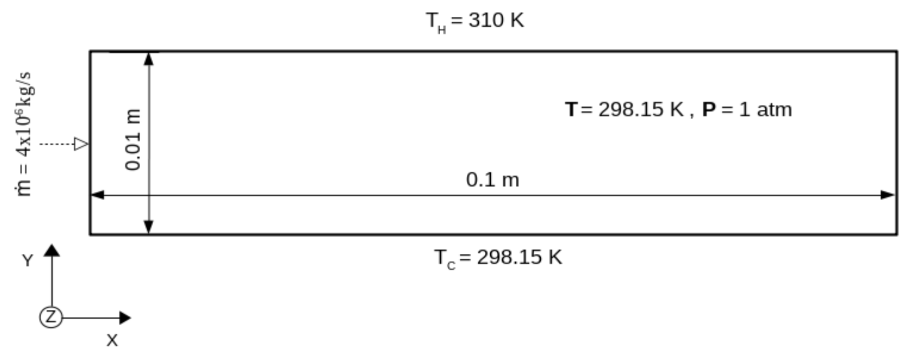

Figure 4.1: Sketch of the thermodiffusion cell, 2-D geometry.

Since the purpose of this tutorial is to show how the *thermophysicalModels* library works with the new defined mass transport coefficients in a compressible solver, only the *thermophysicalProperties* and the custom *initialMassFraction* dictionaries are explained here, the rest of the case files are compelled in the *Appendix* section.

Description	Symbol	Value and units
Mass fraction of He	C_a	0.5
Mass fraction of N_2	C_b	0.5
Mixture mole weight	M	$7.00125[\frac{kg}{mol}]$
Mixture dynamic viscosity	μ	$1.969 \cdot 10^{-5}[\frac{kgm}{s}]$
Mass diffusivity	D_{ab}	$1.969 \cdot 10^{-5}[\frac{m^2}{s}]$
Entalpy of formation	h_f	$0[\frac{kgm^2}{s^2mol}]$
Mixture specific heat	c_p	$3112.452[\frac{m^2}{s^2K}]$
Prandtl	Pr	0.45
Thermodiffusion ratio	K_T	20.33

Table 4.1: Mixture therrmophysical properties

4.1 The *thermophysicalproperties* and *initialMassFraction* dictionaries

In the *thermoPhysicalProperties* dictionary, one has to specify the entries for the new thermophysical and transport models, the reader can compare the values given in table 4.1 with the entry values of the following example of the *thermophysicalProperties* dictionary.

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     location    "constant";
     object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
thermoType
{
     type            thermoDiffheRhoThermo; //- New thermophysical model
     mixture         pureMixture;
     transport       thermoDiffconst;  //- New transport model
     thermo          hConst;
     equationOfState perfectGas;
     specie          specie;
     energy          sensibleEnthalpy;
}
mixture
{
specie
{
   molWeight   7.00125;
}
thermodynamics
{
   Cp          3112.452;
   Hf          0;
}
transport
{
   mu          1.969e-05;
   Pr          0.45;
   Dab         7.5e-5;
   KT          20.3;
}
}
```

As mentioned in the previous chapter, the initial mass fraction of the reference component can be specified in a user-defined dictionary, called *initialMassFraction*, thereby, that parameter can be easily changed without recurring to solver modifications.

```
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   location    "constant";
   object      initialMassFraction;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//Initial mass fraction of component "a" N2 : mixture N2-He
Ca0                    0.5;
```

4.2 Simulation results post-processing

Since the initial mass fraction of the reference component and the maximum and minimum values of temperature are known, the equation (1.6) can be easily integrated to get the final mass fraction value of the reference component. The integrated equation is given below,

$$C_a - C_{a0} = -K_T C_{a0}(1 - C_{a0}) \ln\left[\frac{T_H}{T_C}\right] \tag{4.1}$$

By substituting the values given in table 4.1 the mass fraction value of the reference component is $C_{\{a,calc\}} = 0.5328$, this value can be compared with the maximum value of the component a curve, shown in the figure 4.2 which is approximately $C_{\{a,sim\}} = 0.54$. This gives a relative error value of approximately 1.35%.

Therefore, not only the new implementation works fine with the solver, but also it is possible to get accurate results for a simple steady state simulation case.

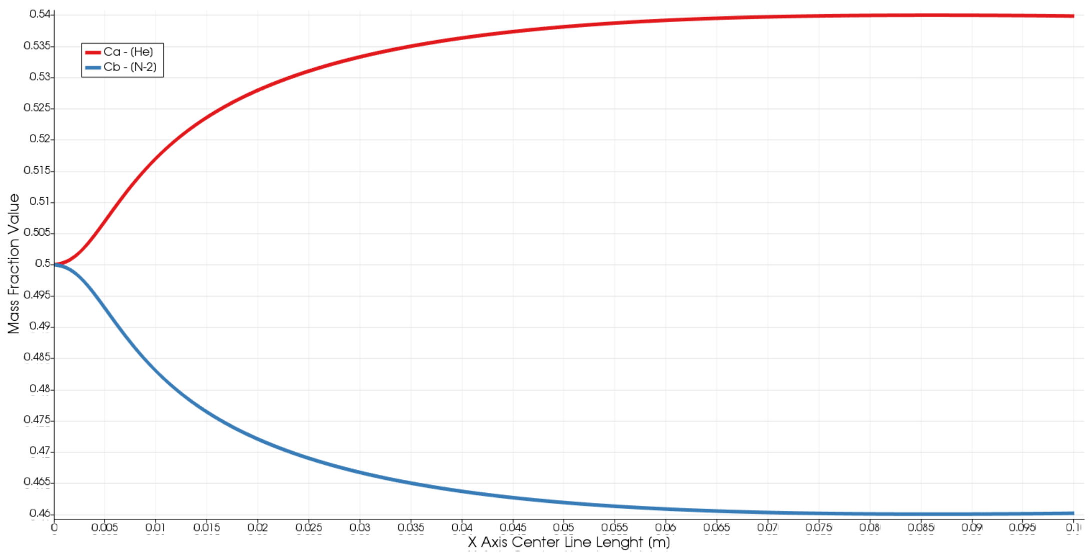

Figure 4.2: Mass fraction variation along the domain center line in the X direction, for both mixture components $(He - N_2)$

With the help of figures 4.3 and 4.4, it is easy to see how the concentration variation depends on the temperature field. While the temperature reaches a uniform gradient value along Y axis, the mass fraction of the reference component settles on an averaged value along the same direction, thus, according to equation 4.1, in a steady state flow, the concentration of species depends on the averaged temperature value and it is independent of the gradient direction. Finally, one last observation is that in accordance to the theory [4], when the value of K_T is positive, the light component will move towards the hotter region and the heavy component will move towards the coldest region, causing a spatial concentration variation of the mixture depending on the local temperature values (the contrary effect takes place when the value of K_T is negative). The reader can run his own test case by copying all the case files given in the Appendix section (Or preferably, by downloading the complementary files that come with this document).

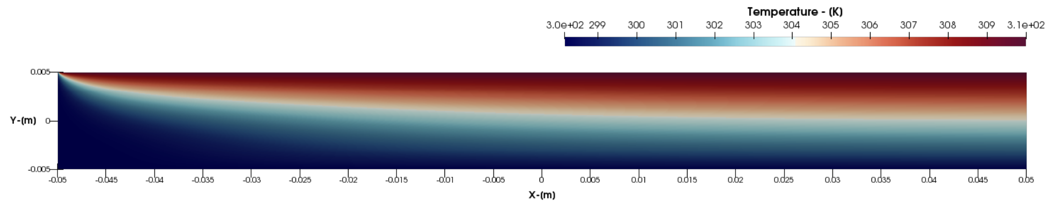

Figure 4.3: Temperature countour field.

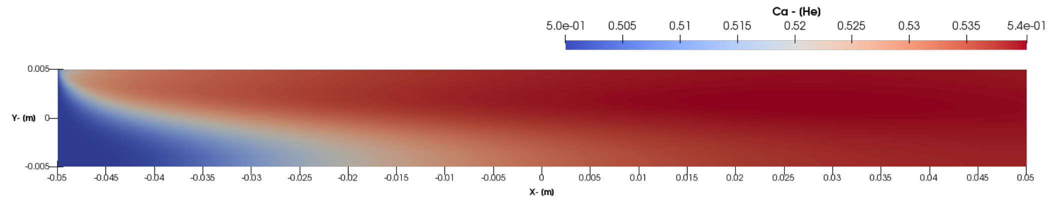

Figure 4.4: Contour field of He mass fraction.

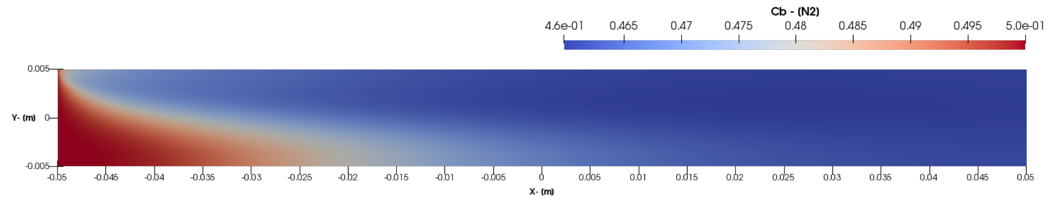

Figure 4.5: Contour field of N_2 Mass fraction.

The method shown in this report can be extended for further additions to the *thermophysicalModels* library, such as new transport models that include calculation models for a variable mass diffusivity and a variable thermodiffusion coefficient. Further coupling between these thermodiffusion transport models, and other thermodynamic and mixture models can be done to assure stronger calculations that can deliver more accurate results. The interested reader is invited to test this new implementations for different fluid mixture types and for different thermodiffusion problems, if there is an interest in extend this work for more complex problems, enquires can be made to the author via email (pmjlab@leeds.ac.uk).

Appendix: Simulation case files

system folder

controlDict file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     location    "system";
     object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     thermoDiffrhoSimpleFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
//endTime         0.4;
//deltaT          1e-04;
writeControl    adjustableRunTime;
//writeInterval   2e-3;
purgeWrite      0;
writeFormat     ascii; //binary;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
adjustTimeStep  yes;
maxCo          1;
// SIMPLE LOOP
endTime         2000;
deltaT          1;
writeInterval   10;
// ************************************************************************* //
```

blockMesh file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 1;
vertices
(
    (-0.05 -0.005 0)
    (0.05 -0.005 0)
    (0.05 0.005 0)
    (-0.05 0.005 0)
    (-0.05 -0.005 0.005)
    (0.05 -0.005 0.005)
    (0.05 0.005 0.005)
    (-0.05 0.005 0.005)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (400 40 1) simpleGrading (
    1
    (
        (0.2 0.35 4)    // 20% y-dir, 35% cells, expansion = 4
        (0.6 0.3 1)     // 60% y-dir, 30% cells, expansion = 1
        (0.2 0.35 0.25) // 20% y-dir, 35% cells, expansion = 0.25 (1/4)
    )
    1
    )
);
edges
(
);
boundary
(
topWall
{
     type wall;
     faces
     (
        (3 7 6 2)
     );
}
bottomWall
{
     type wall;
     faces
     (
      (0 1 5 4)

     );
}
inlet
{
     type patch;
     faces
     (
         (0 4 7 3)
     );
}
outlet
{
     type patch;
     faces
     (
       (2 6 5 1)
     );
}
frontAndBack
{
     type empty;
     faces
     (
         (0 3 2 1)
         (4 5 6 7)
     );
}
);
mergePatchPairs
(
);
// ************************************************************************* //
```

fvSchemes file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     location    "system";
     object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
     default         Euler;
}
gradSchemes
{
     default         leastSquares;
     grad(U)         fourth ;
}
divSchemes
{
     default         none;
     div(phi,U)      Gauss limitedLinearV 1;
     div(phi,e)      Gauss limitedLinear 1;
     div(phi,K)      Gauss limitedLinear 1;
     div(phiv,p)     Gauss limitedLinear 1;
     div(phi,Ca)     Gauss limitedLinear 1;
     div(meshPhi,p)  Gauss limitedLinear 1;
     div(phi,h)      Gauss limitedLinear 1;
     div(phi,Ekp)    Gauss limitedLinear 1;
     div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
     default                        Gauss linear corrected;
}
interpolationSchemes
{
     default     limitedLinear phi 1;
}
snGradSchemes
{
     default     orthogonal;
}
```

fvSolution file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     location    "system";
     object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
"(p|rho|Ca)"
{
    solver          GAMG;
    tolerance       1e-06;
    relTol          0.05;
    smoother        symGaussSeidel;
    nCellsInCoarsestLevel 200;
}
U
```

```
{
    solver          smoothSolver;
    smoother        symGaussSeidel;
    nSweeps         4;
    tolerance       1e-05;
    relTol          0.1;
    minIter         1;
}
h
{
    solver          PBiCGStab;
    preconditioner  DILU;
    nSweeps         2;
    tolerance       1e-05;
    relTol          0.1;
    minIter         1;
}
"(p|rho|Ca)Final"
{
    $p;
    tolerance       1e-05;
    relTol          0;
    minIter         1;
}
UFinal
{
    $U;
    tolerance       1e-05;
    relTol          0.1;
    minIter         1;
}
hFinal
{
    $e;
    tolerance       1e-05;
    relTol          0;
    minIter         1;
}
}
SIMPLE
{
    nNonOrthogonalCorrectors 4;
    rhoMin          0.3;
    rhoMax          1.4;
    pMin            90000;
    pMax            110000;
    transonic       false;
    pRefCell = 0;
    pRefValue = 1e5;
    //consistent      yes;
    residualControl
    {
        p               1e-3;\\
        U               1e-4;\\
        e               1e-3;
        Ca              1e-3;
        // possibly check turbulence fields
        // "(k|epsilon|omega)" 1e-3;
    }
    // 2.4.x
     rhoMin  rhoMin [1 -3 0 0 0 0 0] 0.3;
     rhoMax  rhoMax [1 -3 0 0 0 0 0] 1.4;
}
relaxationFactors-SIMPLE
{
     fields
     {
     p               0.3;
     rho             0.05;
     }
     equations
     {
     U               0.7;
      // "(k|epsilon)"   0.7;
      e               0.5;
     Ca              0.5;
     ".*Final"       1.0;
     }
}
relaxationFactors-PIMPLE
{
     equations
     {
       U               0.95;
     // "(k|epsilon)"   0.95;
      e               0.95;
      Ca              0.95;
        ".*Final"       1.0;
     }
}
relaxationFactors { $relaxationFactors-SIMPLE }
```

constant folder

thermoPhysicalProperties file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     location    "constant";
     object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
thermoType
{
     type            thermoDiffheRhoThermo;
     mixture         pureMixture;
     transport       thermoDiffconst;
     thermo          hConst;
     equationOfState perfectGas;
     specie          specie;
     energy          sensibleEnthalpy;
}
mixture
{
specie
{
    molWeight   7.00125;
}
thermodynamics
{
    Cp          3112.452;
    Hf          0;
}
transport
{
    mu          1.969e-05;
    Pr          0.45;
    Dab         7.5e-5;
    KT          20.3;
}
}
```

initialMassFraction file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     location    "constant";
     object      initialMassFraction;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
//Initial mass fraction of component "a" N2 : mixture N2-He
Ca0                 0.5;
```

turbulenceProperties

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       dictionary;
     location    "constant";
     object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType  laminar;
```

0 folder

Ca file

```
FoamFile
{
     version     2.0;
   format      ascii;
   class       volScalarField;
   location    "0";
   object      Ya;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 0 0 0 0];
internalField   uniform 0.5;
boundaryField
{
inlet
{
    type            uniformFixedValue;
    uniformValue    constant 0.5;
}
outlet
{
    type            inletOutlet;
    value           $internalField;
    inletValue      uniform      0;
}
topWall
{
    type            zeroGradient;
}
bottomWall
{
    type            zeroGradient;
}
frontAndBack
{
    type            empty;
}
}
```

U file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       volVectorField;
     object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
inlet
{
    type            flowRateInletVelocity;
    massFlowRate    constant 4e-06;
    value           uniform (0 0 0);
}
outlet
{
    type            inletOutlet;
    value           $internalField;
    inletValue      uniform   (0 0 0);
}
topWall
{
    type    noSlip;
}
bottomWall
{
    type     noSlip;
}
frontAndBack
{
    type     empty;
}
}
```

P file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       volScalarField;
     object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [1 -1 -2 0 0 0 0];
internalField   uniform 101325;
boundaryField
{
inlet
{
    type            zeroGradient;
}
outlet
{
    type            fixedValue;
    value           $internalField;
}
topWall
{
    type            zeroGradient;
}
bottomWall
{
    type            zeroGradient;
}
frontAndBack
{
    type            empty;
}
}
```

T file

```
FoamFile
{
     version     2.0;
     format      ascii;
     class       volScalarField;
     location    "0";
     object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 1 0 0 0];
internalField   uniform 298.15;
boundaryField
{
inlet
{
    type            uniformFixedValue;
    uniformValue    constant 298.15;
}
outlet
{
    type            inletOutlet;
    value           $internalField;
    inletValue      uniform   298.15;
}
topWall
{
    type            uniformFixedValue;
    uniformValue    constant  310;
}
bottomWall
{
    type            uniformFixedValue;
    uniformValue    constant 298.15;
}
frontAndBack
{
    type            empty;
}
}
```

Study questions

1. How do temperature gradients affect mass diffusivity in binary gas mixtures?
2. What is the thermodiffusion ratio? How is it related to mass diffusivity processes?
3. Describe briefly the structure of the *thermophysicalModels* library, how are the thermophysical properties of a fluid obtained/calculated when running a simulation case?
4. In the *thermophysicalModels* code context, why is it necessary to declare the data members and member functions of a transport model in a base thermophysical model? (it can also be a thermodynamic or state equation model).
5. In a solver code where thermophysical models are required, how are thermophysical models declared? Which file contains that declaration?
6. Explain briefly the entries of the *thermophysicalProperties* dictionary (this is related with the structure of *thermophysicalModels* library) .

Bibliography

[1] V.I. Kaljasin. Thermal diffusion. Thermopedia, http://www.thermopedia.com/content/1189/, 2019

[2] R.B. Bird, W.E. Stewart, and E.N. Lightfoot. *Transport Phenomena*. Wiley International edition. Wiley, 2006.

[3] I. Choquet. ThermophysicalModels library in openFOAM-2.3.x (or 2.4.x), www.tfd.chalmers.se/~hani/kurser/OS_CFD_2015/IsabelleChoquet/thermophysicalModels-OF-2.3-or-2.4.x.pdf, 2015.

[4] P. Costeséque, A. Mojtabi, and J.K. Platten. Thermodiffusion phenomena. *Comptes Rendus Mécanique*, 339(5):275 – 279, 2011.

[5] H. Davarzani, M. Marcoux, P. Costeséque, and M. Quintard. Experimental measurement of the effective diffusion and thermodiffusion coefficients for binary gas mixture in porous media. *Chemical Engineering Science*, 65(18):5092–5104, 2010.

[6] D. Humel. *Modifying buoyantPimpleFoam for the Simulation of Solid-Liquid Phase Change with Temperature-dependent Thermophysical Properties*, Proceedings of CFD with OpenSource Software, dx.doi.org/10.17196/OS_CFD#YEAR_2017, 2017.

[7] OpenCFD. *OpenFOAM - The Open Source CFD Toolbox - User Guide*. OpenCFD Ltd., United Kingdom, v1906 edition, 25 June 2019.

[8] J. K. Platten. The Soret Effect: A Review of Recent Experimental Results. *Journal of Applied Mechanics*, 73(1):5–15, 04 2005.

[9] M.A. Rahman and M.Z. Saghir. Thermodiffusion or soret effect: Historical review. *International Journal of Heat and Mass Transfer*, 73:693–705, 2014.