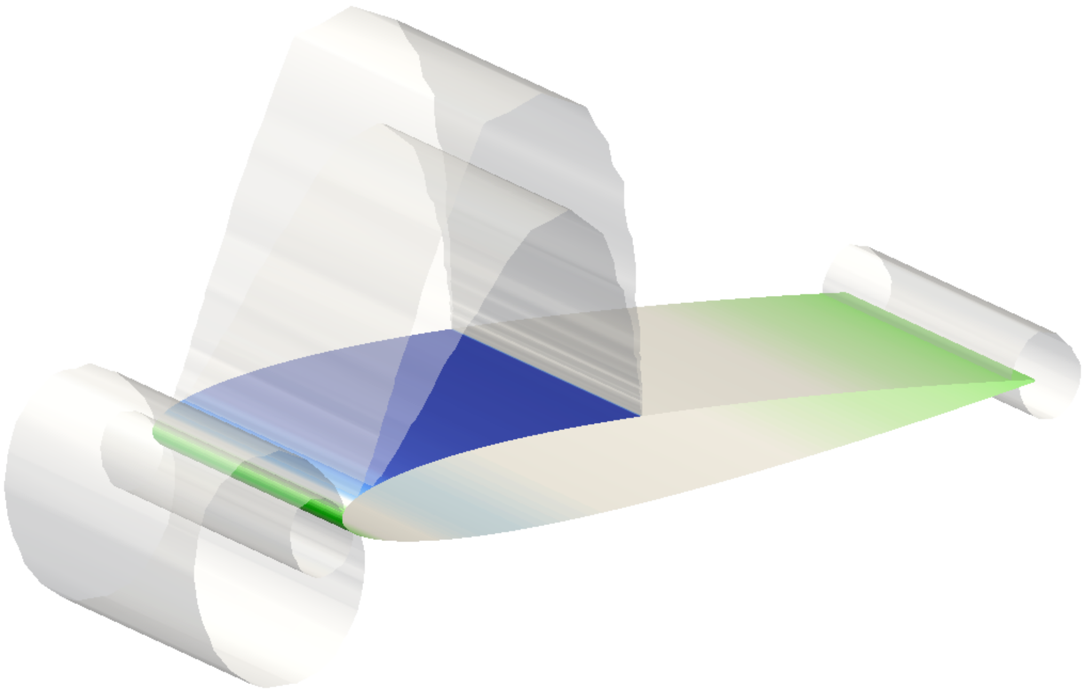
hisa

High Speed Aerodynamic Solver

User Guide



our future through science

Meiring Naude Road, Brummeria, Pretoria, South Africa
www.csir.co.za

First printing: August 2015
This edition: July 2021

Contents

Introduction

This document serves as a guide for new users of the High Speed Aerodynamic (HiSA) solver, providing them with the best practices for using the software. The solver was developed to provide users with an improved aerodynamic solver which is not only efficient and robust, but to extend the range of physics that can be modelled. To model transonic and supersonic flows the unsteady, compressible Navier-Stokes equations are solved using a density-based approach. Density-based solvers are typically preferred for high-speed flows as they allow for the effective handling of sharp numerical discontinuities in the primary variables, typically encountered with propagating shock waves. These solvers are, however, not well suited to problems where the flow tends toward the incompressible limit were the coupling between pressure and density becomes weak. For these cases it is suggested that pressure-based solvers be employed. Some of the current capabilities of the HiSA solver include:

- Assembly of a coupled system for the construction of an implicit matrix system;
- GMRES solver with LU-SGS preconditioning for the efficient solution of the coupled system;
- A selective range of shock capturing schemes;
- ALE formulation for moving meshes;
- Improved coupling algorithm for FSI problems;
- Characteristic boundary conditions for a non-reflective farfield approximation; and
- Effective handling of multiple regions.

To improve usability, the solver automatically detects whether dynamic meshing should be activated based on the user input. The solver employs the standard dynamic mesh motion class of OpenFOAM® , thereby allowing for the use of any of the existing dynamic mesh movement libraries currently available.

Where possible, the guide aims to provide best practices and guidelines for the use of the solver. It should, however, be noted that the solver and libraries are relatively new and still under active development. For this reason it can not be guaranteed that a solution will be obtained in all cases. Please note that this guide considers only the settings required by the solver and does not detail mesh generation and post-processing tools.

1.1 Formatting Conventions

A set of simple formatting rules is used in this document to facilitate reading:

`sample`	Keywords, commands, utilities and solvers.
`sample`	Contents of files.
sample	File names and folders, including HiSA related files.
<sample>	User-required input.

Comments in all HiSA files follow the usual C++ standard. Therefore, any lines prefixed with // or text placed between /* and */ are ignored by the code. Lastly, all relevant environmental variables are preceded by the $ sign.

1.2 Components

A summary of the solvers and libraries provided is given below.

1.2.1 Solvers

The following solver executables are provided:

`hisa`	High speed aero solver for modelling internal and external compressible flow in a single region.
`multiRegionSolver`	Multi-region harness for running different solvers in different meshes, e.g. for conjugate heat transfer (CHT) analysis

1.2.2 Libraries

As part of the coupled aero solver the user is provided with a number of new or extended libraries.

finiteVolume	Contains additional boundary conditions and numerical schemes used by the solver.
modules	Solver libraries used by the single- and multi-region solvers – currently includes *hisa* and *thermalSolid*; the latter for computing heat conduction in solids.
functionObjects	Function objects that can be run alongside the solver, e.g. *shockRefinement* for detecting cells in the vicinity of a shock to be refined with adaptive mesh refinement.
modal	Function object and boundary condition for computing modal solid motion.
getPatchPoints	Utility to extract point co-ordinates from a patch – intended for use in calculation of mode shapes.

1.2.3 Tutorials

To assist the user in getting started with HiSA, a number of example test cases are provided.

Euler

shockTube	Sod's shock tube problem is used to evaluate the ability of **shock capturing schemes** to track the transient evolution of numerical discontinuities in shock waves.
nacaAirfoil	This test case is based on the well-known NACA 0012 2D airfoil case also given in the standard release and is included to serve as a basic guide for setting up a **steady-state** case.
forwardStep	Mach 3 flow over a forward-facing step. Details the settings for a **transient analysis**.

RANS

supersonicFlatPlate	This test case demonstrates the settings for RANS analysis and assess the turbulent implementation by comparing the boundary layer formation for Mach 4.5 flow over a flat plate.
rae2822	Viscous analysis of a transonic airfoil in 2D with **Spalart-Allmaras** turbulence modelling and **low-y+** mesh.
oneraM6	Viscous analysis of a transonic wing in 3D with **Spalart-Allmaras** and k-ω **SST** turbulence modelling and **wall functions**.
agardModelB	AGARD B calibration model - k-ω **SST** turbulence modelling with **low-y+** mesh.
basicFinner, *modifiedFinner*	Army-Navy finned projectile validation cases - k-ω **SST** turbulence modelling with **low-y+** mesh.

Advanced

agard445	AGARD 445.6 aeroelastic analysis. This case uses the **ALE** formulation of the shock capturing scheme as well as **adaptive mesh refinement** and **modal solid modelling**.
aerodynamicHeating	Conjugate heat transfer case demonstrating the **multi-region framework**.

Solver setup

This section details the solver setup for HiSA. Please note that this document will focus on solver settings specific to the HiSA solver only. The reader is referred to the OpenFOAM® User Guide for a general introduction. It is suggested that projects are initialised by copying an existing tutorial and modifying the settings by following the recommendations in this guide.

In this guide, the Spalart-Allmaras and k-ω SST turbulence models will be referred to for concreteness; however, the solver places no restriction on the turbulence model used.

2.1 Initial and boundary conditions

The following variables or fields are required by the solver and need to be specified in the *0* directory:

Field	Description	Analysis required
U	Velocity	All cases
p	Pressure	All cases
T	Temperature	All cases
nut	Turbulent kinematic viscosity	Compressible RANS
alphat	Turbulent thermal diffusivity	Compressible RANS
nuTilda	Spalart-Allmaras turbulent diffusivity	Spalart-Allmaras
k	Turbulent kinetic energy	Menter's k-ω SST model
omega	Turbulent specific dissipation rate	Menter's k-ω SST model

Below, an example initial velocity field, *U*, is shown which is located in the *0* directory. First, the keyword `dimensions` defines the dimensional units of the variable. A detailed description of the dimensional units are given in the OpenFOAM® User Guide. Next, the `internalField` or initial condition of the field should be specified. As this will act as the starting point of the analysis, careful attention should be paid in selecting the initial conditions as it may effect the convergence of the solution. Generally, it is suggested that internal flow field is initialised using the freestream values.

```
dimensions     [0 1 -1 0 0 0 0];

internalField  uniform (200 0 0);

boundaryField
{
    "(airfoil)"
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            characteristicFarfieldVelocity;
        U               (200 0 0);
        p               60000;
        T               300;
        value           $internalField;
    }
    defaultFaces
    {
        type            empty;
    }
}
```

Most of the fields given above accept the standard boundary conditions, and a list of some of the recommended boundary conditions is given below.

Field	Boundary condition	Remarks
U	`boundaryCorrectedFixedValue` / `slip`	For stationary walls
	`movingWallVelocity / movingWallSlip`	For moving walls
	`characteristicFarfieldVelocity`	For reflectionless free-stream conditions
	`characteristicVelocityInletOutputVelocity`	For inlets and outlets with stipulated velocity
	`characteristicPressureInletOutputVelocity`	For inlets and outlets with stipulated pressure
p	`characteristicWallPressure`	For walls
	`characteristicFarfieldPressure`	For reflectionless free-stream conditions
	`characteristicVelocityInletOutputPressure`	For inlets and outlets with stipulated velocity
	`characteristicPressureInletOutputPressure`	For inlets and outlets with stipulated pressure
T	`boundaryCorrectedFixedValue`	Fixed temperature boundary
	`characteisticWallTemperature`	For adiabatic walls

	`characteristicFarfieldTemperature`	For reflectionless free-stream conditions
	`characteristicVelocityInletOutputTemperature`	For inlets and outlets with stipulated velocity
	`characteristicPressureInletOutputTemperature`	For inlets and outlets with stipulated pressure
nuTilda	`boundaryCorrectedFixedValue` (0)	For walls
	`inletOutlet`	For far-field and inlet/outlet
k	`kLowReWallFunction`	For low-y+ walls
	`kqRWallFunction`	For high-y+ walls
	`turbulentIntensityKineticEnergyInlet`	For far-field and inlet/outlet
omega	`omegaWallFunction`	For low- and high-y+ walls
	`turbulentMixingLengthFrequencyInlet`	For far-field and inlet/outlet
nut	`boundaryCorrectedNutLowReWallFunction`	For low-y+ walls
	`nutUSpaldingWallFunction`	For high-y+ walls
alphat	`boundaryCorrectedFixedValue` (0)	For low-y+ walls
	`compressible::alphatWallFunction`	For high-y+ walls

Characteristic-based boundary conditions

With the aim of finding a balance between accurately describing the farfield conditions while maintaining a computationally efficient solution, characteristic boundary conditions aim to prevent outgoing disturbances from propagating back into the domain. This is achieved by performing a linearised Riemann analysis at the boundaries to obtain the so-called characteristic variables for the advection system of equations [1,3].

Each characteristic boundary condition requires the freestream conditions of all the primary variables. It is therefore suggested that a file *0/include/freestreamConditions* should be created which contains the freestream velocity, pressure and temperature

```
U           (242.284 8.46075 0);
p           85418.9;
T           260;
```

This field can then be included at the beginning of each of the primary variable field files as follows

```
#include        "include/freestreamConditions"
```

The characteristic inlet/outlet boundaries also require the U, p and T settings as above, but these are reference/external values rather than free-stream conditions. Which, if any, of these values is actually imposed on the flow depends on the flow direction and Mach number locally at each patch face.

The characteristic-based boundary conditions for temperature and pressure at walls give a more accurate representation than the traditional `zeroGradient` condition, and are more robust in the presence of high Mach number flows. No free-stream conditions are read.

2.2 Material and domain properties

The solver requires the standard constant dictionaries typically specified. Standard dictionaries as well as optional dictionaries read by the solvers include:

Dictionary	Description
dynamicMeshDict	Describes dynamic movement (if not present the solver automatically assumes `staticFvMesh`)
thermophysicalProperties	Material properties of the fluid
turbulenceProperties	Turbulence model employed

For all analyses the material properties of the fluid under consideration should be provided in *constant/thermophysicalProperties*. An example of the material properties for air is shown below

```
thermoType
{
    type            hePsiThermo;
    mixture         pureMixture;
    transport       sutherland;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        nMoles          1;
        molWeight       28.966;
    }
    thermodynamics
    {
        Cp              1005;
        Hf              0;
    }
    transport
    {
        As              1.4584e-6;
        Ts              110.33;
    }
}
```

where for an ideal gas, such as air, `sutherland` should be specified under `transport` to define the viscosity as a function of temperature. In this case the entries `mu` and `Pr` are neglected and the coefficients `As` and `Ts` need to be defined. The entries are as defined in the OpenFOAM® User Guide.

2.2.1 Turbulence models

In *constant/turbulenceProperties* the user can select any of the standard RANS or LES turbulence models. For inviscid flow the user can set `transport` in *constant/thermophysicalProperties* to `const` while setting viscosity, `mu` to 0 and in *constant/turbulenceProperties* select `laminar`.

```
simulationType laminar;
```

The RANS turbulence models `kOmegaSST` and `SpalartAllmaras` are suited to external aerodynamic applications and have been tested with HiSA, but there is no limit in principle to the models which can be used.

2.2.2 Mesh movement

The solver incorporates the dynamic mesh class and therefore allows for the use of any of the general dynamic mesh options used with the standard OpenFOAM® solvers. For example, the `sixDoFRigidBodyMotion` mesh motion solver allows for the simulation of rigid body dynamics. If no *dynamicMeshDict* dictionary is provided the solver automatically assumes `staticFvMesh`.

2.3 Numerical and solver settings

The solver requires the standard numerical and solver dictionaries typically specified for OpenFOAM® analyses; however, some additional settings required by the HiSA solver are now described.

2.3.1 controlDict

The solver allows for both steady and transient solutions and differentiates between the two based on the options specified by the user. Below an example is shown of the *system/controlDict* for a steady state analysis:

```
application     hisa;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         500;
deltaT          1;
writeControl    timeStep;
writeInterval   1;
```

It is noted that for the steady state case the temporal terms do not correspond to physical values but rather to iteration numbers, whereas for a transient analysis the temporal control settings correspond to real time values. The following is an example of a transient setup:

```
application     hisa;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         4.0e-03;
deltaT          0.5e-05;
adjustTimeStep  yes;
maxCo           200;
maxDeltaT       1e-3;
writeControl    adjustableRunTime;
writeInterval   1e-04;
```

In *system/controlDict* the maximum Courant number in the domain, `maxCo`, may be set together with the boolean `adjustTimeStep`. For transient analyses a value of between 5-100 is typically appropriate depending on the required temporal accuracy, while for steady-state analyses there is no restriction and `adjustTimeStep` should be set to `off`. Transient analysis can also be run in fixed-timestep mode with `adjustTimeStep` set to `off`.

For the `multiRegionSolver` application, an extra section in controlDict defines the regions and solver module to be used for each, e.g.:

```
regions
{
    fluid
    {
        solver hisa;
        libs ( "libhisaModule.so" );
    }
    solid
    {
        solver thermalSolid;
        libs ( "libthermalSolidModule.so" );
    }
}
```

The subdictionaries listed in `regions` define the names of mesh regions in which the listed solver modules are run. These mesh regions are subdirectories of the *0*, *constant* and *system* folders in which the relevant setup files for that region are placed, instead of directly in the folders as for a standard single-region case.

2.3.2 fvSchemes

A number of shock-capturing schemes are available to the user and can be selected under `fluxScheme`

Option	Comments
`AUSMPlusUp`	The latest form of the flux splitting scheme proposed by Liou [2].
`HLLC`	An extension to the HLL scheme as proposed by Toro et al. [5].

Note that `AUSMPlusUp` is currently the only scheme for which the ALE formulation is implemented and therefore the only scheme which can be used for moving meshes. It is also the scheme that has been the most extensively tested to date and is recommended for use with the HiSA solver.

The temporal discretistion is specified under `ddtSchemes`. A dual-timestepping/pseudo-timestepping scheme is most easily employed by specifying

```
ddtSchemes
{
    default         dualTimestepping rPseudoDeltaT <real-time scheme>;
}
```

where `<real-time scheme>` is the time scheme used for the discretisation in real (physical) time. The following options are available:

Option	Comments
`steadyState`	Steady-state analysis where the contribution from the real time term is neglected. For steady-state analyses there is no restriction on the real time step size.
`Euler`	First-order backward-difference which provides the best stability properties.
`backward`	Second-order implicit backward-difference method using two previous time values for improved temporal accuracy.

`Crank-Nicolson` ϕ	Also a second-order implicit method which only uses one old-time value and is therefore better suited to adaptive mesh refinement; however, stability is less assured. To mitigate this the user also needs to specify a weighting factor, ϕ. For a value of 1, second-order Crank-Nicolson is recovered, while the solution reverts to first-order Euler as the weight factor reduces to 0. It is suggested that a value of 0.9 be used to achieve the best balance between stability and accuracy.

The following settings represent the suggested defaults for transonic and supersonic flow on unstructured meshes, for the steady-state case:

```
ddtSchemes
{
    default         bounded dualTime rPseudoDeltaT steadyState;
}

gradSchemes
{
    default         cellLimited faceLeastSquares linear 0.6;
    gradTVD         faceLeastSquares linear;
    grad(nuTilda)   cellLimited Gauss linear 0.9;
    grad(k)         cellLimited Gauss linear 0.9;
    grad(omega)     cellLimited Gauss linear 0.9;
}

divSchemes
{
    default         none;
    div(tauMC)      Gauss linear;
    div(phi,nuTilda) bounded Gauss limitedLinear 1;
    div(phi,k)      bounded Gauss limitedLinear 1;
    div(phi,omega)  bounded Gauss limitedLinear 1;
}

laplacianSchemes
{
    default         Gauss linear corrected;
    laplacian(muEff,U)        Gauss linear compact;
    laplacian(alphaEff,e)     Gauss linear compact;
}

interpolationSchemes
{
    default         linear;
    reconstruct(rho) wVanLeer gradTVD;
    reconstruct(U)  wVanLeerV gradTVD;
    reconstruct(T)  wVanLeer gradTVD;
}

snGradSchemes
{
    default         corrected;
}

wallDist
{
    method          Poisson;
}
```

The recommended settings for transient analyses differ only in the following sub-dictionaries:

```
ddtSchemes
{
    default         dualTime rPseudoDeltaT backward;
}

divSchemes
{
    default         none;
    div(tauMC)      Gauss linear;
    div(phi,nuTilda) Gauss limitedLinear 1;
    div(phi,k)      Gauss limitedLinear 1;
    div(phi,omega)  Gauss limitedLinear 1;
}
```

Notes:

- Settings are given here for Spalart-Allmaras (`nuTilda`) and k-ω turbulence models (`k` and `omega`).
- Use of the `bounded` scheme is to help maintain boundedness of the turbulence variables in the steady-state case.
- Use of the `Poisson` wall-distance method instead of the standard `meshWave` method has shown an improvement in accuracy on non-orthogonal grids.
- `compact` is a compact-stencil evaluation of the surface-normal derivative recommended for use with the coupled variables.
- `wVanLeer` is a version of the van Leer flux limiter which accounts for a face not equidistant from cell centres. It is recommended that the gradient used in this calculation is not limited as this may interfere with the role of the flux limiter itself.
- For other gradients `cellLimited` is used to help bound gradients on poor quality meshes.

2.3.3 fvSolution

The HiSA solver constructs a coupled matrix system of the compressible Navier-Stokes equations. The settings for this are contained in the `flowSolver` subdictionary. The `solver` keyword specifies the coupled solver to use. Currently the only option for this is `GMRES`, which selects the generalised minimal residual method of Saad and Schultz [4]. The settings required by the GMRES solver are contained in the `GMRES` sub-subdictionary of the `flowSolver` subdictionary and are described below:

Entry	Default	Description
`inviscidJacobian`	-	Calculation method for the inviscid flux Jacobian. The suggested value is `LaxFriedrichs`, which approximates the inviscid flux Jacobian based on the Lax-Friedrichs flux.
`viscousJacobian`	-	Calculation method for the viscous flux Jacobian. Supported methods are `laplacian` and `spectralRadius`, which are both based on Laplacian approximations of the viscous term.
`preconditioner`	-	Method used to precondition the coupled matrix. Currently supported is `LUSGS`, the Lower-upper symmetric Gauss-Seidel (LU-SGS) scheme developed by Yoon and Jameson [6].
`maxIter`	20	Maximum number of solver iterations if the solver tolerance is not met.
`nKrylov`	8	The number of Krylov vectors or orthogonal search directions. A higher number of search directions increases storage requirements as well as the computational overhead.
`solverTol`	1e-8	The normalised tolerance for checking solver convergence.
`solverTolRel`	1e-1	An optional convergence check which checks the reduction in the residual relative to the first residual.

With the iterative implicit method a dual-time stepping approach is employed and the following settings are contained in the pseudoTime dictionary to control the iterations in pseudo-time (also known as 'outer correctors'):

Entry	Default	Description
`nPseudoCorr`	20	The maximum number of pseudo iterations if the solver tolerance is not met (ignored in steady-state analysis).

`nPseudoCorrMin`	1	The minimum number of pseudo iterations to perform.
`pseudoTol`	-	The normalised tolerance for checking iterative pseudo convergence.
`pseudoTolRel`	-	An optional convergence check which checks the reduction in the residual relative to the first residual.
`moveMeshOuterCorrectors`	false	Specifies whether to move the mesh at each outer corrector, rather than only once at the beginning of each time step. The former may be required for stability if there is coupling between the flow and the mesh motion.
`pseudoCoNum`	1	The initial pseudotime Courant number value which determines the initial pseudo-time step size.
`pseudoCoNumMin`	0.1	Minimum pseudotime Courant number.
`pseudoCoNumMax`	25	Maximum pseudotime Courant number.
`pseudoCoNumMaxIncreaseFactor`	1.25	The pseudotime Courant number is modified by the switched evolution relaxation (SER) method, whereby it changes in inverse proportion to the residual. This setting limits the maximum increase allowable from one iteration to the next.
`pseudoCoNumMinDecreaseFactor`	0.1	This setting limits the allowable decrease of the pseudotime Courant number from one iteration to the next.
`localTimestepping`	true	If `true`, the time step size is set independently at each cell in order to achieve the current pseudo Courant number. If `false`, it is set globally to the mimimum of the locally calculated values. Generally, `true` is recommended for fastest convergence; however, `false` guarantees conservation regardless of convergence in pseudo-time.
`resetPseudo`	false	Reset the pseudotime Courant number to `pseudoCoNum` at the beginning of each real time step.
`localTimesteppingBounding`	true	If performing local timestepping, whether to check bounded variables ρ and e (internal energy) which are physically bounded below by zero. The pseudo-Courant number is reduced locally in cells where the lower bound is being approached too rapidly. This is useful to prevent unphysical values from being reached in intermediate iterations prior to convergence.

`localTimesteppingLowerBound`	0.95	The decrease in bounded variables which will trigger a halving of the local pseudo Courant number if `localTimesteppingBounding` and `localTimestepping` are enabled.

Under `solvers` only the turbulence equations need to be specified - for example:

```
solvers
{
    "(k|omega|nuTilda)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-10;
        relTol          0.1;
        minIter         1;
    }

    "(k|omega|nuTildaFinal)"
    {
      $k;
      reltol 0;
    }
}
```

It is recommended that some degree of relaxation of the turbulence equations be introduced when considering viscous analysis. A value of 0.5 is recommended:

```
  relaxationFactors
  {
      equations
      {
              "(k|omega|nuTilda)" 0.5;
      }
  }
```

When using the recommended `Poisson` method for wall distance calculation (as specified in `fvSchemes`), the `yPsi` solver must also be specified in the `solvers` subdictionary, with recommended settings as follows:

```
solvers
{
    yPsi
    {
        solver          GAMG;
        smoother        GaussSeidel;
        tolerance       1e-6;
        relTol          0;
    }
}
```

Notes:

- In transient simulations, it is important that sufficient pseudotime iterations are performed to adequately converge the solution at each real time step. It is difficult to give a rule of thumb as this depends on the ratio of the real-time and pseudotime Courant numbers, but setting a residual tolerance can be helpful.
- Reducing the `pseudoCoNum` can help to stabilise a simulation that diverges in the early iterations due to an unphysical initialisation.
- Increasing the value of `localTimesteppingLowerBound` closer to 1 can also aid stability on poor meshes.
- Another possible stability improvement is to use `upwind` for the divergence schemes of the turbulence variables to revert to first-order advection.

Bibliography

[1] J. Blazek. *Computational Fluid Dynamics: Principles and Applications: Principles and Applications*. Elsevier Science, 2001.

[2] M.-S. Liou. A sequel to AUSM, Part II: Ausm$^+$-up for all speeds. *Journal of Computational Physics*, 214(1):137–170, 2006.

[3] R. Löhner. *Applied computational fluid dynamics techniques: An introduction based on finite element methods*. Wiley, 2008.

[4] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[5] E. Toro, M. Spruce, and W. Speares. Restoration of the contact surface in the HLL-Riemann solver. *Shock waves*, 4(1):25–34, 1994.

[6] S. Yoon and A. Jameson. Lower-upper symmetric-Gauss-Seidel method for the Euler and Navier-Stokes equations. *AIAA journal*, 26(9):1025–1026, 1988.