User-manual for COOLFluiD Flux Reconstruction Solver (version 2017.01)

Ray Vandenhoeck, ray.vandenhoeck@vki.ac.be
Von Karman Institute, Aeronautics & Aerospace Dept.

Introduction

The COOLFluiD platform [1, 2, 3, 4, 5] contains a high-order Flux Reconstruction (FR) solver capable of solving the Euler and Navier-Stokes equations in 2D and 3D. This manual gives an overview of the options particular to the FR solver and the general configuration of an FR test case.

1 Test Case Configuration

A Flux Reconstruction test case is constructed in the same way as test cases for other methods. The following libraries must be added to the `CFcase` file in order to use the FR method for the Euler or Navier-Stokes equations, in addition to any other required libraries:

```
# Simulator.Modules.Libs =  libFluxReconstructionMethod
                            libFluxReconstructionNavierStokes
```

In order to use the FR method to solve a test case, the following line must be added:

```
# Simulator.SubSystem.SpaceMethod = FluxReconstruction
```

The FR method can either be used with an explicit time marching scheme such as Forward Euler, or with an implicit time marching scheme, such as Backward Euler. For an explicit scheme solving the Euler equations, the following must be added:

```
# Simulator.SubSystem.FluxReconstruction.SpaceRHSJacobCom = RHS
```

For an implicit scheme solving the Euler equations, the following is used:

```
# Simulator.SubSystem.FluxReconstruction.SpaceRHSJacobCom = RHSJacob
# Simulator.SubSystem.FluxReconstruction.ConvSolveCom = ConvRHSJacob
# Simulator.SubSystem.FluxReconstruction.TimeRHSJacobCom = StdTimeRHSJacob
```

When solving the Navier-Stokes equations with an explicit scheme, the following is used:

```
# Simulator.SubSystem.FluxReconstruction.SpaceRHSJacobCom = RHSNS
# Simulator.SubSystem.FluxReconstruction.ConvSolveCom = ConvRHSNS
# Simulator.SubSystem.FluxReconstruction.DiffSolveCom = DiffRHSNS
```

For an implicit scheme solving the Navier-Stokes equations, the following is added:

```
# Simulator.SubSystem.FluxReconstruction.SpaceRHSJacobCom = RHSJacobNS
# Simulator.SubSystem.FluxReconstruction.ConvSolveCom = ConvRHSJacobNS
# Simulator.SubSystem.FluxReconstruction.DiffSolveCom = DiffRHSJacobNS
# Simulator.SubSystem.FluxReconstruction.TimeRHSJacobCom = StdTimeRHSJacob
```

In addition to the lines provided above, all other options associated with the time marching scheme, not particular to the FR method, should be added.

1.1 Upgrading mesh order

In order to read an input mesh, the standard mesh builder can be used:

```
# Simulator.SubSystem.FluxReconstruction.Builder = StdBuilder
```

This is also the default, so the above line can be omitted. When using the default mesh builder, the order of the FR method will be the order specified in the input mesh file. However, it is also possible to upgrade the order by using the `MeshUpgradeBuilder` in the following way:

```
# Simulator.SubSysMesh.FluxReconstruction.Builder = MeshUpgrade
# Simulator.SubSysMesh.FluxReconstruction.MeshUpgrade.PolynomialOrder = Px
# Simulator.SubSysMesh.FluxReconstruction.MeshUpgrade.GeoPolynomialOrder = Py
# Simulator.SubSysMesh.FluxReconstruction.MeshUpgrade.DivideElements = z
```

The option `PolynomialOrder` must be specified. The integer x determines the order of the FR method. Currently, this can be an integer ranging from 1 to 5.

The option `GeoPolynomialOrder` upgrades the geometric order of the elements of the mesh to order y. If this option is omitted, the order specified in the mesh input file is used. Currently an order of 1 or 2 is supported, meaning in essence that a mesh with geometric order P1 can be upgraded to P2.

The option `DivideElements` is used to increase the amount of elements in a mesh, which is useful for performing convergence studies. If this option is omitted, or z=1, the amount of elements in the mesh remains unchanged. The integer z determines how many new elements are adjacent to one face (2D) or edge (3D) of an element of the original mesh. For example, when set to 2, an original quadrilateral element will be equally divided in 4 new elements and an original hexahedra will be equally divided in 8 new elements.

1.2 Running in Parallel

The FR method can be used in parallel without modification to the test case file when using the standard builder. However, when using `MeshUpgradeBuilder`, the method must be run serially. In order to define a single test case file that first upgrades the order of a mesh and then runs the test case in parallel, a second subsystem must be added:

```
# Simulator.SubSystems = SubSysMesh SubSystem
# Simulator.SubSystemTypes = OnlyMeshSubSystem StandardSubSystem
# Simulator.SubSysMesh.Namespaces = MeshNamespace
# Simulator.SubSysMesh.Ranks = 0:0
# Simulator.SubSysMesh.MeshNamespace.MeshData = MeshMeshData
# Simulator.SubSysMesh.MeshNamespace.SubSystemStatus = MeshSubSystemStatus
```

Here `SubSysMesh` is the new subsystem used to upgrade the order of the mesh and `SubSystem` is the subsystem used to solve the test case in parallel, to which no particular modifications need to be made. The test case can then be executed in parallel.

All options relevant to reading an input mesh, upgrading its order and writing the output mesh must be added to `SubSysMesh`. Because no simulation actually needs to be run in this subsystem, the `SpaceMethod` can be set to `Null`:

```
    # Simulator.SubSysMesh.SpaceMethod = Null
```

1.3 Options of the FR method

The FR method has four characterizing parameters. The first is the approximate Riemann solver used for the convective interface flux. This can be chosen as follows:

```
    # Simulator.SubSystem.FluxReconstruction.Data.RiemannFlux = ...
```

The current options that can be chosen are: `CentredFlux`, `RoeFlux`, `LaxFriedrichsFlux`, `AUSMPlusFlux2D`, `AUSMPlusFlux3D`, `AUSMPlusUpFlux2D` and `AUSMPlusUpFlux3D`.

The second parameter, is the correction function. Currently, the VCJH scheme is implemented. The characterizing c parameter of this scheme can be chosen as follows:

```
    # Simulator.SubSystem.FluxReconstruction.Data.CorrectionFunctionComputer = VCJH
    # Simulator.SubSystem.FluxReconstruction.Data.VCJH.CFactor =  c
```

Here, c is a real number.

The third and fourth parameters are the distributions of solution and flux points. These can be chosen as follows:

```
    # Simulator.SubSystem.FluxReconstruction.Data.SolutionPointDistribution = ...
    # Simulator.SubSystem.FluxReconstruction.Data.FluxPointDistribution = ...
```

The current options are `GaussLegendre`, `Lobatto` and `Equidistant`, being respectively the Gauss-Legendre, Gauss-Lobatto points and an equidistant distribution. For tensor elements (Quadrilaterals, Hexahedra), it is strongly advised to use the same distributions for solution and flux points, since this greatly reduces the necessary computational time.

1.4 Boundary conditions and initial condition

In order to initialize the solution, the following default line can be used:

```
    # Simulator.SubSystem.FluxReconstruction.InitComds = StdInitState
```

Additionally, all other commands used to initialize the solution must be added. Alternatively, this option can be set to `Null` in order to use the initial condition specified in the input mesh file.

The boundary conditions are specified in the usual way:

```
# Simulator.SubSystem.FluxReconstruction.BcNames = Name1 Name2
# Simulator.SubSystem.FluxReconstruction.Name1.applyTRS = TRS1
# Simulator.SubSystem.FluxReconstruction.Name2.applyTRS = TRS2
# Simulator.SubSystem.FluxReconstruction.Data.BcTypes = Type1 Type2
# Simulator.SubSystem.FluxReconstruction.Data.BcNames = Name1 Name2
```

When solving the Navier-Stokes equations, currently the following lines must be added as well:

```
# Simulator.SubSystem.FluxReconstruction.BcNamesDiff = Name1Diff Name2Diff
# Simulator.SubSystem.FluxReconstruction.Name1Diff.applyTRS = TRS1
# Simulator.SubSystem.FluxReconstruction.Name2Diff.applyTRS = TRS2
```

The currently implemented boundary conditions are `Dirichlet`, `MirrorEuler2D`, `MirrorEuler3D`, `NoSlipWallHeatFluxNS2D`, `NoSlipWallHeatFluxNS3D`, `SuperOutlet`, `SubInletEulerTtPtAlpha2D`, `SubInletEulerTtPtAlpha3D`, `SubOutletEuler2D`, `SubOutletEuler3D`. `SuperOutlet` being a Neumann boundary condition.

References

[1] 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.

[2] 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.

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

[4] 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.

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