CDMATH-CoreFlows user guide

M. Ndjinga
email : michael dot ndjinga at cea dot fr

21-03-2016

1 Presentation of CDMATH-CoreFlows

CDMATH-CoreFlows is an open source C++/Python library intended at solving PDE systems arising from the thermalhydraulics of two phase flows in power plant boilers. It is a simple environment meant at students and researchers to test new numerical methods on general geometries with unstructured meshes. It is developped at CEA Saclay by Michael Ndjinga and its students since 2014 and proposes a few basic models and finite volume numerical methods. Some of the main objectives are the study of

- Numerical schemes for compressible flows at low Mach numbers
- Well balanced schemes for stiff source terms (heat source, phase change, pressure losses)
- Flow inversion and counter-current two phase flows
- Schemes that preserve the phasic volume fraction $\alpha \in [0, 1]$
- Convergence of finite volume methods
- New preconditioners for implicit methods for two phase flows
- The coupling of fluid models or multiphysics coupling (eg thermal hydraulics and neutronics or thermal hydraulics and solid thermics)

CDMATH-CoreFlows relies on the toolbox [23] of the project CDMATH [22] for the handling of meshes and fields, and on the library Petsc [21] (version 3.4.5) for the handling of large sparse matrices.

2 Software structure

CDMATH-CoreFlows is composed of 6 concrete classes dealing with specific models. They are listed in chronological order

- SinglePhase implementing the compressible Navier-Stokes equations (section 4.2)
- DriftModel implementing the 4 equation drift model (section 4.3.1)

- IsothermalTwoFluid implementing the isentropic two-fluid model (section 4.3.2)
- FiveEqsTwoFluid implementing the equal temperature two fluid model (section 4.3.3)
- TransportEquation implementing a scalar advection equation for the fluid enthalpy (section 4.1.1)
- DiffusionEquation implementing a scalar heat equation for the Uranium rods temperature (section 4.1.2)

On top of these classes there are two abstract classes that mutualise functions that are common to several models.

- ProblemFluid which contains the methods that are common to the non scalar models : SinglePhase DriftModel IsothermalTwoFluid and FiveEqsTwoFluid
- ProblemCoreFlows which contains the methods that are common to the scalar and non scalar models: ProblemFluid, TransportEquation and DiffusionEquation

Here follows an inheritance diagram of CoreFlows

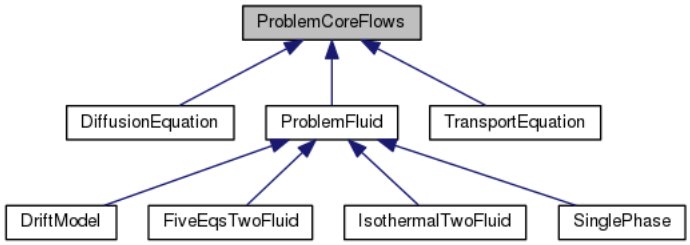

Figure 1: Inheritance diagram of CoreFlows

The program can build simple geometries and meshes using the library CDMATH [22] or read complex geometries and meshes written with the MED file system (see [18]). The output files containing the fields resulting from the calculation can be either of VTK ([20]) or MED type. One can use Paraview [19] or Salome [18] to visualise the results.
Vector and matrices structures come from the Petsc [21] library. The matrices are stored in a block sparse format (type baij in Petsc conventions). The default linear solver is GMRES and the default preconditioner is ILU, both provided by Petsc.

3 Installation and use of CDMATH-CoreFlows

CDMATH-CoreFlows is currently developped and maintained on Fedora and Ubuntu distributions. You will need the packages

- *cmake* [27] and its prerequisite *hdf5* [25],

- *doxygen* [26] if you want to generate the documentation
- *swig* [24] and *numpy* [29] if you want to use CDMATH-CoreFlows in python scripts

3.1 Download and compilation of Petsc

You need to download and compile the sources of Petsc 3.4.5. The sources of Petsc 3.4.5 can be downloaded from

- `http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.4.5.tar.gz`

Then type on the commands

```
./configure  --with-mpi=0 --download-f-blas-lapack=1
make all
make install
```

For the moment, CDMATH-CoreFlows is a sequential library (no parallelism).

3.2 Download and compilation of CDMATH

In order to download CDMATH [22]

- either unzip the following file to a directory cdmath_src
  - `https://github.com/PROJECT-CDMATH/CDMATH/archive/master.zip`
- or

```
git clone https://github.com/PROJECT-CDMATH/CDMATH.git cdmath_src
```

In order to compile CDMATH [22] you will need the libraries *cmake* [27], and *hdf5* [25] plus *swig* [24] and *numpy* [29] if you intend to use CDMATH functions in your python scripts. First create build and install repositories:

```
mkdir ~/workspace/cdmath
cd ~/workspace/cdmath
mkdir cdmath_build
mkdir cdmath_install
```

Go to the build directory

```
cd cdmath_build
```

Then run the commands

```
cmake ../cdmath_src/ -DCMAKE_INSTALL_PREFIX=../cdmath_install -DCMAKE_BUILD_TYPE=Release
make
make install
```

3.3 Download and compilation of CoreFlows

In order to download CoreFlows source files, unzip the following file to a directory CoreFlows-master (not yet available)

- `https://github.com/PROJECT-CoreFlows/CoreFlows/archive/master.zip`

The following steps assume that petsc (version 3.4.5) [21] and CDMATH [22] are installed on your computer.

In CoreFlows-master, open the file CoreFlows.sh and set the variables :

- *PETSC_DIR*, the path to your PETSC installation
- *PETSC_ARCH*, the type of installation used (usually arch-linux2-c-opt or arch-linux2-c-debug)
- *CDMATH_INSTALL*, the path to your CDMATH installation
- *CoreFlows_ROOT*, the path to the CoreFlows-master directory
- *CoreFlows_PYTHON*, set to 'ON' if you intend to use python script, 'OFF' otherwise
- *CoreFlows_Doc*, set to 'ON' if you want the doxygen documentation to be generated, 'OFF' otherwise

Once the file CoreFlows.sh has been edited, you can open a terminal, go to CoreFlows-master and type

- *./CoreFlows_install.sh* to compile the library CoreFlows

3.4 Use of CoreFlows

First load CoreFlows environment from the CoreFlows-master directory

```
source CoreFlows-master/CoreFlows.sh
```

- If you use C language: edit the file CoreFlows-master/CoreFlows_src/main.cxx then in a terminal type

  ```
  cd CoreFlows-master/CoreFlows_build
  make -j
  make install -j
  ```

  Then you can run the simulation in any directory with the command line

  ```
  $CoreFlows
  ```

- If you use python language: edit your own python file my_file.py following for example the pattern of the file CoreFlows-master/CoreFlows_src/main.py. Then in a terminal type

  ```
  python my_file.py
  ```

- If you use the graphic interface, you need to run a Salomé Unix shell and type the command line

  ```
  runSalome -mCOREFLOWS
  ```

  then click on new study to open CoreFlows interface

4 The physical models

The physical models are presented in order of mathematical complexity: the linear scalar problem (transport and diffusion equations) then the Navier-Stokes model and then the two-phase flow models.

4.1 Scalar models

4.1.1 The transport equation

$$\partial_t h + \vec{u} \cdot \vec{\nabla} h = \Phi + \lambda_{sf}(T_s - T) \tag{1}$$

where

- h the main unknown is the fluid enthalpy field
- $\vec{u}$ is the constant transport velocity, set by the user
- Φ is the heat source term if explicitely known
- T_s is the solid temperature field provided by the user if $\lambda_{sf} \neq 0$
- $T = T_0 + \frac{H - H_0}{c_p}$ is the fluid temperature field
- λ_{sf} is the fluid-solid heat transfer coefficient provided by the user
- c_p is the fluid specific heat, possibly provided by the user and assumed constant

The class TransportEquation implements a scalar advection equation for the enthalpy of a fluid. The fluid can be either steam or liquid water around 1 bar or 155 bars.

4.1.2 The diffusion equation

$$\partial_t T = d \triangle T + \frac{\Phi + \lambda_{sf}(T_f - T)}{\rho c_p} \tag{2}$$

where

- T the main unknown is the solid temperature field
- λ is the solid thermal conductivity, possibly set by the user
- ρ is the solid density assumed constant, possibly set by the user
- c_p is the solid specific heat, possibly by the user and assumed constant
- $d = \frac{\lambda}{\rho c_p}$ is the solid diffusivity

- Φ is the heat source term if explicitely known
- T_f is the fluid temperature field provided by the user

The class DiffusionEquation implementing a scalar diffusion equation for the temperature in a solid. The default values for ρ, c_p, λ are those of Uranium oxyde around $900K$.

4.2 The Navier-Stokes equations

The model consists of the following three balance laws for the mass, the momentum and the energy:

$$\left\{\begin{array}{lcllcl} \frac{\partial \phi\rho}{\partial t} & + & \nabla\cdot\phi\vec{q} & & = & 0 \\ \frac{\partial \phi\vec{q}}{\partial t} & + & \nabla\cdot\left(\phi\vec{q}\otimes\frac{\vec{q}}{\rho}\right)+\phi\vec{\nabla}p & - \quad \nu\nabla\cdot(\phi\vec{\nabla}\vec{u}) & = & \phi\rho\vec{g}-\phi K\rho||\vec{u}||\vec{u} \\ \frac{\partial(\phi\rho E)}{\partial t} & + & \nabla\cdot\left[\phi(\rho E+p)\frac{\vec{q}}{\rho}\right] & - \quad \lambda\nabla\cdot(\phi\vec{\nabla}T) & = & \Phi+\phi\rho\vec{g}\cdot\vec{u}-\phi K\rho||\vec{u}||^3 \end{array}\right. \tag{3}$$

where ρ is the density, $\vec{u}$ the velocity, $\vec{q}=\rho\vec{u}$ the momentum, p the pressure, ρe the internal energy, $\rho E=\rho e+\frac{||\vec{q}||^2}{2\rho}$ the total energy, T the absolute temperature, Φ a heat source term, ν the viscosity and λ the thermal conductivity. We close the system (3) by a stiffened gas law $p=(\gamma-1)\rho e-\gamma p_0$ and a linearised internal energy law $e(T)$ calibrated around the points $(P=1bar, T=300K)$ or $(P=155bars, T=618K)$ depending on the enum pressureEstimate. For the sake of simplicity, we consider constant viscosity and conductivity, and neglect the contribution of viscous forces in the energy equation.
The parameters $\lambda, \nu, \vec{g}, K$ and Φ can be set by the user.

4.3 Two phase flow models

We present the homogeneised two phase flow models implemented in CoreFlows. This models are obtained by averaging the balance equations for each separated phase or for the mixture, using space, time or ensemble averaged quantities (see [9] and [10]). The drift model is used in the thermal hydraulics software Flica 4 (see [5]), whilst the two-fluid models are used in Cathare [6], Neptune_CFD [7], Cobra-TF [2], Relap5 [1].

4.3.1 The Drift model

The model consists in the steam mass balance equation together with the mixture mass conservation, the mixture momentum balance and mixture energy balance equations. The main unknowns are the steam mass concentration c_v, the pressure P, the mixture velocity $\vec{u}_m$, and the common temperature T. The model uses stiffened gas laws $p_g(\rho_g, T)$ and $p_l(\rho_l, T)$ as well as linearised internal energy law $e_k(T)$ calibrated around the saturation points $(P=1bar, T=373K)$ $(P=155bars, T=345K)$ depending on the value of the enum pressureEstimate.
The drift model is a system of four nonlinear equations taking the following

conservative form

$$\left\{\begin{array}{lll}
\partial_t\phi(\alpha_g\rho_g+\alpha_l\rho_l) & +\nabla\cdot\phi(\alpha_g\rho_g{}^t\vec{u}_g+\alpha_l\rho_l{}^t\vec{u}_l) & =0\\
\partial_t\phi(\alpha_g\rho_g) & +\nabla\cdot\phi(\alpha_g\rho_g{}^t\vec{u}_g) & =\phi\Gamma_g(h_m,\Phi)\\
\partial_t\phi(\alpha_g\rho_g\vec{u}_g+\alpha_l\rho_l\vec{u}_l) & +\nabla\cdot\phi(\alpha_g\rho_g\vec{u}_g\otimes\vec{u}_g+\alpha_l\rho_l\vec{u}_l\otimes\vec{u}_l)+\phi\vec{\nabla}p & =p\vec{\nabla}\phi+\phi\rho_m\vec{g}-\phi K_g\alpha_g\rho_g||\vec{u}_g||\vec{u}_g\\
 & & -\phi K_l\alpha_l\rho_l||\vec{u}_l||\vec{u}_l\\
\partial_t\phi(\alpha_g\rho_gE_g+\alpha_l\rho_lE_l) & +\nabla\cdot\phi(\alpha_g\rho_gH_g{}^t\vec{u}_g+\alpha_l\rho_lH_l{}^t\vec{u}_l) & =\Phi+\phi\rho\vec{g}\cdot\vec{u}-\phi K_g\alpha_g\rho_g||\vec{u}_g||^3\\
 & & -\phi K_l\alpha_l\rho_l||\vec{u}_l||^3
\end{array}\right. ,$$

where the total energy and total enthalpy are defined by

$$E_k=e_k+\frac{1}{2}|\vec{u}_k|^2\quad,\qquad H_k=h_k+\frac{1}{2}|\vec{u}_k|^2,\qquad k=v,l,$$

where e_k is the internal energy, and $h_k=e_k+\frac{p}{\rho_k}$ the enthalpy associated to phase k and

$$\begin{aligned}
\rho_m &= \alpha_g\rho_g+\alpha_l\rho_l\\
\vec{u}_m &= \frac{\alpha_g\rho_g\vec{u}_g+\alpha_l\rho_l\vec{u}_l}{\alpha_g\rho_g+\alpha_l\rho_l}\\
h_m &= \frac{\alpha_g\rho_gh_g+\alpha_l\rho_lh_l}{\alpha_g\rho_g+\alpha_l\rho_l}.
\end{aligned}$$

We need a drift correlation for the relative velocity:

$$\vec{u}_r=\vec{u}_g-\vec{u}_l=\vec{f_r}(c_g,\vec{u}_m,\rho_m).$$

The phase change is modeled using the formula

$$\Gamma_g=\left\{\begin{array}{ll}\frac{\Phi}{\mathcal{L}} & \text{if } h_l^{sat}\le h<h_g^{sat} \text{ and } 0<\alpha_g<1\\ 0 & \text{otherwise}\end{array}\right. . \tag{4}$$

The parameters $\lambda_k,\nu_k,\vec{g},K_k$ and Φ can be set by the user.

4.3.2 The isothermal two-fluid model

The model consists in the phasic mass and momentum balance equations. The main unknowns are α, P, $\vec{u}_g$, $\vec{u}_l$. The model uses stiffened gas laws $p_g(\rho_g)$ and $p_l(\rho_l)$ calibratedd around the points $(1bar,300K)$ or $(155bars,618K)$ depending on the value of the enum pressureEstimate.

The subscript k stands for l for the liquid phase and g for the gas phase. The common averaged pressure of the two phases is denoted by p. In our model, pressure equilibrium between the two phases is postulated, and the resulting system to solve is:

$$\left\{\begin{array}{lll}
\frac{\partial m_g}{\partial t} + \nabla\cdot\vec{q}_g & =0,\\
\frac{\partial m_l}{\partial t} + \nabla\cdot\vec{q}_l & =0,\\
\frac{\partial\vec{q}_g}{\partial t} + \nabla\cdot(\vec{q}_g\otimes\frac{\vec{q}_g}{m_g})+\alpha_g\vec{\nabla}p & \\
+\Delta p\nabla\alpha_g-\nu_g\Delta\vec{u}_g & =m_g\vec{g}-K_gm_g||\vec{u}_g||\vec{u}_g\\
\frac{\partial\vec{q}_l}{\partial t} + \nabla\cdot(\vec{q}_l\otimes\frac{\vec{q}_l}{m_l})+\alpha_l\vec{\nabla}p & \\
+\Delta p\nabla\alpha_l-\nu_l\Delta\vec{u}_l & =m_l\vec{g}-K_lm_l||\vec{u}_l||\vec{u}_l,
\end{array}\right. \tag{5}$$

where $\alpha_g + \alpha_l = 1$, $m_k = \alpha_k \rho_k$ and $\vec{q}_k = \alpha_k \rho_k \vec{u}_k$. Here, ν_k is the viscosity of phase k, and Δp denotes the pressure default $p - p_k$ between the bulk average pressure and the interfacial average pressure.
The parameters $\lambda_k, \nu_k, \vec{g}, K_k$ and Φ can be set by the user.

4.3.3 The five equation two-fluid model

The model consists in the phasic mass and momentum balance equations and one mixture total energy balance equation. The main unknowns are α,P,$\vec{u}_g$,$\vec{u}_l$ and $T = T_g = T_l$. The model uses stiffened gas laws $p_g(\rho_g, T)$ and $p_l(\rho_l, T)$ as well as linearised internal energy law $e_k(T)$ calibratedd around the saturation points $(P = 1bar, T = 373K)$ or $(P = 155bars, T = 345K)$ depending on the enum pressureEstimate.

$$\left\{\begin{array}{lcll} \frac{\partial m_g}{\partial t} & + & \nabla \cdot \vec{q}_g & = \Gamma_g(h_g, \Phi), \\ \frac{\partial m_l}{\partial t} & + & \nabla \cdot \vec{q}_l & = \Gamma_l(h_l, \Phi), \\ \frac{\partial \vec{q}_g}{\partial t} & + & \nabla \cdot (\vec{q}_g \otimes \frac{\vec{q}_g}{m_g}) + \alpha_g \nabla p & \\ & + & \Delta p \nabla \alpha_g - \nu_g(\Delta \frac{\vec{q}_g}{m_g}) & = m_g \vec{g} - K_g m_g ||\vec{u}_g|| \vec{u}_g \\ \frac{\partial \vec{q}_l}{\partial t} & + & \nabla \cdot (\vec{q}_l \otimes \frac{\vec{q}_l}{m_l}) + \alpha_l \nabla p & \\ & + & \Delta p \nabla \alpha_l - \nu_l(\Delta \frac{\vec{q}_l}{m_l}) & = m_l \vec{g} - K_l m_l ||\vec{u}_l|| \vec{u}_l, \\ \partial_t \rho_m E_m & + & \nabla \cdot (\alpha_g \rho_g H_g{}^t \vec{u}_g + \alpha_l \rho_l H_l{}^t \vec{u}_l) & = \Phi + \rho \vec{g} \cdot \vec{u} - K_g m_g ||\vec{u}_g||^3 - K_l m_l ||\vec{u}_l||^3 \end{array}\right. \tag{6}$$

where

$$\begin{aligned} \rho_m &= \alpha_g \rho_g + \alpha_l \rho_l \\ E_m &= \frac{\alpha_g \rho_g E_g + \alpha_l \rho_l E_l}{\alpha_g \rho_g + \alpha_l \rho_l}. \end{aligned}$$

The phase change is modeled using the formula

$$\Gamma_g = \left\{\begin{array}{cc} \frac{\Phi}{\mathcal{L}} & \text{if } h_l^{sat} \leq h < h_g^{sat} \text{ and } 0 < \alpha_g < 1 \\ 0 & \text{otherwise} \end{array}\right. . \tag{7}$$

The parameters $\lambda_k, \nu_k, \vec{g}, K_k$ and Φ can be set by the user.

5 The numerical methods

CDMATH-CoreFlows proposes a variety of finite volume methods (see [17] for an introduction). The time discretisation can be explicit or implicit, whereas the space discretisation can be upwind or centred as in [16] or lowMach, pressureCorreection, staggered for low Mach flows. The flux formulation for non scalar fluid model is either the Roe formulation [11], the VFRoe formulation [12] or the VFFC formulation [13]. It is also possible to add an entropic correction [15] and/or use source upwinding [14].
The finite volume discretisation allows an easy handling of general geometries and meshes generated by Salomé.

Explicit schemes are used in general for fast dynamics solved with small time steps while implicit schemes allow the use of large time step to quickly reach the stationary regime. The implicit schemes result in nonlinear systems that are solved using a Newton type method.
The upwind scheme is the basic scheme but options are available to use a centered scheme (second order in space) or entropic corrections.

Our models can be written in generic form as a nonlinear system of balance laws:

$$\frac{\partial U}{\partial t} + \nabla \cdot (\mathcal{F}^{conv}(U)) + \nabla \cdot \left(\mathcal{F}^{diff}(U)\right) = S(U,x), \tag{8}$$

where U is the vector of conservative unknowns, $\mathcal{F}^{conv}$ is the convective flux and $\mathcal{F}^{diff}$ the diffusive flux.

We decompose the computational domain into N disjoint cells C_i with volume v_i. Two neighboring cells C_i and C_j have a common boundary ∂C_{ij} with area s_{ij}. We denote $N(i)$ the set of neighbors of a given cell C_i and $\vec{n}_{ij}$ the exterior unit normal vector of ∂C_{ij} . Integrating the system (8) over C_i and setting $U_i(t) = \frac{1}{v_i}\int_{C_i} U(x,t)dx$, the semi-discrete equations can be written:

$$\frac{\mathrm{d}U_i}{\mathrm{d}t} + \sum_{j\in N(i)} \frac{s_{ij}}{v_i}\left(\overrightarrow{\Phi}_{ij}^{conv} + \overrightarrow{\Phi}_{ij}^{diff}\right) = S_i(U,x). \tag{9}$$

with: $\overrightarrow{\Phi}_{ij}^{con} = \frac{1}{s_{ij}}\int_{\partial C_{ij}} \mathcal{F}^{conv}(U).\vec{n}_{ij}ds$, $\overrightarrow{\Phi}_{ij}^{diff} = \frac{1}{s_{ij}}\int_{\partial C_{ij}} \mathcal{F}^{diff}(U).\vec{n}_{ij}ds$.

To approximate the convection numerical flux $\overrightarrow{\Phi}_{ij}^{conv}$ we solve an approximate Riemann problem at the interface ∂C_{ij}.

- Using the Roe local linearisation of the fluxes [11], we obtain the following formula:

$$\begin{aligned}\overrightarrow{\Phi}_{ij}^{conv} &= \frac{\mathcal{F}^{conv}(U_i)+\mathcal{F}^{conv}(U_j)}{2}.\vec{n}_{ij} - \mathcal{D}(U_i,U_j)\frac{U_j-U_i}{2} \\ &= \mathcal{F}^{conv}(U_i)\vec{n}_{ij} + A^-(U_i,U_j)(U_j-U_i),\end{aligned} \tag{10}$$

- Using the VFRoe [12] local linearisation of the fluxes the formula is:

$$\overrightarrow{\Phi}_{ij}^{conv,VFRoe} = \mathcal{F}^{conv}\left(\frac{U_i+U_j}{2} - \mathcal{D}(U_i,U_j)\frac{U_j-U_i}{2}\right)\vec{n}_{ij}, \tag{11}$$

- Using the VFFC [13] local linearisation of the fluxes, the formula is:

$$\overrightarrow{\Phi}_{ij}^{conv,VFFC} = \frac{\mathcal{F}^{conv}(U_i)+\mathcal{F}^{conv}(U_j)}{2}\vec{n}_{ij} - \mathcal{D}(U_i,U_j)\frac{\mathcal{F}^{conv}(U_j)-\mathcal{F}^{conv}(U_i)}{2}\vec{n}_{ij} \tag{12}$$

where $\mathcal{D}$ is an upwinding matrix, $A(U_i,U_j)$ the Roe matrix and $A^- = \frac{A-\mathcal{D}}{2}$. The choice $\mathcal{D}=0$ gives the centered scheme, whereas $\mathcal{D}=|A|$ gives the upwind scheme.

The diffusion numerical flux $\overrightarrow{\Phi}_{ij}^{diff}$ is approximated on structured meshes using the formula:

$$\overrightarrow{\Phi}_{ij}^{diff} = D(\frac{U_i+U_j}{2}, \vec{n}_{ij})(U_j-U_i), \tag{13}$$

where $D(U,\vec{n}_{ij}) = \nabla\mathcal{F}^{diff}(U).\vec{n}_{ij}$ is the matrix of the diffusion tensor. (13) is not accurate for highly non structured or non conforming meshes. However, since we are mainly interested in convection driven flows, we do not ask for a very precise scheme.

Finally, since $\sum_{j\in N(i)} \mathcal{F}^{conv}(U_i).\vec{n}_{ij} = 0$, using (10) and (13) the equation (9) of the semi-discrete scheme becomes:

$$\frac{\mathrm{d}U_i}{\mathrm{d}t} + \sum_{j\in N(i)} \frac{s_{ij}}{v_i}\{(A^- + D)(U_i,U_j)\}(U_j - U_i) = S_i(U,x). \tag{14}$$

The source term in (14) can be approximated using either a

$$\text{Centered source (default)} S_i = S(U_i) \tag{15}$$

or an

$$\begin{aligned}\text{Upwind source (WellBalanced option) } S_i &= \frac{1}{2}(Id - signe(A^{Roe}_{i,i+1}))\frac{S(U_i)) + S(U_{i+1})}{2} \\ &+\frac{1}{2}(Id + signe(A^{Roe}_{i-1,i}))\frac{S(U_{i-1})) + S(U_i)}{2}.\end{aligned} \tag{16}$$

5.1 Explicit schemes

In explicit schemes, in order to compute the values U_i^{n+1}, the fluxes Φ_{ij}^{conv}, Φ_{ij}^{diff} and the source term $S(U,x)$ in (9) are evaluated at time n :

$$\begin{aligned}\frac{U_i^{n+1} - U_i^n}{\Delta t} &+ \sum_{j\in N(i)} \frac{s_{ij}}{v_i}\left(\frac{1}{2}(\mathcal{F}^{conv}(U_i^n) + \mathcal{F}^{conv}(U_j^n)).\vec{n}_{ij} - \mathcal{D}(U_i^n,U_j^n,\vec{n}_{ij})\frac{U_j^n - U_i^n}{2}\right) \\ &+ \frac{s_{ij}}{v_i}D(\frac{U_i+U_j}{2},\vec{n}_{ij})(U_j - U_i) = S(U^n,x_i),\end{aligned}$$

or equivalently using (10) and (13)

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + \sum_{j\in N(i)} \frac{s_{ij}}{v_i}\{(A^- + D)(U_i^n,U_j^n,\vec{n}_{ij})\}(U_j^n - U_i^n) = S(U^n,x_i). \tag{17}$$

From the system (17) we can obtain U_i^{n+1} easily using matrix-vector products and vector sum. However the time step of explicit methods is constrained by the CFL condition for stability reasons.

5.2 Implicit schemes

In implicit schemes, in order to compute the values U_i^{n+1}, the fluxes Φ_{ij}^{conv}, Φ_{ij}^{diff} and the source term $S(U,x)$ in (9) are evaluated at time $n+1$:

$$\begin{aligned}\frac{U_i^{n+1} - U_i^n}{\Delta t} &+ \sum_{j\in N(i)} \frac{s_{ij}}{v_i}\left(\frac{1}{2}(\mathcal{F}^{conv}(U_i^{n+1}) + \mathcal{F}^{conv}(U_j^{n+1})).\vec{n}_{ij} - \mathcal{D}(U_i^{n+1},U_j^{n+1},\vec{n}_{ij})\frac{U_j^{n+1} - U_i^{n+1}}{2}\right) \\ &+ \frac{s_{ij}}{v_i}D(\frac{U_i+U_j}{2},\vec{n}_{ij})(U_j - U_i) = S(U^{n+1},x_i),\end{aligned}$$

or equivalently using (10) and (13)

$$\frac{U_i^{n+1}-U_i^n}{\Delta t}+\sum_{j\in N(i)}\frac{s_{ij}}{v_i}\{(A^-(U_i^{n+1},U_j^{n+1},\vec{n}_{ij})+D)\}(U_j^{n+1}-U_i^{n+1})=S(U^{n+1},x_i). \tag{18}$$

The system (18) is nonlinear. The computation of U_i^n is more expensive but we can expect to use larger time steps than would be allowed with the explicit scheme.
We use the following Newton iterative method to obtain the required solutions:

$$\begin{aligned}\frac{\delta U_i^{k+1}}{\Delta t} &+ \sum_{j\in N(i)}\frac{s_{ij}}{v_i}\left[(A^-(U_i^k,U_j^k)+D)\right]\left(\delta U_j^{k+1}-\delta U_i^{k+1}\right)\\ &= -\frac{U_i^k-U_i^n}{\Delta t}-\sum_{j\in N(i)}\frac{s_{ij}}{v_i}\left[(A^-(U_i^k,U_j^k)+D)\right](U_j^k-U_i^k),\end{aligned}$$

where $\delta U_i^{k+1}=U_i^{k+1}-U_i^k$ is the variation of the k-th iterate that approximate the solution at time $n+1$. Defining the unknown vector $\mathcal{U}=(U_1,\ldots,U_N)^t$, each Newton iteration for the computation of $\mathcal{U}$ at time step $n+1$ requires the numerical solution of the following linear system:

$$\mathcal{A}(\mathcal{U}^k)\delta\mathcal{U}^{k+1}=b(\mathcal{U}^n,\mathcal{U}^k). \tag{19}$$

5.3 Numerical scheme for the Navier-Stokes equations

For the Navier-Stokes equations with constant porosity $\phi=1$, $U=(\rho,\vec{q},\rho E)^t$ and the fluxes in (3) write $\mathcal{F}^{conv}(U)=\begin{pmatrix}\vec{q}\\ \vec{q}\otimes\frac{\vec{q}}{\rho}+p\mathbb{I}_d\\ (\rho E+p)\frac{\vec{q}}{\rho}\end{pmatrix}$, $\mathcal{F}^{diff}(U)=\begin{pmatrix}0\\ -\nu\vec{\nabla}(\frac{\vec{q}}{\rho})\\ -\lambda\vec{\nabla}T\end{pmatrix}$.

For the Euler equations, we can build the Roe matrix $A(U_i,U_j)$ explicitly using the Roe averaged state $U_{Roe}(U_i,U_j)=(\tilde{\rho},\tilde{\rho}\tilde{u},\tilde{\rho}\tilde{E}=\tilde{\rho}\tilde{H}-\tilde{p})^t$ defined by

$$\begin{aligned}\tilde{\rho} &= \sqrt{\rho_i\rho_j}\\ \tilde{u} &= \frac{\sqrt{\rho_i}u_i+\sqrt{\rho_j}u_j}{\sqrt{\rho_i}+\sqrt{\rho_j}}\\ \tilde{H} &= \frac{\sqrt{\rho_i}H_i+\sqrt{\rho_j}H_j}{\sqrt{\rho_i}+\sqrt{\rho_j}}.\end{aligned}$$

The Roe matrix writes (see [17])

$$A_{Roe}(U_i,U_j)=\nabla\mathcal{F}^{conv}(U_{Roe}(U_i,U_j))\vec{n}_{ij}=\begin{pmatrix}0 & 1 & 0\\ \tilde{\chi}+(\frac{1}{2}\tilde{\kappa}-1)\tilde{u}^2 & (2-\tilde{\kappa})\tilde{u} & \tilde{\kappa}\\ (\tilde{\chi}+\frac{1}{2}\tilde{\kappa}\tilde{u}^2-\tilde{H})\tilde{u} & \tilde{H}-\tilde{\kappa}\tilde{u}^2 & (\tilde{\kappa}+1)\tilde{u}\end{pmatrix}$$

The diffusion numerical flux $\overrightarrow{\Phi}_{ij}^{diff}$ is approximated with the formula:

$$\overrightarrow{\Phi}_{ij}^{diff}=D(\frac{U_i+U_j}{2})(U_j-U_i) \tag{20}$$

with the matrix $D(U) = \begin{pmatrix} 0 & \vec{0} & 0 \\ \frac{\nu\vec{q}}{\rho^2} & \frac{-\nu}{\rho}\mathbb{I}_d & 0 \\ \frac{\lambda}{c_v}\left(\frac{c_v T}{\rho} - \frac{||\vec{q}||^2}{2\rho^3}\right) & \frac{\vec{q}^{\,t}\lambda}{\rho^2 c_v} & -\frac{\lambda}{c_v \rho} \end{pmatrix}$, where c_v is the heat capacity at constant volume.

5.4 Boundary conditions

CoreFlows can manage four types of boundary conditions: Neumann, Wall, Inlet and Outlet. Inlet and outlet boundary condition require some input parameters from the user that depend on the model.
Boundary conditions are treated via a fictitious ghost cell located on the other side of the boundary face. The conservative and primitive states $U_b(U), V_b(V)$ of the ghost cell can be determined from the corresponding values U, V in the boundary cell located inside the domain. The functions $U_b(U)$ and $V_b(V)$ may require data provided by the user (inlet and outlet boundary conditions). The numerical flux can then be then obtainedat the boundary between the ghost cell and the cell inside the domain.
If the scheme is implicit, we need to provide the jacobian matrix ∇U_b of the function U_b. In order to determine the jacobian ∇U_b for Inlet and oulet boundary conditions, we assume a stiffened gas equation of state $p = (\gamma - 1)\rho e - \gamma p_0$ and a linear internal energy law depending only on the temperature : $e(T) = e_0 + c_0^k(T - T_0), k = g, l$ with a constant specific heat c_0^k.

The boundary conditions are specific to each model (see for example 6.7). In the following we describe we give specific detail of the treatment of the convection part of each model.

5.4.1 Navier-Stokes

In this case the system consists in three balance laws (see section 4.2).

- Neumann :

$$V_b(V) = V \quad , \qquad U_b(U) = U$$

$$\nabla U_{b,Neumann} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \mathbb{I}_d & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Wall : we assume that the wall has local unit normal $\vec{n}$

$$V_b : \begin{pmatrix} p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} p \\ \vec{u}_m - 2(\vec{u}_m \cdot \vec{n})\vec{n} \\ T \end{pmatrix} \quad , \qquad U_b : \begin{pmatrix} \rho_m \\ \rho_m \vec{u}_m \\ \rho_m E_m \end{pmatrix} \to \begin{pmatrix} \rho_m \\ \rho_m(\vec{u}_m - 2(\vec{u}_m \cdot \vec{n})\vec{n}) \\ \rho_m E_m \end{pmatrix}$$

$$\nabla U_{b,Wall} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \mathbb{I}_d - 2\vec{n} \otimes \vec{n} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Inlet :
  - First case: the velocity $\vec{u}_e$ and temperature T_e are imposed

$$V_b : \begin{pmatrix} p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} p \\ \vec{u}_e \\ T_e \end{pmatrix}, \quad U_b : \begin{pmatrix} \rho \\ \rho\vec{u}_m \\ \rho E \end{pmatrix} \to \begin{pmatrix} \rho_e \\ \rho_e\vec{u}_e \\ \rho_e E_e \end{pmatrix}$$

with

$$\begin{aligned} \rho_e &= \rho(p, T_e) \\ E_e &= e(T_e) + \frac{1}{2}|\vec{u}_e|^2 \end{aligned}$$

and

$$\begin{aligned} \mathrm{d}\rho_e &= \frac{1}{(\gamma-1)e(T_e)}\mathrm{d}p(\rho, \rho E) \\ &= \frac{1}{e(T_e)}\mathrm{d}\left(\rho E - \frac{1}{2}\rho||\vec{u}_m||^2\right) \\ &= \frac{||\vec{u}_m||^2}{2e(T_e)}\mathrm{d}\rho - \frac{\vec{u}_m}{e(T_e)} \cdot \mathrm{d}\vec{q} + \frac{1}{e(T_e)}\mathrm{d}(\rho E). \end{aligned}$$

Hence

$$\nabla U_{b,Inlet\,u} = \begin{pmatrix} \frac{||\vec{u}_m||^2}{2e(T_e)} & -\frac{{}^t\vec{u}_m}{e(T_e)} & \frac{1}{e(T_e)} \\ \frac{||\vec{u}_m||^2}{2e(T_e)}\vec{u}_e & -\frac{\vec{u}_e \otimes \vec{u}_m}{e(T_e)} & \frac{1}{e(T_e)}\vec{u}_e \\ \frac{E_e||\vec{u}_m||^2}{2e(T_e)} & -\frac{E_e\,{}^t\vec{u}_m}{e(T_e)} & \frac{E_e}{e(T_e)} \end{pmatrix}$$

The final expression obtained for $\nabla U_{b,Inlet\,u}$ does not involve any parameter of the stiffened gas law.

  - Second case: the pressure p_e and temperature T_e are imposed

$$V_b : \begin{pmatrix} p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} p_e \\ \vec{u}_m \\ T_e \end{pmatrix}, \quad U_b : \begin{pmatrix} \rho \\ \rho\vec{u}_m \\ \rho E \end{pmatrix} \to \begin{pmatrix} \rho_e \\ \rho_e\vec{u}_m \\ \rho_e E_e \end{pmatrix}$$

with

$$\begin{aligned} \rho_e &= \rho(p_e, T_e) \\ E_e &= e(T_e) + \frac{1}{2}||\vec{u}_m||^2 \end{aligned}$$

and

$$\begin{aligned} \mathrm{d}\rho_e\vec{u}_m &= \rho_e\mathrm{d}\frac{\rho\vec{u}_m}{\rho} \\ &= \frac{\rho_e}{\rho}\mathrm{d}(\rho\vec{u}_m) - \frac{\rho_e\vec{u}_m}{\rho}\mathrm{d}\rho \\ \mathrm{d}(\rho_e E_e) &= \rho_e\vec{u}_m \cdot \mathrm{d}\vec{u}_m \\ &= \frac{\rho_e\vec{u}_m}{\rho} \cdot \mathrm{d}(\rho\vec{u}_m) - \frac{\rho_e||\vec{u}_m||^2}{\rho}\mathrm{d}\rho. \end{aligned}$$

Hence

$$\nabla U_{b,Inlet\,p} = \begin{pmatrix} 0 & 0 & 0 \\ -\frac{\rho_e}{\rho}\vec{u}_m & \frac{\rho_e}{\rho}\mathbb{I}_d & 0 \\ -\frac{\rho_e||\vec{u}_m||^2}{\rho} & \frac{\rho_e}{\rho}{}^t\vec{u}_m & 0 \end{pmatrix}$$

The derivation of $\nabla U_{b,Inlet\,p}$ does not require the assumption of a stiffened gas law.

- Outlet : pressure p_s is imposed

$$V_b : \begin{pmatrix} p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} p_s \\ \vec{u}_m \\ T \end{pmatrix}, \quad U_b : \begin{pmatrix} \rho \\ \rho\vec{u}_m \\ \rho E \end{pmatrix} \to \begin{pmatrix} \rho_s \\ \rho_s\vec{u}_m \\ \rho_s E \end{pmatrix}$$

with

$$\rho_s = \rho(p_s, T).$$

and

$$\begin{aligned} \mathrm{d}\rho_s &= \mathrm{d}\left(\frac{p_s+\gamma p_0}{(\gamma-1)e}\right) = \frac{p_s+\gamma p_0}{\gamma-1}\mathrm{d}\frac{\rho}{\rho e} \\ &= \frac{p_s+\gamma p_0}{\gamma-1}\left(\frac{1}{\rho e}\mathrm{d}\rho - \frac{1}{\rho e^2}\mathrm{d}(\rho e)\right) \\ &= \frac{p_s+\gamma p_0}{(\gamma-1)\rho e}\left(\mathrm{d}\rho - \frac{1}{e}\mathrm{d}(\rho E - \frac{1}{2}\rho||\vec{u}_m||^2)\right) \\ &= \frac{(\gamma-1)\rho_s e}{(\gamma-1)\rho e}\left(\mathrm{d}\rho - \frac{1}{e}\left(\frac{||\vec{u}_m||^2}{2}\mathrm{d}\rho - \vec{u}_m\cdot\mathrm{d}\vec{q} + \mathrm{d}(\rho E)\right)\right) \\ &= \frac{\rho_s(p_s,T)}{\rho(p,T)}\left(\mathrm{d}\rho - \frac{||\vec{u}_m||^2}{2e}\mathrm{d}\rho + \frac{\vec{u}_m}{e}\cdot\mathrm{d}\vec{q} - \frac{1}{e}\mathrm{d}(\rho E)\right). \end{aligned}$$

Then from

$$\begin{aligned} \mathrm{d}\rho_s\vec{u}_m &= \vec{u}_m\mathrm{d}\rho_s + \rho_s\mathrm{d}\frac{\vec{q}}{\rho} = \vec{u}_m\mathrm{d}\rho_s + \frac{\rho_s}{\rho}\mathrm{d}\vec{q} - \frac{\rho_s}{\rho}\vec{u}_m\mathrm{d}\rho \\ \mathrm{d}\rho_s E &= E\mathrm{d}\rho_s + \rho_s\mathrm{d}\frac{\rho E}{\rho} = E\mathrm{d}\rho_s + \frac{\rho_s}{\rho}\mathrm{d}(\rho E) - \frac{\rho_s}{\rho}E\mathrm{d}\rho \end{aligned}$$

we deduce

$$\nabla U_{b,Outlet} = \begin{pmatrix} \frac{\rho_s}{\rho}\frac{1-||\vec{u}_m||^2}{2e} & \frac{\rho_s}{\rho}\frac{{}^t\vec{u}_m}{e} & -\frac{\rho_s}{\rho}\frac{1}{e} \\ \frac{\rho_s}{\rho}\frac{||\vec{u}_m||^2}{2e}{}^t\vec{u}_m & \frac{\rho_s}{\rho}(\frac{\vec{u}_m\otimes\vec{u}_m}{e} - \mathbb{I}_d) & -\frac{\rho_s}{\rho}\frac{1}{e}{}^t\vec{u}_m \\ \frac{\rho_s}{\rho}\frac{||\vec{u}_m||^2}{2e}E & \frac{\rho_s}{\rho}\frac{{}^t\vec{u}_m}{e}E & \frac{\rho_s}{\rho}(1-\frac{E}{e}) \end{pmatrix}$$

As was the case with the inlet boundary condition matrix $\nabla U_{b,Inlet}$, we observe that the final expression obtained for $\nabla U_{b,Outlet}$ does not involve any parameter of the stiffened gas law.

5.4.2 The drift model

In this case the system consists in four balance laws (see section 4.3.1).

- Neumann :

$$V_b(V) = V \quad , \qquad U_b(U) = U$$

$$\nabla U_{b,Neumann} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \mathbb{I}_d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Wall : we assume that the wall has local unit normal $\vec{n}$

$$V_b : \begin{pmatrix} c_v \\ p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} c_v \\ p \\ \vec{u}_m - 2(\vec{u}_m \cdot \vec{n})\vec{n} \\ T \end{pmatrix} \quad , \qquad U_b : \begin{pmatrix} \rho_m \\ \rho_m c_v \\ \rho_m \vec{u}_m \\ \rho_m E_m \end{pmatrix} \to \begin{pmatrix} \rho_m \\ \rho_m c_v \\ \rho_m(\vec{u}_m - 2(\vec{u}_m \cdot \vec{n})\vec{n}) \\ \rho_m E_m \end{pmatrix}$$

$$\nabla U_{b,Wall} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \mathbb{I}_d - 2\vec{n} \otimes \vec{n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Inlet :
  - First case the concentration $c_{ve} = 1 - c_{le}$, velocity $\vec{u}_e$, and temperature T_e are imposed

$$V_b : \begin{pmatrix} c_v \\ p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} c_{ve} \\ p \\ \vec{u}_e \\ T_e \end{pmatrix}, \quad U_b : \begin{pmatrix} \rho \\ \rho c_v \\ \rho \vec{u}_m \\ \rho E \end{pmatrix} \to \begin{pmatrix} \rho_e \\ \rho_e c_{ve} \\ \rho_e \vec{u}_e \\ \rho_e E_e \end{pmatrix}$$

with

$$\begin{aligned} \rho_e &= \frac{\rho_v(p,T_e)\rho_l(p,T_e)}{\rho_v(p,T_e)c_{le} + \rho_l(p,T_e)c_{ve}} = \frac{1}{\frac{c_{ve}}{\rho_v(p,T_e)} + \frac{c_{le}}{\rho_l(p,T_e)}} \\ E_e &= c_{ve}(e_v(T_e) + \frac{1}{2}|\vec{u}_{ve}|^2) + c_{le}(e_l(T_e) + \frac{1}{2}|\vec{u}_{le}|^2) \\ \vec{u}_{ve} &= \vec{u}_e + c_{le}\vec{u}_r(c_{ve}, \vec{u}_e, \rho_e) \\ \vec{u}_{le} &= \vec{u}_e - c_{ve}\vec{u}_r(c_{ve}, \vec{u}_e, \rho_e). \end{aligned}$$

The pressure differential can be expressed as

$$\begin{aligned} \mathrm{d}p &= \chi \mathrm{d}\rho + \xi \mathrm{d}(\rho c_v) + \kappa \mathrm{d}(\rho e) \\ &= (\chi + \kappa \frac{||\vec{u}_m||^2}{2})\mathrm{d}\rho + \xi \mathrm{d}(\rho c_v) - \kappa \vec{u}_m \cdot \mathrm{d}\vec{q} + \kappa \mathrm{d}(\rho E), \end{aligned}$$

and setting

$$\omega = \left(\frac{c_{ve}}{(\gamma_v - 1)\rho_v(p,T_e)^2 e_v(T_e)} + \frac{c_{le}}{(\gamma_l - 1)\rho_l(p,T_e)^2 e_l(T_e)} \right).$$

we obtain the jacobian matrix as

$$\nabla U_{b,Inlet\,u} = \begin{pmatrix} \rho_e^2\omega(\chi+\kappa\frac{||\vec{u}_m||^2}{2}) & \rho_e^2\xi\omega & -\rho_e^2\omega\kappa^t\vec{u}_m & \rho_e^2\omega\kappa \\ \rho_e^2\omega(\chi+\kappa\frac{||\vec{u}_m||^2}{2})c_{ve} & \rho_e^2\xi\omega c_{ve} & -\rho_e^2\omega\kappa^t\vec{u}_m c_{ve} & \rho_e^2\omega\kappa c_{ve} \\ \rho_e^2\omega(\chi+\kappa\frac{||\vec{u}_m||^2}{2})\vec{u}_e & \rho_e^2\xi\omega\vec{u}_e & -\rho_e^2\omega\kappa^t\vec{u}_e\otimes\vec{u}_m & \rho_e^2\omega\kappa\vec{u}_e \\ \rho_e^2\omega(\chi+\kappa\frac{||\vec{u}_m||^2}{2})E_e & \rho_e^2\xi\omega E_e & -\rho_e^2\omega\kappa {E_e}^t\vec{u}_m & \rho_e^2\omega\kappa E_e \end{pmatrix}$$

– Second case the concentration $c_{ve} = 1 - c_{le}$, pressure p_e, and temperature T_e are imposed

$$V_b : \begin{pmatrix} c_v \\ p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} c_{ve} \\ p_e \\ \vec{u}_m \\ T_e \end{pmatrix}, \quad U_b : \begin{pmatrix} \rho \\ \rho c_v \\ \rho\vec{u}_m \\ \rho E \end{pmatrix} \to \begin{pmatrix} \rho_e \\ \rho_e c_{ve} \\ \rho_e\vec{u}_m \\ \rho_e E_e \end{pmatrix}$$

with

$$\begin{aligned} \rho_e &= \frac{\rho_v(p_e,T_e)\rho_l(p_e,T_e)}{\rho_v(p_e,T_e)c_{le}+\rho_l(p_e,T_e)c_{ve}} \\ E_e &= c_{ve}(e_v(T_e)+\frac{1}{2}|\vec{u}_m|^2)+c_{le}(e_l(T_e)+\frac{1}{2}|\vec{u}_m|^2) \\ &= c_{ve}e_v(T_e)+c_{le}e_l(T_e)+\frac{1}{2}|\vec{u}_m|^2. \end{aligned}$$

From

$$\begin{aligned} \mathrm{d}\rho_e\vec{u}_m &= \rho_e\mathrm{d}\frac{\rho\vec{u}_m}{\rho} \\ &= \frac{\rho_e}{\rho}\mathrm{d}(\rho\vec{u}_m)-\frac{\rho_e\vec{u}_m}{\rho}\mathrm{d}\rho \\ \mathrm{d}(\rho_e E_e) &= \rho_e\vec{u}_m\cdot\mathrm{d}\vec{u}_m = \rho_e\vec{u}_m\cdot\mathrm{d}\frac{\rho\vec{u}_m}{\rho} \\ &= \frac{\rho_e}{\rho}\vec{u}_m\cdot\mathrm{d}(\rho\vec{u}_m)-\frac{\rho_e}{\rho}||\vec{u}_m||^2\mathrm{d}\rho \end{aligned}$$

we obtain the jacobian matrix as

$$\nabla U_{b,Inlet\,p} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{\rho_e\vec{u}_m}{\rho} & 0 & \frac{\rho_e}{\rho}\mathbb{I}_d & 0 \\ -\frac{\rho_e}{\rho}||\vec{u}_m||^2 & 0 & \frac{\rho_e}{\rho}{}^t\vec{u}_m & 0 \end{pmatrix}$$

- Outlet : pressure p_s is imposed

$$V_b : \begin{pmatrix} c_v \\ p \\ \vec{u}_m \\ T \end{pmatrix} \to \begin{pmatrix} c_v \\ p_s \\ \vec{u}_m \\ T \end{pmatrix}, \quad U_b : \begin{pmatrix} \rho \\ \rho c_v \\ \rho\vec{u}_m \\ \rho E \end{pmatrix} \to \begin{pmatrix} \rho_s \\ \rho_s c_v \\ \rho_s\vec{u}_m \\ \rho_s E \end{pmatrix}$$

with

$$\rho_s = \frac{\rho_v(p_s,T)\rho_l(p_s,T)}{\rho_v(p_s,T)c_l+\rho_l(p_s,T)c_v} = \frac{1}{\frac{c_v}{\rho_v(p_s,T)}+\frac{c_l}{\rho_l(p_s,T)}}.$$

The temperature differential can be expressed as

$$\begin{aligned} \mathrm{d}T &= \chi_T \mathrm{d}\rho + \xi_T \mathrm{d}(\rho c_v) + \kappa_T \mathrm{d}(\rho e) \\ &= (\chi_T + \kappa_T \frac{||\vec{u}_m||^2}{2})\mathrm{d}\rho + \xi_T \mathrm{d}(\rho c_v) - \kappa_T \vec{u}_m \cdot \mathrm{d}\vec{q} + \kappa_T \mathrm{d}(\rho E), \end{aligned}$$

and setting

$$\omega_T = \left(\frac{c_v c_0^v}{\rho_v(p_s,T)e_v(T)} + \frac{c_l c_0^l}{\rho_l(p_s,T)e_l(T)} \right).$$

we obtain the jacobian matrix as $\nabla U_{b,Outlet} =$

$$\begin{pmatrix} -\rho_s^2\omega_T(\chi_T + \kappa_T\frac{||\vec{u}_m||^2}{2}) & -\rho_s^2\xi_T\omega_T & \rho_s^2\omega_T\kappa_T{}^t\vec{u}_m & -\rho_s^2\omega_T\kappa_T \\ -\rho_s^2\omega_T(\chi_T + \kappa_T\frac{||\vec{u}_m||^2}{2})c_v - \frac{\rho_s}{\rho}c_v & -\rho_s^2\xi_T\omega_T c_v + \frac{\rho_s}{\rho} & \rho_s^2\omega_T\kappa c_v{}^t\vec{u}_m c_v & -\rho_s^2\omega_T\kappa c_v \\ -\rho_s^2\omega_T(\chi_T + \kappa_T\frac{||\vec{u}_m||^2}{2})\vec{u}_m - \frac{\rho_s}{\rho}\vec{u}_m & -\rho_s^2\xi_T\omega_T\vec{u}_m & \rho_s^2\omega_T\kappa_T{}^t\vec{u}_m \otimes \vec{u}_m + \frac{\rho_s}{\rho}\mathbb{I}_d & -\rho_s^2\omega_T\kappa_T\vec{u}_m \\ -\rho_s^2\omega_T(\chi_T + \kappa_T\frac{||\vec{u}_m||^2}{2})E - \frac{\rho_s}{\rho}E & -\rho_s^2\xi_T\omega_T E & \rho_s^2\omega_T\kappa_T E^t\vec{u}_m & -\rho_s^2\omega_T\kappa E + \frac{\rho_s}{\rho} \end{pmatrix}$$

6 Simulation options

6.1 First two parameters

The first two parameters are the one with a strong influence on the remaining options:

- the space dimension Ndim: affects the size of boundary conditions, initial data and gravity vectors
- the physical model: determines the size of the initial data size, boundary condition size and available types and physical parameters.

6.2 Meshes

The mesh creation building requires depends on whether it is structured or unstructured. An unstructured mesh implies that a med input file should be provided containing the mesh and possibly the initial data. Structured mesh implies that the mesh is a regular grid generated by CDMATH and the user should provide the box sizes (xinf, xsup, yinf, ysup, zinf, zsup) and cell numbers (nx, ny, nz) according to the dimension Ndim.

Therefore the mesh M is built either

- using the CDMATH structured mesh constructor Mesh M(xmin,xmax,nx)
- loading a med file generated by Salomé platform Mesh M("filename")

All the boundary faces of the mesh sould be gathered into groups of FACES that will be used to set boundary conditions. These groups of faces can be set

- using the CDMATH face group generator M.setGroupAtPlan(value,direction,tolerance,groupName), where direction is an integer with value 0 for the plane $x = value$, 1 for the plane $y = value$ and 1 for the plane $z = value$
- using Salome platform when generating the med file

For the numerical treatment of the boundaries, each boundary group should be associated with one of the boundary condition types: “Wall”, “Neumann”, “Inlet”, “Outlet” or “Dirichlet”.

6.3 Initial data or power fields

The initial field structure depends strongly on the model, space dimension and whether the mesh is structured or unstructured. The initial data and power fields can be defined on the mesh using the class Field of CDMATH either

- starting from a null field using the command Field(fieldname,ON_CELLS,numberOfComponents), then setting the values on cell i for each component j with the command $F(i,j) = fij$
- loading a field stored in a med file using the command Field(filename,ON_CELLS, filename)

The initial data should in general be a vector field containing parameters that are specific to each model

- SinglePhase : $(p, \vec{v}, T)$, ie pressure, velocity, temperature
- DriftModel : $(c_g, p, \vec{v}, T)$, ie phase number one concentration, pressure, velocity, temperature
- IsothermalTwoFluid : $(\alpha_g, p, \vec{v}_g, \vec{v}_l)$, ie phase number one void fraction, pressure, phase number one velocity, phase number two velocity
- FiveEqsTwoFluid : $(\alpha_g, p, \vec{v}_g, \vec{v}_l, T)$, ie phase number one void fraction, pressure, phase number one velocity, phase number two velocity, common temperature
- TransportEquation : h the fluid enthalpy
- DiffusionEquation : T the solid temperature

Whatever the model chosen, the initial data can be set with the function ProblemCoreFlows::setInitialField(initialField), and the heat power field is set with the function ProblemCoreFlows::setHeatSourceField(heatPowerField) (or ProblemCoreFlows::setHeatSource(heatPower) for a constant power field).

6.4 Models

The model can be set as

- SinglePhase (fluidType, pressureEstimate, spaceDimension) where fluidName is either ”Liquid” or ”Steam”, and pressureEstimate is ”around1bar” or ”around155bars”
- DriftModel (pressureEstimate, spaceDimension) where pressureEstimate is ”around1bar” or ”around155bars”
- IsothermalTwoFluid (pressureEstimate, spaceDimension) where pressureEstimate is ”around1bar” or ”around155bars”

- FiveEqsTwoFluid (pressureEstimate, spaceDimension) where pressureEstimate is "around1bar" or "around155bars"
- TransportEquation (fluidType, pressureEstimate, $\vec{v}$)
- DiffusionEquation (spaceDim, ρ,c_p,λ)

6.5 Physical options

- in order to set a constant heat source : myProblem.setHeatSource(heatSource) where heat source is a double
- in order to set the viscosities : setViscosity(viscosity) where viscosity is a $vector < double >$ (default value 0)
- in order to set the conductivities : setConductivity(conductivity) where conductivity is a $vector < double >$ (default value 0)
- in order to set the gravity : myProblem.setGravity(gravity) (default value 0)
- in order to set the friction coefficients : myProblem.setDragCoeffs(dragCoeffs) where dragCoeffs is a $vector < double >$ (default value 0)

6.6 Numerical options

These options can be set independantly from the model, dimension and mesh type (structured vs unstructured).

- The nonlinear formulation can be Roe [11], VFRoe [12] or VFFC [13] and is set by the method setNonLinearFormulation
- The time discretisation can be Explicit or Implicit, and the space discretisation can be upwind, lowMach, pressureCorrection, staggered or centered. The numerical configuration can be set using the command setNumericalScheme(upwind, Explicit); for instance.
- Advanced options can be set to enforce a well balanced scheme with setWellBalanceCorrection or an entropic scheme with setEntropicCorrection.
- in order to set the cfl number: myProblem.setCFL(cfl);
- in order to set the tolerance number : myProblem.setPrecision(precision);
- in order to set the maximum number of time steps : myProblem.setMaxNbOfTimeStep(MaxNbOfTimeS
- in order to set the maximum time of the computation : myProblem.setTimeMax(maxTime);
- in order to set the linear solver and preconditioner : myProblem.setLinearSolver("GMRES","ILU") The usual linear solver are GMRES and BICGSTAB, and the usual preconditionners are ILU and LU.

6.7 Boundary conditions

The boundary conditions field structure depends strongly on the model, space dimension.

Boundary conditions can be of type

- Wall
  - simply set with the command myProblem.setWallBoundaryCondition(...)
  - possible for all fluid models : SinglePhase, DriftModel, IsothermalTwoFluid, FiveEqsTwoFluid
- Neumann
  - simply set with the command myProblem.setNeumannBoundaryCondition(...)
  - possible for all models
- Inlet
  - simply set with the command myProblem.setInletBoundaryCondition(...)
  - possible for TransportEquation, SinglePhase, DriftModel, IsothermalTwoFluid, FiveEqsTwoFluid (not possible for DiffusionEquation)
- Outlet
  - simply set with the command myProblem.setOutletBoundaryCondition(...)
  - possible for fluid models : SinglePhase, DriftModel, IsothermalTwoFluid, FiveEqsTwoFluid
- Dirichlet
  - simply set with the command myProblem.setDirichletBoundaryCondition(...)
  - possible for DiffusionEquation

The former way to set the boundary conditions was to create a C++ map $map < string, LimitField > boundaryFields$ that associates the boundary name to a field. Once the map is filled with the appropriate content, the boundary conditions are set with the command ProblemCoreFlows::setBoundaryFields(boundaryFields).

The LimitField structure is specific to each model and each type of boundary condition.

- for a Neumann boundary condition : LimitField limitNeumann; limitNeumann.bcType=Neumann;
- for an Inlet boundary condition (not possible for DiffusionEquation): LimitField limitInlet; limitInlet.bcType=Inlet; then
  - For TransportEquation :
    limitInlet.h=1.3e6
  - For SinglePhase (3D case):
    limitInlet.T = 600;
    limitInlet.v_x = $vector < double > (1, 0)$;
    limitInlet.v_y = $vector < double > (1, 0)$;
    limitInlet.v_z = $vector < double > (1, 0)$;

  - For DriftModel (3D case):
    limitInlet.c=0;
    limitInlet.T = 600;
    limitInlet.v_x = $vector < double > (1, 0)$;
    limitInlet.v_y = $vector < double > (1, 0)$;
    limitInlet.v_z = $vector < double > (1, 0)$;
  - For IsothermalTwoFluidModel (3D case):
    limitInlet.alpha=0;
    limitInlet.v_x = $vector < double > (2, 0)$;
    limitInlet.v_y = $vector < double > (2, 0)$;
    limitInlet.v_z = $vector < double > (2, 0)$;
  - For FiveEqsTwoFluid (3D case):
    limitInlet.alpha=0;
    limitInlet.T = 600;
    limitInlet.v_x = $vector < double > (2, 0)$;
    limitInlet.v_y = $vector < double > (2, 0)$;
    limitInlet.v_z = $vector < double > (2, 0)$;
  - then boundaryFields["Inlet"]= limitInlet;

- for an Outlet boundary condition (for all fluid models): LimitField limitOutlet; limitOutlet.bcType=Outlet; then
  - limitOutlet.p = 155e5;
  - then boundaryFields["Outlet"]= limitOutlet;

- for a Wall with friction and/or heat conduction : : LimitField limitWall; limitWall.bcType=Wall; then
  - For SinglePhase (3D case):
    limitWall.T = 600;
    limitWall.v_x = $vector < double > (1, 0)$;
    limitWall.v_y = $vector < double > (1, 0)$;
    limitWall.v_z = $vector < double > (1, 0)$;
  - For DriftModel (3D case):
    limitWall.T = 600;
    limitWall.v_x = $vector < double > (1, 0)$;
    limitWall.v_y = $vector < double > (1, 0)$;
    limitWall.v_z = $vector < double > (1, 0)$;
  - For IsothermalTwoFluidModel (3D case):
    limitWall.v_x = $vector < double > (2, 0)$;
    limitWall.v_y = $vector < double > (2, 0)$;
    limitWall.v_z = $vector < double > (2, 0)$;
  - For FiveEqsTwoFluid (3D case):
    limitWall.T = 600;
    limitWall.v_x = $vector < double > (2, 0)$;
    limitWall.v_y = $vector < double > (2, 0)$;
    limitWall.v_z = $vector < double > (2, 0)$;
  - then boundaryFields["Wall"]= limitWall;

- for a Dirichlet boundary condition (only for DiffusionEquation) : LimitField limitDirichlet; limitDirichlet.bcType=Dirichlet; then

  - limitDirichlet.T = 600;
  - boundaryFields["Dirichlet"]= limitDirichlet;

References

[1] A. S. Shieh, V. H. Ransom, R. Krishnamurthy, Validation of numerical techniques in RELAP5/MOD3, RELAP5/MOD3 code manual volume 6, October 1994

[2] `http://www.casl.gov/COBRA-TF.shtml`

[3] `http://www.ansys.com/Products/Simulation+Technology/Fluid+Dynamics/Fluid+Dynamics+Products/ANSYS+CFX`

[4] J. J. jeong, H. Y. Yoon, I. K. Park, H. K. Cho, J. Kim, A semi-implicit numerical scheme for transient two-phase flows, Nuclear Engineering and Design 236 (2008) 3403-3412

[5] I. Toumi, A. Bergeron, D. Gallo, E. Royer, D. Caruge, "FLICA-4: a three-dimensional two-phase flow computer code with advanced numerical methods for nuclear applications," Nuclear Engineering and Design, Volume 200, Issues 1-2, August 2000, Pages 139-155.

[6] D. Bestion, "The Physical Closure Laws in The CATHARE Code, Nuclear Engineering and Design," vol. 124, pp. 229-245, 1990.

[7] N. Méchitoua, M. Boucker, J. Laviéville, J.-M. Hérard, S. Pigny, and G. Serre. An Unstructured Finite Volume Solver for Two-Phase Water/Vapour Flows Modelling Based on an Elliptic Oriented Fractional Step Method. In NURETH 10, International Meeting on Nuclear Reactor Thermal-Hydraulics, Seoul, South Korea, 2003.

[8] Propriétés thermophysiques des systèmes fluides, `http://webbook.nist.gov/chemistry/fluid/`

[9] M. Ishii, "Thermo-Fluid Dynamic Theory of Two-Phase Flow," Eyrolles, Paris, 1975.

[10] D.A. Drew and S.L. Passman, "Theory of Multicomponent Fluids," Springer-Verlag, New York, 1999.

[11] Ph. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, J. Comput. Phys., 43, 357, 1981

[12] J. M. Masella, I. Faille and T. Gallouët, "On an Approximate Godunov Scheme", Intl. J. Computational Fluid Dynamics, 1999, Vol. 12, pp. 133-149.

[13] J.M. Ghidaglia, A. Kumbaro, G. Le Coq, Une méthode volumes finis à flux caractéristiques pour la résolution numérique des systèmes hyperboliques de lois de conservation, Comptes Rendus de l'Acad. Sciences Paris, Série 1, vol 322, pp. 981 988, 1996.

[14] "Upwind methods for hyperbolic conservation laws with source terms", A. Bermudez, E. Vazquez, Comp. Fluids, vol 23, issue 8, pp. 1049-1071,1994

[15] M. Ndjinga, T.-P.-K. Nguyen, C. Chalons, Numerical simulation of an incompressible two-fluid model, Springer Proc. Math. & Stat., Vol. 78, Finite Volumes for Complex Applications FVCA7, 2014

[16] T.-H. Dao, M. Ndjinga, F. Magoules, Comparaison of Upwind and Centered Schemes for Low Mach Number Flows, Finite Volumes for Complex Applications VI - Problems & Perspectives, Springer Proceedings in Mathematics 4, 2011

[17] R. J. LeVeque, Finite Volume Methods for Hyperbolic Problems, Cambridge University Press, 2002

[18] The open-source integration platform for numerical simulation `www.salome-platform.org`

[19] The open-source, parallel data analysis and visualization application `www.paraview.org`

[20] The Visualization ToolKit `www.vtk.org`

[21] Parallel storage and manipulation of large sparse matrices `www.mcs.anl.gov/petsc`

[22] Projet "Ce sont Des Mathématiques Appliquées à la THermohydraulique": `http://cdmath.jimdo.com/`

[23] Toolbox du projet CDMATH : `https://github.com/PROJECT-CDMATH/CDMATH`

[24] Simplified Wrapper and Interface Generator : `http://www.swig.org`

[25] HDF5 - Data model and file format of large volume : `http://www.hdfgroup.org/HDF5`

[26] DOXYGEN - Generate documentation from source code : `http://www.doxygen.org`

[27] CMAKE - Open-source build test and package software : `http://www.cmake.org`

[28] MED - Modèle d'échange de données `http://www.code-aster.org/outils/med/html/introduction.html`

[29] NUMPY - Package for scientific computing with Python `http://www.numpy.org/`