User-manual for COOLFluiD (version 2017.10)

Andrea Lani, lani@vki.ac.be
Von Karman Institute, Aeronautics & Aerospace Dept.

Introduction

The COOLFluiD platform (`https://github.com/andrealani/COOLFluiD/wiki`) [2, 3, 4, 9, 10] is an object-oriented framework for high-performance computing (HPC) on unstructured grids and is the main in-house computational tool for CFD applications at the Von Karman Institute. COOLFluiD consists of a set of plug-in libraries that can be linked at run-time to a kernel where the basic parallel data structure and interface functionalities are defined. The platform is currently able to handle complex multi-physics simulations with a wide range of spatial discretization algorithms and time marching methods, both explicit and implicit. Linear systems arising from Newton linearizations of the space/time residual are efficiently solved in parallel with dedicated software packages that have been interfaced within COOLFluiD as plug-in libraries. Thanks to an extremely modular and scalable design, different functionalities (and corresponding library components) can be easily combined together to get more complex capabilities which will, in most cases, automatically work both serial and in parallel. Some of the available features offered by the COOLFluiD software environment are:

- **multiple space discretizations:** Cell Centered Finite Volume (FV), Residual Distribution (RD) Finite Element (FE), Spectral Finite Volume (SV), Spectral Finite Difference (SD), Discontinuous Galerkin (DG);
- **multiple time integration schemes:** Runge Kutta, 1- and 3-point Backward Euler, Crank-Nicholson, Time limited schemes;
- **multiple parallel linear system solvers:** PETsc, Trilinos, SAMG, Pardiso, jacobian free methods;
- **parallel infrastructure:** parallel I/O capabilities and domain decomposition;
- **multiple physical models:** Ideal Magneto Hydro Dynamics (MHD), RANS ($k - \omega$, SST, BSL, Spalart-Allmaras models), Linear Elasticity, Heat Transfer, Compressible and Incompressible high-enthalpy flows in thermo-chemical nonequilibrium (TCNEQ) or in Local Thermodynamic Equilibrium (LTE) (with fixed or variable elemental fractions), Aeroacustics (LEE), LES;
- **algorithms for loosely coupled multi-domain multi-physics simulations**: different numerical methods applied to different models on full-non matching unstructured meshes for Aeroelasticity, Aerothermoelasticity, Conjugate heat transfer, etc.
- **ALE formulation for unsteady simulations on moving meshes**: high quality parallel mesh movement and deformation algorithms.

The COOLFluiD platform has been **designed to grow according to the needs** and has demonstrated its flexibility by incorporating progressively more and more complex algorithms and physical models. Some key strong points are:

- **Flexible data-structure** that allows to implement complex numerical methodologies and equations and get them automatically working in parallel (all the described functionalities work in parallel!) to take full profit of computational power. The extreme flexibility is proved by the fact that is hard to find another software package, commercial or not, that can offer such a variety of space discretization algorithms (each one with its own data-structure) working in parallel *within the same framework*.
- **Scalable design** that allows researchers to integrate new functionalities by fully reusing existing ones and take immediate profit of others' work. As an example, physicists can work in the Mutation library to refine the transport and thermodynamic modeling and numericists can improve the numerical algorithms independently one from the other, or with extremely limited interaction. At the end, the functionalities work both separately, as new independent framework components, and together.
- **Multi-physics** simulations can be customized at will, reusing the available components or incrementally implement new ones.
- **State-of-the-art numerical algorithms** and **extremely advanced physical modeling** for high-enthalpy flow properties (rigorously derived from kinetic theory or statistical and quantum mechanics) are combined together.
- Possibility of accurately simulating high-enthalpy flows **from incompressible regime to hyper-velocity** (up to Mach 40 or more).

1 Getting started

Detailed installation instructions are available online at `https://github.com/andrealani/COOLFluiD/wiki/HOWTO`. The following summarizes just the main steps for a standard installation and running of testcases.

1.1 Installation instructions

1. Download COOLFluiD sources:

   ```
   svn co https://github.com/andrealani/COOLFluiD/trunk coolfluid
   ```

2. Running the script to install dependencies:

   ```
   cd coolfluid/tools/scripts
   ./install-coolfluid-deps.pl --tmp-dir=TDIR --install-dir=LDIR --install-mpi-
       dir=MDIR
   ```

   where `TDIR` is the full path to the directory where dependencies files will be unpacked, `LDIR` is the installation directory and `MDIR` is the directory where the Message Passing Interface (MPI) libraries will be installed. PETSc and ParMetis libraries will be installed inside the MPI directory. Different MPI installations can coexist: the user-defined *coolfluid.conf* file, the `PATH` and `LD_LIBRARY_PATH` environmental variables will decide which actual installation to use.

3. Updating environmental variables to include the appropriate paths to the dependency libraries in the `.bashrc`:

```
export PATH=LDIR/bin:MDIR/bin:$PATH
export LD_LIBRARY_PATH=LDIR/lib:MDIR/lib:MDIR/petsc/lib:$LD_LIBRARY_PATH
```

4. Setting up the file *coolfluid.conf* with the appropriate `coolfluid_dir`, `basebuild_dir`, `install_dir`, paths to all dependencies (check example files in `coolfluid/tools/conf`) and modules to download.

5. Checking out the selected modules with

   ```
   ./prepare.pl --config-file=coolfluid.conf
   ```

6. Generating the build (make) files in `basebuild_dir` in *debug* (full debugging options, very slow), *optim* (some debug, some optimization, **recommended**) or *release* (no debugging, full optimization) mode:

   ```
   ./prepare.pl --config-file=coolfluid.conf --build=optim
   ```

7. Compiling (typically on multiple cores, 4 in our example) and creating all COOLFluiD libraries:

   ```
   cd basebuild_dir ; make -j4 ; make install
   ```

8. Setting the appropriate paths to the COOLFluiD libraries in the `.bashrc`:

   ```
   export PATH=install_dir/bin:$PATH
   export LD_LIBRARY_PATH=install_dir/lib:$LD_LIBRARY_PATH
   ```

WATCH OUT: *Steps 1 and 2 are only needed if dependency libraries are not installed in your system yet. Internal users at the VKI do not need those steps, since public installations are available. In particular, lammpi, openmpi and mpich2 are all supported at the VKI.*

WATCH OUT: *Using "make install" in step 8 is not necessary if you choose to use soft links to the coolfluid-solver executable located in* `basebuild_dir/src/Solver` *directory.*

1.2 How to run a simulation (from inside a testcase folder)

The command line to run COOLFluiD is:

```
mpirun -np N ./coolfluid-solver --scase ./myfile.CFcase
```

from inside the testcase directory. The parameter N must be replaced by the number of processors. The format of the input file (called `myfile.CFcase` in our example) is described here after. For a serial run (N=1) the user can also use:

```
./coolfluid-solver --scase ./myfile.CFcase
```

WATCH OUT: *In order to be able to run successfully, a soft link to (or copy of) the* `coolfluid-solver` *executable and file* `coolfluid-solver.xml` *(providing useful info on the path to the COOLFluiD shared libraries) must be present in the working directory.*

1.3 Configuration file description

The format of the input file (with extension .CFcase) consists of lines in the form `KEY` = `VALUE`:

```
Simulator.OptionA = Value1                  # use Value1 as value for OptionA
Simulator.Value1.OptionB = Value2           # use Value2 as value for OptionB
Simulator.Value1.Value2.OptionC = Value3 # use Value3 as value for OptionC
Simulator.Value1.Value2.OptionD = Value4 # use Value4 as value for OptionD
```

where, in each line, the whole LHS is the keyword and the RHS is the value. The latter, depending of the actual types defined in the code for each configurable parameter, can be:

- an alpha-numerical string
- an integer
- a boolean (`true` or `false`)
- a floating point number
- an arbitrarily complex analytical function
- an array of all the previous.

The keyword is composed of literal strings separated by ".", corresponding to different *entities* (configurable objects or parameters) defined inside the actual code. The configuration is hierarchical and recursive from top to lowest level. **The order in which the options are declared in the file are irrelevant**. If needed, the value can be broken into different lines by using the continuation character (back slash) at the end of each line (note that the number of spaces at the end or before the line is irrelevant):

```
Simulator.Example.arrays = 4 4 10 \
                           10 4 \
                           4 3
```

Comments start with "#": they can occupy full lines or be placed at the end of the line.

2 Environment options

This section summarizes the main options available for setting up the kernel components of COOLFluiD (i.e. Environment and Framework libraries).

```
CFEnv.ErrorOnUnusedConfig = true
```

If activated this option makes the simulation crash if there are spelling mistakes in the given options. This option must always be inactivated when using a mesh converter (e.g. Gambit2CFmesh, Gmsh2CFmesh, Tecplot2CFmesh).

```
CFEnv.ExceptionDumps = true
CFEnv.ExceptionOutputs = true
```

If activated, those options will catch exceptions and show the error message. The simulation will crash only if the exception is not caught, but at least it will indicate what went wrong. It should be deactivated to reduce outputs in case things work.

```
Simulator.Modules.Libs = libCFmeshFileReader libNavierStokes libFiniteVolume ...
```

List of the COOLFluiD dynamic libraries needed for the present simulation. In the following description, each section will indicate the required libraries whenever applicable.

```
Simulator.Paths.WorkingDir = ./
Simulator.Paths.ResultsDir = ./RESULTS
```

Paths to the working directory and to the directory where output files (convergence history, Tecplot files, CFmesh files) should be written.

2.1 Interactive file

Some parameters can be changed interactively during the simulation by editing a separate file where the full option setting (key and value) has to be present.

```
Simulator.SubSystem.InteractiveParamReader.FileName = ./out.inter
```

tells the path to the interactive file and

```
Simulator.SubSystem.InteractiveParamReader.readRate = 10
```

specifies how often the file should be read by the solver in order to update the corresponding interactive parameters.
WATCH OUT: *In a parallel run, this rate must be defined with a safe margin (depending of the speed of the iterative process), allowing the user to quickly edit, modify and close the file before the solver tries to read the file as well.*

2.2 Stop Condition

The simulation can be stopped by prescribing a maximum number of steps:

```
Simulator.SubSystem.StopCondition = MaxNumberSteps
Simulator.SubSystem.MaxNumberSteps.nbSteps = 2
```

of by looking at the norm of the residual

```
Simulator.SubSystem.StopCondition          = Norm
Simulator.SubSystem.Norm.valueNorm         = -3.0
```

A threshold of $<= -3.$ is reasonably good for most cases, if the temperature is used as variable to monitor (see MonitoredVarID below).

```
Simulator.SubSystem.SubSystemStatus.TimeStep = 5.0
```

defines the time step in an unsteady simulation.

3 Physical Model

3.1 Ideal MHD

Required libs: libMHD.

```
Simulator.SubSystem.Default.PhysicalModelType = MHD2DProjection (or
    MHD3DProjection)
```

defines a generic 2D (or 3D) ideal MHD model that uses hyperbolic divergence cleaning method as the $\nabla \cdot \vec{B} = 0$ constraint satisfying technique.

```
Simulator.SubSystem.Default.PhysicalModelType = MHD2D (or MHD3D)
```

defines a generic 2D (or 3D) ideal MHD model that uses Powell's source term method as the $\nabla \cdot \vec{B} = 0$ constraint satisfying technique.

```
Simulator.SubSystem.MHD3DProjection(or MHD3D or MHD2DProjection or MHD2D).
    ConvTerm.gamma = 1.666666667
```

defines the ratio of specific heats for the plasma which we take most of the time to be equal to 5/3.

```
Simulator.SubSystem.MHD2DProjection(or MHD3DProjection).ConvTerm.refSpeed = 3.0
#Simulator.SubSystem.MHD2DProjection(or MHD3DProjection).ConvTerm.dissipCoeff =
    3.0
#Simulator.SubSystem.MHD2DProjection(or MHD3DProjection).ConvTerm.correctionType
    = Mixed
```

defines a constant reference speed for the hyperbolic divergence cleaning method with hyperbolic Lagrange multiplier. The two commented out lines should be uncommented while using the mixed hyperbolic/parabolic Lagrange multiplier [11] which we never had to use so far. The constant reference speed is chosen to be equal to the freestream flow speed or the maximum speed in the initial conditions for unsteady simulations where there is no freestream flow in the setup of the testcase (see OrszagTang vortex, Smooth Alfvén Wave and Rotor testcases [17]).

3.1.1 $\vec{B}_0 + \vec{B}_1$ splitting model

For solar wind/planetary magnetosphere interaction simulations, the planetary intrinsic magnetic field is modelled as a dipole ($\vec{B}_0$).

```
Simulator.SubSystem.MHD3DProjection(or MHD2DProjection or MHD2D or MHD3D).
    ConvTerm.mX = 0.0
Simulator.SubSystem.MHD3DProjection(or MHD2DProjection or MHD2D or MHD3D).
    ConvTerm.mY = 0.0
Simulator.SubSystem.MHD3DProjection(or MHD3D).ConvTerm.mZ = -3000.0
```

define the magnetic dipole moment of the planetary magnetic field with the center of the dipole located at the origin.

3.2 Thermochemical nonequilibrium

Required libs: libNavierStokes, libNEQ.

```
Simulator.SubSystem.Default.PhysicalModelType = NavierStokes2DNEQ
```

defines a generic 2D thermo-chemical nonequilibrium model.

```
Simulator.SubSystem.NavierStokes2DNEQ.refValues = \
  1e-12 1e-6 1e-6 0.00002854 1e-6 0.00000866 1e-6 1e-6 \
  1e-6 1e-6 1e-6 11360. 1000. 195. 195.
```

provides values of the order of the free stream quantities for all stored variables (see `.updateVar` below), one per equation.
WATCH OUT: *None of those values can be zero, since they are actually used as denominator in scaling for numerical finite difference while computing numerical jacobians.*

```
Simulator.SubSystem.NavierStokes2DNEQ.nbSpecies = 11
Simulator.SubSystem.NavierStokes2DNEQ.nbEulerEqs = 3
Simulator.SubSystem.NavierStokes2DNEQ.nbVibEnergyEqs = 1
```

specify the number of chemical species, the total number of equations excluding species continuity and vibrational/electronic equations (it must be 3 for 2D cases, 4 for 3D cases), the number of vibrational energy equations.

3.2.1 Mutation 2.0

Required libs: libMutation2OLD, libMutation2OLDI.

```
Simulator.SubSystem.NavierStokes2DNEQ.PropertyLibrary = Mutation2OLD
```

specifies Mutation 2.0.0 (slower but stable version of Mutation 2.0) [5, 7] as the physico-chemical library for computing thermodynamic, transport, chemical kinetics properties. This version of Mutation supports arbitrary chemical mixtures (neutral and ionized), chemical equilibrium models with fixed and variable elemental fractions, thermal and chemical nonequilibrium multi-temperature models, including full and reduced Collisional Radiative (CR) models for air [8].

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.mixtureName = air11
```

specifies the name of the mixture corresponding to a file `air11.mix` defined inside `PATH_TO_MUTATION/Mutation2.0.0I/data/mixture`.

WATCH OUT: *The mixture file specifies the order of the chemical species as they are used and stored by the flow solver.*

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.reactionName = parkair93
```

specifies the name of the chemical reactions model corresponding to a file `parkair93` defined inside `PATH_TO_MUTATION/Mutation2.0.0I/data/chemistry/gasreact`.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.transfName = air11
```

specifies the name of the energy transfer model corresponding to a file `air11` defined inside `PATH_TO_MUTATION/Mutation2.0.0I/data/chemistry/transfer`.

WATCH OUT: *The detailed description of the format for Mutation data files is out of the scope of this tutorial. The user is referred to the Mutation manual (contact* `magin@vki.ac.be`*) instead.*

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.TminFix = 100.
```

defines minimum temperature allowed inside Mutation routines.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.dynViscAlgo = CG
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.thermCondAlgo = Direct
```

specifies the transport algorithms to use for the computation of the dynamic viscosity and the thermal conductivity. Those settings are the most stable and should not be changed.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.includeElectronicEnergy = true
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.electrEnergyID = 0
```

Those options should always be activated when running ionized cases and deactivated otherwise.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.path = \
   /data1/andrea/COOLFLUID/plugins/Mutation2.0.0I/
```

provides the full path (no environmental variables are allowed here!) to the Mutation 2.0 installation.

4 Output Format

```
Simulator.SubSystem.OutputFormat     = Tecplot CFmesh
```

defines list of requested output files (only `Tecplot` and `CFmesh` are supported for nonequilibrium flows).

4.1 CFmesh writer

Required libs: libCFmeshFileWriter

```
Simulator.SubSystem.OutputFormat     = Tecplot CFmesh
```

defines list of requested output files (only `Tecplot` and `CFmesh` are supported for nonequilibrium flows).

```
# in parallel runs, a file out-P0.CFmesh is written
Simulator.SubSystem.CFmesh.FileName = out.CFmesh

# every how many iterations the file is saved
Simulator.SubSystem.CFmesh.SaveRate = 1000

# append iteration number to the file name
Simulator.SubSystem.CFmesh.AppendIter = true
```

specifies the settings for the CFmesh file format (the internal format of COOLFluiD, including both mesh and solution). Only one CFmesh file is written even in a parallel simulation.

4.2 Tecplot writer

Required libs: libTecplotWriter.

```
# in parallel runs, one file per processor out-P*.plt is written
Simulator.SubSystem.Tecplot.FileName         = out.plt

# output variables name (RhoivtTv correspond to [rho_i v T T_v])
Simulator.SubSystem.Tecplot.Data.outputVar = RhoivtTv

# write also density, total enthalpy, Mach number and pressure
Simulator.SubSystem.Tecplot.Data.printExtraValues = true

# name of the boundary patch for which a file out-P*-surf.plt will be saved
Simulator.SubSystem.Tecplot.Data.SurfaceTRS = Wall

# every how many iterations the file is saved
```

```
Simulator.SubSystem.Tecplot.SaveRate = 1000

# append iteration number to the file name
Simulator.SubSystem.Tecplot.AppendIter = false
```

specifies the settings for the Tecplot file format. One file per processor will be written.

5 Mesh Creator

5.1 CFmesh reader

Required libs: libCFmeshFileReader.

```
Simulator.SubSystem.Default.listTRS = InnerFaces Wall Symmetry Inlet Outlet
```

specifies the list of all Topological Region Sets (TRS), i.e. the boundary patches as defined in the mesh file. `InnerFaces` dose not need to be included in the list.

```
Simulator.SubSystem.MeshCreator = CFmeshFileReader
Simulator.SubSystem.CFmeshFileReader.Data.FileName = ./input.CFmesh
```

specify the reading from a file called `Restart.CFmesh` in CFmesh format.

```
Simulator.SubSystem.CFmeshFileReader.Data.ScalingFactor = 1000.
```

specifies a factor for which the input mesh must be *divided* (the name "Scaling" is misleading here).
WATCH OUT: *the scaling factor must be used only when starting from scratch and not from a CFmesh file containing the solution.*

```
Simulator.SubSystem.CFmeshFileReader.ParReadCFmesh.ParCFmeshFileReader.
    NbOverlapLayers = 2
```

This option is obsolete, but kept for compatibility with older versions of the code. It specifies the number of overlap layers in parallel computations. This value should be 2 for second-order calculations, but it is now automatically calculated.

5.2 Converting from Gambit files

Required libs: libGambit2CFmesh.

If the mesh file is not yet in CFmesh format and it's coming from the ANSYS Gambit mesh generator, the following settings must e defined:

```
Simulator.SubSystem.CFmeshFileReader.convertFrom = Gambit2CFmesh
Simulator.SubSystem.CFmeshFileReader.Gambit2CFmesh.Discontinuous = true
Simulator.SubSystem.CFmeshFileReader.Gambit2CFmesh.SolutionOrder = P0
```

In this case the solver expects a file called `input.neu` placed inside the working directory. **WATCH OUT:** *all Gambit settings must be commented out when restarting from a previous CFmesh solution (see Restart option).*

5.3 Converting from Gmsh files

Required libs: libGmsh2CFmesh.

If the mesh file is not yet in CFmesh format and it's coming from the Gmsh mesh generator, the following settings must e defined:

```
Simulator.SubSystem.CFmeshFileReader.convertFrom = Gmsh2CFmesh
Simulator.SubSystem.CFmeshFileReader.Gmsh2CFmesh.Discontinuous = true
Simulator.SubSystem.CFmeshFileReader.Gmsh2CFmesh.SolutionOrder = P0
```

In this case the solver expects `input.msh` and `input.SP` files placed inside the working directory
WATCH OUT: *all Gmsh settings must be commented out when restarting from a previous CFmesh solution (see Restart option).*

5.3.1 Converting from THOR files

Required libs: libTHOR2CFmesh.

If the mesh file is not yet in CFmesh format and it's coming from the THOR code mesh format, the following settings must be defined:

```
Simulator.SubSystem.CFmeshFileReader.THOR2CFmesh.Discontinuous = true
Simulator.SubSystem.CFmeshFileReader.THOR2CFmesh.SolutionOrder = P0
Simulator.SubSystem.CFmeshFileReader.convertFrom = THOR2CFmesh
```

In this case the solver expects a mesh file called `input.thor` and a super patch file called `input.SP` placed inside the working directory.
WATCH OUT: *all THOR settings must be commented out when restarting from a previous CFmesh solution (see Restart option).*

6 Convergence Method

In the following `<CMETHOD>` must be substituted with the concrete ConvergenceMethod name (e.g. BwdEuler, NewtonIterator, BDF2).

6.1 Convergence file

```
 #the filename into which the convergence history is to be written
 Simulator.SubSystem.<CMETHOD.ConvergenceFile = ./convergence.plt
```

6.2 CFL

6.2.1 Interactive CFL

```
Simulator.SubSystem.<CMETHOD>.Data.CFL.ComputeCFL = Interactive
```

declares the CFL interactive and its value will be read from the interactive file (`out.inter` in our example). In this case the line

```
Simulator.SubSystem.<CMETHOD>.Data.CFL.Interactive.CFL = 10.0
```

must be present and, if needed, modified in the interactive file.

6.2.2 Function CFL

```
Simulator.SubSystem.<CMETHOD>.Data.CFL.Value = 1.0
Simulator.SubSystem.<CMETHOD>.Data.CFL.ComputeCFL = Function
Simulator.SubSystem.<CMETHOD>.Data.CFL.Function.Def = \
   if(i<1000,1.0,if(i<2000.,1.01*cfl , min(1200.,1.05*cfl)))
```

In order to automatize the iterative process, a function specifying an arbitrarily complex CFL law can be provided. The variable that can appear in this expression are: `i` (iteration number), `cfl` (previous CFL value), `r` (current residual), `ri` (initial residual), `rl` (last residual), `rmax` (maximum residual).

WATCH OUT: *no spaces are allowed within the expression and the supported operators and mathematical functions are indicated in [1].*

6.3 Backward Euler

Required libs: libBackwardEuler libBackwardEulerMHD libNewtonMethod.

We consider here both the steady implicit time stepping case corresponding to a first-order accurate backward Euler integration and the unsteady implicit time stepping case corresponding to the second-order accurate Crank-Nicholson and BDF.

```
Simulator.SubSystem.ConvergenceMethod = BwdEuler
#the filename into which the convergence history is to be written
Simulator.SubSystem.BwdEuler.ConvergenceFile = ./convergence.plt
```

6.3.1 Backward Euler-MHD

```
Simulator.SubSystem.BwdEuler.UpdateSol = UpdateSolMHD
Simulator.SubSystem.BwdEuler.UpdateSolMHD.pressureCorrectionValue =
     0.000000000001
```

define a correction value for negative pressure occurrences especially encountered during the solar wind/planetary magnetosphere simulations with sufficiently harsh initial conditions and planetary dipole field in the vicinity of the planet when started from an initial uniform solution. Thus, the simulation does not blow up due to the increase in the number of negative pressure occurrences and most of the time the negative pressure values are eliminated as the simulation continues.

6.4 Newton Method

Required libs: libNewtonMethod.

We consider here only the steady implicit time stepping case, corresponding to a first-order accurate Backward Euler integration.

```
Simulator.SubSystem.ConvergenceMethod = NewtonIterator

# this value must be > 1 only for unsteady simulations
Simulator.SubSystem.NewtonIterator.Data.MaxSteps = 1
```

The CFL parameter which controls the stability of the calculation can be specified in two ways: interactively or with a user-defined function.

```
Simulator.SubSystem.NewtonIterator.StdUpdateSol.Relaxation = 1.0
```

provides a relaxation parameter (<= 1.0): a single value for all equations or an array of values (with size equal to the number of equations). This typically can be kept equal to 1.

```
Simulator.SubSystem.NewtonIterator.Data.L2.MonitoredVarID = 13
```

indicates the ID of the variable to be monitored for convergence (see StopCondition). In chemically reacting flows, the ID corresponding to the temperature (total energy equation) is recommended.

```
Simulator.SubSystem.NewtonIterator.Data.L2.ComputedVarID = 13
```

indicates the ID of the variable whose norm will be computed and written to screen. If this line is commented out, residuals for all variables will be computed.

```
Simulator.SubSystem.NewtonIterator.Data.FilterState = Max

# flags (0 or 1) to tell which variables must be clipped
Simulator.SubSystem.NewtonIterator.Data.Max.maskIDs = \
 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1

# real values to tell the minimum values to be imposed for each variable
# only values given for flagged variables (maskID = 1) will be clipped
Simulator.SubSystem.NewtonIterator.Data.Max.minValues = \
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

provide some filtering of the solution values before the solution update.

6.5 BDF2

```
# this value must be > 1 only for unsteady simulations where it will represent
    the number of subiterations to be made at each timestep
Simulator.SubSystem.BDF2.Data.MaxSteps = 1

Simulator.SubSystem.BDF2.Data.L2.MonitoredVarID = 8
```

indicates the ID of the variable to be monitored for convergence (see Stop condition).

```
Simulator.SubSystem.BDF2.Data.L2.ComputedVarID = 8
```

indicates the ID of the variable whose norm will be computed and written to screen. If this line is commented out, residuals for all variables will be computed.

```
Simulator.SubSystem.BDF2.Data.Norm = -3.
Simulator.SubSystem.BDF2.Data.PrintHistory = true
```

7 Linear System Solver

7.1 PETSc

Required libs: libPetsc.

We include here only the settings corresponding to the PETSC linear system solver library, even though Trilinos is also interfaced within COOLFluiD.

```
Simulator.SubSystem.LinearSystemSolver = PETSC
Simulator.SubSystem.LSSNames = NewtonIteratorLSS
Simulator.SubSystem.NewtonIteratorLSS.Data.PCType = PCASM
Simulator.SubSystem.NewtonIteratorLSS.Data.KSPType = KSPGMRES
Simulator.SubSystem.NewtonIteratorLSS.Data.MatOrderingType = MATORDERING_RCM
```

specify the basic settings for a GMRES solver combined with a parallel Additive Schwartz preconditioner.

```
Simulator.SubSystem.NewtonIteratorLSS.Data.MaxIter = 1000
```

defines the maximum allowed number of GMRES iterations: should be typically kept <= 1000.

```
Simulator.SubSystem.NewtonIteratorLSS.Data.RelativeTolerance = 1e-4
```

defines the relative tolerance for the GMRES solver: $1e-4$ or $1e-3$ are recommended values, since with higher values convergence can be very slow and requiring many more GMRES iterations per time step.

8 Space Method: Finite Volume

Required libs: libFiniteVolume

In order to restart from a previous CFmesh file (with saved solution), the following option must be set to true, otherwise must be commented out.

```
Simulator.SubSystem.CellCenterFVM.Restart = true
```

8.1 Initial Field

The initial field conditions are typically prescribed on the full domain with a set of user-defined analytical functions depending on the position vector (x,y,z) [1], one for each of the update variables (`RhoivtTv` in our current example).

The following settings are always required:

```
# "InitState" is the name of the object implementing an initial state
Simulator.SubSystem.CellCenterFVM.InitComds = InitState

# "InField" is a user-defined alias that will be used to configure InitState
Simulator.SubSystem.CellCenterFVM.InitNames = InField

# from now on, only "InField" is used for the initialization settings
# "InnerFaces" is the boundary patch (TRS) on which InField is active
Simulator.SubSystem.CellCenterFVM.InField.applyTRS = InnerFaces
```

The following settings define the analytical functions.

```
# independent variables (use "x y z" in 3D)
Simulator.SubSystem.CellCenterFVM.InField.Vars = x y

# arbitrarily complex function definitions (one per update variable, 15 in this
   case)
# no space allowed within a single function
# NOTE: this is just an illustrative example with no physical sense!
Simulator.SubSystem.CellCenterFVM.InField.Def = \
 0. 0. 0. if(sqrt(x^2+y^2)<1.,0.00002854,0.00002854/2.) \
 0. if(sqrt(x^2+y^2)<1.,0.00000866,0.00000866/2.) 0. 0. 0. 0. 0.
 if(x<1.0,11360.,100.*(sqrt(x^2+y^2)-1.)) 0. 195. 195.
```

If analytical expressions are particularly complex, the user can use a more advance 2-step initializer. The previous example can be simplified as:

```
# "InitStateAddVar" is used instead of "InitState"
Simulator.SubSystem.CellCenterFVM.InitComds = InitStateAddVar
Simulator.SubSystem.CellCenterFVM.InitNames = InField
Simulator.SubSystem.CellCenterFVM.InField.applyTRS = InnerFaces

Simulator.SubSystem.CellCenterFVM.InField.InitVars = x y
Simulator.SubSystem.CellCenterFVM.InField.InitDef = sqrt(x^2+y^2)

# here "rad" is a new user-defined variable that can be used
# to simplify the final expressions
Simulator.SubSystem.CellCenterFVM.InField.Vars = x y rad
Simulator.SubSystem.CellCenterFVM.InField.Def = \
 0. 0. 0. if(rad<1.,0.00002854,0.00002854/2.) \
 0. if(rad<1.,0.00000866,0.00000866/2.) 0. 0. 0. 0. 0.
 if(x<1.0,11360.,100.*(rad-1.)) 0. 195. 195.
```

8.2 Polynomial reconstruction

The following options should be kept frozen:

```
Simulator.SubSystem.CellCenterFVM.SetupCom = LeastSquareP1Setup
Simulator.SubSystem.CellCenterFVM.SetupNames = Setup1
Simulator.SubSystem.CellCenterFVM.Setup1.stencil = FaceVertexPlusGhost
Simulator.SubSystem.CellCenterFVM.UnSetupCom = LeastSquareP1UnSetup
Simulator.SubSystem.CellCenterFVM.UnSetupNames = UnSetup1
Simulator.SubSystem.CellCenterFVM.Data.PolyRec = LinearLS2D
Simulator.SubSystem.CellCenterFVM.Data.Limiter = Venktn2D
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.coeffEps = 1.0
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.useFullStencil = true
# set true the following for backward compatibility, but false should behave
    better
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.useNodalExtrapolationStencil =
    false
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.length = 1.0
```

The following factor determines if the simulation is of first, second or in-between order:

```
# 0 <= gradientFactor <= 1, with  0. (first order), 1. (second order)
Simulator.SubSystem.CellCenterFVM.Data.LinearLS2D.gradientFactor = 0.
```

This is an interactive parameter that can be placed into the interactive file. Another interactive parameter, important for second order computations, is

```
Simulator.SubSystem.CellCenterFVM.Data.LinearLS2D.limitRes = -4.0
```

This corresponds to the minimum residual at which the freezing of the flux limiter should be applied for flows exhibiting discontinuities or steep gradients (e.g., in temperature). In practice, `limitRes` can be kept at -4 till when the simulation reaches a limit cycle and then can be increased to 8. in order to exit the cycle. This cure is not always effective and it often depends on the moment when `limitRes` is increased.

WATCH OUT: *Before restarting a simulation from a second order solution, if the limiter has not been explicitly saved in the CFmesh file (see below),* `limitRes` *has to be set back to* -4.

In order to save the limiter in second order calculations (when `gradientFactor = 1.`), the following options must be added to the CFcase **before** starting the computation:

```
Simulator.SubSystem.CFmesh.Data.ExtraStateVarNames = limiter

# the following must be the total number of equations
Simulator.SubSystem.CFmesh.Data.ExtraStateVarStrides = 15
```

Finally, in order to restart from a file in which the limiter **has been already saved**, the following line should be included in the CFcase file:

```
Simulator.SubSystem.CFmeshFileReader.Data.ExtraStateVarNames = InitLimiter
```

8.2.1 Boundary conditions (physics-independent examples)

Boundary conditions fields will be applied also during initialization on the corresponding boundary TRS, in such a way that *ghost states* (dummy cell centers that lie outside the computational domain) are set consistently before starting computing numerical fluxes.
The following example shows how to specify a full set of boundary conditions (four in this case, but real settings will obviously depend on the mesh in use).

```
# list of the names of the objects defining each boundary condition
Simulator.SubSystem.CellCenterFVM.BcComds = \
    MirrorVelocityFVMCC SuperInletFVMCC SuperOutletFVMCC

# list of aliases that the use must define for configuring each BC
Simulator.SubSystem.CellCenterFVM.BcNames = Mirror SInlet SOutlet



# apply MirrorVelocityFVMCC to the Symmetry TRS
Simulator.SubSystem.CellCenterFVM.Mirror.applyTRS = Symmetry

# IDs corresponding to the velocity components within the state vector
# (they dependent on the model in use, here 2D Euler example)
Simulator.SubSystem.CellCenterFVM.Mirror.VelocityIDs = 1 2

# array of flags where "1" correspond to variables for which
# a zero gradient has to be imposed (2D Euler case here)
Simulator.SubSystem.CellCenterFVM.Mirror.ZeroGradientFlags = 1 0 0 1




# apply SuperInletFVMCC to the Inlet TRS
Simulator.SubSystem.CellCenterFVM.SInlet.applyTRS = Inlet

# analytical functions can be defined here as for InitState
# (2-step option is not available though)
Simulator.SubSystem.CellCenterFVM.SInlet.Vars = x y
Simulator.SubSystem.CellCenterFVM.SInlet.Def = 1000. 11360. 0. 195.
```

specify the settings for a supersonic inlet: all update variables (Puvt, Rhoivt, Cons, etc. depending on the actual case) have to be prescribed. Additional interactive parameters (to be put in the interactive file) can be used for running stiff cases (typically 3D), for which, for instance, it may be impossible to start with the full velocity:

```
# the following settings tell the solver to multiply the variable with
# ID = 11 (x-velocity) by a factor (must <= 1.0)
Simulator.SubSystem.CellCenterFVM.SInlet.InteractiveVarIDs = 1
Simulator.SubSystem.CellCenterFVM.SInlet.InteractiveFactor = 1.0
```

```
Simulator.SubSystem.CellCenterFVM.SOutlet.applyTRS = Outlet

# array of flags where "1" correspond to variables for which
# a zero gradient has to be imposed
Simulator.SubSystem.CellCenterFVM.SOutlet.ZeroGradientFlags = 1 1 1 1
```

8.3 Finite Volume Navier-Stokes

Required libs: libFiniteVolume, libNavierStokes, libFiniteVolumeNavierStokes.

```
Simulator.SubSystem.SpaceMethod = CellCenterFVM
Simulator.SubSystem.CellCenterFVM.ComputeRHS = NumJacob
Simulator.SubSystem.CellCenterFVM.ComputeTimeRHS = PseudoSteadyTimeRhs
```

8.3.1 Convective flux schemes

The user must choose one of the following convective flux schemes:

The following work for both single- and multi-temperatures:

```
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = AUSMPlus2D
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlus2D.choiceA12 = 1

# it includes some built-in preconditioning to handle low Mach flows
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = AUSMPlusUp2D
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusUp2D.choiceA12 = 1
# the free stream Mach number must be specified
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusUp2D.machInf = 2.

# HUS flux works for all NEQ models but is generally less stable and carbuncle
    prone
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = HUS2D
Simulator.SubSystem.CellCenterFVM.Data.HUS2D.isNatural = true

# Van Leer scheme is good enough for inviscid flows but inaccurate for viscous
    cases
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = VanLeer2D

# standard Roe scheme
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = Roe
```

```
# Roe scheme with entropy fix
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = RoeEntropyFix
Simulator.SubSystem.CellCenterFVM.Data.RoeEntropyFix.entropyFixID = 1 #2

# Roe scheme (Sanders' carbuncle fix) works only for neutral mixtures
# in thermo-chemical NEQ
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = RoeSA
Simulator.SubSystem.CellCenterFVM.Data.RoeSA.entropyFixID = 1  #2 or 3
# the following option if activated could be more stable in some cases
#Simulator.SubSystem.CellCenterFVM.Data.RoeSA.oldStencil = true
```

8.3.2 Variable sets for Navier-Stokes

The following settings define some variable that are needed in different phases of the simulation. In particular, the user should substitute `<VARSET>` with `Puvt` in 2D or `Pvt` in 3D:

```
# variables in which the solution is stored and updated
# (use Rhoivt for thermal equilibrium)
Simulator.SubSystem.CellCenterFVM.Data.UpdateVar = <VARSET>

# variables in which the equations are formulated must ALWAYS be Cons
Simulator.SubSystem.CellCenterFVM.Data.SolutionVar = Cons

# variables in which the diffusive fluxes are computed
# (use Rhoivt for thermal equilibrium)
Simulator.SubSystem.CellCenterFVM.Data.DiffusiveVar = <VARSET>

# diffusive flux must be NavierStokes for viscous computations
Simulator.SubSystem.CellCenterFVM.Data.DiffusiveFlux = NavierStokes
```

8.3.3 2D axisymmetric settings

```
Simulator.SubSystem.CellCenterFVM.Data.isAxisymm = true
```

should always be present in 2D axisymmetric computations.

```
# source terms for the axisymmetric case
Simulator.SubSystem.CellCenterFVM.Data.SourceTerm = NavierStokes2DAxiST

# IDs corresponding to the velocity variables (momentum equations)
Simulator.SubSystem.CellCenterFVM.Data. NavierStokes2DAxiST.uvIDs = 11 12
```

8.3.4 Boundary conditions examples for Navier-Stokes

```
# ...BcComds = SubInletEuler2DUVTFVMCC
# ...BcNames = BcInlet
Simulator.SubSystem.CellCenterFVM.BcInlet.applyTRS = Inlet
# inlet velocity components
Simulator.SubSystem.CellCenterFVM.BcInlet.Vx = 121.151
Simulator.SubSystem.CellCenterFVM.BcInlet.Vy = 0.0
# inlet temperature
Simulator.SubSystem.CellCenterFVM.BcInlet.T = 298.15
```

```
# ...BcComds = SubOutletEuler2DFVMCC
# ...BcNames = BcOutlet
Simulator.SubSystem.CellCenterFVM.BcOutlet.applyTRS = Outlet
Simulator.SubSystem.CellCenterFVM.BcOutlet.P = 986.369
```

```
# ...BcComds = FarFieldEuler2DFVMCC #FarFieldEuler3DFVMCC
# ...BcNames = Infarfld
Simulator.SubSystem.CellCenterFVM.Infarfld.applyTRS = FarField
Simulator.SubSystem.CellCenterFVM.Infarfld.Tinf = 288.15
Simulator.SubSystem.CellCenterFVM.Infarfld.Pinf = 43.489948
Simulator.SubSystem.CellCenterFVM.Infarfld.Uinf = 170.131324
Simulator.SubSystem.CellCenterFVM.Infarfld.Vinf = 0.0
#Simulator.SubSystem.CellCenterFVM.Infarfld.Winf = 0.0
# specify analytical functions for each variable given by InputVar
# a variable transformation from input to update will be applied if
# InputVar is different from Simulator.SubSystem.CellCenterFVM.UpdateVar
#Simulator.SubSystem.CellCenterFVM.Infarfld.InputVar = Puvt
#Simulator.SubSystem.CellCenterFVM.Infarfld.Vars = x y #z
#Simulator.SubSystem.CellCenterFVM.Infarfld.Def = if(x<0,43.489948,30*y)
    170.131324 0.0 288.15
```

```
# ...BcComds = FarFieldEulerChar2DFVMCC #FarFieldEulerChar3DFVMCC
# ...BcNames = Infarfld
Simulator.SubSystem.CellCenterFVM.Infarfld.applyTRS = FarField
Simulator.SubSystem.CellCenterFVM.Infarfld.Tinf = 288.15
Simulator.SubSystem.CellCenterFVM.Infarfld.Pinf = 43.489948
Simulator.SubSystem.CellCenterFVM.Infarfld.Uinf = 170.131324
Simulator.SubSystem.CellCenterFVM.Infarfld.Vinf = 0.0
#Simulator.SubSystem.CellCenterFVM.Infarfld.Winf = 0.0
# specify analytical functions for each variable given by InputVar
# a variable transformation from input to update will be applied if
# InputVar is different from Simulator.SubSystem.CellCenterFVM.UpdateVar
#Simulator.SubSystem.CellCenterFVM.Infarfld.InputVar = Puvt
#Simulator.SubSystem.CellCenterFVM.Infarfld.Vars = x y #z
#Simulator.SubSystem.CellCenterFVM.Infarfld.Def = if(x<0,43.489948,30*y)
    170.131324 0.0 288.15
```

```
# ...BcComds = NoSlipWallIsothermalNSvtFVMCC
# ...BcNames = NSWall
# apply NoSlipWallIsothermalNSvtFVMCC to the Wall
# (TRS name coming from the initial mesh file)
Simulator.SubSystem.CellCenterFVM.NSWall.applyTRS = Wall

# if an adiabatic condition is needed set this flag to true
Simulator.SubSystem.CellCenterFVM.NSWall.Adiabatic = false

# imposed wall temperature
Simulator.SubSystem.CellCenterFVM.NSWall.TWall = 615.0

In order to impose a radiative equilibrium condition, additional options are
    needed:
\begin{lstlisting}[breaklines]
Simulator.SubSystem.CellCenterFVM.NSWall.RadEquilibrium = true

# emissivity
Simulator.SubSystem.CellCenterFVM.NSWall.Emissivity = 0.9

# maximum allowable change in temperature between two consecutive time step
Simulator.SubSystem.CellCenterFVM.NSWall.MaxRadEqDTwall = 100.

# temperature of the distant body (typically 0 or free stream value)
Simulator.SubSystem.CellCenterFVM.NSWall.DistantBodyTemp = 0.
```

8.4 Nodal Extrapolation

Since flow solution is computed in the cell centers, nodal values must be extrapolated from cell centers to the mesh vertices for visualization purposes or for computing viscous gradients. This is accomplished by `NodalExtrapolation` objects. In viscous cases, where a slip condition and, possibly, a temperature are imposed at the wall, the following settings must be added in order to strongly impose the desired values.

```
# this specifies the name of the nodal extrapolator object
Simulator.SubSystem.CellCenterFVM.Data.NodalExtrapolation = DistanceBasedGMove

# the name(s) of the boundary TRS(s) on which imposing values strongly
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMove.TRSName = Wall

# IDs of the variables to be imposed strongly on the boundaries listed in TRSName
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMove.ValuesIdx = 1 2 3 #uID
    vID TID

# values of the selected variables to be imposed strongly
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMove.Values = 0. 0. 615. #u
    v T at the wall

# list determines the priority of one TRS (and BC) over another in corner nodes
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMove.TrsPriorityList = Wall
    Symmetry Inlet Outlet
```

WATCH OUT: *Nodal extrapolation settings must be consistent with the boundary conditions (same velocity IDs, same temperature at the wall, etc.).*

8.5 Finite Volume MHD

Required libs: libFiniteVolume, libMHD, libFiniteVolumeMHD.

The following standard settings for the implicit FV solver should not be changed except the save rate which should be the same as the save rate of the Tecplot solution data file in order to be able to write the $\nabla \cdot \vec{B}$ error values correctly in the data file:

```
Simulator.SubSystem.SpaceMethod = CellCenterFVM
Simulator.SubSystem.CellCenterFVM.ComputeRHS = NumJacobMHD
Simulator.SubSystem.CellCenterFVM.NumJacobMHD.SaveRate = 100
Simulator.SubSystem.CellCenterFVM.ComputeTimeRHS = StdTimeRhs (or BDF2TimeRhs
    when using BDF2)
Simulator.SubSystem.CellCenterFVM.BDF2TimeRhs.zeroDiagValue = 0 0 0 0 0 0 0 0 1
```

Moreover, with the last line, we make the time derivative of the last equation in the modified ideal MHD system due to the hyperbolic divergence cleaning vanish. Hence, we apply a pure Newton iteration on the last equation which reduced to the $\nabla \cdot \vec{B} = 0$ constraint which is linear in $\vec{B}$ even after discretization. Thus, we assure convergence of the constraint upto machine accuracy even at each subiteration when using BDF2 [17].

8.5.1 Interactive file

For the present solver, the interactive parameters mainly utilized are the CFL number, diffusion reduction coefficient for the modified Rusanov scheme with tunable dissipation and the residual threshold value below which either the limiter is frozen or historical modification of limiter is applied as a remedy for convergence hampering for TVD schemes. The content of the interactive file in its most general form is given as follows:

```
Simulator.SubSystem.BwdEuler(or BDF2).Data.CFL.Interactive.CFL = 1000.0
Simulator.SubSystem.CellCenterFVM.Data.LinearLS3D(or LinearLS2D).limitRes = -4.0
Simulator.SubSystem.CellCenterFVM.Data.LinearLS2D(or LinearLS3D).gradientFactor =
      0.
Simulator.SubSystem.CellCenterFVM.Data.Centred.LaxFried.DiffCoeff = 0.3
Simulator.SubSystem.CellCenterFVM.Data.Centred.MHD3DProjectionConsLaxFriedTanaka(
    or MHD2DProjectionConsLaxFriedTanaka).DiffCoeff = 0.3
Simulator.SubSystem.CellCenterFVM.Data.Centred.MHD3DConsLaxFriedTanaka(or
    MHD2DConsLaxFriedTanaka).DiffCoeff = 0.3
```

8.5.2 Convective flux schemes

The user must choose one of the following (Rusanov scheme with tunable dissipation is recommended for most cases):

```
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = Centred
```

defines the option for Rusanov scheme. Specific cases that involve additional options were explained in detail earlier.

```
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = Roe
Simulator.SubSystem.CellCenterFVM.Data.Roe.Flux = MHD2DProjectionConsRoe (or
    MHD2DConsRoe or MHD3DProjectionConsRoe or MHD3DConsRoe)
```

define the option for Roe's scheme.

$\vec{B}_0 + \vec{B}_1$ splitting method For solar wind/planetary magnetosphere interaction simulations, the planetary intrinsic magnetic field is modelled as a dipole ($\vec{B}_0$).

```
Simulator.SubSystem.MHD3DProjection(or MHD2DProjection or MHD2D or MHD3D).
    ConvTerm.mX = 0.0
Simulator.SubSystem.MHD3DProjection(or MHD2DProjection or MHD2D or MHD3D).
    ConvTerm.mY = 0.0
Simulator.SubSystem.MHD3DProjection(or MHD3D).ConvTerm.mZ = −3000.0
```

define the magnetic dipole moment of the planetary magnetic field with the center of the dipole located at the origin.

```
Simulator.SubSystem.CellCenterFVM.Data.Centred.Flux =
    MHD3DProjectionConsLaxFriedTanaka (or MHD3DConsLaxFriedTanaka or
    MHD2DConsLaxFriedTanaka or MHD2DProjectionConsLaxFriedTanaka)
Simulator.SubSystem.CellCenterFVM.Data.Centred.MHD3DProjectionConsLaxFriedTanaka(
    or MHD3DConsLaxFriedTanaka or MHD2DConsLaxFriedTanaka or
    MHD2DProjectionConsLaxFriedTanaka).NameFluxFunction = Powell99 (or Tanaka94)
```

define the necessary additional lines when using the Rusanov scheme together with the abovementioned type of problems. Note that in this type of problems Roe's scheme is not preferred as it was not observed to have the necessary robustness level. Therefore, we stick to the Rusanov scheme with tunable dissipation. Moreover, "Powell99" implementation made according to [13] for only 3D problems is preferred over "Tanaka94" implementation made according to [14] for both 2D and 3D problems as the latter implementation which involves matrix transformations is slower in comparison with the former.

8.5.3 Variable sets

The following settings do not change for the present solver:

```
Simulator.SubSystem.CellCenterFVM.Data.UpdateVar   = Cons
Simulator.SubSystem.CellCenterFVM.Data.SolutionVar = Cons
Simulator.SubSystem.CellCenterFVM.Data.LinearVar   = Cons
```

8.5.4 Source terms

```
Simulator.SubSystem.CellCenterFVM.Data.hasSourceTerm = true
Simulator.SubSystem.CellCenterFVM.Data.SourceTerm = MHDConsACAST
```

should always be present in simulations involving hyperbolic divergence cleaning technique.

```
Simulator.SubSystem.CellCenterFVM.Data.hasSourceTerm = true
Simulator.SubSystem.CellCenterFVM.Data.SourceTerm = MHD2DPowellST (or
    MHD3DPowellST)
```

should always be present in simulations involving Powell's source term technique.

8.5.5 Boundary conditions examples for MHD

Boundary condition fields will be applied also during initialization on the corresponding boundary TRS, in such a way that *ghost states* (dummy cell centers that lie outside the computational

domain) are set consistently before starting computing numerical fluxes.
The following example shows how to specify a full set of boundary conditions (three in this case, but real settings will obviously depend on the mesh in use).

```
# list of the names of the objects defining each boundary condition
Simulator.SubSystem.CellCenterFVM.BcComds = \
      MirrorMHD3DProjectionTanakaPFixFVMCC \
      SuperInletFVMCC \
      SuperOutletMHD3DProjectionFVMCC

# list of aliases that the user must define for configuring each BC
Simulator.SubSystem.CellCenterFVM.BcNames = Wall Inlet Outlet
```

- Ionosphere/magnetosphere boundary condition [16, 18]

  ```
  # apply MirrorMHD3DProjectionTanakaPFixFVMCC to the SlipWall
  # (TRS name coming from the initial mesh file)
  Simulator.SubSystem.CellCenterFVM.Wall.applyTRS = SlipWall

  # imposed density and pressure
  Simulator.SubSystem.CellCenterFVM.Wall.rhoFixed = 1.0
  Simulator.SubSystem.CellCenterFVM.Wall.pFixed = 8.0 (This value should be 8
      times the freestream solar wind plasma pressure value which is equal to
      1.0 in this example.)
  ```

  This boundary condition is based on [13].

- Superfast inlet boundary condition [15, 16, 17, 18]

  ```
  # apply SuperInletFVMCC to the SuperInlet TRS
  Simulator.SubSystem.CellCenterFVM.Inlet.applyTRS = SuperInlet
  Simulator.SubSystem.CellCenterFVM.Inlet.Vars = x y z
  Simulator.SubSystem.CellCenterFVM.Inlet.Def =  1.05100 \
                                                 -3.99981 \
                                                 0.283107 \
                                                 -0.0381893 \
                                                 0.403907 \
                                                 0.399628 \
                                                 -0.489089 \
                                                 8.09741 \
                                                 (0.0 in case of 9 equations)
  ```

- Superfast outlet boundary condition [15, 16, 17, 18]

  ```
  Simulator.SubSystem.CellCenterFVM.Outlet.applyTRS = SuperOutlet
  Simulator.SubSystem.CellCenterFVM.Outlet.refPhi = 0.0
  ```

  where the scalar potential function, ϕ, value is kept constant in the ghost cells typically equal to 0 in all the testcases tried so far.

8.6 Finite Volume NEQ (ATD)

As far as the simulation of aerothermodynamics is concerned (see [4, 7] for technical details), COOLFluiD offers:

- 2D / axisymmetric / 3D FV solver for thermo-chemical equilibrium (LTE with or without demixing effect) and NEQ (different multi-temperature models) viscous flows on unstructured hybrid grids with various schemes (AUSM family, HUS, Roe, modified Steger-Warming, etc.).
- The same solver can also handle incompressible inductively coupled plasmas (ICP) in LTE (extension to thermo-chemical NEQ is underway) where the Navier-Stokes equations are weakly coupled with the electro-magnetic induction equations.
- COOLFluiD interfaces the Mutation F77 (version 2.0) and Mutation++ for the accurate computation of thermodynamic, transport and chemical kinetics properties in all temperature regimes, with different LTE and thermo-chemical NEQ models, including pioneering collision-radiative models with > 100 chemical species [6].
- Simulations on neutral or ionized mixtures of argon, air, CO_2, nitrogen is available.
- A new generation Residual Distribution solver for improving accuracy of thermo-chemical NEQ flows simulations on unstructured simplex-element meshes (with triangles or tetrahedra) is currently under development [4].
- Possibility of reusing all the available coupling algorithms to get arbitrarily complex multi-physics steady or unsteady simulations on deforming meshes [10].

Required libs: libFiniteVolume, libNavierStokes, libFiniteVolumeNavierStokes, libNEQ, libFiniteVolumeNEQ.

The following standard settings for the implicit Finite Volume solver should not be changed:

```
Simulator.SubSystem.SpaceMethod = CellCenterFVM
Simulator.SubSystem.CellCenterFVM.ComputeRHS = NumJacobFast
Simulator.SubSystem.CellCenterFVM.NumJacobFast.FreezeDiffCoeff = true
Simulator.SubSystem.CellCenterFVM.ComputeTimeRHS = PseudoSteadyTimeRhs
```

8.6.1 Convective flux schemes

The user must choose one of the following convective flux schemes (AUSM+ is recommended for most cases):

The following work for both single- and multi-temperatures:

```
# AUSM+ is the most stable and works for all equilibrium and NEQ models
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = AUSMPlusMS2D
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusMS2D.choiceA12 = 5

# AUSM+up flux works for all equilibrium and nonequilibrium models
# it includes some built-in preconditioning to handle low Mach flows
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = AUSMPlusUpMS2D
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusUpMS2D.choiceA12 = 5
# the free stream Mach number must be specified
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusUpMS2D.machInf = 30.

# HUS flux works for all NEQ models but is generally less stable and carbuncle
    prone
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = HUSMS2D
Simulator.SubSystem.CellCenterFVM.Data.HUSMS2D.isNatural = true
```

The following scheme works only for two-temperatures and w/o ionization:

```
# Roe scheme (Sanders' carbuncle fix) works only for neutral mixtures
# in thermo-chemical NEQ
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = RoeTCNEQ2DSA
Simulator.SubSystem.CellCenterFVM.Data.RoeTCNEQ2DSA.entropyFixID = 1  #2 or 3
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.noElectronicEnergy = true
```

8.6.2 Variable sets for NEQ

The following settings define some variable that are needed in different phases of the simulation. In particular, the user should substitute `<VARSET>` with `Rhoivt` for single-temperature or `RhoivtTv` for multi-temperature.

```
# variables in which the solution is stored and updated
# (use Rhoivt for thermal equilibrium)
Simulator.SubSystem.CellCenterFVM.Data.UpdateVar = <VARSET>

# variables in which the equations are formulated must ALWAYS be Cons
Simulator.SubSystem.CellCenterFVM.Data.SolutionVar = Cons

# variables in which the diffusive fluxes are computed
# (use Rhoivt for thermal equilibrium)
Simulator.SubSystem.CellCenterFVM.Data.DiffusiveVar = <VARSET>

# diffusive flux must be NavierStokes for viscous computations
Simulator.SubSystem.CellCenterFVM.Data.DiffusiveFlux = NavierStokes
```

8.6.3 2D and axisymmetric settings for NEQ

```
Simulator.SubSystem.CellCenterFVM.Data.isAxisymm = true
```

should always be present in 2D axisymmetric computations.

The following options must be activated only for axisymmetric calculations in thermo-chemical NEQ:

```
# source terms for the axisymmetric case
Simulator.SubSystem.CellCenterFVM.Data.SourceTerm = \
  NavierStokes2DTCNEQAxiST Euler2DCTNEQST

# IDs corresponding to the velocity variables (momentum equations)
Simulator.SubSystem.CellCenterFVM.Data.NavierStokes2DTCNEQAxiST.uvIDs = 11 12
```

The following options must be activated only for axisymmetric calculations in chemical NEQ (thermal equilibrium):

```
# source terms for the axisymmetric case
Simulator.SubSystem.CellCenterFVM.Data.SourceTerm = \
  NavierStokes2DNEQAxiST Euler2DCNEQST

# IDs corresponding to the velocity variables (momentum equations)
Simulator.SubSystem.CellCenterFVM.Data.NavierStokes2DNEQAxiST.uvIDs = 11 12
```

WATCH OUT: *In 2D non-axisymmetric cases, only* `Euler2DCTNEQST` *or* `Euler2DCNEQST` *should be declared as source term.*

8.6.4 Boundary Conditions examples for ATD

```
# ...BcComds = NoSlipWallIsothermalNSrvtMultiFVMCC
# ...BcNames = NSWall
# apply NoSlipWallIsothermalNSrvtMultiFVMCC to the Wall
# (TRS name coming from the initial mesh file)
Simulator.SubSystem.CellCenterFVM.NSWall.applyTRS = Wall

# if an adiabatic condition is needed set this flag to true
Simulator.SubSystem.CellCenterFVM.NSWall.Adiabatic = false

# imposed wall temperature
Simulator.SubSystem.CellCenterFVM.NSWall.TWall = 615.0

In order to impose a radiative equilibrium condition, additional options are
    needed:
\begin{lstlisting}[breaklines]
Simulator.SubSystem.CellCenterFVM.NSWall.RadEquilibrium = true

# emissivity
Simulator.SubSystem.CellCenterFVM.NSWall.Emissivity = 0.9

# maximum allowable change in temperature between two consecutive time step
Simulator.SubSystem.CellCenterFVM.NSWall.MaxRadEqDTwall = 100.

# temperature of the distant body (typically 0 or free stream value)
Simulator.SubSystem.CellCenterFVM.NSWall.DistantBodyTemp = 0.
```

A super-catalytic wall condition, imposing LTE at the wall, can be imposed by replacing the BC object name `NoSlipWallIsothermalNSrvtMultiFVMCC` with `NoSlipWallIsothermalNSrvtLTEMultiFVMCC`.

8.6.5 Nodal Extrapolation

Since flow solution is computed in the cell centers, nodal values must be extrapolated from cell centers to the mesh vertices for visualization purposes or for computing viscous gradients. This is accomplished by `NodalExtrapolation` objects. In viscous NEQ cases, where a slip condition and, possibly, a temperature are imposed at the wall, the following settings must be added in order to strongly impose the desired values.

```
# this specifies the name of the nodal extrapolator object
Simulator.SubSystem.CellCenterFVM.Data.NodalExtrapolation =
    DistanceBasedGMoveRhoivt

# the name(s) of the boundary TRS(s) on which imposing values strongly
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.TRSName = Wall

# IDs of the variables to be imposed strongly on the boundaries listed in TRSName
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.ValuesIdx = \
  11 12 13 14

# values of the selected variables to be imposed strongly
```

```
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.Values = \
   0. 0. 615. 615.

# list determines the priority of one TRS (and BC) over another in corner nodes
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.TrsPriorityList =
     \
    Wall Symmetry Inlet Outlet
```

WATCH OUT: *Nodal extrapolation settings must be consistent with the boundary conditions (same velocity IDs, same temperature at the wall, etc.).*

Radiative equilibrium case When radiative equilibrium is imposed at the wall, besides specifying appropriate boundary condition settings (see above), the nodal extrapolation must also be adapted consistently, as follows.

```
# only velocity IDs and values must be prescribed at the wall nodes, since
# temperature will be computed on-the-fly by an iterative procedure
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.ValuesIdx = 11 12
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.Values = 0. 0.

# this flag must be activated
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.RadEquilibrium =
     true
```

Super-catalytic case (LTE at the wall) A super-catalytic BC requires setting `DistanceBasedGMoveRhoivtLTE` instead of `DistanceBasedGMoveRhoivt`.

8.6.6 Post-processing: radiation coupling and surface quantities computation

We declare two post-processing numerical commands.

```
# name of the post-processing objects
Simulator.SubSystem.DataPostProcessing = DataProcessing DataProcessing

# user-defined configuration names for the post-processing object
Simulator.SubSystem.DataPostProcessingNames = DataProcessing2 DataProcessing3
```

Radiation coupling **Required libs:** libparade, libradiation, libParadeI, libRadiativeTransfer.

The following options control the postprocessing:

```
# how often the post-processing is applied (this is an interactive option,
# it can go to the interactive file)
Simulator.SubSystem.DataProcessing2.ProcessRate = 100

# flag telling if to skip the coupling for the first iteration (advised)
Simulator.SubSystem.DataProcessing2.SkipFirstIteration = true
```

The following options allow for solving the radiation transport on the stagnation line (works only in serial mode) using PARADE or to solve optically thin in the whole domain.

```
# name of the command object implementing the radiation coupling
# Slab1DFVMCC corresponds to 1D infinite tangent slab method
Simulator.SubSystem.DataProcessing2.Comds = Slab1DFVMCC

# user-defined configuration name for "Slab1DFVMCC"
Simulator.SubSystem.DataProcessing2.Names = Radiation1D

# wall TRS boundary required by the algorithm
Simulator.SubSystem.DataProcessing2.Radiation1D.applyTRS = Wall

# if this option is present, providing the x,y coordinates of the stagnation
    point,
# the RTE will be solved ONLY on the stagnation line
# this setting should be commented out in case all mesh lines are needed (not
    supported)
Simulator.SubSystem.DataProcessing2.Radiation1D.StagnationPoint = 0. 0.

# radiation library (only Parade is available at the moment)
Simulator.SubSystem.DataProcessing2.Radiation1D.RadiationLibrary = Parade

# flag telling if the flow is emission dominated (optically thin assumption)
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.EmisDom = true

# temperature to impose on the wall boundary
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.Tb1 = 615.0

# free stream temperature to impose on the inlet boundary
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.Tb2 = 195.0

# under-relaxation factor (must be <= 1)
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.UndRel = 1.0
```

The following options allow for solving the radiation transport with the **Monte-Carlo** algorithm and to couple back the computed $\nabla \cdot \mathbf{q}_{rad}$ to the flow energy conservation equation.

```
# name of the command object implementing the radiation transport via Monte-Carlo
Simulator.SubSystem.DataProcessing2.Comds = RadiativeTransferMonteCarloFVMCC

# user-defined configuration name for "RadiativeTransferMonteCarloFVMCC"
Simulator.SubSystem.DataProcessing2.Names = RT

# Name of the topological region where to apply the algorithm
Simulator.SubSystem.DataProcessing2.RT.applyTRS = InnerFaces

# Names of the topological regions corresponding to solid walls
Simulator.SubSystem.DataProcessing2.RT.wallTrsNames = Wall1 Wall2

# Names of the topological regions corresponding to symmetry planes
Simulator.SubSystem.DataProcessing2.RT.symmetryTrsNames = Symmetry

# Names of the topological regions corresponding to other boundaries
Simulator.SubSystem.DataProcessing2.RT.boundaryTrsNames = Inlet Outlet

# User-defined number of rays (photons) shot by each computational cell
Simulator.SubSystem.DataProcessing2.RT.numberOfRays = 20

# Maximum number of visited cell during ray tracing
Simulator.SubSystem.DataProcessing2.RT.MaxNbVisitedCells = 200
```

```
# Number of wavelengths to be considered to obtain a reduced spectra
Simulator.SubSystem.DataProcessing2.RT.ReducedSpectralSize = 500

# Wall emissivity
Simulator.SubSystem.DataProcessing2.RT.WallEmissivity = 0.

# Wall absorption coefficient
Simulator.SubSystem.DataProcessing2.RT.WallAbsorption = 0.644

# ID of the temperature in the state vector
# (typically = number of species + spatial dimension)
Simulator.SubSystem.DataProcessing2.RT.TemperatureID = 13

# Free stream temperature (if>0, it will impose q_rad=0 for the cell where active
    )
#Simulator.SubSystem.DataProcessing2.RT.FreeStreamTemperature = 640.

# Flag to tell to use the Planck function at the wall
# (alternative to specifying WallEmissivity)
#Simulator.SubSystem.DataProcessing2.RT.PlanckFunction = true
```

In axisymmetric cases, the following must be specified:

```
# Specify if to use the axisymmetric ray tracing
Simulator.SubSystem.DataProcessing2.RT.Axi = true

# Specify the number of cells in the streamwise direction
Simulator.SubSystem.DataProcessing2.RT.nCx = 80

# Specify the number of cells normal to the wall
Simulator.SubSystem.DataProcessing2.RT.nCy = 150
```

PARADE settings to be used in combination with Monte-Carlo:

```
# radiation library (only Parade is available at the moment)
Simulator.SubSystem.DataProcessing2.RT.RadiationLibrary = Parade

# flag telling if the flow is emission dominated (optically thin assumption)
Simulator.SubSystem.DataProcessing2.RT.Parade.EmisDom = false

# Minumum wavelength (must be consistent with "wavlo" declared in parade.con)
Simulator.SubSystem.DataProcessing2.RT.Parade.WavelengthMin = 2000.

# Maximum wavelength (must be consistent with "wavhi" declared in parade.con)
Simulator.SubSystem.DataProcessing2.RT.Parade.WavelengthMax = 40000.

# number of wavelengths for which radiative properties are computed at a time
# if < "npoints" declared in parade.con, Monte-Carlo will compute radiative
    properties
# for this number of wavelengths at once
Simulator.SubSystem.DataProcessing2.RT.Parade.WavelengthStride = 10000

# Path where PARADE is installed with all data files
Simulator.SubSystem.DataProcessing2.RT.Parade.path = /home/myuser/PARADEv3.1/
    parade31

# when set to 1, this allows for restarting from previously computed radiative
    properties
# as long as the number of processors remains the same as in the previous run
Simulator.SubSystem.DataProcessing1.RT.Parade.ReuseProperties = 0
```

The flow radiation cupoling is controlled by one interactive parameter (can be changed interactively inside the .inter file)

```
# RadRelaxationFactor = 0        uncoupled case
# 0 < RadRelaxationFactor <= 1   coupled case, this is an under-relaxation factor
Simulator.SubSystem.CellCenterFVM.Data.Euler2DCTNEQST.RadRelaxationFactor = 1.0
```

WATCH OUT: *In order to run the flow-radiation coupling the executable* `parade` *together with all required PARADE input files (parade.con) must be present in the working directory.*

8.7 Surface quantities

Required libs: libAeroCoefFVM, libAeroCoefFVMNEQ.

Surface quantities such as surface pressure, temperature, heat flux and skin friction can be computed and saved to **one single Tecplot file** with the following settings.

```
# how often the post-processing is applied (this is an interactive option,
# it can go to the interactive file)
Simulator.SubSystem.DataProcessing3.ProcessRate = 10

# name of the command object implementing the post-processing
Simulator.SubSystem.DataProcessing3.Comds = NavierStokesSkinFrictionHeatFluxCCNEQ

# user-defined configuration name for "NavierStokesSkinFrictionHeatFluxCCNEQ"
Simulator.SubSystem.DataProcessing3.Names = SkinFriction

# boundary TRS on which applying the post-process
Simulator.SubSystem.DataProcessing3.SkinFriction.applyTRS = Wall

# output Tecplot data file on which surface quantities will be written
Simulator.SubSystem.DataProcessing3.SkinFriction.OutputFileWall = walldata.plt

# ALL the following free stream values and update variable IDs MUST be specified
Simulator.SubSystem.DataProcessing3.SkinFriction.rhoInf = 0.0000372 # density
Simulator.SubSystem.DataProcessing3.SkinFriction.pInf = 2.1         # pressure
Simulator.SubSystem.DataProcessing3.SkinFriction.uInf = 11360.      # x-velocity
Simulator.SubSystem.DataProcessing3.SkinFriction.TInf = 195.        # temperature
Simulator.SubSystem.DataProcessing3.SkinFriction.UID = 11           # x-velocity
    ID
Simulator.SubSystem.DataProcessing3.SkinFriction.VID = 12           # y-velocity
    ID
Simulator.SubSystem.DataProcessing3.SkinFriction.TID = 13           # temperature
     ID
```

9 Mesh Fitting Algorithms

Required libs: libMeshTools, libMeshToolsFVM.

The COOLFLuiD framework contains an autonomous physics-driven mesh deformation algorithms capable of performing mesh r-adaptive simulations based on both local physical and geometrical properties, respectively, of the flow field and of the mesh elements. This section presents the different configuration options needed to apply the mesh deformation.
The mesh fitting algorithms options are simply added to a classical `CFcase` file.

9.1 Linear System Solver (LSS)

Required libs: libPetscI.

The mesh fitting algorithms reuse the Linear System Solver, described in section 7.

```
# setting for PETSC linear system solver
Simulator.SubSystem.LinearSystemSolver = PETSC
Simulator.SubSystem.LSSNames = MeshAlgoLSS
```

The option `UseNodalBased` ensures that the system is nodal-based and not cell-centred based

```
Simulator.SubSystem.MeshAlgoLSS.Data.UseNodeBased = true
# preconditioner types: PCILU for serial, PCASM for serial/parallel
Simulator.SubSystem.MeshAlgoLSS.Data.PCType = PCASM
Simulator.SubSystem.MeshAlgoLSS.Data.KSPType = KSPGMRES
Simulator.SubSystem.MeshAlgoLSS.Data.MatOrderingType = MATORDERING_RCM
Simulator.SubSystem.MeshAlgoLSS.Data.MaxIter = 1000
Simulator.SubSystem.MeshAlgoLSS.Data.SaveSystemToFile = false
Simulator.SubSystem.MeshAlgoLSS.MaskEquationIDs = 0 1
# Krylov method is chosen
Simulator.SubSystem.MeshAlgoLSS.Data.NbKrylovSpaces = 50
```

WATCH OUT: *The option* `MaskEquationIDs` *is set to 0 1 for a 2D test case and 0 1 2 for a 3D test case. Not using correctly one of the aforementioned settings will result in a system crush*

9.2 Wall Distance computations

The wall distance computation is used to evaluate the nodal distance from a user-defined boundary and stored in a data array (a.k.a data socket inside COOLFluiD)
The output of the wall distance computations is written and plottable with **Tecplot**

```
# setting wall distance socket
Simulator.SubSystem.Tecplot.Data.DataHandleOutput.CCSocketNames = wallDistance
Simulator.SubSystem.Tecplot.Data.DataHandleOutput.CCVariableNames = wdistance
Simulator.SubSystem.Tecplot.Data.DataHandleOutput.CCBlockSize = 1
Simulator.SubSystem.Tecplot.WriteSol = ParWriteSolutionBlock
```

The following configuration options ensure the wall distance computations.

```
Simulator.SubSystem.DataPreProcessing = DataProcessing
Simulator.SubSystem.DataPreProcessingNames = DataProcessing1
Simulator.SubSystem.DataProcessing1.RunAtSetup = true
```

The computation is not done at the first iteration

```
Simulator.SubSystem.DataProcessing1.SkipFirstIteration = true
```

To reduce the computational time and memory cost, the computation is done after a `ProcessRate` iteration(s)

```
Simulator.SubSystem.DataProcessing1.ProcessRate = 50
```

The option `Comds` specifies the name of the class responsible of the wall distance computations.

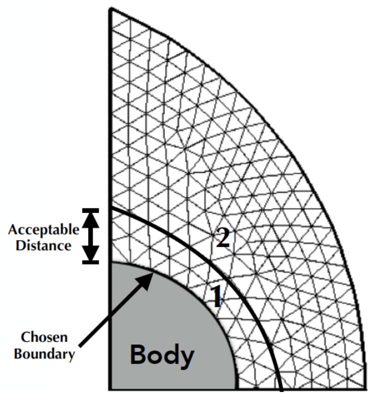
```
Simulator.SubSystem.DataProcessing1.Comds = ComputeWallDistanceVector2CCMPI
```

The user can specify the boundary from which the computations are done in the option `BoundaryTRS`.

```
Simulator.SubSystem.DataProcessing1.Names = WallDistance
Simulator.SubSystem.DataProcessing1.WallDistance.BoundaryTRS = SlipWall
Simulator.SubSystem.DataProcessing1.WallDistance.CentroidBased = true
```

A particular user-defined distance, denoted Acceptable Distance, from the specified boundary in the `BoundaryTRS` results in defining two separate region that incorporate improved stiffness concepts or specific nodal displacement behaviors. As a result, the regions 1 & 2 will be subject to two different spring analogies

```
# Setting the Acceptable distance
Simulator.SubSystem.DataProcessing1.WallDistance.AcceptableDistance= 1.
```



WATCH OUT: *If the option* `AcceptableDistance` *is set on 0, this means that only one spring analogy will be applied to all the mesh*

9.3 Mesh Fitting Parameters

The mesh fitting techniques are defined as a post-process step.

```
Simulator.SubSystem.DataPostProcessing           = DataProcessing
Simulator.SubSystem.DataPostProcessingNames      = MeFiAlgo
Simulator.SubSystem.MeFiAlgo.Comds               = MeshFittingAlgorithm
Simulator.SubSystem.MeFiAlgo.Data.CollaboratorNames = MeshAlgoLSS
```

The algorithm has a several characterizing parameters:

The Mesh adaptation is activated at a specific iteration.

```
Simulator.SubSystem.MeFiAlgo.StartIter           = 0
Simulator.SubSystem.MeFiAlgo.SkipFirstIteration  = true
```

The algorithm can be stopped by prescribing a maximum number of steps.

```
Simulator.SubSystem.MeFiAlgo.StopIter            = 5000
```

The option `ProcessRate` will control the frequency of the use of the mesh refinement within the simulation.

```
Simulator.SubSystem.MeFiAlgo.ProcessRate         = 20
```

The option `updateVar` for the mesh fitting process needs to be consistent with the option `updateVar` of the flow field computations (e.g. Rhoivt, cons)

```
Simulator.SubSystem.MeFiAlgo.Names              = MeshFitting
Simulator.SubSystem.MeFiAlgo.Data.updateVar     = Cons
```

The linear springs are truncated and bounded between a minimum and maximum value based on a P^2 method, denoted respectively, minimum percentile and maximum percentile

```
Simulator.SubSystem.MeFiAlgo.MeshFitting.minPercentile     = 0.30
Simulator.SubSystem.MeFiAlgo.MeshFitting.maxPercentile     = 0.55
```

An under-relaxation factor, having a similar behavior as a mesh velocity, is added to the mesh adaptation solver to smooth the nodal displacements and avoid mesh node overlaps.

```
Simulator.SubSystem.MeFiAlgo.MeshFitting.meshAcceleration = 0.05
```

WATCH OUT: *The* `meshAcceleration` *option affects negatively the convergence rate. A trade-off between convergence issues in the flow field solver and convergence rate gives an order of magnitude of the under-relaxation factor of* $\mathcal{O}(10^{-2})$

The physics-driven adaptation, based on a physical flow field variable, is defined in the following options (e.g. density, pressure or temperature etc...)

```
Simulator.SubSystem.MeFiAlgo.MeshFitting.monitorVarID      = 4
Simulator.SubSystem.MeFiAlgo.MeshFitting.monitorPhysVarID = 0
```

WATCH OUT: *If the* `monitorPhysVarID` *is specified, it cancels the effect of the* `monitorVarID`

The equilibrium spring length concerns the inner nodes and the multiplication factor `ratioBoundaryToInnerEquilibriumSpringLength` tends to stiffen the boundary mesh nodes.

```
Simulator.SubSystem.MeFiAlgo.MeshFitting.equilibriumSpringLength = 2e-4
Simulator.SubSystem.MeFiAlgo.MeshFitting.
                        ratioBoundaryToInnerEquilibriumSpringLength  = 0.01
```

The nodes on specific boundaries can be unlocked and therefore moving along a boundary line for 2D or a boundary surface for 3D.

```
Simulator.SubSystem.MeFiAlgo.MeshFitting.unlockedBoundaryTRSs = SuperOutlet \
                                                                SlipWall
```

The last step will be to update the mesh

```
Simulator.SubSystem.CellCenterFVM.AfterMeshUpdateCom = StdMeshFittingUpdate
```

9.4 Mesh Quality Indicator (MQI)

A mesh quality indicator is a tool to qualitatively grade an adapted mesh. One need to assign a specific value to the Mesh Quality Indicator depending on the mesh element type.
NOTE: *The Mesh Quality Indicator is mesh type dependant*

```
Simulator.SubSystem.MeFiAlgo.MeshFitting.MQIvalue = 0
```

Value	MQI
0	deactivated option
2	2D Triangular meshes
3	Aspect Ratio 2D Quads
4	Skewness 2D Quads
5	3D tetrahedral meshs

9.5 Refinement Stop Indicator (RSI)

A Refinement Stop Indicator is a quantitative function aiming to stopping the refinement process at the best moment autonomously. It is based on user-defined tolerance on the mesh movement, expressed in percentage.
NOTE: *The Refinement Stop Indicator is deactivated if the option* `tolerance = 0`.

```
Simulator.SubSystem.MeFiAlgo.MeshFitting.tolerance = 0.01
```

9.6 Post-Processing - Outputs

This section presents the different plottable outputs with **Tecplot** based on the predefined socket filled during the refinement process.
The option `SocketNames` can be defined as following:

- `stiffness` : For visualizing the mesh stiffness.
- `iradius` : MQI for 2D triangular meshes
- `skewness` : MQI for 2D quadrilateral meshes based on the skewness criteria
- `AR` : MQI for 2D quadrilateral meshes based on the aspect ratio criteria
- `isphere` : MQI for 3D tetrahedral meshes
- `relativeError` : For visualizing the value of the relative movement of the mesh nodes

```
Simulator.SubSystem.Tecplot.Data.DataHandleOutput.SocketNames   = stiffness
Simulator.SubSystem.Tecplot.Data.DataHandleOutput.VariableNames = kstiff
Simulator.SubSystem.Tecplot.Data.DataHandleOutput.isNodal = true
```

References

[1] J. Nieminen, J. Yliluoma. *Function Parser for C++*, http://warp.povusers.org/ Function-Parser/, 2009.

[2] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, *The COOLFluiD Framework - Design Solutions for High-Performance Object Oriented Scientific Computing Software*, International Conference Computational Science 2005, Atlanta (GA), LNCS 3514, Vol.1, pp. 281-286, Springer-Verlag, 2005.

[3] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, S. Vandewalle and S. Poedts, *Reusable Object-Oriented Solutions for Numerical Simulation of PDEs in a High Performance Environment*, Scientific Programming. ISSN 1058-9244, Vol. 14, N. 2, pp. 111-139, IOS Press, 2006.

[4] A. Lani, *An Object Oriented and High Performance Platform for Aerothermodynamics Simulation*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2008.

[5] T. E. Magin, *A model for Inductive Plasma Wind Tunnels*, Ph.D. thesis, von Karman Institute, Rhode-Saint-Genèse, Belgium, 2004.

[6] A. Munafo, M. Panesi, R. Jaffe, A. Lani, T. Magin, *Vibrational State to State Kinetics in Expanding and Compressing Nitrogen Flows*, AIAA-2010-4335, 10th AIAA/ASME Joint Thermophysics and Heat Transfer Conference, Chicago, Illinois, June 28-July 1, 2010.

[7] M. Panesi, *Physical Models for Nonequilibrium Plasma Flow Simulations at High Speed Re-entry Conditions*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2009.

[8] M. Panesi, A. Lani and O. Chazot, *Reduced Kinetic Mechanism for CFD Applications*, AIAA-2009-3920, 41st AIAA Thermophysics Conference, San Antonio, Texas, June 22-25, 2009.

[9] T. L. Quintino. *A Component Environment for High-Performance Scientific Computing. Design and Implementation*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2008.

[10] Thomas Wuilbaut, *Algorithmic Developments for a Multiphysics Framework*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2008.

[11] A. Dedner, F. Kemm, D. Kröner, C. D. Munz, T. Schnitzer and M. Wesenberg, *Hyperbolic Divergence Cleaning for the MHD Equations*, Journal of Computational Physics, Vol. 175, pp. 645-673, 2002, doi:10.1006/jcph.2001.6961.

[12] K. G. Powell, P. L. Roe, R. S. Myong, T. I. Gombosi and D. L. De Zeeuw, *An Upwind Scheme for Magnetohydrodynamics*, 12^{th} AIAA Computational Fluid Dynamics Conference, p. 661-674, Am. Inst. of Aeron. and Astron., San Diego, Calif., 1995.

[13] K. G. Powell, P. L. Roe, T. J. Linde, T. I. Gombosi and D. L. De Zeeuw, *A Solution-Adaptive Upwind Scheme for Ideal Magnetohydrodynamics*, Journal of Computational Physics, Vol. 154, pp. 284-309, 1999, doi:10.1006/jcph.1999.6299.

[14] T. Tanaka, *Finite Volume TVD Scheme on an Unstructured Grid System for Three-Dimensional MHD Simulation of Inhomogeneous Systems Including Strong Background Potential Fields*, Journal of Computational Physics, Vol. 111, pp. 381-389, 1994, doi:10.1006/jcph.1994.1071.

[15] M. S. Yalim, D. Vanden Abeele and A. Lani, *Simulation of Field-aligned Ideal MHD Flows around Perfectly Conducting Cylinders Using an Artificial Compressibility Approach*, Proceedings of the 11^{th} International Conference on Hyperbolic Problems held in Ecole Normale Supérieure, Lyon, France, July 17-21, 2006, pp. 1085-1092, Springer-Verlag, 2008.

[16] M. S. Yalim, *An Artificial Compressibility Analogy Approach for Compressible Ideal MHD: Application to Space Weather Simulation*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2008.

[17] M. S. Yalim, D. Vanden Abeele, A. Lani, T. Quintino and H. Deconinck, *A Finite Volume Implicit Time Integration Method for Solving the Equations of Ideal Magnetohydrodynamics for the Hyperbolic Divergence Cleaning Approach*, Journal of Computational Physics, Vol. 230, N. 15, pp. 6136-6154, 2011, doi:10.1016/j.jcp.2011.04.020.

[18] M. S. Yalim, J. Majewski, H. Deconinck and S. Poedts, *Adaptive Unstructured Grid Simulation of Interaction of Solar Wind with Planetary Magnetosphere Using an Implicit Finite Volume Method*, Journal of Geophysical Research Space Physics, submitted, 2011.