Loci-Stream

Release 2.1.9

Streamline Numerics, Inc.

Apr 24, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

The Loci-Stream code is a product of the coupling of a CFD flow solver (Stream) to an open-source parallel-computing framework (Loci). Together the two components make up the Loci-Stream code. The Loci framework is a highly scalable framework that powers the Stream solver to run on large-scale machines with thousands of processors. Loci-Stream is designed to take advantage of existing commercially available front-end technology (standard grid generators) and back-end technology (post-processing software such as Tecplot, FieldView, EnSight etc.) which is relatively inexpensive compared to proprietary pre-processing and post-processing software provided by the major CFD vendors. The diagram below illustrates how typical users interact with Loci-Stream during a simulation workflow.

Loci-Stream is a pressure-based finite-volume solver that operates with generalized unstructured meshes. Some of the major simulation capabilities include the following:

- Incompressible flows
- Compressible flows with shock, rarefaction, and expansion waves
- Turbulent flows using RANS, DES, and LES turbulence closure methods
- Cavitating flows using REFPROP tabulated fluids
- Combusting flows using finite-rate chemistry and flamelet-based tabulation methods
- Volume-of-fluid (VOF) simulations with multiple liquid materials
- Lagrangian particle simulations
- Generalized moving mesh simulations with overset grids

There are two files that are required for running a simulation: a grid file (`.vog` extension) and a run control file (`.vars` extension). The grid and control files are keyed to the case name which is supplied to the execution command. The following sections provide some basic information about these data files.

The general philosophy of this user manual is to teach by example with a set of examples that are sequentially more complex. Each new example introducing another layer of functionality in *Stream.* As such the earlier portions of the manual will contain the most detail and discussion of the details of using *Stream*, with subsequent sections relying on the user having read through the previous sections and building on that knowledge base.

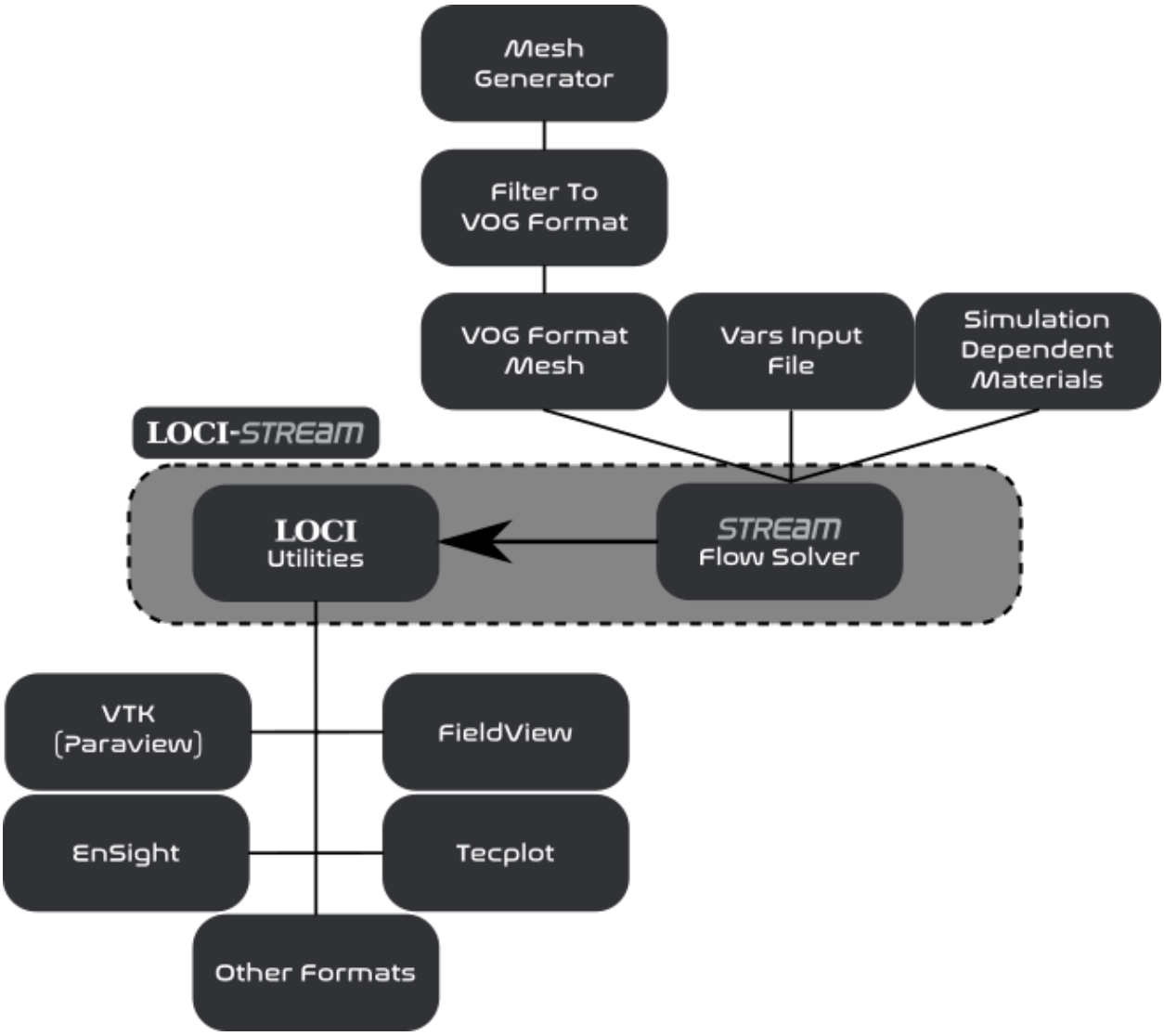

Fig. 1: The organization of the Loci-Stream code.

1.1 Run Control File

The run control file contains all the input variables required to specify a simulation. The *run control file* sample in this section illustrates a lid-driven cavity simulation, where flow in a 2-D cavity is driven by a sliding lid. All variables in the run control file must be located within a single pair of squiggly brackets {}. Other than this constraint, there are no constraints on the location or order of the variables in the file. Generally, however, it is useful to keep the variables arranged in some logical groupings as shown in the run control file example so that information can be easily located. Detailed explanation of the input variables in each of the groupings can be found in later sections of this guide. Note that in general, one can include comments throughout the run control file using the // characters. All file contents located between the // and the end of the line will be ignored by the file parser.

The run control file variables that are required for a simulation are governed by the hierarchy of options that are selected. Some options will require a particular set of variables to specified, whereas other options may have different variables that need to be specified. This hierarchy includes elements such as the compressibility of the simulation, the presence of combustion or cavitation, the numerical methods selected to solve the governing equations, the forms of the governing equations, and the models used in the governing equations.

Listing 1: Sample run control file for lid-driven cavity flow

```
// Cavity flow, Re=1000.
{
// Grid file information.
grid_file_info: <file_type=VOG,Lref=1 m>

// Boundary condition information.
boundary_conditions:
<
BC_1=noslip, BC_2=noslip, BC_3=noslip, // left, right, bottom walls
BC_4=incompressibleInlet(v=-1.0 m/s), // lid
BC_5=symmetry,BC_6=symmetry // symmetry boundaries
>

// Initial condition.
initialCondition: <rho=1.0 kg/m/m/m,v=0.0,p=0.0>

// Thermodynamic and transport properties.
chemistry_model: air_1s0r
transport_model: const_viscosity
mu: 0.001

// Flow properties.
flowRegime: laminar
flowCompressibility: incompressible

// Time integration.
timeIntegrator: BDF
timeStep: 1.0e+30
numTimeSteps: 1001
convergenceTolerance: 1.0e-30
maxIterationsPerTimeStep: 1

// Fluxes and gradient limiting.
inviscidFlux: SOU
```

(continues on next page)

(continued from previous page)

```
limiter: none

// Equation options.
momentumEquationOptions: <linearSolver=SGS,relaxationFactor=0.7, maxIterations=5>

pressureCorrectionEquationOptions: <linearSolver=PETSC, relaxationFactor=0.1,␣
↪maxIterations=20>

// Output.
print_freq: 100
plot_freq: 100
restart_freq: 1000
}
```

1.2 Grid File

The only grid file format currently fully supported by *Stream* is the volume grid format (`.vog` extension). The only means of obtaining a grid in this format is to use the translators provided in the *Loci* distribution. These can be found in the `/bin` directory within the *Loci* installation directory.

Table 1: Translators for generation of `.vog` files

Grid Format	Translation Procedure
CFD++	cfd++2xdr → xdr2vog
Cobalt Solutions	cobalt2vog
Fluent	fluent2vog
Plot-3D	plot3d2vog
SolidMesh/AFLR3	ugrid2vog

The table above shows some of the existing translators that are available, and the command used to produce a `.vog` file from the input grid file. As an example, if the *AFLR3* mesh generator was used to generate a volume mesh, one would have a file called either case_name.ugrid or `case_name.b8.ugrid`, depending on whether the file was output in ASCII or binary form. To translate the file, one would issue the following command (assuming the grid was generated in units of inches):

```
ugrid2vog -in case_name
```

Following the completion of the translation, one should see the file `case_name.vog` in the directory where the command was issued. Note that while the grid may be generated in any units, upon translation, the internal units of the coordinates are in meters within the `.vog` file. All the translators shown in the *table* above require the units of the input grid so that coordinates can be scaled to meters for the output grid.

1.3 Execution Command

Below is an example command to run *Stream* within a Linux environment.

```
mpirun -np 10 stream --scheduleoutput -q solution case_name >>&run.log &
```

A breakdown of the components of the command above is as follows:

- `mpirun -np 10 stream`: Execute stream in parallel using 10 processes. The number of processes to use in executing the computation is specified with the `-np` option. The number of processes that can be used depends on the size of the grid being used. On typical high-performance computing (HPC) systems, *Stream* exhibits excellent scalability down to about 5000 grid cells per process. Thus, for a grid of a million cells, one would typically use no more than 200 processes.
- `--scheduleoutput`: The schedule file is a text-based file that contains the actual series of rules that are being executed by *Stream*. For single-process runs, this file is named `.schedule`, and will appear in the `/debug` directory if the `--scheduleoutput` argument is included on the command line. For multi-process runs, a series of files named `.schedule-\*` is generated, where the wildcard represents the process number(starting from 0). The schedule files are not typically of concern to the user during normal code usage, however, if a problem occurs, these files can be helpful to knowledgeable users in determining the cause of the problem.
- `-q solution`: The query value is passed to the code using the `-q` option on the command line. While other queries are possible with *Stream*, in practice, one will always pass the value `solution`, which indicates to the code that one wants the complete flow simulation solution.
- `>>& run.log &`: *Stream* writes out information to both standard-output and, should an error occur during code execution, to standard-error. This information can be redirected to a log file for permanent reference using the syntax `>>&` before the name of the log file on the command line. With this syntax, the file `run.log` will be created if it is not already present. If the file is already present, new information will be appended to the end of the existing `run.log` file. This is useful when conducting a restart so that the entire history of the run may be contained in a single file. If one uses the alternate `syntax >&`, any existing log file of the specified name will be deleted and a new log file created, thus destroying the information in the previous log file.

CHAPTER

TWO

SIMULATION OF LAMINAR INCOMPRESSIBLE FLOWS

The *Stream* solver is exceptionally suited for incompressible flows due to the use of a pressure-based solution framework. The incompressible flow assumption is appropriate for flows with low Mach number in which the density variation with pressure is minimal.

2.1 Example Case: Karman Vortex

Consider the case of an incompressible laminar viscous flow of water around a cylinder. Details about this case as well as the grid and run control files can be found on our website: https://www.snumerics.com/karman-vortex-example. In this example, a flow moving at 2.1930 m/s with a density of 1000 kg/m^3 passes over a cylinder and causes a time-dependent shedding pattern of vortices behind the cylinder. This case illustrates the physical phenomenon called a Karman vortex street. A schematic of the geometry and boundaries of the case is shown in the figure below.

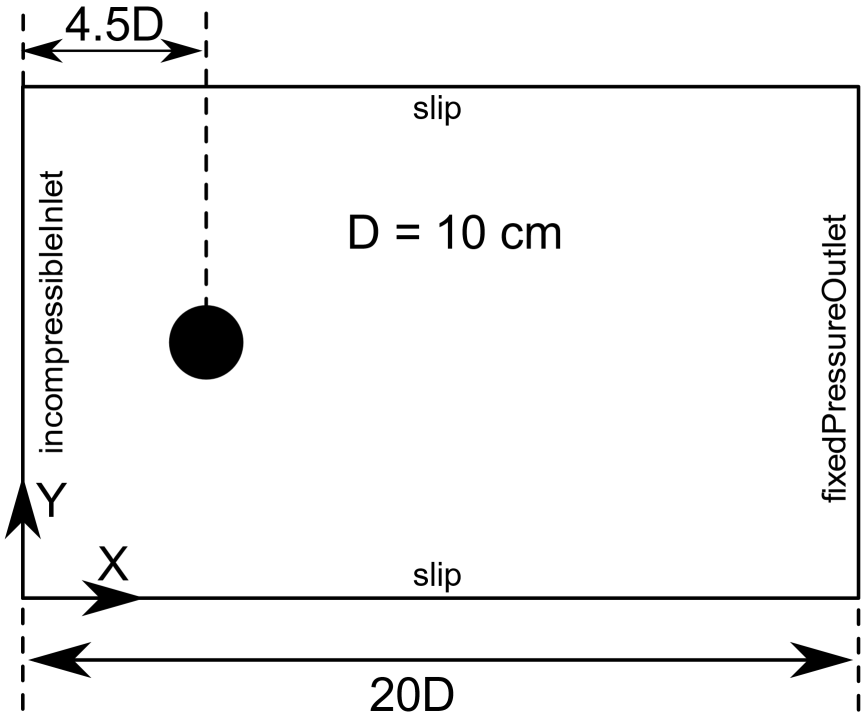

Fig. 1: Domain diagram for the Karman vortex case.

The run control file used for this simulation is shown below.

Listing 1: Sample run control file for the Karman vortex case.

```
{
// Grid file information.
grid_file_info: <file_type=VOG, Lref=1 m, pieSlice>

boundary_conditions: <
Inlet=incompressibleInlet(v=2.1930 m/s),
Outlet=fixedPressureOutlet(p=1 Pa),
Walls=slip,
Particle=noslip,
BackWall=symmetry,
FrontWall=symmetry
>

// initial conditions in nozzle
initialCondition: <rho=1000 kg/m^3, p=1 Pa, v=0.0 m/s>

// Flow properties
flowRegime: laminar
flowCompressibility: incompressible

// Transport properties
transport_model: const_viscosity
mu: 8.77205e-4

// Time-stepping (timeIntegrator[Euler,PISO])
timeIntegrator: BDF2
timeStep: 1.0e-2
numTimeSteps: 501
convergenceTolerance: 1.0e-30
maxIterationsPerTimeStep: 30

// InviscidFlux [FOU,SOU,Roe(compressible only)]
inviscidFlux: SOU

// Gradient limiting.
limiter: venkatakrishnan

// HYPRE solver parameters
linearSolverTolerance: 5.0e-02
hypreSolverName: AMG

// Momentum equation (linearSolver[SGS,PETSC],0.0<relaxationFactor<1.0)
momentumEquationOptions: <linearSolver=SGS, relaxationFactor=0.7, maxIterations=3>

// Pressure equation
pressureCorrectionEquationOptions:<linearSolver=HYPRE, relaxationFactor=0.2,␣
↪maxIterations=20>
pressureBasedMethod: SIMPLEC

// Printing, plotting and restart parameters.
```

(continues on next page)

(continued from previous page)

```
print_freq: 10
plot_freq: 1
plot_output: pResidualTT
restart_freq: 200
}
```

2.1.1 Boundary Conditions

The boundary condition used for the inflow in this case is the `incompressibleInlet`. Only the velocity on this boundary needs to be specified. The velocity specification can take many forms, many of which are described *here*. A velocity directed into a domain normal to a boundary can be specified by simply providing the `v=` input option.

```
Inlet=incompressibleInlet(v=2.1930 m/s)
```

For incompressible simulations the only boundary condition that can be used is the `fixedPressureOutlet`. A fixed static pressure along the boundary is set in this case. Other options for this boundary condition are detailed *here*. The boundary is intentionally placed far from the cylinder in a location where specifying a fixed pressure will not adversely affect the flow near the cylinder.

```
Outlet=fixedPressureOutlet(p=1 Pa)
```

The meaning of pressure in incompressible flows is not intuitively obvious. The governing equations for incompressible flow do not demand the specification of a pressure reference; only the gradient of the pressure is of importance. However, when using the `fixedPressureOutlet` boundary condition, the reference pressure will be set at the boundary using the value provided from the `p=` input option.

The boundary condition on the cylinder surface is set to `noslip` to enforce the zero-velocity condition. Since we are not concerned with the boundary layer effects on the upper and lower walls, the boundary condition for these boundaries is set to `slip`. The front and back (going into the page) boundaries are set to `symmetry` to model zero variation in the flow field in the z-direction.

2.1.2 Initial Conditions

In this example the initial flow is a quiescent flow of water with a pressure of 1 Pascal and a density of 1000 kg/m^3 , which is specified as follows:

```
initialCondition:  <rho=1000 kg/m^3, p=1 Pa, v=0.0m/s>
```

For incompressible simulations one should always initialize the pressure in the domain to the same value as the `fixedPressureOutlet` to prevent the development of large velocities at the boundary due to a discontinuous change in the pressure level.

2.1.3 Numerics

The second-order accurate `timeIntegrator` option `BDF2` is chosen over the first-order accurate `BDF` option for this simulation since we are concerned with time accuracy. The `BDF2` time-stepping scheme achieves time-accurate results at much larger timesteps compared to the `BDF` scheme. The second-order `inviscidFlux` option `SOU` selected here is generally preferred over `FOU` due to its lower numerical dissipation characteristics and should always be used if possible. Here we are using the Venkatakrishnan limiter to limit the second order convective fluxes; more information about the flux limiters can be found *here*.

For incompressible flows, only the momentum and pressure-correction equations need to be solved. The specification for the momentum equation options is shown below. Here we are using the symmetric Gauss-Seidel (`SGS`) method to

solve the linear system. A relaxation factor of 0.7 has been chosen, which indicates the 70% of the current iteration solution and 30% of the previous iteration solution will be averaged and used as the new value. Lower values for the `relaxationFactor` can be used if numerical stability issues arise. The `maxIterations` option specifies the maximum number of iterations to use in the linear solver. For the `SGS` solver, the maximum number of iterations is always used. Typically, one should not use more than 5 iterations in the linear solver for the momentum equation.

```
momentumEquationOptions:  <linearSolver=SGS, relaxationFactor=0.7, maxIterations=3>
```

Options for pressure-correction equation are shown below. The HYPRE linear solver is generally the preferred linear solver for the pressure-correction equation due to the Poisson-like nature of this equation. Typically, more iterations are required for the pressure-correction equation than the other governing equations.

```
pressureCorrectionEquationOptions:<linearSolver=HYPRE, relaxationFactor=0.2,
maxIterations=20>
```

Stream supports both the `SIMPLE` and `SIMPLEC` pressure-based methods, the selection of which is made using the `pressureBasedMethod` variable.

```
pressureBasedMethod:  SIMPLEC
```

2.1.4 Miscellaneous

In *Stream*, the method of constraining a flow to be two-dimensional is to provide the `pieSlice` input in the `grid_file_info` section. This disables the z-component of the equations. **It is important to note that this option presumed a grid that exists in an x-y plane.**

2.2 Helpful Guidance

- Residuals are output to the log file (lines that start with `R:`). These lines contain the residuals for each of the governing equations being solved. See the following *Appendix* for more information. A two to three order of magnitude drop in the residuals from the starting to the ending iteration within every timestep is generally considered acceptable for a time-accurate simulation.
- *The table below* shows the field variables that are available for output in laminar incompressible flow simulations. Default variables are output automatically and do not need to be specified in the `` `plot_output line `` in the run control file. The variables `pResidualTT` and `vResidualTT` shown in the table are useful for gauging solution convergence. A discussion of the principle behind the turn-over time can be found *here*.

Table 1: Field Variable Output for Laminar Incompressible Flows

Variable	Description	Default
laminarViscosity	Fluid viscosity	No
pg	Gauge pressure	Yes
pPrime	Pressure-correction	No
pResidual	Pressure-correction eq. residual	No
pResidualTT	Pressure-correction eq. turn-over time	No
r	Density	Yes
v	Velocity	Yes
vort_mag	Vorticity magnitude	No
vResidual	Velocity eq. residual	No
vResidualTT	Velocity eq. turn-over time	No

CHAPTER

THREE

SIMULATION OF LAMINAR COMPRESSIBLE FLOWS

The *Stream* solver can simulate flows where compressible effects are present, including flows with shock waves. Compressibility of a flow is usually categorized by observing a change of the density of a fluid as a response to a change in the pressure. Compressibility should be considered where flows may have Mach numbers greater than 0.3 typically. Another class of flows that should be solved by considering compressibility effects are Rayleigh-type flows where heat transfer through boundaries is considered.

For compressible flows without shock waves, the user should use the `FOU` and `SOU` flux schemes. For flows with shock waves, the `SLAU` flux scheme is available. Details about the flux schemes that are available in are detailed *here*. The run control file specification mostly remains the same from incompressible cases. An additional data file, called a chemistry model file(`.mdl`), is required for compressible flows. See the following *Appendix* contains more information about the contents of a chemistry model file and information about how to create one.

3.1 Example Case: Prandtl-Meyer Fan

Consider the case of a compressible laminar inviscid flow around a corner as shown below. For this case, the fluid in the domain is air. Details about the problem and grid and run control file can be found on our website. In this example, a flow of air at a temperature of 550 Rankine (305 Kelvin) moving at Mach 2.5 flows past a bend in the geometry of the domain. The supersonic flow produces a Prandtl-Meyer expansion fan near the corner in response to the bend.

A sample run control file used to run the Prandtl-Meyer simulation is shown below.

Listing 1: Sample run control file for the Prandtl-Meyer expansion fan problem.

```
{
grid_file_info: <file_type=VOG, Lref=1m, pieSlice>

boundary_conditions: <
Inlet=supersonicInlet(v=2874 ft/s, p=12 psia, T=550 R),
Outlet=extrapolatedPressureOutlet,
Wall=slip,
Top=slip,
BC_5=symmetry(),
BC_6=symmetry()
>

initialCondition: <T=550 R, p=12 psia, v=2874.0322 ft/s>

// Flow properties
flowRegime: laminar
```

(continues on next page)

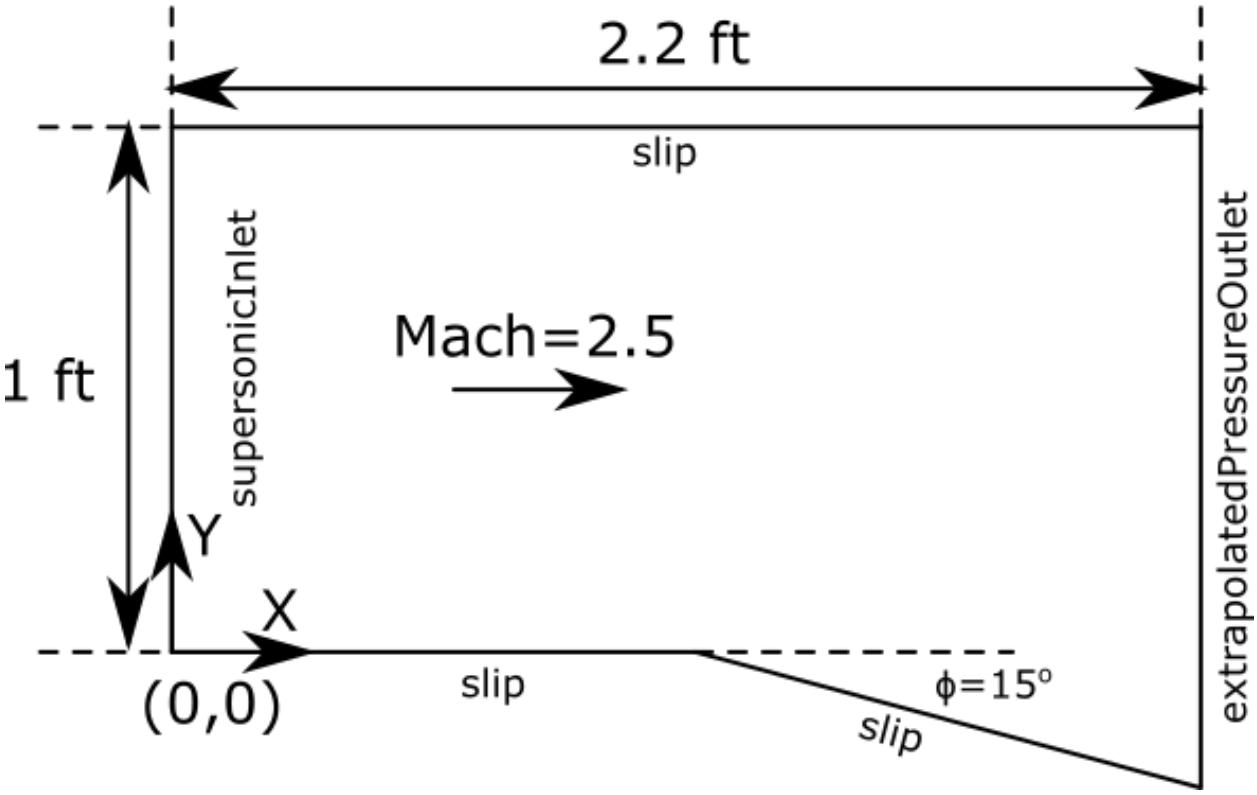

Fig. 1: Domain diagram for the Prandtl-Meyer expansion fan problem.

(continued from previous page)

```
flowCompressibility: compressible

// Gas Model & Transport Properties
chemistry_model: air_1s0r
transport_model: const_viscosity
mu: 1.0e-15
kcond: 1.0e-15

// Time-Stepping
timeIntegrator: BDF2
timeStep: 1.0e-5
numTimeSteps: 10001
convergenceTolerance: 1.0e-30
maxIterationsPerTimeStep: 30

// Fluxes
inviscidFlux: SOU
limiter: venkatakrishnan

hypreSolverName: GMRES
linearSolverTolerance: 1.0e-02

// Momentum equation (linearSolver[SGS], 0.0<relaxationFactor<1.0)
```

(continues on next page)

(continued from previous page)

```
momentumEquationOptions: <linearSolver=SGS,relaxationFactor=0.5,maxIterations=5>

// Pressure equation (linearSolver[SGS, PETSC, HYPRE], 0.0<relaxationFactor<1.0)
pressureCorrectionEquationOptions: <linearSolver=HYPRE, relaxationFactor=0.1,␣
↪maxIterations=50>
pressureBasedMethod: SIMPLEC

// Energy equation (linearSolver[SGS], 0.0<relaxationFactor<1.0)
energyEquationOptions: <linearSolver=SGS, relaxationFactor=0.3, maxIterations=5,␣
↪form=temperature>
TclipMax: 2000.0

// Printing, plotting and restart parameters.
print_freq: 100
plot_freq: 500
plot_output: pResidualTT
restart_freq: 1000
}
```

3.1.1 Boundary Conditions

The boundary condition used for the inflow in this case is the `supersonicInlet`. This boundary condition is used for flows with Mach numbers greater than one. The velocity, pressure, and temperature must be specified for this inlet. The velocity specification `v=` assumes that the velocity is directed along the normal vector pointing into the domain. See the following *Appendix* for more information about all the ways to provide specifications to this boundary condition.

```
Inlet=supersonicInlet(v=2874.0322 ft/s, p=12 psia, T=550 R)
```

For supersonic flows, the location of the outlet boundary can be much closer to the area of interest because disturbances near the boundary do not travel back into the domain due to the supersonic nature of the flow. The outlet boundary is located just downstream of the corner in this example and is an `extrapolatedPressureOutlet`. This boundary condition is a zero-gradient boundary that is permissible in the presence of flows that are supersonic.

```
Outlet=extrapolatedPressureOutlet
```

All solid surfaces are set to the `slip` boundary condition because of the inviscid nature of the flow. The front and back (going into the page) boundaries are set to `symmetry` to model zero variation in the flow field in the z-direction.

3.1.2 Initial Conditions

In this example the initial flow is air in motion with a velocity of 2874 ft/s. The pressure is 12 psia and the temperature is 550 Rankine. These initial conditions are specified as follows:

```
initialCondition:  <v=2874 ft/s, p=12 psia, T=550 R>
```

English units are used in this example to show that different unit types are supported for many of the inputs. **All inputs are converted to S.I. units within the code.**

3.1.3 Transport Properties

For this case, the fluid is air, and the viscosity is set to a small value to approximate an inviscid flow. The thermal conductivity is also provided for use in the energy equation.

```
transport_model:  const_viscosity
mu:  1.0e-15
kcond:  0.0265
```

A specification of the species that are present in the domain is needed when running compressible flows. This provides information about the equation of state to use for the species. The species information for the air is encoded in the chemistry model file (`.mdl`), which has its specification provided *here*. The contents of the file are reproduced below for ease of replicating this case.

Listing 2: The `.mdl` file for the air species used in the Prandtl-Meyer expansion fan case.

```
// Model for Air as an ideal gas
species = {
_Air = < m=28.89, n=2.5, href=0, sref=0, Tref=298.0, Pref=10000.0, mf=1.0 >;
};
reactions = {
};
```

3.1.4 Numerics

The first-order accurate `BDF` time-stepping scheme is used for this case because we are primarily interested in the steady-state behavior of the expansion fan within the domain. The second-order `inviscidFlux` option `SOU` are used is used because of its low numerical dissipation characteristics. For supersonic flows with shockwaves, the `SOU` flux scheme is not appropriate because it is unstable in the presence of discontinuities in a flow, but it is ok for this case because there is only a smoothly varying isentropic expansion fan in the domain for this case.

The momentum, pressure correction, and energy equations need to be solved for compressible laminar flows. The specification for the momentum equation options is shown below. Here we are using the symmetric Gauss-Seidel (`SGS`) method to solve the linear system. A relaxation factor of 0.5 has been chosen, which indicates the 50% of the current iteration solution and 50% of the previous iteration solution will be averaged and used as the new value. Lower values for the `relaxationFactor` can be used if numerical stability issues arise. The `maxIterations` option specifies the maximum number of iterations to use in the linear solver. For the `SGS` solver, the maximum number of iterations is always used. Typically, one should not use more than 5 iterations in the linear solver for the momentum equation.

```
momentumEquationOptions:  <linearSolver=SGS, relaxationFactor=0.5, maxIterations=5>
```

Options for pressure-correction equation are shown below. The PETSC linear solver is used in this case to show that other linear solvers are supported. Typically, more iterations are required for the pressure-correction equation than the other governing equations.

```
pressureCorrectionEquationOptions:<linearSolver=PETSC, relaxationFactor=0.5,
maxIterations=5>
```

For compressible flows *Stream* supports three forms of the energy equation: a temperature form, a total energy form, and a total enthalpy form. For this case, the temperature form of the energy equation is used by specifying form=temperature in the `energyEquationOptions`. The other forms of the energy equation are discussed in the *Appendix*.

```
energyEquationOptions:  <linearSolver=SGS, relaxationFactor=0.3, maxIterations=5,
form=temperature>
```

3.1.5 Miscellaneous

In *Stream*, an inviscid simulation can be approximated by setting the `transport_model` variable to `const_viscosity` and providing a very small value of the viscosity, as was done in this example.

3.2 Helpful Guidance

- *The table below* shows additional field variables that are available for output in laminar compressible flow simulations. These are in addition to the field outputs that are available for the laminar incompressible simulations. Default variables are output automatically and do not need to be specified in the `plot_output` line in the run control file. The variables `pResidualTT` and `vResidualTT` shown in the table are useful for gauging solution convergence. A discussion of the principle behind the turn-over time can be found *here*.

Table 1: Field Variable Output for Laminar Compressible Flows

Variable	Description	Default
a	Speed of sound	Yes
cfl	CFL number	No
cp	Specific heat	No
kineticEnergy	Kinetic energy	No
hResidual	Energy eq. residual	No
hResidualTT	Energy eq. turn-over time	No
kconduct	Thermal conductivity	No
mix	Species mass fractions	Yes
t	Temperature	Yes

CHAPTER

FOUR

SIMULATION OF TURBULENT FLOWS

In this section, we describe the capabilities within *Stream* for simulating turbulent flows. *Stream* contains a variety of turbulence models that are suitable for both steady-state and unsteady turbulent flow analysis. Here we assume that the user has a sufficient understanding of the terminology used in the description of turbulent flows and knowledge of the various analytical turbulence closure models that are employed in the CFD field. For a more comprehensive description of the governing equations of turbulent flow and the specific turbulence models used in *Stream*, see [Pope2000] as well as the *Stream* Theory Manual.

4.1 RANS Models

Stream contains a selection of Reynolds-Averaged Navier-Stokes (RANS) turbulence models from both the k-epsilon and k-omega families. The governing assumptions behind the derivation of these models generally limits their accurate predictive capability to relatively benign turbulent flow scenarios in which flows remain predominantly attached to the walls of the domain (no separation). Mean-flow properties of separated flows (such as drag coefficients) can be accurately predicted, including for flows with massive separation under certain conditions (i.e., post-transition, supercritical turbulence), however, the prediction of intermittent and transitional behavior of flows cannot be predicted with these models. The paper of Hart [Hart2016] contains a detailed comparison of RANS turbulence models and Scale Resolving Simulation (SRS) models (which includes the DES and LES models to be discussed later in this chapter), regarding their predictive capability for massively separated flow. This information may be of use in helping users decide whether RANS models are applicable to their problems of interest.

One major addition to the run control file for turbulent flow is the variable `turbulenceEquationOptions`. This variable is used for specifying the turbulence model, the relaxation and linear solver parameters for the turbulence equations and any other auxiliary input required by the turbulence models. Turbulence model selection is made using the model option. The table below shows the allowable values for this option for RANS simulations.

Freestream values for the turbulence variables may be set using the `kFreestream`, `omegaFreestream` and `epsilonFreestream` variables. These variables are commonly set to the same values that have been specified at the inlet boundary and have default values of 1.0e-08, 10.0 and 10.0, respectively. These variables are only used to ensure that the computed turbulence values remain realizable (i.e. k, omega, epsilon > 0) and will have no impact on the final solution unless clipping of the turbulent variables is occurring.

Table 1: RANS Turbulence Model Options for `turbulenceModelOptions` variable

Model Family	Model Names	Notes
k-ϵ	• `StandardKE` • `RealizableKE`	• `StandardKE` from Launder & Spaulding [LaSp1974] • `RealizableKE` model of Shih et. al. [SLSY1995]
k-ω BSL	• `menterBSL` • `menterBSL_m` • `menterBSL_V` • `menterBSL_Vm` • `menterBSL_KL` • `menterBSL_KLm`	• `menterBSL` is the original 1994 Baseline Model of Menter • For other variants of the Menter baseline models, see NASA BSL
k-ω SST	• `menterSST` • `menterSST_m` • `menterSST_V` • `menterSST_Vm` • `menterSST_KL` • `menterSST_KLm`	• `menterSST` is the original 1994 Shear Stress Transport Model of Menter • For other variants of the 2003 Menter shear-stress transport model, see NASA SST
k-ω-SST-2003	• `menterSST2003` • `menterSST2003_m` • `menterSST2003_V` • `menterSST2003_Vm` • `menterSST2003_KL` • `menterSST2003_KLm`	• `menterSST2003` is the improved 2003 Shear Stress Transport Model of Menter • For other variants of the Menter shear-stress transport model, see NASA SST

4.2 DES Models

While the RANS models described above may be used in an unsteady mode with small time step to simulate complex turbulence flows with separation and bluff-body-type recirculation zones, they are generally found to be lacking for these flows. Typically, in zones of separation, one expects to find a collection of eddies that span a range of length scales, from the largest energy-containing eddies to the smallest dissipative eddies of the turbulence cascade. However, due to the modeling assumptions made during closure of the ensemble-averaged equations, solutions for separated flows are almost universally found to exhibit a time-averaged character, whereby small-scale eddy content is severely diminished or even completely absent, and the point-by-point frequency spectrum is devoid of any chaotic or high-frequency content. To provide a remedy for this deficiency, *Stream* provides several turbulence models in the Detached-Eddy Simulation (DES) family, which can be used to simulate turbulent flows with significant separation. While the analytical formulation these models is not based on the strict filtering process used in the traditional Large-Eddy Simulation (LES) approach, these DES models provide turbulent solutions that are similar in character to LES models and are thought of most conveniently as hybrid models that blend the characteristics of LES in separated an interior region while transitioning to traditional Unsteady RANS (URANS) behavior in the near-wall region. A great discussion about detached eddy simulation methodology can be found here.

4.2.1 A Brief DES History

A brief history of DES family of models is presented below for readers who may be interested in the evolution of the DES models.

Table 2: History of DES Family of Models

Reference	Main Features	Comments
Spalart 1997	• Original motivation for **DES** • Basic equations (with C_{des} constant undetermined) • Two-dimensional examples.	• First paper on DES concept • In the context of Spalart-Almaras (SA) model only
Shur 1999	• First true 3D application of DES • Calibration of C_{des} • Successful prediction of airfoil forces	• Usage of DES in 3D
Travin 2000	• First DES with grid refinement • Fair agreement on the drag crisis • Refined definition of DES	
Nitkin 2000	• Discussion of DES for wall modeling inside LES	• One of the first attempts of employing DES as an LES using wall modeling
Strelets 2001	• Wide range of applications, including models other than Spalart-Almaras.	• First **SST-DES** formulation
Menter and Kuntz 2002	• Demonstrate grid-induced separation • Propose *shielding* for SST model to "preserve RANS mode" or "delay LES mode" in boundary layer by using blending functions F_1 or F_2	• First demonstration of Grid-Induced Separation (GIS) with DES • They refer to it as "Zonal SST-DES" • Ambiguity in "shielding" function – can use F_1 or F_2 $\rightarrow$ Spalart 2006 improves this • Provides motivation for the DDES model of Strelets (2006)
Spalart 2006	• Introduced **Delayed DES (DDES)** to avoid grid-induced separation (GIS) • Coined the phrase **Modeled-Stress Depletion (MSD)** and showed it to be the cause of GIS • Use more generic shielding function based on eddy viscosity and wall distance (using parameter r_d and function f_d)	• MSD results from insufficient flow instabilities downstream of the switch from the RANS to the LES model formulation => the switch from the RANS to the LES model inside wall boundary layers is not desirable • Applied to S-A model in this paper • But also applicable to SST model (done in Gritskevich-Menter 2012)
Shur 2008 & Travin 2006	• **Improved DDES (IDDES)**	• Original DES results in a

4.2.2 Activating DES Mode

To active DES mode in *Stream*, the user should provide an addition option in the `turbulenceEquationOptions` options list of the form `des=<ModelName>`. An example is shown below.

```
turbulenceEquationOptions:  <model=menterSST2003, des=IDDES2012, linearSolver=SGS,
relaxationFactor=0.5, maxIterations=5>
```

The table below shows the allowable values for the des option for running simulations with DES.

Table 3: DES Turbulence Model Selection for `turbulenceModelOptions` variable

Model Value	Description
`DES2001`	2001 Strelets DES model [Stre2001]
`DDES`	Menter Delayed DES model [MeKu2002]
`DDES2012`	2012 SST-DDES model [GGSM2012]
`IDDES2012`	Improved 2012 SST-IDDES model [GGSM2012]

A recommended combination is the `menterSST2003` turbulence model with the `IDDES2012` des model. For cases that experience numerical instability, try the `DDES2012` des option.

4.3 Example Case: Backward Step

Consider the case of a compressible turbulent flow over a backward step as shown in Figure 8. For this case, the fluid in the domain is air. Details about the problem and grid and run control file can be found on our website.

In this example a turbulent flow of air at around 537 Rankine (298 Kelvin) is moving at 41.7 m/s and it passes over a sudden drop. This drop causes the flow to create a circulating region near the drop. The size and character of this recirculation bubble is the subject of the backward step problem. The RANS solution of the flow field is what will be solved for in this example.

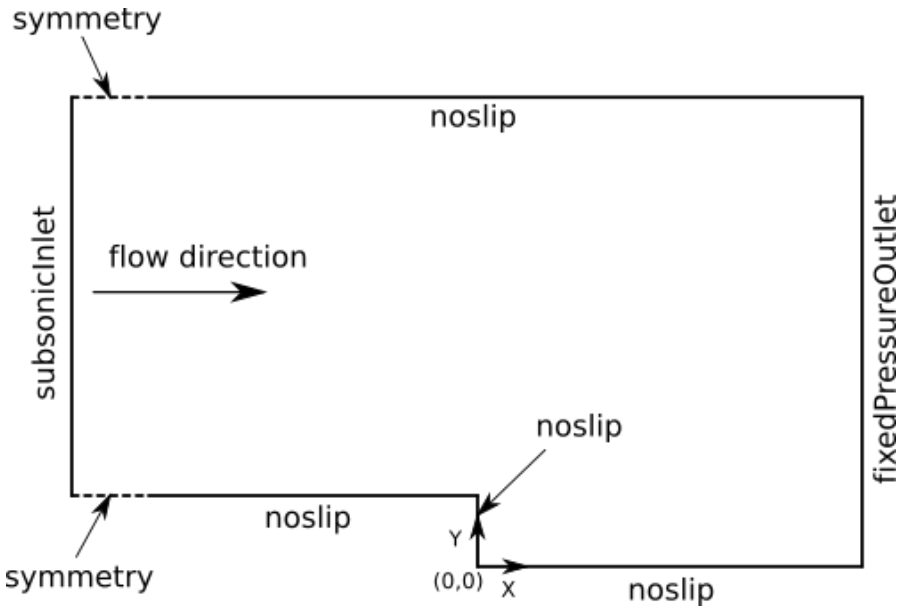

Fig. 1: Domain diagram for the backward step problem.

When running turbulent simulations, most of the run control file inputs described in previous chapters remain the same, with minor changes. Some additional input required for the specification of data needed for the governing turbulence equations is required. A sample run control file used to run the backward step simulation is shown below.

Listing 1: Sample run control file for the backward step problem.

```
{
grid_file_info: <file_type=VOG, Lref=1m, pieSlice>

boundary_conditions: <
BC_1=noslip(adiabatic), //Upstream Bottom Wall
BC_2=noslip(adiabatic), // Top Wall
BC_3=subsonicInlet(T=537.0 R, v=41.7096 m/s, k=0.00097, omega=5091), //Inlet
BC_5=symmetry, //Backwall
BC_6=symmetry, // Frontwall
BC_13=noslip(adiabatic), // Bottom of Downstream section
BC_15=noslip(adiabatic), // Vertical Step Face
BC_22=fixedPressureOutlet(pMean=1.0 atm), // Outlet
>

initialCondition: <p=1 atm, T=537.0 R, v=41.7096 m/s, k=0.00097, omega=5091>

// Flow properties
flowRegime: turbulent
flowCompressibility: compressible

// Gas model.
chemistry_model: air_1s0r
transport_model: const_viscosity
mu: 2.498e-5
kcond: 4.175e-2

// Time-stepping
timeIntegrator: BDF
timeStep: 1e-3
numTimeSteps: 10001
convergenceTolerance: 1.0e-30
maxIterationsPerTimeStep: 30

// InviscidFlux
inviscidFlux: SOU
turbulenceInviscidFlux: SOU
limiter: venkatakrishnan

linearSolverTolerance: 1.0e-02

// Momentum equation (linearSolver[SGS,PETSC],0.0<relaxationFactor<1.0)
momentumEquationOptions: <linearSolver=SGS,relaxationFactor=0.5,maxIterations=5>

// Pressure equation
pressureCorrectionEquationOptions: <linearSolver=PETSC,relaxationFactor=0.1,
↪maxIterations=50>
pressureBasedMethod: SIMPLEC
```

(continues on next page)

(continued from previous page)

```
// Energy equation (linearSolver[SGS,PETSC],0.0<relaxationFactor<1.0)
energyEquationOptions: <linearSolver=SGS,relaxationFactor=0.5,maxIterations=5,
↪form=temperature>

// Turbulence equation
turbulenceEquationOptions: <model=menterSST2003, linearSolver=SGS, relaxationFactor=0.5,␣
↪maxIterations=5>
kFreestream: 0.00097
omegaFreestream: 5091
eddyViscosityLimit: 10000

// Printing, plotting and restart parameters.
print_freq: 250
plot_freq: 2000
plot_output: a, pResidualTT, laminarViscosity, viscosityRatio, k, omega
restart_freq: 2000
}
```

4.3.1 Boundary Conditions

All boundary condition entries remain the same except for the inlet boundary, where one must now specify inlet values for the turbulent kinetic energy, `k`, and either the turbulent dissipation, `epsilon`, or the specific turbulent dissipation, `omega`. Since we are using a model in the k-omega family for this example, `omega` must be provided.

The boundary condition used for the inflow in this turbulent example is the `subsonicInlet`. This boundary condition is like the one used for *compressible flows* with the addition of turbulence parameters. See the following *Appendix* for more information about this boundary condition.

```
BC_3=subsonicInlet(T=537.0 R, v=41.7096 m/s, k=0.00097, omega=5091)
```

The boundary condition requires two turbulence parameters, `k` and `omega`. The choice of turbulence inputs depends on the model that is selected in the `turbulenceEquationOptions` variable. More details can be found in the *Appendix* regarding the `turbulenceEquationOptions` variable. The outlet is a `fixedPressureOutlet` with a mean pressure constraint set on it as shown below.

```
BC_22=fixedPressureOutlet(pMean=1.0 atm)
```

All solid surfaces are set to `noslip(adiabatic)` because of the viscous nature of the flow. The keyword `adiabatic` must be passed to the `noslip` boundary condition for compressible flow simulations. More details on the `noslip` boundary condition can be found in the following *Appendix*. The front and back (going into the page) boundaries are set to `symmetry` to model zero variation in the flow field in the z-direction. A small section of the domain upstream is set to `symmetry` near the inlet because the backward step is a canonical validation problem provided by the NASA Langley turbulence modeling resource.

4.3.2 Initial Conditions

In this example the initial flow is a flow of air, already in motion, with a pressure of 1 atm, a temperature of 537 Ranine, and a velocity of 41.7096 m/s, which is specified as follows:

```
initialCondition:  <p=1 atm, T=537.0 R, v=41.7096 m/s, k=0.00097, omega=5091>
```

Depending on the turbulence model selected for the `turbulenceEquationOptions` like the options shown in the *RANS options table*, the name of the turbulence variable provided in the initial condition will change (will either be `omega` or `epsilon`).

4.3.3 Transport Properties

For this case, the fluid is air, and the viscosity and thermal conductivity are set to be constant values that are selected to model the typical values for air at the temperature and pressure of the example case under consideration. The `const_viscosity` value for the `transport_model` option allows for a single value of the viscosity and thermal conductivity to be set, as shown below.

```
transport_model:  const_viscosity
mu:  2.498e-5
kcond:  4.175e-2
```

A specification of the species is needed when running compressible flows. This provides information about the equation of state to use for the species in the domain. The species information for the air is encoded in the chemistry model file (`.mdl`) file for this case, whose specification format can be found *here*. The contents of the file are reproduced here for ease of replicating this case.

Listing 2: The `.mdl` file for the air species used in the backward step case.

```
// Model for Air as an ideal gas
species = {
_Air = < m=28.89, n=2.5, href=0, sref=0, Tref=298.0, Pref=10000.0, mf=1.0 >;
};
reactions = {
};
```

4.3.4 Numerics

For turbulent flows, the user must set the value of the variable `flowRegime` to turbulent in the run control file.

```
flowRegime:  turbulent
```

In *Stream,* the inviscid flux used for the turbulence equations may be set independently from the main inviscid flux used for the momentum, energy, and species equations by using the `turbulenceInviscidFlux` variable. The default value is `SOU`, which provides for the second-order upwinding of the independent variables k, omega and epsilon in the convection term of the turbulence equations. To achieve second-order spatial accuracy, one should use a value of `SOU` for both `inviscidFlux` and `turbulenceInviscidFlux`. Under certain extreme flow scenarios, convergence of the system of equations may be difficult to obtain using second-order convective fluxes for the turbulence equations, requiring one to downgrade to first-order upwinding by specifying a value of `FOU` for the `turbulenceInviscidFlux` variable. This, however, has rarely been found to be needed in practice.

The first-order accurate BDF time-stepping scheme and the second-order inviscidFlux option SOU are used is used in this example. BDF is used because the time-accurate evolution of the flowfield is not of interest, only the steady state solution is desired.

The momentum, pressure correction, energy, and turbulence closure equations need to be solved for compressible turbulent flows.

The new addition is the turbulenceEquationOptions which controls the turbulence model that is used as well as the numerical controls for solving the turbulence closure equations.

```
turbulenceEquationOptions:  <model=menterSST2003, linearSolver=SGS, relaxationFactor=0.5,
maxIterations=5>
```

The linear solver options are like the other governing equations that have been previously covered. All linear solvers discussed(*here*) are supported for the value of linearSolver, however one should rarely need to use anything other than the SGS solver.

This example uses the MenterSST2003 turbulence model, which is a Reynolds Averaged Navier-Stokes (RANS) model, because that is what the NASA validation example called for.

4.3.5 Output Variables

The table below shows additional field variables that are available for output in turbulent flow simulations. **NOTE:** The variables kClip, omegaClip and epsilonClip are boolean variables that are assigned a value of 0 for no clipping and a value of 1.0 when clipping is active. These are in addition to the field outputs that are available for the laminar compressible/incompressible simulations. Default variables are output automatically and do not need to be specified in the plot_output line in the run control file. The variables kResidualTT, omegaResidualTT, and epsilonResidualTT are useful for gauging solution convergence.

Table 4: Field Variable Output for Turbulent Flows

Variable	Description	Default
k	Turbulent kinetic energy	No
kclip	Turbulent kinetic energy clipped flag	No
kResidual	Turbulent kinetic energy eq. residual	No
kResidualTT	Turbulent kinetic energy eq. turn-over time	No
omega	Specific dissipation rate	No
omegaclip	Specific dissipation rate clipped flag	No
omegaResidual	Specific dissipation rate eq. residual	No
omegaResidualTT	Specific dissipation rate eq. turn-over time	No
epsilon	Turbulent dissipation	No
epsilonClip	Turbulent dissipation clipped flag	No
epsilonResidual	Turbulent dissipation eq. residual	No
epsilonResidualTT	Turbulent dissipation eq. turn-over-time	No
viscosityRatio	Turbulent/laminar viscosity ratio	No
eddyViscosity	Turbulent viscosity	No

References

CHAPTER

FIVE

SIMULATION OF CAVITATING FLOWS

Stream can be used to simulate flows with cavitation by loading the cavitation module at run-time. This is done by inserting the following `loadModule` directive into the run control file as shown below.

```
loadModule: cavitation_nist
{
... standard run control file content
}
```

The `cavitation_nist` module uses tabulated *REFPROP* data for the equation of state and the fluid transport properties. The following sections discuss the analytical models for the cavitation source terms and provide information on the run control file variables required to perform simulations with the cavitation module.

5.1 Cavitation Model

Stream uses a homogeneous equilibrium model (HEM) to simulate the onset and evolution of cavitation within the fluid system. With such a model, the phases in the system (liquid, vapor, and non-condensable gas) are spatially averaged and assumed to occupy the same volume (homogeneously mixed). In addition, all phases are assumed to be in both mechanical and thermodynamic equilibrium, whereby the pressure, temperature, and velocity of the phases at any point are identical. The HEM is considered a good approximation for cavitating flows where the disturbance time scales and frequencies are long compared to the cavitation equilibration time. With these approximations, the continuity, momentum, and energy equations are solved for the mixture, rather than for the individual constituents of the flow. In addition, a transport equation for the vapor mass fraction is solved which contains source terms which generate vapor when the local flow pressure becomes lower than the vapor pressure of the fluid at the local temperature. A transport equation is also solved for the non-condensable gas (NCG) mass fraction. The transport equations for these two quantities are shown below:

$$\frac{\partial(\rho\alpha_v)}{\partial t} + \frac{\partial(\rho v_j \alpha_v)}{\partial x_j} = F(P, P_{\min}, F_{\max}, n) \cdot S(P, P_{vap}(T))$$

$$\frac{\partial \rho \alpha_{NCG}}{\partial t} + \frac{\partial \rho v_j \alpha_{NCG}}{\partial x_j} = 0$$

where ρ is the mixture density, v_j is the flow velocity, P is the static pressure, T is the temperature, α_V is the vapor mass fraction, and α_{NCG} is the non-condensable mass fraction . The following subsections describe the models that are available for the net vapor production rate source term $S(P, P_{vap}(T))$ and the source term scaling factor $F(P, P_{\min}, F_{\max}, n)$, which is can be activated in certain instances to prevent flow pressures from reaching unphysically low values.

5.1.1 Merkle Source Term Model

The Merkle source term model [MeFB1998] [HoAU2007] is a simple heuristic model in which the net vapor production rate is linearly proportional to the difference between the local flow pressure and the local vapor pressure. The expression for the net vapor production rate is given by the following:

$$S(P, P_{vap}(T)) = \rho(K_f \alpha_L + K_b \alpha_V)$$

where ρ is the mixture density, α_L is the liquid mass fraction, and α_V is the vapor mass fraction. The forward production and backward destruction rate coefficients K_f and K_b are given by:

$$K_f = \begin{cases} 0, & \text{if } P > P_{vap} \\ \frac{1}{\tau_{vap}} \left(\frac{P_{vap} - P}{\frac{1}{2}\rho_\infty U_\infty L_\infty} \right), & \text{if } P \leq P_{vap} \end{cases}$$

$$K_b = \begin{cases} 0, & \text{if } P < P_{vap} \\ \frac{1}{\tau_{\text{cond}}} \left(\frac{P_{vap} - P}{\frac{1}{2}\rho_\infty U_\infty L_\infty} \right), & \text{if } P > P_{vap} \end{cases}$$

In the expressions above, τ_{vap} is the vaporization rate constant, τ_{cond} is the condensation rate constant, and ρ_∞, U_∞, and L_∞ represent the freestream fluid density, velocity, and the reference length scale, respectively. These parameters can be specified in the run control file using the names shown in the table below.

Table 1: Merkle Source Term Model Tunable Parameters

Variable	Description	Default Value
`tauVap`	Vaporization rate constant	1
`tauCond`	Condensation rate constant	0.0125
`L`	Reference Length	None
`V`	Reference Velocity	None
`rho`	Reference density	None

The Merkle model can be selected by specifying `source=Merkle` in the `cavitationEquationOptions` variable as shown below.

```
cavitationEquationOptions:  <linearSolver=SGS, relaxationFactor=0.9, maxIterations=5,
source=Merkle>
```

The corresponding parameters for the model can then be set using the `MerkleSourceParameters` variable as follows.

```
MerkleSourceParameters:  <tauVap=1.0, tauCond=0.0125, L=0.7, V=1.5, rho=1.15>
```

5.1.2 Zwart Source Term Model

The functional form of the Zwart source term model [GeZB2004] is derived from the Rayleigh-Plesset equation which describes the growth or collapse of a vapor bubble in a pressurized fluid environment. Although more formally based on the physics of nucleation and vaporization, this model is none-the-less also ultimately heuristic in nature due to the introduction of constants that allow tuning of the vaporization and condensation rates for the problem of interest. The net vapor production rate is given by the following:

$$S(P, P_{vap}(T)) = \begin{cases} F_{vap} \frac{3 r_{nuc}(1-\alpha_V)\rho_V}{R_B} \sqrt{\frac{2}{3} \frac{P_{vap} - P}{\rho_L}}, & \text{if } P \leq P_{vap} \\ F_{cond} \frac{3 \alpha_V \rho_V}{R_B} \sqrt{\frac{2}{3} \frac{P - P_{vap}}{\rho_L}}, & \text{if } P > P_{vap} \end{cases}$$

where P is the local pressure, P_{vap} is the local vapor pressure, α_V is the local vapor mass fraction, ρ_V is the local vapor density, ρ_L is liquid density, R_B is the bubble nucleation radius, and r_{nuc} is the nucleation volume fraction. The table

below shows the default values for the user-tunable parameters for the model. Generally, unless specific knowledge is available regarding nucleation for a particular problem, one should only adjust the rate constants F_{vap} and F_{cond} to tune the model.

Table 2: Zwart Source Term Model Tunable Parameters

Variable	Description	Default Value
FVap	Vaporization rate constant	50
FCond	Condensation rate constant	0.01
RB	Bubble nucleation radius	1.0e-06
aNuc	Nucleation volume fraction	5.0e-04

5.1.3 Sauer-Schnerr Source Term Model

The Sauer-Schnerr source term model [ScSS2008] is based on the Rayleigh-Plesset bubble dynamics equation. The expression for the net vapor production rate is given by the following:

$$S(P, P_{vap}(T)) = \begin{cases} F_{vap}\frac{\rho_v \rho_L}{\rho_m}\alpha_V(1-\alpha_V)\frac{3}{R_B}\sqrt{\frac{2}{3}\frac{P-P_{vap}}{\rho_L}}, & \text{if } P < P_{vap} \\ -F_{\text{cond}}\frac{\rho_V \rho_L}{\rho_m}\alpha_V(1-\alpha_V)\frac{3}{R_B}\sqrt{\frac{2}{3}\frac{P_{vap}-P}{\rho_L}}, & \text{if } P > P_{vap} \end{cases}$$

Where ρ_L is the liquid density, ρ_V is the vapor density, ρ_m is the mixture density, α_V is the vapor mass fraction, and R_B is the bubble radius. The bubble radius has the following definition.

$$R_B = \left(\frac{\alpha_V}{1-\alpha_V}\frac{3}{4\pi n}\right)^{\frac{1}{3}}$$

Where n is the bubble number density, and often takes a value around 1e+08. The table below describes the user tunable parameter for the model.

Table 3: Sauer-Schnerr Source Term Model Tunable Parameters

Variable	Description	Default Value
rNuc	Nucleation radius	1.0e-5
n	Bubble number density	None

5.1.4 Source Term Scaling Factor

From the description of the source term models above, one can see that a specific functional relationship between the vapor production rate and the difference between the local flow pressure and the local vapor pressure is postulated. For the Merkle model a linear dependence is assumed while for the Zwart and Sauer-Schnerr models a square root dependence is assumed. For certain flow problems, the nature of the cavitation is such that neither functional dependence can provide an ample amount of vapor generation in certain regions without the pressure decreasing to values significantly below the vapor pressure and in certain cases becoming negative. This can result in the demise of the simulation. In practice, in any real device in which the flow is cavitating, the minimum pressure in the domain should remain in the vicinity of the local vapor pressure. In order that this can be achieved in numerical simulations without having to adjust the vaporization rate parameters to unrealistically high values (which would thus cause large amounts of vapor to be produced at all cells where $P < P_{vap}$), a source term scaling factor has been included in the formulation. This factor is only applied to locations in the domain where the pressure departs significantly from the local vapor pressure. The source term scaling factor $F(P, P_{min}, F_{max}, n)$ is defined by the following expression:

$$F(P, P_{\min}, F_{\max}, n) = \begin{cases} 1 + (F_{\max} - 1)\left[\frac{e^{-n(P)}}{\frac{P_{min}}{e^n - 1}} - 1\right], & \text{if } P < P_{\min} \\ 1, & \text{if } P > P_{\min} \end{cases}$$

The image below shows a plot of the function, while the table shows the default values the scaling factor parameters. Since the primary goal is to prevent the simulation from departing too far at any point from the local vapor pressure, the model should have its parameters set such that below the value $P = P_{min}$ the source term scaling factor ramps up rapidly. This is achieved typically with the default value of `n`.

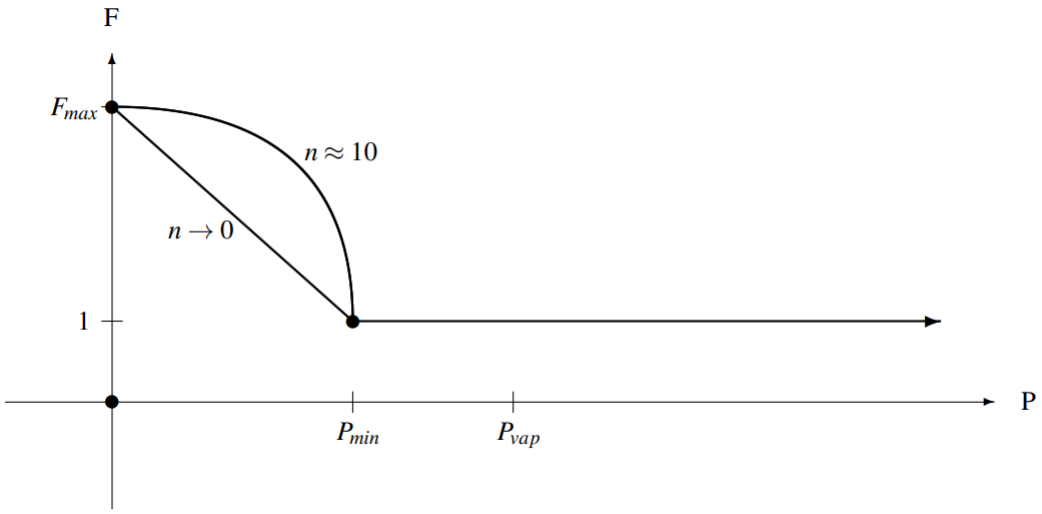

Fig. 1: Source term scaling factor for controlling minimum simulation pressure. Scaling becomes active at user specified value of P_{min}. User specified value F_{max} controls maximum scaling, while user specified exponent n adjusts the scaling profile.

Table 4: Source Term Scaling Parameters

Variable	Description	Default Value
`cavitationMaxSourceFactor`	Maximum scaling factor	1
`cavitationSourceFactorExponent`	Exponent	1
`cavitationMinPressure`	Scaling activation pressure	1

The maximum scaling factor F_{max} is usually problem dependent, however, the value in the table is a good starting point. The selection of P_{min} should be guided by actual experimental measurements on similar flow configurations if possible, however, if such information is not available, values in the range of no more than several hundred Pascals below the vapor pressure are recommended. In general, it is noted that the minimum simulation pressure will be somewhat below the specified value of P_{min} because this is merely the point at which the source term scaling is activated.

5.2 Control File Setup

Setting up a run control file for the simulation of a cavitating flow is like setting up a standard compressible flow problem. It is important to note that cavitating flows can only be simulated in compressible mode. A few important guidelines must be observed when setting up the run control file for a cavitation simulation:

- Boundary conditions must be set for the vapor mass fraction variable `vapor_y` and non-condensable gas mass fraction variable, `NCG_y`, on all inlet boundaries.
- Initial conditions must be set for `vapor_y` and `NCG_y`.
- The temperature form of the energy equation must be used by specifying `form=temperature` in the variable `energyEquationOptions`. The total enthalpy and total energy forms of the energy equation are not supported.

The specifications for `vapor_y` and `NCG_y` must always be provided. If there is no NCG in the flow, specify a value of 0. The following subsections detail each section of the run control file.

5.2.1 Boundary Conditions

Because cavitating flows must be simulated using compressible mode, the only inlet boundary conditions that should be in use are `subsonicInlet`, `supersonicInlet`, and `inflow`. The following shows an example of the specification of the `vapor_y` and `NCG_y` boundary condition for a subsonic inlet with pure liquid with a small amount of NCG:

```
Inlet=subsonicInlet(v=10.11 m/s, T=300.0 K, k=0.0, omega=1000.0, vapor_y=0.0, NCG_y=1.
0e-06)
```

Similar specification would be made for the other inlet types.

5.2.2 Initial Conditions

The initial condition for the vapor and NCG mass fractions are set as follows:

```
initialCondition:  <vapor_y=0, NCG_y=0.0, v=10.11 m/s, p=28515.8 Pa, T=300.0 K, k=0.0,
omega=1000>
```

Similar specification of these values can be made using the run control file variables `ql` and `qr` or the variable `initialConditionRegions` as discussed in the following *Appendix*.

5.2.3 Equation of State and Transport Properties

The cavitation_nist module utilizes tabulated thermophysical property data from the NIST *REFPROP* software. A python utility is provided with the *Stream* source code that can generate tabulations for liquid, vapor, and saturation data files for any *REFPROP* species. The utility will generate three output files with suffixes of `.liq`, `.vap`, and `.sat`. The prefixes for the liquid and vapor files must be provided using the `liquid_model` and `vapor_model` variables in the run control file. The prefix for the saturation file will be taken from the `liquid_model` variable. The equation-of-state section in the run control file should look as follows:

```
liquid_model: NITROGEN
vapor_model: NITROGEN
transport_model: module
```

Setting the value of the `transport_model` to `module` indicates that all transport data is to be obtained from data in the tabulated liquid, vapor, and saturation files.

5.2.4 Cavitation Equation Options

An example of the cavitation equation section of the run control file with all available options is shown below:

```
cavitationEquationOptions: <linearSolver=SGS, relaxationFactor=0.9, maxIterations=5,
                            source=Zwart, vaporProductionRateRelaxationFactor=0.7>

ZwartSourceParameters: <RB=1.0e-06, aNuc=5.0e-04, fVap=50.0, fCond=0.01>
cavitationInviscidFlux: SOU
AlphaViscosity: 1.0e-05
liquid_rho: 736
vapor_rho: 17
```

(continues on next page)

(continued from previous page)

```
NCG_rho: 15
cavitation: on
cavitationSourceScaling: <Pmin=3200, Fmax=100, n=10>
turbulentVapourPressureCorrection: on
```

The `.vars` file variables used are summarized in the table below.

Table 5: Some Cavitation Module Options

Parameter	Description
`AlphaViscosity`	Artificial viscosity added to vapor and NCG mass fraction transport equations for numerical stability purposes.
`cavitationInviscidFlux`	Flux scheme used for vapor and NCG transport equations. Supported values are: `FOU` and `SOU`.
`liquid_rho`	Reference value for liquid-phase density used in normalizing the liquid term in the pressure-correction equation.
`vapor_rho`	Reference value for vapor-phase density used in normalizing the vapor term in the pressure-correction equation.
`NCG_rho`	Reference value for non-condensable-gas density used in normalizing the NCG term in the pressure-correction equation.
`turbulentVapourPressureC`	Activates the turbulent correction to the vapor pressure.

The linear solver options are like the other governing equations that have been previously covered. All linear solvers discussed (*here*) are supported for the value of `linearSolver`, however one should rarely need to use anything other than the `SGS` solver. Source term model selection is made using either `source=Merkle`, `source=SauerSchnerr`, or `source=Zwart`. Default parameters for these three models are shown below:

```
MerkleSourceParameters: <tauVap=1.0, tauCond=0.0125>
ZwartSourceParameters:<RB=1.0e-06, aNuc=5.0e-04, fVap=50.0, fCond=0.01>
SauerSchnerrSourceParameters: <rNuc=3.0e-06, n=4.8e8>
```

To explicitly turn off the source terms in the vapor mass fraction equation, one can specify `cavitation: off` in the run control file. The default value for this variable is `on`. Parameters for the source scaling model can be specified using the variable `cavitationSourceScaling`. No source term scaling will be performed is this variable is not present in the run control file. If the `liquid_rho`, `vapor_rho`, and `NCG_rho` variables are not included (or are set to -1), the code will use local cell values to normalize the pressure-correction equation at each cell.

5.2.5 Non-Condensable Gas Specification

All cavitation simulations involve NCG terms in the governing equations, even if the user does not use NCG. The default fluid properties used for the `nonCondensableGasProperties` variable is air as shown below. To use a different fluid, the user must specify its molar mass `m`, specific heat at constant pressure `cp`, laminar dynamic viscosity `mu`, and thermal conductivity `kcond` in the `nonCondensableGasProperties` variable.

```
nonCondensableGasProperties: <m=28.9647 g/mol, cp=1005 J/kg/K, mu=1.805e-5 kg/m/s,
                              kcond=0.02476 W/m/K>
```

5.2.6 Output Variables

The table below shows additional field variables that are available for output in cavitating flow simulations. Default variables are output automatically and do not need to be specified in the plot_output line in the run control file.

Table 6: Field Variable Output for Cavitating Flows.

Variable	Description	Default
`bt``	Bulk expansion coefficient * temperature	No
`F`	Source term scaling factor	No
`liquid_vof`	Liquid volume fraction	No
`liquid_y`	Liquid mass fraction	No
`NCG_vof`	NCG volume fraction	No
`NCG_y`	NCG mass fraction	No
`NCG_yResidual`	NCG mass fraction eq. residual	No
`pV``	Vapor pressure	No
`rhoL`	Liquid density	No
`rhoV`	Vapor density	No
`vapor_y`	Vapor mass fraction	No
`vapor_vof`	Vapor volume fraction	No
`vapor_yResidual`	Vapor mass fraction eq. residual	No

5.3 REFPROP Tabulation Utility

The Python *REFPROP* tabulation utility is located under the `cavitation_nist/utilities` directory in the source code. The name of the utility is `cav_tab_tool.py`. This tool is for building necessary liquid, vapor, and saturation input files.

The Python tabulation utility has a functionality for making calls directly to REFPROP 11. This allows for the tool to tabulate as many pressure levels as the user desires. The user specifies the REFPROP material name, the pressure range, temperature range, and the number of data points to use over each range. The user can also pass a flag (`--log_p`) to uniformly distribute the pressure points in the log-space to ensure that an equal number of points resolve each pressure decade (this is good for large pressure ranges).

A set of `.liq`, `.vap`, and `.sat` files are generated by the tool, and these are the inputs used by the `cavitation_nist` module in *Stream*. Sample calls to the utility for the liquid/vapor files and the saturation files are given below.

There are two modes that the tool can be used in. One is for tabulating properties outside of a two-phase region in the thermodynamic space. The other is for tabulating the saturation curve properties of the liquid and vapor states.

5.3.1 Liquid/Vapor Tabulation

To generate tabulated liquid and vapor files, one should pass the liquidVapor argument to the tabulation tool. The presence of this argument enables a required set of arguments which are described below:

Table 7: Liquid/Vapor Tabulation Arguments

Argument	Description
t_min	The minimum temperature range value
t_max	The maximum temperature range value
num_t	The number of temperature levels
p_min	The minimum pressure range value
p_max	The maximum pressure range value
num_p	The number of pressure levels
log_p	An optional argument that distributes the pressure levels uniformly across the logarithmic space of the pressure range

If the `log_p` argument is not provided, a linear tabulation in the pressure dimension is performed. The logarithmic spacing enabled by the `log_p` argument creates a uniform spacing in the logarithmic space. This type of spacing is useful when there is a large variation in the pressure range, as a linear tabulation would result in a low resolution in the low-pressure region of the tabulation. An example command to generate liquid and vapor tabulation files for nitrogen in the temperature range of 65K-300K and pressure range of 50kPa-70MPa is shown below.

```
python cav_tab_tool.py liquidVapor --fluid_name NITROGEN --t_min 65
--t_max 300 --p_min 50000 --p_max 70e6 --log_p --num_p 200 --num_t 200
```

In the example above, the `liquidVapor` argument sets the tool into the mode for tabulating `.liq` and `.vap` tables. The fluid_name argument must match an available REFPROP `FLD` file in the REFPROP fluid database. The tabulation levels are done at constant pressure, and so the `t_min` and `t_max` arguments are the temperature range over which to tabulate properties. The `p_min` and `p_max` arguments are the inclusive bounds of the pressure range over which you want to tabulate data. The `log_p` argument converts the pressure range into a logarithmic range and distributes points equally in that space and then converts back to pressure levels. This ensures an equal number of pressure levels per decade of pressure. This is sometimes a desirable option. The `num_p` and `num_t` arguments are the number of pressure levels to tabulate and the number of temperature levels to tabulate.

5.3.2 Saturation Tabulation

To generate a tabulated saturation file, one should pass the saturation argument to the tabulation tool. The presence of this argument enables a required set of arguments which are described below:

Table 8: Liquid/Vapor Tabulation Arguments

Argument	Description
`t_min`	The minimum temperature range value
`t_max`	The maximum temperature range value
`num_t`	The number of temperature levels

An example command to generate a saturation tabulation file for nitrogen in the temperature range of 65K-300K with 150 tabulation points is shown below.

```
python cav_tab_tool.py saturation --fluid_name NITROGEN --t_min 65 --t_max 300 --num_t␣
↪150
```

In the example above, the saturation argument activates the saturation curve tabulation mode of the script. This mode has fewer arguments than the `liquidVapor` mode. The temperature range and number of points to include are the only

arguments that are necessary. These arguments have the same meanings as they do for the `liquidVapor` mode of the tabulation tool that was discussed earlier.

5.3.3 Using Cavitation Tabulation Tool

One must have version 11 of the *REFPROP* software to use the cavitation tabulation tool. A short tutorial on how to install the dependencies for the cavitation tabulation tool is presented below.

1. Download the *REFPROP* 11 software using the NIST downloader. This must be done on a Windows machine. The software will be downloaded into a directory named REFPROP.
2. Copy the REFPROP directory over to a Linux machine. The software can be put in a directory such as `/home/<User>/software/refprop`, where `<User>` would be your username on the machine). **Note**: do not rename the REFPROP source code directory because that may cause issues. For example, the path should look as follows: `/home/<User>/software/refprop/REFPROP`
3. Compile a local version of REFPROP on your Linux machine
    a. Do a recursive clone of the following repository: `git clone --recursive https://github.com/usnistgov/REFPROP-cmake.git`
    b. Go to the root of the cloned repository: `cd REFPROP-cmake`
    c. Create a build directory: `mkdir build`
    d. Change to the build directory: `cd build`
    e. Run cmake to configure the build: `cmake .. -DCMAKE_BUILD_TYPE=Release -DREFPROP_FORTRAN_PATH=/home/<User>/software/refprop/REFPROP/FORTRAN`
    f. Build the project: `cmake --build .`
4. Once the build is complete, a `librefprop.so` file should be found in the build directory. This shared library contains all the *REFPROP* functions that the Python wrapper will call. Move this file to the *REFPROP* directory created in step 2.
    a. Move the shared library file: `mv librefprop.so /home/<User>/software/refprop/REFPROP`
5. Set the environment variable `RPPREFIX` to point to the *REFPROP* directory that contains the shared library file.
    a. Add the following line to your .bashrc file: `export RPPREFIX=/home/<User>/software/refprop/REFPROP`.
    b. Make sure to source your .bashrc file after adding this line: `source ~/.bashrc`.
6. Clone the *REFPROP* wrappers directory into the `/home/<User>/software/refprop` directory using the following command: `git clone https://github.com/usnistgov/REFPROP-wrappers`
7. Create a *Python3* environment with the *REFPROP* interface installed. This will provide a portable and dependable Python environment for running the tabulation tool. Go to your `refprop` directory and follow the steps below:
    a. Create a virtual environment: `python3 -m venv refprop-env`
    b. Activate the environment: `source refprop-env/bin/activate`
    c. Install numpy: `pip install numpy`
    d. Navigate to the *REFPROP* wrappers directory: `cd /home/<User>/software/refprop/REFPROP-wrappers/wrappers/python/ctypes`
    e. Install the python wrapper into the environment: `python setup.py install`
    f. To leave the virtual environment type: `deactivate`

One you are in the python environment, you can run the tabulation script and it should execute using calls to the REFPROP library.

References

CHAPTER

SIX

SIMULATION OF COMBUSTING FLOWS WITH FLAMELET METHOD

Stream supports solving combustion problems via using flamelet tables that are created from solving flamelet equations.

6.1 Flamelet Method

The flamelet method tries to balance computational speed and accuracy by pre-tabulating reaction data for a class of representative combustion model problems that can accumulated to build a database that can be queried to quickly provide information such as heat release and species production rates. A limitation of this method is that it is not accurate for problems that do not lend themselves to being easily represented by a series of flamelet solutions. The flamelet generation process and the theory behind the flamelet equations is documented in a separate guide. A much more detailed introduction to flamelet modeling can be found in Peters' book Turbulent Combustion [Pete2000]. Diffusion flamelet tables are heavily used with in the flamelet model due to their applicability for injector combustion problems.

To use the compressible flamelet module, the run control file should appear as follows:

```
loadModule: compressible_flamelet_tf
{
... standard vars file content
}
```

All flamelet simulations are parameterized by at least two variables: the mixture fraction(Z) and the progress variable(C). The mixture fraction is a variable that represents the state in the fuel and oxidizer streams where a value of Z=0 in the oxidizer stream and Z=1 in the fuel stream. The progress variable is usually defined as the summation of a series of combustion product mass fractions (e.g., C= YH20 + YCO2), and as such is used as a measure of the completeness of a reaction. A value of zero would represent a state where there are no products of combustion and a value of 1 would be a state where there are only products of combustion. Practically, C rarely will go to 1 as there is often a limit to the combustion process turning all the reactants into products for engineering applications.

6.2 Control File Setup

A few important guidelines must be observed when setting up the run control file for a flamelet combustion simulation:

- Boundary conditions must be set for the mixture fraction variable `Z` and the progress variable `C` on all inlet boundaries.
- Initial conditions must be set for `Z` and `C`.

The following subsections detail the specifics of each section of the vars file required for flamelet simulations.

6.2.1 Boundary Conditions

The following shows an example of the specification of the flamelet boundary condition for a subsonic inlet for an oxidizer stream with an inert species present in the stream:

```
Oxidizer_Inlet=subsonicInlet(mdot=2.51243E-04 kg/s, T=700.0 K, Z=0.0, C=0.0551,
                             k=4.2, omega=12900.0)
```

This specification has an oxidizer inlet with an inert species present in the oxidizer stream (not pure O2), and a definition of the progress variable that includes the species that is mixed with the oxidizer; therefore, the value of C is non-zero for this case. For cases with totally pure oxidizer and fuel streams, the value of C will be zero. A specification of an inlet boundary condition for a pure fuel stream is shown below:

```
Fuel_Inlet=subsonicInlet(mdot=9.1978E-05 kg/s, T=811.0 K, Z=1.0, C=0.0,
                         k=0.5, omega=37950.0)
```

6.2.2 Initial Conditions

The initial conditions for Z and C are set as follows:

```
initialCondition: <v=0.0 m/s, p=5.4e+06 Pa, T=3174.0 K, Z=0.34, C=0.912,
                   k=0.5, omega=37950.0>
```

Similar specification of Z and C can be made using the run control file variables `ql` and `qr` or the variable `initialConditionRegions`.

6.2.3 Transport Properties

For flamelet calculations, all transport properties are obtained from the flamelet table. Specify the `transport_model` as module like below.

```
transport_model:  module
```

6.2.4 Flamelet Equation Options

The flamelet equation section of the run control file with all available options appears as follows:

```
flameletEquationOptions: <linearSolver=SGS, relaxationFactor=0.5, maxIterations=5>
combustion: on
flameletInviscidFlux: SOU
```

6.2.5 Output Variables

The table below shows additional field variables that are available for output in flamelet simulations. Default variables are output automatically and do not need to be specified in the `plot_output` line in the run control file.

Table 1: Field Variable Output for Combusting Flamelet Flows

Variable	Description	Default
ZZ	Mixture fraction	No
C	Mixture progress variable	No
ZResidual	Mixture fraction eq. residual	No
CResidual	Mixture progress variable eq. residual	No

References

CHAPTER

SEVEN

SIMULATION WITH OVERSET GRIDS

Stream can use overset grids for simulating cases that have complex geometries or require mesh movements. This capability is enabled by inserting the following loadModule directive into the run control file as shown below.

```
loadModule: overset
{
  ... standard run control file content
}
```

The following sections discuss the overset method used in *Stream* and provide information on the run control file variables required to perform simulations with the overset module.

7.1 Overset Method

Stream uses a standard overset method where a background mesh has component meshes overlayed on it, and appropriate hole-cutting and blanking of cells is performed. At the interfaces of the components, a buffer region of cells exists to provide an implicit interpolation between the overlapping meshes. Some definitions that are helpful when working with the overset module are:

Table 1: Definitions of terms used in the overset method.

Term	Description
Blanked cells	Cells that have been assigned a flag that controls how the cells are to be used in the context of the overset method (the nomenclature for these markers is `iblank=1`, `iblank=2`, etc).
Hole cutting	The removal of irrelevant or unused cells from an overset computation
Component geometry	The rough geometric specification (provided in the control file) of a component mesh's shape that is used to remove background mesh cells that "inside" the component mesh's empty area.
Interface boundary	The outermost boundary of a mesh.
iblank=0 State	Cells where the governing equations are solved normally. Values are donated to interpolation.
iblank=1 State	Cells where the governing equations are solved normally. Values are **not** donated to interpolation.
iblank=2 State	Cells of this type are not computed (no equations are solved within those cells), they receive their values by interpolating from a cloud of points that comes from a set of cells with `iblank=0` (which are from the background mesh).
iblank=3 State	Cells are not computed. Their current value is simply advanced with time.

The `iblank=3` cells are often ones that lie outside the background mesh, or background cells that are deep within a component mesh beyond the interpolation layer. The `iblank=1` state is not used often during the overset simulations and is included for possible future uses. The most relevant states are the `iblank=0,2,3` states.

The overset method marks **roughly three layers** of cells inwards from the interface boundary on a component geometry's mesh with the `iblank=2` state. This means that the values in that buffer layer of cells on the component geometry's mesh obtain their values from an interpolation from the cloud of points on the background mesh. For example, the pressure correction equation is solved on the `iblank=0` cells of the component mesh, which are adjacent to the `iblank=2` cells. When an `iblank=0` cell requires information about its `iblank=2` neighbor, a list of cells that are used to construct the neighbor's value, and the corresponding interpolation weights from each of those cells is collected. These stencil weights are directly inserted into the matrix equation for the pressure correction equation to provide implicit coupling between a component mesh and the background mesh.

7.2 Overview of the method

What follows in this section is a more detailed review the overset framework. Start by examining the figure below. Here the gray cells are `iblank=0`, the green cells are `iblank=2`, and the red cells are `iblank=3`. The right element in the image is the component mesh, and a solid component boundary is shown on the far right side of the element. For this simple illustration the two grids overlap in a 1D manner. **Note**: The image shown is as if the two meshes were overlapping, but retaining the coloring scheme. They are shown separated in this figure first to help the reader understand the marking and coloring scheme.

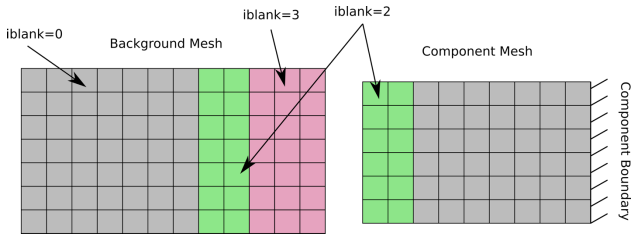

Fig. 1: A background and component mesh that are separated. The coloring shows the `iblank` state as if they were on top of each other (as will be shown in later figures).

The figure below shows the two grids overlaid properly. The boundary of the component mesh has its outer layer marked as `iblank=2`, which draws from the `iblank=0` background cells. The background mesh cells close to the component geometry's solid surface are set to `iblank=3`, and a set of buffer layer cells on the background mesh around the `iblank=3` cells are set to `iblank=2`, which they obtain their values from the `iblank=0` cells on the component mesh.

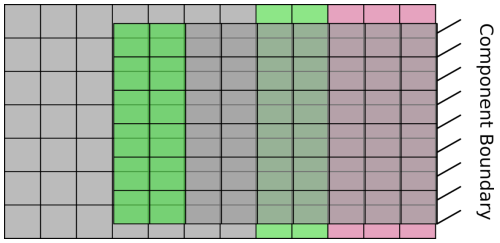

Fig. 2: Both background and component meshes over each other. The iblank colors are consistent with the grid positioning.

The figure below shows an example case of how the implicit treatment is performed for the pressure correction equation. In this case, consider the cell A, which is located on the component mesh. It is gray, so it has a `iblank=0` state, and the governing equations are solved on that cell. It requires information from its neighbor cells. The data about the cell to its right is easily obtained. The cell to its left (cell B) however is a green `iblank=2` cell. So that cell's data is coming

from an interpolation that uses data from the gray background `iblank=0` cells around it. The background cell centers are shown and numbered. The arrows point to the stencil of cells that hypothetically go into the interpolation for cell B. To construct the matrix equation for the pressure equation for cell A, one needs coefficient information from cells that contribute to the solution of cell A. This is how the equation $[A][P'] = [b]$ is constructed. The difference here is that for cell A, the neighbor value of cell B is not used directly. Cell B's data is a weighted sum of the values of the pressure correction of cell 1,2,3,4,5,6 in the diagram. These weights are directly inserted into the coefficient matrix for the pressure correction equation such that the equation for cell A contains entries from its right neighbor, and cells 1,2,3,4,5,6. This is then the equation that is solved.

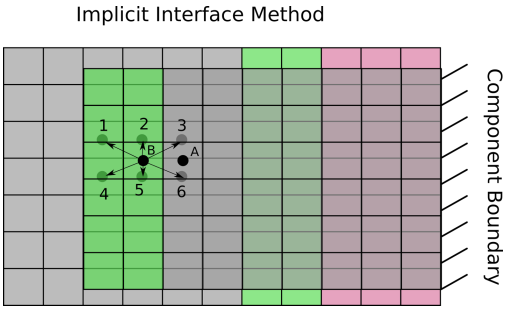

Fig. 3: An example showing how the information for data from an `iblank=2` cell is obtained from the cloud of points.

Pressure contours for a 2D plug/pipe case can be seen in the image below. Continuity of the solution across the overset mesh interface can be observed using the implicit method in this flow where a high pressure on left side of the domain flows over a plug and towards the right.

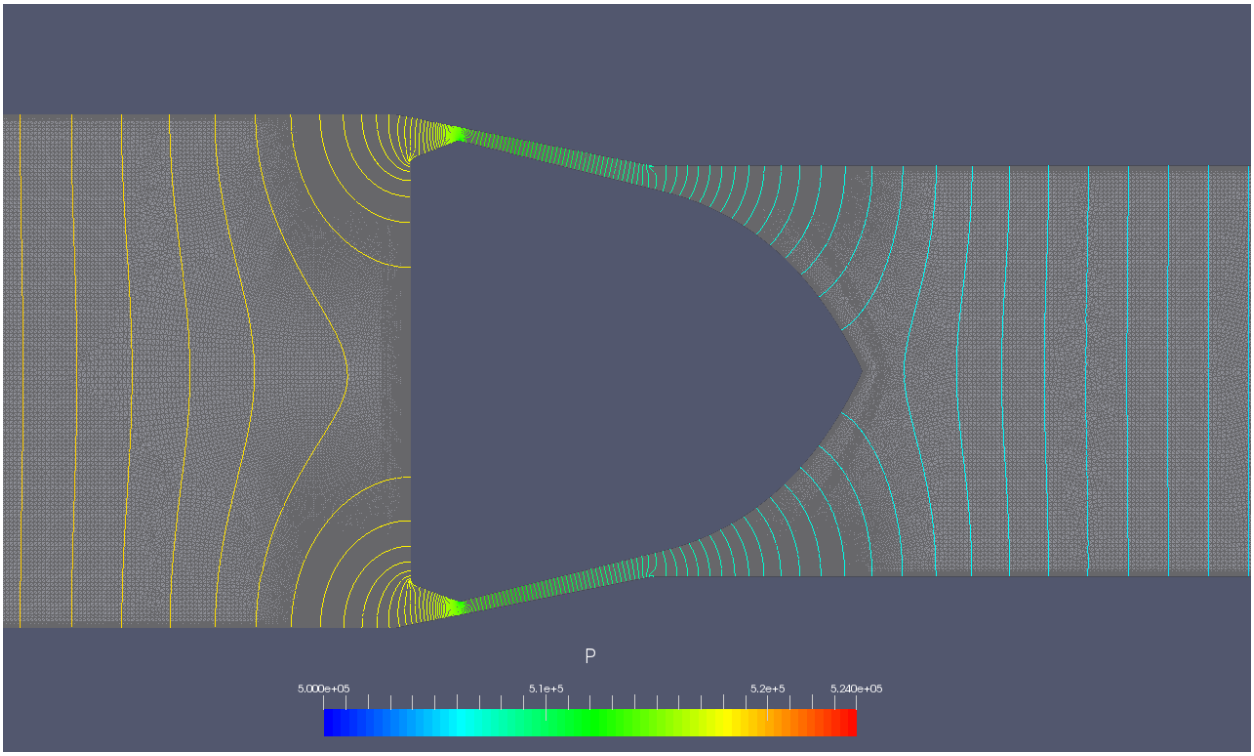

Fig. 4: Pressure contours across the overset 2D sideways plug example case. Contours vary smoothly across the overset interface.

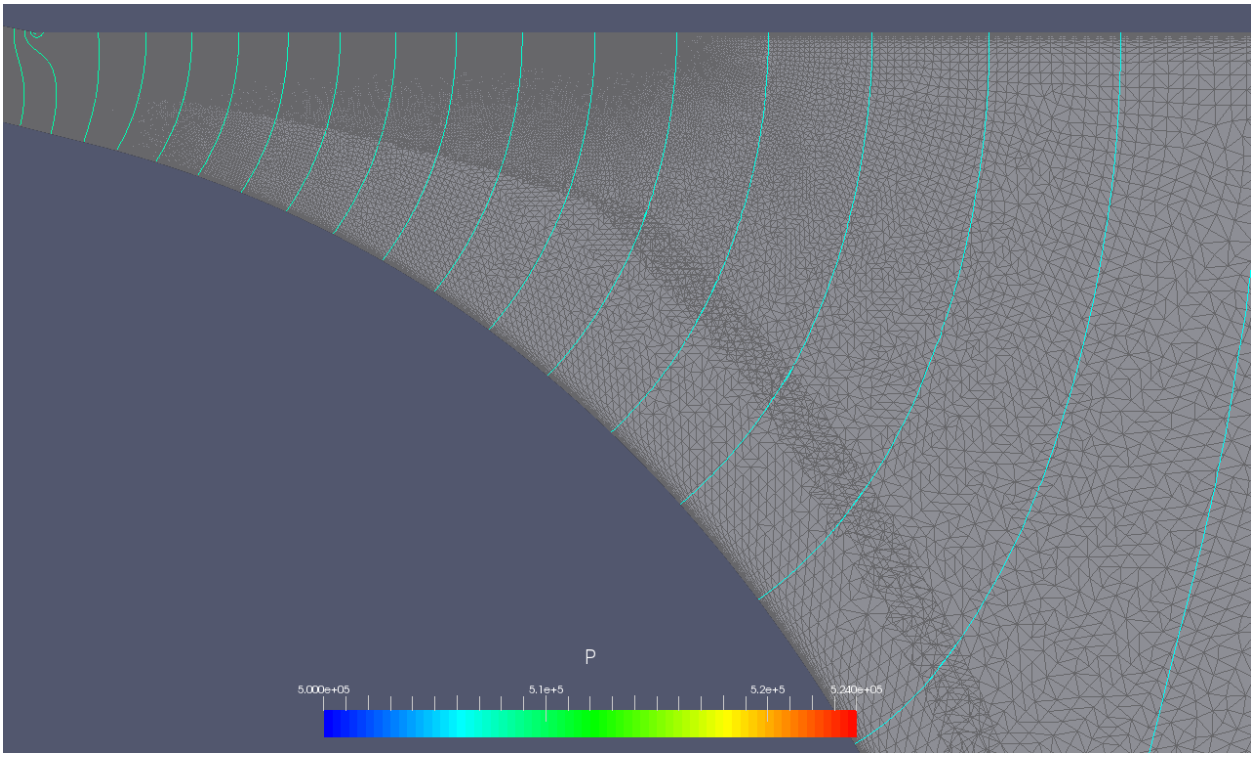

Fig. 5: Pressure contours across the implicitly handled overset grid interface.

7.3 Control File Setup

Only a few additions need to be made to an existing run control file to set up an overset simulation. One thing to be aware of when setting up for an overset simulation is the specification of the `componentGeometry` variable as the overset algorithms depend on this variable being properly specified. The following subsections detail each section of the run control file.

7.3.1 Boundary Conditions

The boundary conditions for all the surfaces of both the background mesh and component meshes need to be specified in the `boundary_conditions` section of the run control file. The external mesh boundary of the component mesh needs to have an `interface` boundary specification as shown below for example.

```
overset_boundary=interface
```

In the above example the boundary name on the left of the = is whatever boundary name you have chosen for the mesh external boundary. No other adjustments need to be made to the boundary condition specifications section. Specify the other boundary conditions as you would for a non-overset simulation.

7.3.2 Initial Conditions

No changes to the initial conditions need to be made to accommodate an overset simulation.

7.3.3 Component Geometry

An important run control file specification that is required for overset simulations is the `componentGeometry` variable. This variable defines a region of the domain that should contain the inner volume of a component's geometry. It does not have to be exact, but a close approximation will help the overset algorithm to properly mark and assign cells. The figure below shows an example diagram of an overset mesh configuration with the most common features named with the nomenclature used in this manual.

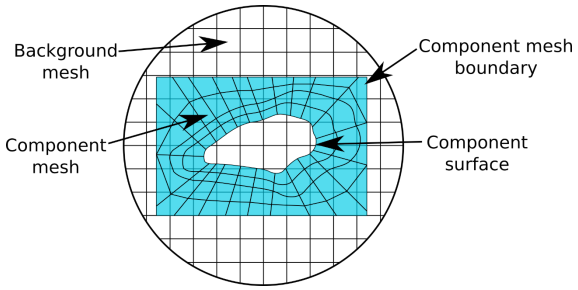

Fig. 6: Diagram showing the important features of an overset simulation and the names used to refer to these features typically.

In keeping with the example shown above, the specification of a rectangular `componentGeometry` is shown the figure below. This rectangular region is what the overset algorithm uses to determine which background mesh cells should be marked with the non-participating `iblank=3` state.

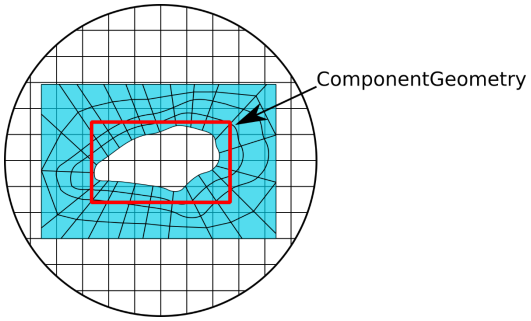

Fig. 7: This is an example showing a rectangular `componentGeometry` specified to approximate the complex shape contained within it. The red rectangular region is what the overset algorithm uses to determine which background mesh cells to mark with the `iblank=3` state.

An example of specifying the `componentGeometry` variable in the run control file is shown below.

```
componentGeometry: <body1=cylinder(p1=[-1, 0, 0], p2=[1, 0, 0], radius=3),
body2= sphere(center=[2, 2, 0], radius=1.5),
body3=revolution(p1=[5, 1, 1], p2=[7, 1, 1], radius=[2, 4, 3],
offsets=[0, 0.62, 1.0])>
```

Component geometries can be specified in terms of cylinders, spheres, bodies of revolution, or a list of planes. The table below contains examples of how to specify a component geometry of each type.

Table 2: Types of primitives for defining a `componentGeometry` specification.

Variable	Description
cylinder(p1,p2,radi	`p1` and `p2` define two end points of the cylinder axis and radius specifies the cylinder radius.
sphere(center,	`center` and `radius` define the sphere's center and radius.
revolution(p1,p2,rad	Defines a body of revolution about an axis defined by points `p1` and `p2`. `radius` specifies a list of radii along the axis between `p1` and `p2`. `offsets` provides a list of the relative distance between `p1` and `p2` that the associated radii corresponds to.
planeList(list=	Defines a region that is on the left side of all the planes in the list. A plane is defined by a point, `p`, and a normal vector, `n`. The normal vector points out of the volume.

Examples of the component geometry forms described in the table above are show below.

A cylinder with an x-axis orientation from x=1 to x=2 with a radius of 0.5:

```
cylinder(p1=[1,0,0], p2=[2,0,0], radius=0.5)
```

A sphere centered at the origin with a radius of 1.5:

```
sphere(center=[0,0,0], radius=1.5)
```

A body of revolution with an x-axis orientation from x=1 to x=4.

```
revolution(p1=[1,0,0], p2=[4,0,0], radius=[3,2,4], offsets[0.0,0.6,1.0])
```

The body of revolution specification can be difficult to understand, so a diagram showing how each argument corresponds to a practical example is shown in below.

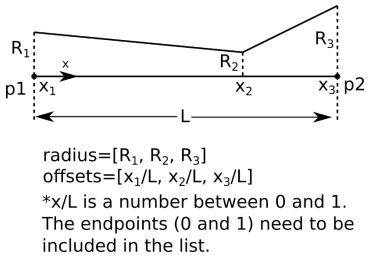

Fig. 8: Body of revolution diagram showing the important quantities that must be specified in the `componentGeometry` entry.

A cube of edge length 1 centered at the origin:

```
planeList(list=[
   //x-coordinate planes
   plane(p=[1.0,0,0],n=[1,0,0]),plane(p=[-1.0,0,0],n=[-1,0,0]),
   // y-coordinate planes
   plane(p=[0,1.0,0],n=[0,1,0]),plane(p=[0,-1.0,0],n=[0,-1,0]),
   // z coordinate planes
   plane(p=[0,0,1.0],n=[0,0,1]),plane(p=[0,0,-1.0],n=[0,0,-1])
])
```

7.3.4 Component Motion

The `overset` module supports grid motions of the overset grids. This grid motion is not restricted solely to overset simulations, but this section is included here because often overset and grid motion are used together. The run control file entry for using moving grids is shown below for a case of a background and single component.

```
componentMotion:
<
  moving_part=prescribed,
  background_mesh=stationary
>
```

Three options are available to be assigned to each mesh tag in the run control file. The `prescribed` assignment means that the motion will be provided by an external input file. A rotation assignment can be given of the type `rotation(axis=[1,0,0],center=[0,0,0],speed=400 rpm)`. The non-moving mesh should have the `stationary` assignment.

Any component mesh that is given the `prescribed` keyword in the run control file needs to have a corresponding file named `motion_<meshTagName>.dat` present at the same location at the run control

The elements of the motion file are described in the table below.

Field	Description
Number of interpolants	Number of interpolant inputs in the motion file
Time	Time in seconds
Position	Position in meters (3 components: x, y, z)
Quaternion	Four components of a normalized quaternion describing rotations about the preceding position

A cubic spline is used to interpolate between the given values. A discontinuity in the spline can be created if the condition for the same time is repeated twice.

A sample motion file is shown below. This one specified a translation in the x-direction of 63.5 mm in a span of 6 milliseconds. Note that the final digit of the quaternion should be 1.0 unless you are familiar with quaternions.

```
3
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
0.006 0.0 0.0635 0.0 0.0 0.0 0.0 1.0
10 0.0 0.0635 0.0 0.0 0.0 0.0 1.0
```

7.3.5 Output Variables

- The table below shows additional field variables that are available for output in overset flow simulations. Default variables are output automatically and do not need to be specified in the `plot_output` line in the run control file.

Table 3: Field Variable Output for Overset Simulations

Variable	Description	Default
	No current outputs for overset module supported	

7.4 Creating the Overset VOG Mesh with Tags

Overset meshes have multiple distinct meshes, but a single `vog` formatted mesh file is still required when using the overset module. This means that all the grids that compose the background and component meshes must be encoded into a single `vog` mesh file. Consider a case where a background mesh called `backgroundMesh.vog` and a component mesh called `componentMesh.vog` are to be used for an overset simulation. The following *Loci* utility is used to merge these two meshes into a single vog mesh named `mergedMesh.vog` with tags assigned to each mesh.

```
vogmerge -g backgroundMesh.vog -tag background -g componentMesh.vog-tag component -o
mergedMesh.vog
```

The tags are important because they are what is used to reference the difference meshes when describing the different components in the run control file. The tag names are arbitrary.

7.5 Known Issue with Overset Module & Hole Cutting

While using the `overset` module you may notice a situation in your domain where there are gaps in the domain: cells missing like what is shown in the figure below. This is very often an effect of an interaction in the *Loci* framework between the `componentGeometry` specification provided in a run control file and the mesh of a component. The hole cutting (removal of cells) is done to save computational resources because there is no reason to simulate cells in the background mesh that would be wholly contained within body of a component mesh. The `componentGeometry` specification is what informs the overset algorithm about which cells are potentially within the body of a component mesh's geometry, and it is this specification which often leads to situations like what is shown below.

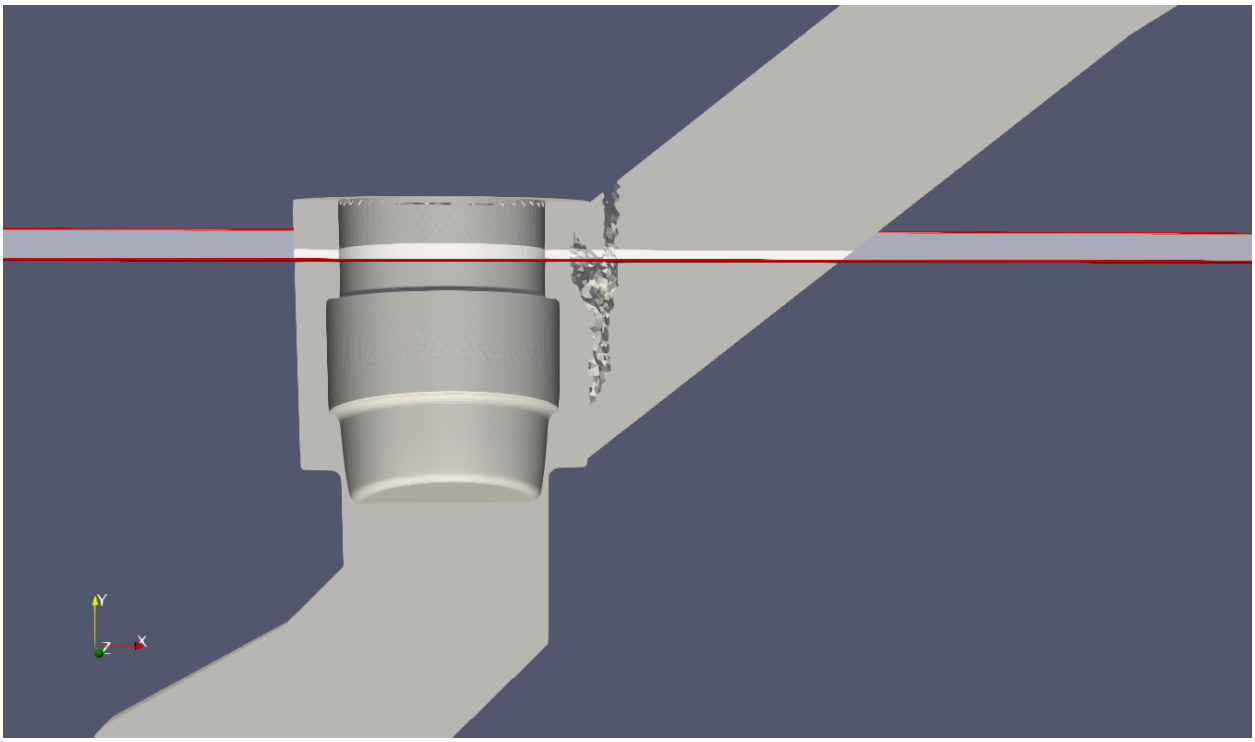

Fig. 9: A y-plane cute that is made through a domain to illustrate the hole cutting issue.

The region with the missing cells can be seen better in the image shown below, which is a top view of the plane cut shown above. In this figure, the component geometry was set to a large diameter (seen as the gap on the far-right side) for illustrative purposes. The gap seen on the left semicircular region is where the issue being discussed occurs.

Fig. 10: Overset hole cutting leaving a gap over on the left between the plug and valve. The hole on the right was expected to be there due to the component geometry being set to that location.

The blanking of the cells close to the interface boundary of a component mesh reduces the amount of usable grid near the overlap region and this interacts with the *Loci* framework's medial cut procedure which causes there to be insufficient overlap to fully connect the grid system. For this illustration, if the radius of the `componentGeometry` cylinder in this example is reduced from 0.056m to 0.047m, then there is sufficient overlap to prevent the gap. **The user should err on the side of greater overlap between the component mesh's interface boundary and componentGeometry specified surface.** This can be done in this case either by slightly increasing the grid resolution in the overlap region to allow more cells to overlap, or by keeping the same resolution and pushing the `componentGeometry` further away from the interface as changing radius from 0.056m to 0.047m does in this example.

Another alternative approach to alleviate this issue is to simply **extend the actual spatial extent of the component mesh to extend completely outside of the background mesh.** This is more costly than the example above but may be an effective and fast solution for cases where adjusting the component geometry specification continues to cause issues with the gaps in the mesh.

7.6 Visualizing the iblank state of a simulation

A `vars` file option `plot_overset_freq` is available. This operates on the same principle as the `plot_output` vars file variable. If this variable is included, the output will be `.csv` files. There will be one file at each timestep output for each `iblank` state.

A helpful thing to keep in mind is that the `iblank=3` cells are often ones outside of the domain, or within a component's geometry. For the purposes of illustration, we will consider the case of a simple valve and plug configuration as shown below. The valve is the orange and is the background mesh, and the plug is the blue and is the component geometry.

Any overset simulation can be visualized using *Paraview*. **Note: This visualization is for determining the state of the cells of the overset meshes in the event any debugging may be necessary. The standard extract utility works**

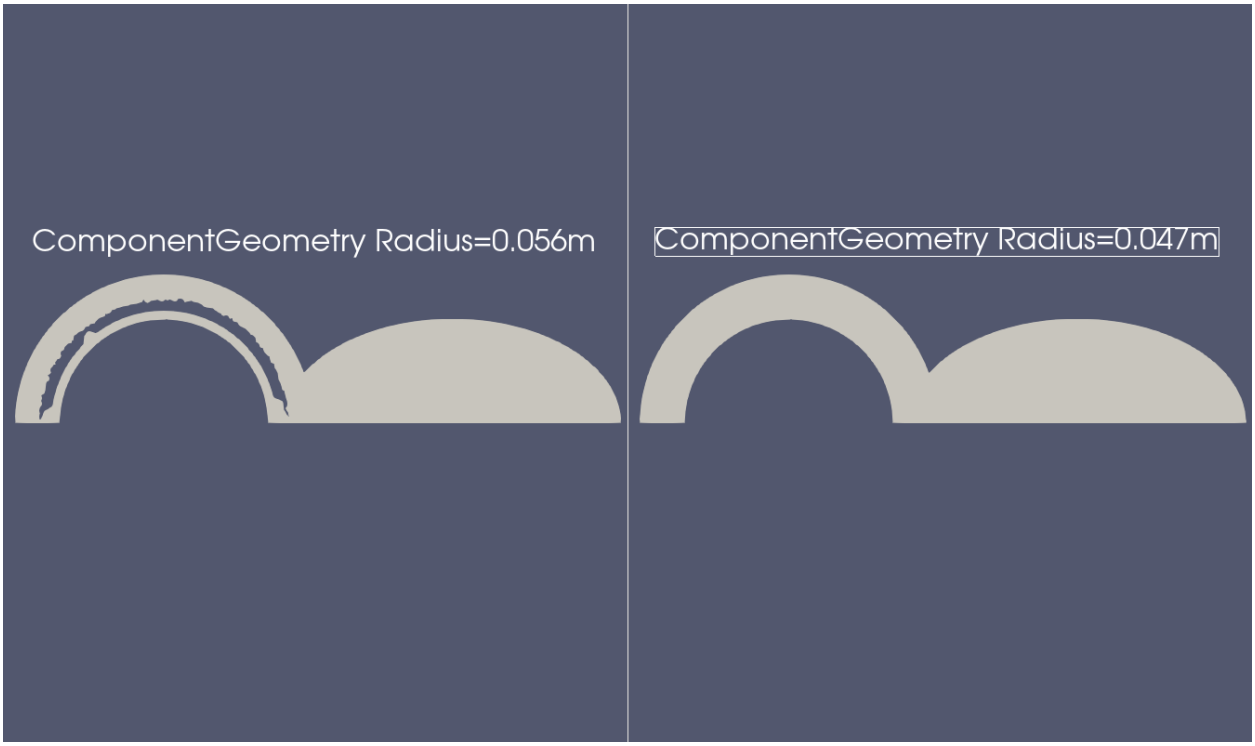

Fig. 11: The left cut-plane shows the effect of setting the componentGeometry surface too close to the location of the boundary of the mesh for the component. On the right, the componentGeometry boundary definition was moved inwards, which allowed for more of an overlap region between the background mesh cells and the component mesh.

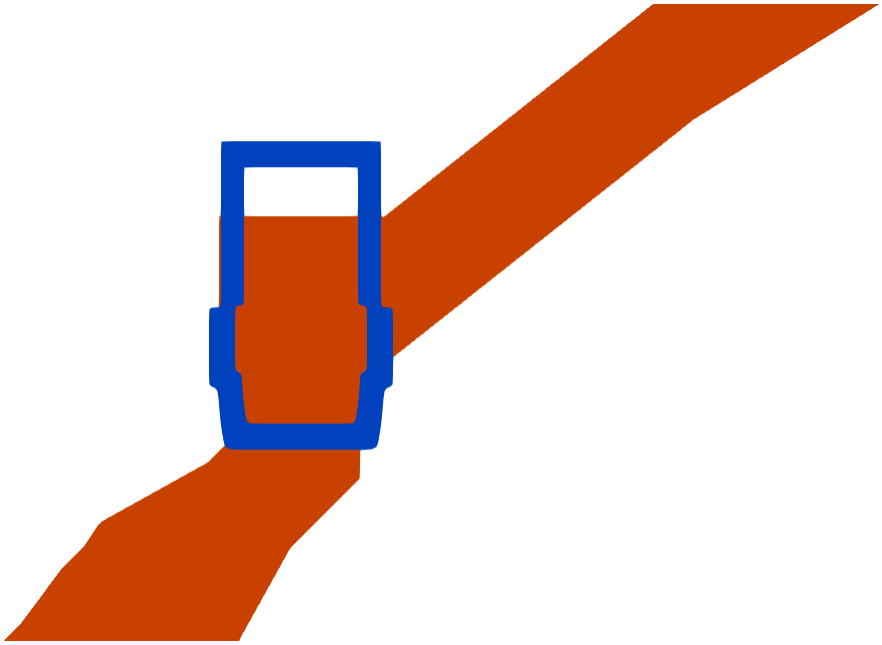

Fig. 12: 2D valve overset domain. Blue is the plug, and orange is the fluid(background) domain.

for overset simulations and presents the cumulative picture of the overset interpolating/hole cutting algorithm. The method for examining overset cell marker data in *Paraview* is:

1. Load the CSV file into Paraview (make sure to set the `Field Delimiter Characters` option to be a space)
2. Apply a `Table to Points` Filter to the loaded dataset.
3. Select the `X Column`, `Y Column`, and `Z Column` to be the `x`, `y`, `z` headers from the CSV file. If the file was read correctly in Step 1, the dropdown menu should show you x,y,z as data you can select for this part.

It is recommended to load both geometry files (fluid background mesh and plug component mesh) into *Paraview* along with the CSV data loaded using the process outlined above. Careful attention must be paid to which cells centers the red and green dots are aligned with. In the image below, if the red dots align with the cell centers of the left grid, then they are representing the iblank=3 state of the fluid mesh cells.

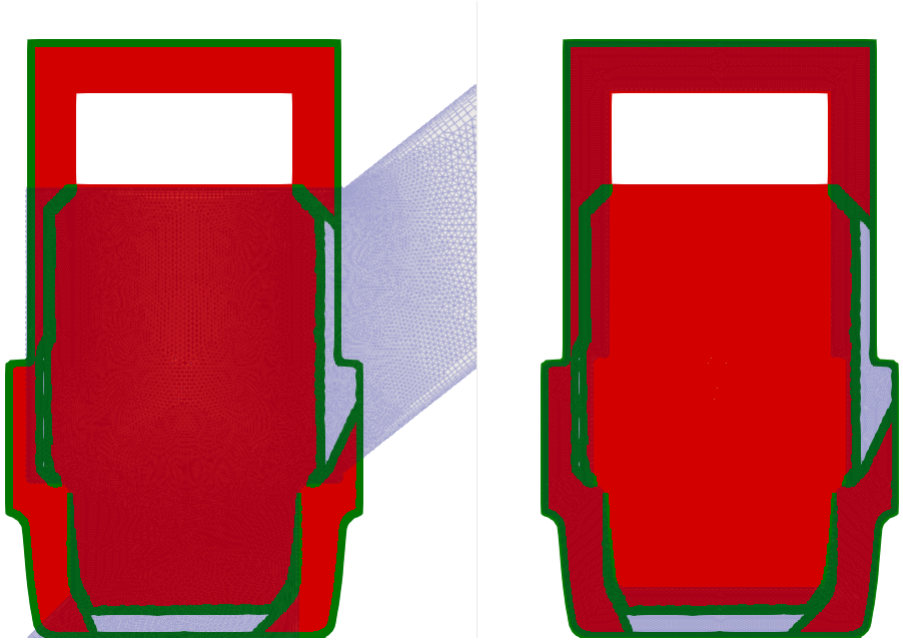

Fig. 13: Side-by-side image of the iblank dataset (red and green dots) overlayed on the fluid(left side) and plug(right side) meshes.

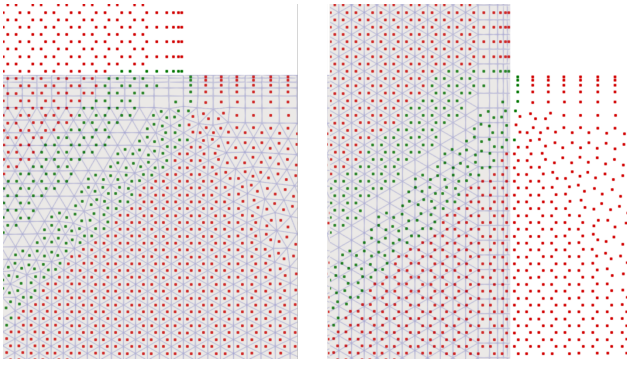

Fig. 14: Left is the fluid mesh, and right is the plug mesh. Note the alignment of the red and green cells on the left and right meshes.

In the image aboe, the left image is the background fluid mesh, and the right images if the plug component geometry mesh. As was mentioned earlier, this image was created by loading the background fluid mesh, and the plug overset

mesh separately into *Paraview* and displaying the iblank dataset over both grids. This is the region where the plug intersects the fluid mesh, on the left side of the plug. The vertical red/green lines that represent the background fluid mesh's boundary layer are helpful for visually estimating where the plug is located on the background mesh. The background mesh cells that are within the plug's no-slip surface boundary are marked with `iblank=3`. There is also a region around the plug akin to a buffer layer that is specified using the `componentGeometry` vars file specification which cuts out any of the background mesh that is contained within the boundaries of its specification. Ideally in this example, we only need an overlapping region of cells near where the outer boundary of the plug mesh's grid is, so if the plug mesh has a much larger mesh compared to the geometry that is represented by the plug, the `componentGeometry` can be specified to remove additional cells.

Going back the left image, therefore the red markers don't follow a simple straight line upwards. Because the `compoenentGeometry` is defined to be offset from the actual plug surface.

On the left image there are green markers that are aligned with the background mesh. These are `iblank=2` cells. They are cells that are receiving their values from the `iblank=0` cells that are used to make up the cloud of points for interpolation.

CHAPTER

EIGHT

PIMPLE MODULE

Stream has a module that provides users with the option to utilize the PIMPLE algorithm for their simulation. This capability is enabled by inserting the following `loadModule` directive into the run control file as shown below.

```
loadModule: pimple
{
  ... standard run control file content
}
```

The following sections discuss the PIMPLE method used in *Stream* and provide information on the run control file variables required to perform simulations with the PIMPLE module.

8.1 PIMPLE Algorithm

The PIMPLE algorithm that is implemented in *Stream* uses a combination of the SIMPLE and PISO algorithms. The SIMPLE algorithm being tuned for solving steady-state problems and the PISO algorithms being particularly suited for unsteady problems. The combination of the two algorithms allows for the larger timestep permitted by SIMPLE and the speed of the PISO to be combined into a hybrid algorithm. The figure below outlines the PIMPLE algorithm as it is implemented in *Stream*.

8.2 Control File Setup

Only a few additions need to be made to an existing run control file to set up a simulation using the PIMPLE module.

8.2.1 Numerics

From PIMPLE diagram above, we can see that the PIMPLE algorithm involved a SIMPLE-type predictor step, and a PISO-type corrector stage. The run control file variable that controls the total number of times that the algorithm goes through the SIMPLE+PISO stages is the standard `numIterationsPerTimeStep` variable. This value can be set as usual, but with the PIMPLE module, usually the value of the variable is a fraction of what would typically be used when running with just SIMPLE. The number of times that the PIMPLE algorithm iterates on the pressure correction equation using the PISO corrector can be controlled by the run control file variable shown below. **Note: there is not default value, so you must include this in the run control file if the pimple module is loaded.**

```
numCorrectorIterations:  1
```

The recommendation is to use only one corrector iteration unless cavitation equations are being solved. In that case, two corrector iterations are required for stability. Another feature that is available in the pimple module is the ability

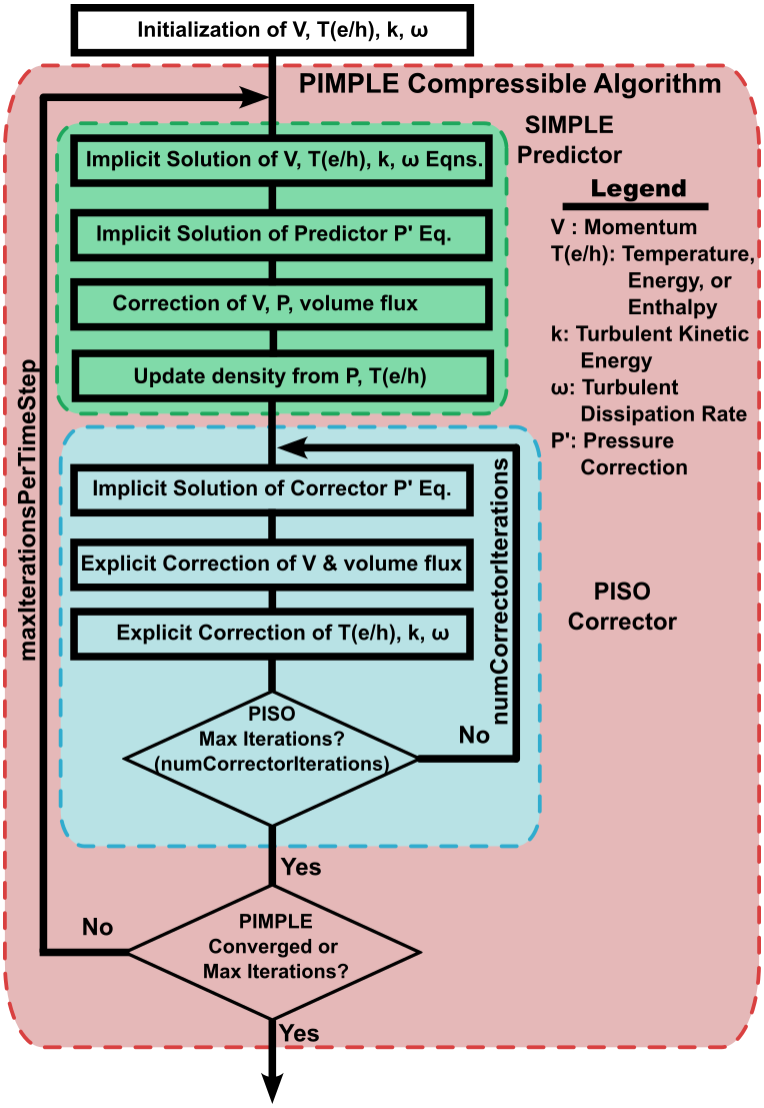

Fig. 1: Summary of the PIMPLE algorithm for compressible flows that is implemented in stream.

to control which equations are included in the explicit corrector step in the PISO corrector stage and can be specified in the run control file as shown below.

```
explicitCorrectorEquations:  <cavitation, energy, turbulence>
```

Including `cavitation` includes the vapor mass fraction and NCG mass fraction equations. The `energy` option includes either the temperature, total enthalpy, or total energy equations depending on the `form=` value in the `energyEquationOptions` run control file variable. The turbulence keyword includes the `k` equation and either the `epsilon` or `omega` equations, depending on the turbulence model specified in the `turbulenceEquationOptions` run control file variable.

A final variable determines the form of the volume flux that is used in the corrector. It is set by the `correctorVolumeFluxForm` variable in the run control file and has the value of old and new. An example of the specification specification for this variable is shown below (default shown).

```
correctorVolumeFluxForm:  new
```

A practical matter regarding the pimple module that user should consider is that the PIMPLE algorithm is more aggressive than the SIMPLE algorithm when solving equations, which may result in numerical instability if the PIMPLE algorithm is used for cases with bad initial pressure field guesses. **An easy fix is to run a simulation with a bad initial condition using SIMPLE for several timesteps and then restart and switch to the PIMPLE module.**

8.2.2 Output Variables

The table below shows additional field variables that are available for output in PIMPLE simulations. Default variables are output automatically and do not need to be specified in the `plot_output` line in the run control file.

Table 1: Field Variable Output for PIMPLE Module

Variable	Description	Default
	No current outputs for PIMPLE module supported	

CHAPTER

NINE

SPACETIMEINTERPOLATION MODULE

Stream has a module that provides users with a general capability for specifying spatially and temporally varying boundary conditions for the various inflow boundary types. This capability is enabled by inserting the following `loadModule` directive into the run control file as shown below.

```
loadModule: spaceTimeInterpolation
{
... standard run control file content
}
```

The following sections discuss the space-time interpolation module and provide information on the run control file variables required to perform simulations with the `spaceTimeInterpolation` module.

9.1 Space-Time Interpolated BCs

The `spaceTimeInterpolation` module provides facilities for describing spatial and temporally varying boundary conditions. This boundary condition is enabled by loading this module and specifying the directory that contains the interpolation source files with the variable `spaceTimeInterpolateFile`. All inflow type boundary conditions that are assigned the flag `spaceTimePrescribed` will receive the interpolated values. The data in the directory consists of HDF5 files with a description file called `data.info`. Files in this directory are described with a format of `filename:hdf5var` where `filename` is the name of the HDF5 file in the directory and `hdf5var` is the internal HDF5 dataset name. The filename or dataset name may contain special # symbols which will be substituted for the time step number of the file used for temporal interpolation. If multiple consecutive # characters are used, then the number will be padded to that width using leading 0 characters. The # characters can be used in either the `filename` or `hdf5var` portion of the file specifier. The `data.info` file contains the following variables shown in the table below.

Table 1: `data.info` File Contents

Variable	Description
positions	The positions variable specifies the file that contains the spatial locations of the input points. Note that the positions are expected to remain fixed in time and so the # symbol is not used in this file specifier.
initialStep	The initial step is the number of the first file in the time series.
lastStep	The last step is the number of the final file in the time series.
stepIncrement	The step increment gives how much the step increases between input files.
startTime	The start time is the time associated with the initial step.
deltaTime	The delta time is the temporal increment between steps.
Pambient	The reference pressure used for the gage pressure data files.
pg	The file that describes the spatially varying gage pressure at all the points described in positions.
t	The file that describes the spatially varying temperature at all the points described in positions.
v	The file that describes the spatially varying velocity vector at all the points described in positions.
k	The file that describes the spatially varying turbulent kinetic energy at all of the points described in positions.
w	The file that describes the spatially varying turbulent specific dissipation (omega) at all the points described in positions.
fspecies	Any variable that starts with an "f" character is assumed to be a mass fraction file. If no mass fraction files are specified, then the default mixture fraction specified in the chemistry mdl file will be used.

An example of a representative `data.info` file is given below. The data for this example was extracted from a separate simulation using the `clipSurfaces` functionality. The files output from this facility using a `clipFreq` of 10 and run for 600 timesteps were directly used by specifying the following input file:

Listing 1: Sample data.info file

```
positions: pos.0:data
initialStep: 10
lastStep: 600
stepIncrement: 10
startTime: 0
deltaTime: 1e-3
Pambient: 101325
pg: pg_sca.#:data
t: t_sca.#:data
v: v_vec.#:data
k: k_sca.#:data
w: w_sca.#:data
f_Air: fH2_sca.#:data
fH2: f_Air_sca.#:data
```

The clip surfaces capability allows a user to place an arbitrary infinite cutting plane within the computational domain. At the intersection of this plane with the CFD volume mesh, data is spatially interpolated using a cloud-of-points interpolator. Data extraction for the plane is controlled during a simulation by using the variables `plot_output` and

plot_freq.

9.2 Control File Setup

Only a few additions need to be made to an existing run control file to set up a simulation using the spaceTimeInterpolation module.

```
// Clip surfaces.
clipFreq: 100
clipSurfaces: < xcut=plane(point=[19.999,0,0],normal=[1,0,0]) >
```

CHAPTER

TEN

POROUS MEDIA MODULE

To use the porous media module, the run control file should appear as follows:

```
loadModule: porous
{
... standard vars file content
}
```

10.1 Armour Cannon Cady (ACC) Mesh Model

The mesh model based on the work of Armour, Cannon, and Cady [ArCa1968] [Cady1973] (ACC) is one of the available mesh models that can be applied to regions of the domain. The model adds a source term to the momentum equation to account for the effect of the porous media on the flow. The source term is given by the following expressions:

$$Kl = 2.0 \cdot \left(\frac{B}{BCFD}\right) \cdot alpha \cdot Q \cdot \left(\frac{\mu}{\rho}\right) \cdot a^2$$

$$Kq = 2.0 \cdot \left(\frac{B}{BCFD}\right) \cdot beta \cdot Q \cdot \frac{1}{Dp}$$

$$\text{SourceTerm} = \left(\frac{\rho}{2 \cdot \epsilon^2} \cdot (Kl + Kq \cdot \|v\|)\right) \cdot v$$

where ρ is the fluid density, μ is the dynamic viscosity of the fluid, and v is the velocity vector.

These parameters can be specified in the run control file using the names shown in the table below.

Table 1: Armour, Cannon, and Cady (ACC) Model Source Term Tunable Parameters

Variable	Description	Default Value
`a`	Screen surface area to volume ratio	None
`B`	Empirical constant	None
`BCFD`	Empirical constant	None
`Dp`	Hydraulic pore diameter	None
`Q`	Mesh tortuosity factor	None
`alpha`	Empirical constant	None
`beta`	Empirical constant	None
`epsilon`	Screen void fraction	None

The ACC model can be selected by specifying `ACC` as a mesh model type in the `porousScreens` variable as shown below.

```
porousScreens: <
    mesh1=ACC(a=510235, B=0.000089, BCFD=1000.0, Dp=0.000005, Q=1.3, alpha=3.2, beta=0.
↪19, epsilon=0.245)
>
```

10.2 Numerically Determined Resistance (NDR) Mesh Model

The mesh model is an engineering model with numerically determined resistance (NDR). The model adds a source term to the momentum equation to account for the effect of the porous media on the flow. The source term is given by the following expressions:

$$\text{SourceTerm} = \left(\frac{0.5 \cdot \rho \cdot \|v\| \cdot Kq}{\epsilon^2} \right) \cdot v$$

These parameters can be specified in the run control file using the names shown in the table below.

Table 2: Numerically Determined Resistance (NDR) Model Source Term Tunable Parameters

Variable	Description	Default Value
`Kq`	Empirical Constant	None
`epsilon`	Screen void fraction	None

The NDR model can be selected by specifying `NDR` as a mesh model type in the `porousScreens` variable as shown below.

```
porousScreens: <
    NDR(Kq=1742.782, epsilon=0.4)
>
```

10.3 Control File Setup

A porous media set of variables must be defined in the run control file. These definitions are used to specify the locations of the porous media screens and the parameters that define the porous media model used for the porous media regions. The following section shows an example specification of the porous media variables in the run control file.

```
porousScreens: <
  mesh1=ACC(a=510235, B=0.000089, BCFD=1000.0, Dp=0.000005, Q=1.3, alpha=3.2, beta=0.19,␣
↪epsilon=0.245),
  mesh2=NDR(Kq=1742.782, epsilon=0.4),
  screens=[
    box(center=[5.0,0.5,0.05], axis1=[1.0,0.0,0.0], axis2=[0.0,1.0,0.0], w1=0.1, w2=1.0,␣
↪w3=0.1, material=mesh2)
  ]
>
```

In the above specification of the `porousScreens` variable, the `mesh1` and `mesh2` variables are used to define the different types of mesh models that can be used in the porous media regions. The `screens` variable is used to define the regions of the domain where the porous media models are applied. The available supported screen geometries are detailed in the following section.

An additional run control file variable named `porositySourceOptions` can be used to specify whether to use an implicit or explicit source term treatment for the porous media model. The calculation type of the source term can also be specified to use a standard or conservative approach. The two methods are:

- The standard method, which evaluates the source term at the cell center, specified using `standard`.
- Mencinger's conservative method which requires a face evaluation of the porosity source term per unit volume, specified using `conservative`.

An example of specifying the source term treatment and the source term computation is shown below.

```
porositySourceOptions:  <implicit, conservative>
```

The default values for the source term treatment and the source term computation are `implicit` and `conservative`, respectively.

10.3.1 Screen Geometries

Circular Screen

The circular screen type is specified using the `circle` keyword in the `screens` variable. The parameters for the circular screen are the radius, center, and normal vector to the plane of the circular screen. The parameters are shown in the table below.

Parameter	Unit	Description
`radius`	m	Radius of the circular screen.
`center`	m	Center point of the circular screen.
`normal`		Normal vector to the plane of the circular screen.

An example of specifying a circular screen is shown below.

```
circle(radius=1.0, center=[0.0, 0.0, 0.0], normal=[0.0, 1.0, 0.0],
material=mesh1)
```

Cylindrical Screen

The cylindrical screen type is specified using the `cylinder` keyword in the `screens` variable. The parameters for the cylindrical screen are the radius, length, center, and axis vector along the length of the cylinder. The parameters are shown in the table below.

Parameter	Unit	Description
`radius`	m	Radius of the cylindrical screen.
`length`	m	Length of the cylindrical screen.
`center`	m	Center point of the cylindrical screen.
`axis`		Axis vector along the length of the cylinder.

An example of specifying a cylindrical screen is shown below.

```
cylinder(radius=1.0, length=1.0, center=[0.0, 0.0, 0.0], axis=[0.0, 1.0, 0.0],
material=mesh1)
```

Box Screen

The box screen type is specified using the `box` keyword in the `screens` variable. The parameters for the box screen are the width along the three axes, center, and two axis vectors defining the edges of the box. The parameters are shown in the table below.

Parameter	Unit	Description
`w1`	m	Width of the box along axis1.
`w2`	m	Width of the box along axis2.
`w3`	m	Width of the box along axis3 (automatically determined by the cross product of axis1 and axis2).
`center`	m	Center point of the box screen.
`axis1`		First axis vector defining one edge of the box.
`axis2`		Second axis vector defining another edge of the box perpendicular to axis1.

An example of specifying a box screen is shown below.

```
box(center=[0.0, 0.0, 0.0], axis1=[1.0, 0.0, 0.0], axis2=[0.0, 1.0, 0.0], w1=1.
0, w2=1.0, w3=1.0, material=mesh1)
```

10.3.2 Output Variables

The table below shows additional field variables that are available for output in simulations that use the porous media module. Default variables are output automatically and do not need to be specified in the `plot_output` line in the run control file.

Table 3: Field Variable Output for Porous Media Flows

Variable	Description	Default
`screenNumber`	Screen integer identification	No

References

CHAPTER

ELEVEN

APPENDICES

11.1 Appendix: Time Integration

The run control file variables that control the time integration of the system of governing equations are discussed in the following subsections.

11.1.1 Integration Method

The time-integration method used in *Stream* is specified with the run control file variable `timeIntegrator`. Three methods are available. The first method is the standard first-order backward differencing method. This method can be chosen by using the value `BDF`. If the `timeIntegrator` variable is not specified in the `vars` file, the value `BDF` is chosen by default. This method is commonly used for marching simulations to a steady-state condtion, but is not generally acceptable for unsteady flows due to the requirement for excessively small time steps in order to reduce numerical dissipation. For unsteady flows, *Stream* offers two second-order differencing methods. The first of these methods is the second-order backward differencing method, which can be chosen by using the value `BDF2`. The second is the Crank-Nicolson method, which can chosen by using the value `CN`. An example of setting the time integrator in a `vars` file is shown below.

```
timeIntegrator:  BDF2
```

Blended Crank-Nicholson Method

While both the `BDF2` and `CN` are formally second-order accurate methods, `CN` has been found in our experience to be less dissipative than `BDF2`. However, the pure `CN` scheme is not always robust for complex flows unless very small timesteps are used, which can lead to prohibitive run times. To mitigate this problem, *STREAM* provides a blended CN/BDF scheme, which is activated by a blending factor using the run control file variable `CNBDFBlend`, when using the `CN` time itegrator. This blending factor can be set anywhere between 0 (resulting scheme is 100% BDF) and 1 (resulting scheme is 100% CN). If this variable is not provided, the value defaults to 1. In practice, it has been found that blending factor above 0.9 can help stabilize the `CN` scheme at higher time step, while still providing slightly lower dissipation characteristics than `BDF2`. An example of using the blending factor is shown below.

```
CNBDFBlend:  0.9
```

For finite-rate chemistry simulations, *Stream* uses a second-order Strang-splitting procedure, which is not compatible with the specification of `BDF2` for the `timeIntegrator` variable. For unsteady DES and LES simulations with *Stream*, one should always use the `CN` scheme with the highest stable value of the variable `CNBDFBlend` as possible.

11.1.2 Time Step Selection

The time step for a simulation can be set in two different ways. To run a simulation with a single constant time step value, use the run control file variable `timeStep` in the following manner:

```
timeStep:  1.0e-01
```

If one is interested in only a steady-state solution, often the best approach is to simply eliminate the temporal terms from the governing equations by specifying a large time step value, say `1.0e+30`, which is the default value for this variable. One can then control the iterative process by only using the relaxation factors specified via the variables for the individual governing equations. This is the preferred method for steady-state flows. As a measure of last resort however, one can use a finite time step to perform temporal relaxation, which can often be effective at stabilizing a convergence path where iterative relaxation alone is not sufficient.

A second method of setting the time step is also available, whereby one can specify several so-called ramps in which the time step remains at a fixed value for a certain number of time steps before moving to a new ramp value. At the end of this process the time step will take on the value specified by the variable `timeStep`. Consider the following `vars` file specification:

Listing 1: time step ramp

```
timeStep: 1.0e-01
timeStepRamp: <ramp0=[n=1000,dt=1.0e-04],ramp1=[n=100,dt=1.0e-03],
ramp2=[n=10,dt=1.0e-02]>
```

In this case, the simulation will start off by performing 1000 time steps at $dt = 10^{-4}s$, followed by 100 time steps at $dt = 10^{-3}s$, followed by 10 time steps at $dt = 10^{-2}s$, and then assume the constant value of $dt = 10^{-1}s$ for the remainder of the simulation. It is important to note that the time step ramp will be active if the time step number in the simulation is lower than the total number of time steps in the ramp. So, for example, using the case above, if one ran a simulation of 1000 time steps and then stopped, upon restart the time step ramp would proceed to immediately do 100 time steps at $dt = 10^{-3}s$, followed by 10 time steps at $dt = 10^{-2}s$ and then use the constant value of $dt = 10^{-1}s$ for the remainder of the restart simulation. On the other hand, if one had run a simulation with 2000 time steps and then stopped, upon restart, the time step ramp would no longer be active, having stopped at time step 1110. One may use an arbitrary number of ramps. In addition, the names of the ramps (`ramp0`, `ramp1`, ...) are not important. One can choose any names. Only the order of the ramps is important.

11.1.3 Number of Time Steps

The number of time steps to run in the simulation is specified by the run control file variable `numTimeSteps`, as follows:

```
numTimeSteps:  1001
```

Once a simulation is initiated, it will not terminate until the number of specified time steps is run. If one wishes to terminate the solution at an arbitrary point in an orderly manner before the maximum number of time steps specified by `numTimeSteps` has been executed, one can use the so-called `touch stop` utility, by simply executing the command `touch stop` while in the directory from which the simulation was initiated. This will create a file called `stop` in the directory. When the code detects the presence of this file, it will terminate in an orderly manner, writing both output and restart files at the last time step executed. The code will automatically delete the file `stop` so that it will not linger around and cause problems with future runs from the same directory.

11.1.4 Time Step Convergence

To achieve the proper temporal accuracy of a time integration scheme, one must converge the system of equations within each time step to the level required to eliminate the iterative (SIMPLE) or corrective (PIMPLE) error. The topic of convergence estimation is discussed in more detail *here*. Convergence within any time step is controlled using the run control file convergence tolerance and maximum iteration variables. Two type of convergence specification are available. If one wishes to have a single absolute tolerance value for all governing equations, the following specification should be used (default values shown):

Listing 2: convergence tolerance

```
convergenceTolerance: 1.0e-30
maxIterationsPerTimeStep: 50
```

Using this form, if the total residuals (the residual values printed to standard output) at an iteration all become lower than the value specified by `convergenceTolerance`, the code will automatically advance to the next time step after that iteration. If such a convergence level is not obtained within the maximum number of iterations specified by the value of `maxIterationsPerTimeStep`, the code will in any case advance to the next time step. Note that the variable `convergenceTolerance` can only be effectively used if residuals have been normalized using reference values, as discussed *here*. Common practice is normally to forego the usage of `convergenceTolerance` by setting its value to a small number, say 10^{-30}, and simply using the variable `maxIterationsPerTimeStep` to control convergence within the time step. This is done generally to avoid the complication of having to compute reference values that provide meaningful nondimensionalization of all total residuals simultaneously. In addition, for engineering geometries, it is often impossible to achieve a consistent convergence tolerance limit on all residuals at every time step. For steady-state simulations, one would typically run a fixed number of time steps by setting values like the following:

Listing 3: steady-state simulation

```
convergenceTolerance: 1.0e-30
maxIterationsPerTimeStep: 1
numTimeSteps: 500
```

To control convergence within the time step in a more refined manner, one can use the run control file absolute and relative convergence tolerance variables as follows (all possible specifiable tolerances shown):

Listing 4: individual equation convergence criteria

```
convergenceAbsoluteTolerance: <default=1.0e-03, momentum=1.0e-04, pressure=1.0e-05,
                               energy=1.0e-06, k=1.0e-07, omega=1.0e-03, epsilon=1.0e-04,
↪ species=1.0e-02>

convergenceRelativeTolerance: <default=1.0e-03, momentum=1.0e-04, pressure=1.0e-04,
                               energy=1.0e-04, k=1.0e-04, omega=1.0e-04, epsilon=1.0e-04,
↪ species=1.0e-04>

maxIterationsPerTimeStep: 100
```

In the above options, the default value is first assigned to all governing equations. Subsequent entries for each of the specific governing equations are then specified to override the default value, if desired. If one does not specify a default value, the default value for the default value is 10^{-30}. One need only specify override tolerance values for the specific equations of interest. For example, one could specify an absolute convergence tolerance of 10^{-3} for all equations, but a tolerance of 10^{-4} for pressure, as follows:

Listing 5: specific absolute tolerance on pressure, generic on all other equations

```
convergenceAbsoluteTolerance: <default=1.0e-03, pressure=1.0e-04>
```

One may use `convergenceAbsoluteTolerance` and `convergenceRelativeTolerance` either separately or together. For example, to specify that convergence is to be determined by the satisfaction of only relative tolerances, one would specify:

Listing 6: specific relative tolerance on pressure, default on all other equations

```
convergenceRelativeTolerance: <default=1.0e-03, pressure=1.0e-04>
```

This specification is equivalent to the following:

```
convergenceAbsoluteTolerance: <default=1.0e-30>
convergenceRelativeTolerance: <default=1.0e-03, pressure=1.0e-04>
```

whereby the small default absolute convergence tolerance guarantees that absolute convergence is never satisfied, and only relative convergence determines convergence within the time step. **Convergence within a time step is declared when all the active governing equations are individually converged. Convergence for any governing equation is declared when either the absolute tolerance or the relative tolerance is satisfied.** If all active governing equations have not converged within the maximum number of iterations specified by the value of `maxIterationsPerTimeStep`, the code will automatically advance to the next time step.

11.2 Appendix: Initial Conditions

The run control file variables that are used to specify initial conditions for a simulation are discussed in the following subsections.

11.2.1 Uniform Initial Conditions

The most basic method of assigning initial conditions is to assign uniform conditions throughout the entire domain using the `initialCondition` variable in the run control file. An example of this is shown below for an incompressible, turbulent flow. Such a problem requires the specification of the initial state for the density, pressure, velocity and turbulence quantities. Here we assume that a k-omega turbulence model is in use. We note that for any given flow problem, additional variables that are specified beyond what is necessary will be ignored.

Listing 7: incompressible, turbulent, uniform initial condition

```
initialCondition:<rho=1.0 kg/m/m/m, p=10.0 atm, v=0.0 m/s, k=0.001, omega=1000.0>
```

The table below summarizes all variables for the main code that can be specified, as well as their definitions and default units. Additional variables for the various *Stream* modules are described in their respective chapters of this document.

Table 1: Variable Specification for `initialCondition`

Variable	Description	Default Units
`rho`	Density	$\frac{kg}{m^3}$
`p`	Pressure	Pa
`T`	Temperature	K
`mixture`	Species mass fractions	
`v`	Velocity	$\frac{m}{s}$
`k`	Turbulent kinetic energy	$\frac{m^2}{s^2}$
`omega`	Specific turbulent dissipation rate	s^{-1}
`epsilon`	Turbulent Dissipation rate	$\frac{m^2}{s^3}$

For compressible flows of a pure substance, two thermodynamic properties must be specified to establish the thermodynamic state for an initial condition. In *Stream*, this is done exclusively using the pressure and temperature. Given these two quantities, the initial conditions for other derived thermodynamic quantities such as density and specific internal energy are determined from the equation of state that is being employed in the simulation. Shown below is a typical uniform initial condition specification for a turbulent, compressible flow of a pure substance.

Listing 8: compressible, turbulent, pure substance, uniform initial condition

```
initialCondition: <p=10.0 atm, T=1000.0 K, v=1.0 m/s, k=0.001, omega=1000.0>
```

An initial condition for compressible, turbulent flow of a mixture material is similar specified, however one must now also provide an initial condition for the composition, which in *Stream* is given by the mixture species mass fractions, as shown below.

Listing 9: compressible, turbulent, mixture material, uniform initial condition

```
initialCondition: <p=10.0 atm, T=1000.0 K, v=1.0 m/s, k=0.001, omega=1000.0,
                   mixture=[H2=0.1, O2=0.1, H2O=0.8]>
```

The names of the species provided in the mixture specification must match those given in the `.mdl` file which specifies the mixture material. Species in the mixture material that are not explicitly assigned in the initial condition are given a default value of zero.

11.2.2 Left and Right State Initial Conditions

An initial condition with uniform left and right states may be needed for some problems. The run control file variables `ql` and `qr` are used to specify these states. The specification for `ql` and `qr` is identical to that shown above for `initialCondition`. The left and right initial condition states are separated by a plane whose location is specified at a midpoint by using either the variable `xmid`, `ymid` or `zmid`. If a midpoint is not specified in the `vars` file, the default specification is `xmid` with a value of zero in which case the y-z plane through the origin is the surface separating the left and right uniform initial condition states. If `ymid` is specified, then the cutting plane is an x-z plane located at the `ymid` value, while if `zmid` is specified, the cutting plane is an x-y plane located at the `zmid` value.

With the location of the cutting plane specified, the left state, `ql`, is used for all cells with a cell-center coordinate less than the specified midpoint coordinate value. The right state, `qr`, is used for cells with a cell-center coordinate greater than the specified midpoint coordinate value. The following example shows a typical specification for a laminar compressible flow of a mixture material.

Listing 10: compressible, mixture material, hydrogen/oxygen initial condition split at x=0.5

```
ql: <p=10.0 atm,T=1000.0 K,v=0.0 m/s,mixture=[H2=1.0]>
qr: <p=1.0 atm,T=300.0 K,v=0.0 m/s,mixture=[O2=1.0]>
xmid: 0.5
```

11.2.3 Initial Condition Regions

It may be desirable to assign different initial condition values to different regions of the computational domain. For example, one may have a pipe with a 90-degree bend. In this case, if one wished to assign an initial condition for velocity that is parallel to the walls of the pipe, one would need to be able to independently specify two different conditions. *Stream* provides a general capability based on geometric primitives (shapes) that allows the user to specify an arbitrary number of initial condition states over various regions of the entire domain. This specification is accomplished with the run control file variable `initialConditionRegions`. To illustrate the use of this feature, consider the example specification below for a compressible flow of a mixture material:

Listing 11: compressible, mixture material, hydrogen/oxygen initial condition by region

```
initialConditionRegions:<
  default=state(u=0.0m/s, p=1atm, T=298K, mixture=[N2=1.0]),
  hydrogen=state(u=0.0m/s, p=1atm, T=400K, mixture=[H2=1.0]),
  oxygen=state(u=0.0m/s, p=1atm, T=200K, mixture=[O2=1.0]),
  regions=[inBox(p1=[0,0,0], p2=[1,1,1], composition=hydrogen),
  inSphere(radius=0.1m, center=[0.5,0.5,0.5], composition=oxygen)] >
```

In this example, the default state is first used to assign values to every cell in the computational domain. The regions that are subsequently defined are processed in order of appearance, each overwriting the state of the cells contained inside the region. Thus, for this example, the `default` state is replaced by the `hydrogen` state for cells inside the defined box and then the cells contained inside the defined sphere are assigned the `oxygen` state.

The table below contains a complete listing of the variables for the main code that can specified inside `initialConditionRegions`. As with the other initial condition specifications above, consistent units other than the default units may be specified, in which case numeric values are converted internally to the default units for use in the code. If units are not specified for a quantity, the default units for that quantity are assumed.

Table 2: Variables for Initial Condition Regions Input

Variable	Description	Default Units
`rho`	Density	$\frac{kg}{m^3}$
`p`	Pressure	Pa
`T`	Temperature	K
`mixture`	Species mass fractions	
`u`	Velocity	$\frac{m}{s}$
`M`	Mach Number	
`k`	Turbulent kinetic energy	$\frac{m^2}{s^2}$
`omega`	Specific turbulent dissipation rate	s^{-1}
`epsilon`	Turbulent Dissipation rate	$\frac{m^2}{s^3}$

The geometric primitives that are currently supported for `initialConditionRegions` are shown in the table below.

Table 3: Geometric Primitives for Initial Condition Regions Input

Prim-itive	Description
`inBox`	Takes two arguments p1 and p2, which define two diagonally opposing corners of the box.
`inSph`	Takes two arguments, radius, and center, which define respectively the size and location of the sphere.
`inCyl`	Takes three arguments, radius which defines the cylinder radius, and the points p1 and p2, which define the endpoints of the axis of the cylinder.
`inCon`	Takes four arguments, p1 and p2 which define the endpoints of the axis of the cone, and r1 and r2 which define the radius of the cone at the locations p1 and p2, respectively.
`leftP`	Takes two arguments, point, which defines a point on the plane, and normal, which defines a vector normal to the plane. The region included in this primitive is the semi-infinite region on the side of the plane away from the direction of the specified normal.

11.2.4 Interpolated Initial Conditions

Often after running a simulation for a particular geometry, one decides that a refined mesh is required. In this case, one can use the solution obtained from the old simulation as an initial guess for the new simulation by using the `interpolateInitialConditions` variable in the run control file as follows:

Listing 12: puT interpolated initial condition

```
interpolateInitialConditions: put.1000_cavity
```

When running a simulation, if the user has included the variable `put` in the `plot_output` line of the run control file, *Stream* writes out what is known as the `puT` file in the output directory along with other output files according to the value of the run control file variable `plot_freq`. In the above example, the value `1000` indicates the time-step

number at which data was written in the previous simulation, and the name `cavity` is the case name of the previous simulation. The `puT` file (the name is an acronym for pressure(p), velocity(u), and temperature(T)) contains all the basic flow variable information associated with the run, including velocity, pressure, temperature, species, and turbulence data if the simulation is employing a turbulence model.

If one is running a new case in which the type of simulation remains constant but the grid has changed, then one simply needs to select any `puT` file from a previous run and use it in the above manner. Note that the name of the `puT` file specified in the new run must include the complete path to the file. However, if the simulation type changes between runs, there are a few special cases of interest which require additional input, the details of which follow:

- Consider the case of using a `puT` file (compressible or incompressible) to restart to an incompressible simulation. The density is not stored in a `puT` file, and there is no equation of state for an incompressible flow; therefore, the variable `initialCondition` must be used to provide the value for density. All required values for an incompressible flow simulation (density, velocity, and pressure) must be specified in `initialCondition`, but only the value of density will be taken from this variable. All other values will be obtained from the file specified by `interpolateInitialConditions`.
- For using an incompressible `puT` file as an initial condition to a new compressible run, one must use the variable `initialCondition` to provide the value for the initial temperature. This is because the value for temperature in any incompressible `puT` files is zero, which is non-physical for compressible flow. A complete specification for `initialCondition` appropriate for compressible flow must be given, but only the temperature value will be used.

11.3 Appendix: Boundary Conditions

Boundary conditions are specified by using `boundary_conditions` run control file variable, an example of which is shown below.

Listing 13: run control file boundary condition specification example

```
boundary_conditions:
<
Inlet=totalPressureInlet(p0=135.0 psi, T=164 R, k=0.05, omega=282.46, incompressible),
IPipe=noslip(adiabatic),
OPipe1=noslip(adiabatic),
OPipe2=noslip(adiabatic),
YPart=noslip(adiabatic),
Outlet1=fixedMassOutlet(mdot=-1027.8 lbm/s),
Outlet2=fixedMassOutlet(mdot=-1027.8 lbm/s)
>
```

The names of the boundaries specified on the left-hand sides of the declarations above must match those defined in the `.vog` file for the simulation. In the `.vog` file, each boundary name is identified with a set of boundary faces on the mesh which comprise the boundary. The corresponding boundary condition specified on the right-hand side of the boundary condition declaration is applied to all faces that comprise the boundary. During the grid generation process, it is important that all faces on the boundary of the domain be assigned to a boundary name. Should any faces not be included in a boundary name group, the code has no way of specifying the boundary conditions for those faces, and will terminate due to an insufficient problem specification. The types of boundary conditions supported in *Stream* are described in the following sections.

11.3.1 extrapolatedPressureOutlet

This outlet boundary condition is appropriate for compressible flow problems where the flow is supersonic at the outlet. An example of its usage is shown below:

```
Outlet=extrapolatedPressureOutlet
```

There are no options required by this boundary condition. Velocity, pressure, temperature, and turbulence quantities are all extrapolated from the center of the cell adjacent to the boundary face to the boundary face center. Density is then computed from the equation-of-state. Due to the use of zeroth-order extrapolation, it is prudent that outlet boundaries using this boundary condition be placed in a location where streamwise gradients are small, as well as to ensure that the grid is highly orthogonal, with the line from any boundary cell center to its respective boundary face center very closely aligned with the boundary normal vector.

11.3.2 fixedMassOutlet

This outlet boundary condition is appropriate for both incompressible and compressible flow problems. Two options are available. Using the `mdot` option, the total mass flow rate through the boundary can be specified directly, as shown below:

```
Outlet=fixedMassOutlet(mdot=-10.0 kg/s)
```

Alternately, using the `massFlux` option, the mass flow rate per unit area over the boundary can be specified, as follows:

```
Outlet=fixedMassOutlet(massFlux=-2.0 kg/m/m/s)
```

Internally, *Stream* operates using only mass fluxes for the boundary faces. When the boundary mass flow rate is specified using the `mdot` option, the equivalent `massFlux` is computed automatically by dividing the specified value by the total boundary area. **By convention, a negative value indicates mass flow out of the domain. Positive values for this boundary condition should not be used.** One may specify any units that are consistent with the units shown above, and conversion to SI units will be handled internally by the code. For incompressible flows, density, pressure, and turbulence quantities are extrapolated from the center of the cell adjacent to the boundary face to the boundary face center. For compressible flows, pressure, temperature, and turbulence quantities are extrapolated and then density is computed from the equation-of-state. The velocity on any boundary face is then computed directly from the mass flux and the local face density, and is assigned to be in the direction of the local face outward normal vector.

11.3.3 fixedPressureInlet

This inlet boundary condition is appropriate for both incompressible and compressible flow problems, and is designed to hold either a specified pressure or a specified mean pressure over the boundary. The velocity at the boundary faces is obtained via extrapolation from the adjacent cell centers in the interior of the domain. For incompressible simulations, one should specify the value of the pressure or the mean pressure. For compressible simulations, the temperature at the boundary must also be specified using any of the available scalar specification forms detailed *here* . If the simulation involves turbulent flow, one should also specify the turbulence intensity `I`, and the turbulent to laminar viscosity ratio `muRatio`. These two quantities may only be assigned as constant values. Profiles for `I` and `muRatio` are not currently available. Inlet values for the primitive turbulence variables variables `k`, `omega`, and `epsilon` are then computed based on the following equations:

$$k = -\frac{3}{2}(\vec{v}\cdot\vec{v})I, \quad \omega = \frac{\rho k}{\mu_{\text{lam}}\cdot\mu_{\text{ratio}}}, \quad \epsilon = \frac{C_\mu \rho k^2}{\mu_{\text{lam}}\cdot\mu_{\text{ratio}}}$$

If the simulation contains multiple species, one should also assign the species mass fraction values using the mixture option. When using the mixture option, one need not list values for which the species mass fraction is zero, as this will be automatically handled by the code.

Listing 14: fixed pressure inlet, incompressible, turbulent

```
Inlet=fixedPressureInlet(p=101325.0 Pa, I=0.03, muRatio=100.0)
```

Listing 15: fixed mean-pressure inlet, compressible, turbulent, multi-species

```
Inlet=fixedPressureInlet(pMean=101325.0 Pa, T=300.0K, mixture=[H2=0.5, O2=0.5],
                         I=0.03, muRatio=100.0)
```

11.3.4 fixedPressureOutlet

This outlet boundary condition is appropriate for both incompressible simulations and compressible simulations where the flow always remains subsonic on the outlet boundary. There are two types of pressure constraints which may be employed for this boundary condition. Since there is no default value, one must explicitly specify either one of the constraints. With the first constraint, a constant pressure is specified and maintained on all boundary faces of the mesh comprising the boundary.

Listing 16: constant pressure

```
Outlet=fixedPressureOutlet(p=202650 Pa)
```

With the second type of constraint, a mean pressure is specified and maintained on the boundary. This is implemented internally within the code by extrapolating the current pressure field from the cells next to the boundary to the boundary faces and then adding the same constant pressure correction value to all boundary faces to achieve the desired mean value.

Listing 17: mean pressure

```
Outlet=fixedPressureOutlet(pMean=101320 Pa)
```

All other solver variables including velocity are extrapolated from the interior of the domain to the boundary. By default, this extrapolation will not allow flow to come into the domain through any face on the boundary. In certain instances, such as the case where one is using this boundary condition to approximate a far-field boundary of nearly constant pressure, one may wish to allow entrainment (flow coming into the domain from outside the domain) to occur.

Listing 18: entrainment

```
Outlet=fixedPressureOutlet(p=202650 Pa, entrainment)
```

The `entrainment` option can be used with both the `p=` and `pMean=` options. In general, one will only want to use this option when there is relatively weak recirculation through the boundary. For internal flow problems, if one finds any recirculation zones in the vicinity of the outlet boundary, it is preferable to re-grid the domain in such a way (by including a part of a downstream component, for example) as to eliminate recirculation at the outlet, which otherwise could be a cause of numerical instability preventing the simulation from achieving convergence.

11.3.5 incompressibleInlet

This boundary condition is used only for incompressible flow simulations. For laminar flow simulations, one need only specify the velocity, either directly via the velocity value (using any the of methods detailed *here*) or indirectly via the mass flux or mass flow rate through the boundary. The usage of these three methods is shown below:

Listing 19: incompressible, laminar

```
Inlet=incompressibleInlet(v=[1.0 m/s, 0.5 m/s, 0.1 m/s])
Inlet=incompressibleInlet(massFlux=0.5 kg/m/m/s)
Inlet=incompressibleInlet(mdot=2.7 kg/s)
```

For turbulent flow simulations, one must also specify the turbulence quantities at the inlet using the `k=` and `omega=` options using any of the methods for scalar values detailed *here*.

For compatibility with *Chem*, the option `prescribed=` is also allowed. With this option, one can specify a data file in which the velocity, pressure, temperature, and turbulence quantities are specified at several control points. Values are computed at the centers of the faces on the boundary via a generalized interpolation of the data specified at the control points. For the `incompressibleInlet` boundary condition, only the velocity and turbulence quantities (for turbulent simulations) are used from the data file, since this is the only required information for incompressible flow.

Listing 20: prescribed BC

```
Inlet=incompressibleInlet(prescribed="bc.dat")
```

Any name may be specified for the boundary condition file. If one specifies no name between the quotes, the default file name `bc.dat` is used.

11.3.6 noslip

The `noslip` boundary condition should be used for all solid surfaces in viscous flow simulations. For compressible flow simulations, it is **IMPORTANT** to note that there is no default method that is automatically selected for heat-transfer at the boundary. Thus, one must explicitly assign either an adiabatic, specified temperature or specified heat flux conditions as an option. The following sections detail all the available options for the no-slip boundary condition.

Wall Functions

The default mode of operation is to compute wall viscous stresses and heat transfer using the standard gradient operations with information in the vicinity of the wall. This is the so-called **gridding down to the wall** approach, where wall functions are not employed by default. When using this mode, one should ensure that $y^+ < 3$‘ for accurate viscous stresses and heat fluxes. For incompressible flow, because the energy equation is not solved, if one is gridding down to the wall, there are no options required for the no-slip boundary and the boundary condition can be used in either of the following ways:

Listing 21: no-slip, incompressible

```
Wall=noslip
Wall=noslip()
```

Wall functions can be activated by using the `wallFunction` option.

Listing 22: wall function

```
Wall=noslip(wallFunction)
```

The only wall function model currently available is that of [NiNe2003]. It is important to note that wall functions are enabled on a patch-by-patch basis. Only boundary patches with the `wallFunction` option will employ wall functions, while the remaining no-slip surfaces without the `wallFunction` option will employ the default approach of gridding down to the wall.

Adiabatic Wall

For solid surfaces with no heat flux through any part of the boundary, one should use the `adiabatic` option. If one is gridding down to the wall, one would specify the boundary condition as follows:

Listing 23: adiabatic wall

```
Wall=noslip(adiabatic)
```

Wall functions can also be enabled for adiabatic boundaries, in which case one need only tack on the `wallFunction` option.

Specified Temperature at the Wall

For solid surfaces with a fixed temperature distribution, one should use the option `T=`, with any of the methods specified *here* to assign the temperature distribution. Wall functions may also be used for boundaries with specified temperature by tacking on the `wallFunction` option, in a manner like the following:

Listing 24: specified temperature

```
Wall=noslip(T=500 K, wallFunction)
```

For compatibility with *Chem*, the option `Twall=` is also supported. This option can also be used to specify a constant scalar wall temperature value like the example above.

Specified Heat Flux at the Wall

For solid surfaces with a fixed heat flux ($\frac{W}{m^2}$) through the wall, one should use the option `qwall=` in the following manner (default units shown):

Listing 25: specified heat flux

```
Wall=noslip(qwall=1024 W/m/m)
```

A single constant heat flux value is the only means of specification currently supported. Any units consistent with $\frac{W}{m^2}$ may be used. **By convention, the value specified is defined to be the heat flux from the fluid domain to the wall.** Thus, one should specify a negative value to have positive heat transfer from the wall to the fluid. At the current time, wall functions cannot be used with the specified heat flux condition, so one must use this option only when gridding down to the wall.

Specified Wall and Reservoir Conditions

If one wants to approximate the condition in which a wall of known effective thermal resistance separates the fluid domain from a reservoir held at constant temperature, one can specify the wall and reservoir conditions in the following manner (default units shown):

Listing 26: specified wall and reservoir conditions

```
Wall=noslip(Treservoir=400 K, Rwall=10.0 m*m*K/W)
```

At the current time, only constant value specifications for both `Treservoir` and `Rwall` are supported. With the supplied reservoir temperature and wall effective thermal resistance, the code will perform a 1-D wall heat-transfer balance to compute the wall temperature that balances the flux leaving the fluid domain with that entering the reservoir. At the current time, wall functions cannot be used with this specification, so one must use this option only when gridding down to the wall.

11.3.7 slip

The `slip` boundary condition should be used for all solid surfaces in inviscid flow simulations. All flow variables are extrapolated to the boundary from the cell values in the interior of the domain. There are no options required for this boundary condition.

Listing 27: slip

```
Wall=slip
```

11.3.8 subsonicInlet

This boundary condition is only used for compressible flow simulations. For numerical stability purposes, the flow should remain subsonic on the boundary during the entire simulation. Pressure on the boundary is obtained by extrapolation from the interior of the domain. Density on the boundary is computed from the equation of state once the temperature at the boundary is known. The velocity on the boundary may be specified either directly via the velocity value (using any the of methods detailed *here*) or indirectly via the mass flux or mass flow rate through the boundary. The usage of these three methods is shown below:

Listing 28: compressible, subsonic

```
Inlet=subsonicInlet(v=[0.5 m/s, 0.1 m/s, 1.0 m/s])
Inlet=subsonicInlet(massFlux=0.5 kg/m/m/s)
Inlet=subsonicInlet(mdot=2.7 kg/s)
```

Note that unlike for `incompressibleInlet`, the explicit specification of the boundary velocity via the `v=` option does not result in a fixed-mass inlet because the density at the boundary will evolve during the simulation. The temperature on the boundary must also be specified with the `T=` option using any the of methods detailed *here* for scalar specification. For turbulent flow simulations, one must also specify the turbulence quantities at the inlet with the `k=` and `omega=` options, again using any of the methods detailed *here* for scalar specification. If the simulation contains multiple species, one should also assign the species mass fraction values using the `mixture=` option. When using the `mixture=` option, one need not list values for which the species mass fraction is zero, as this will be automatically handled by the code.

Listing 29: compressible, turbulent, multi-species, subsonic

```
Inlet=subsonicInlet(mdot=2.7 kg/s, T=400 K, k=0.05 m*m/s/s, omega=500.0 1/s,
                    mixture=[H2=0.5, O2=0.5])
```

For compatibility with *Chem*, the option `prescribed=` is allowed. With this option, one can specify a data file in which the velocity, pressure, temperature, mixture species mass fractions and turbulence quantities are specified at several control points. Values are computed at the centers of the faces on the boundary via a generalized interpolation

of the data specified at the control points. For the `subsonicInlet` boundary condition, the only quantity not used from the data file is the pressure because pressure is extrapolated from the interior of the domain.

Listing 30: prescribed BC

```
Inlet=subsonicInlet(prescribed="bc.dat")
```

Any name may be specified for the boundary condition file. If one specifies no name between the quotes, the default file name `bc.dat` is used.

11.3.9 supersonicInlet

This boundary condition is only used for compressible flow simulations. For numerical stability purposes, the flow should remain supersonic on the boundary during the entire simulation. The velocity on the boundary may be specified either directly via the velocity value (using any the of methods detailed *here*) or indirectly via the mass flux or mass flow rate through the boundary. The usage of these three methods is shown below:

Listing 31: compressible, supersonic

```
Inlet=supersonicInlet(v=[0.5 m/s, 0.1 m/s, 1.0 m/s])
Inlet=supersonicInlet(massFlux=0.5 kg/m/m/s)
Inlet=supersonicInlet(mdot=2.7 kg/s)
```

The temperature and pressure on the boundary must also be specified with the `T=` and `p=` options, respectively. Temperature may be specified using any the of methods detailed *here* for scalar specification, however, at the current time only a single constant value may be specified for pressure. For turbulent flow simulations, one must also specify the turbulence quantities with the `k=` and `omega=` options, again using any of the methods detailed *here* for scalar specification. If the simulation contains multiple species, one should also assign the species mass fraction values using the `mixture=` option. When using the `mixture=` option, one need not list values for which the species mass fraction is zero, as this will be automatically handled by the code.

Listing 32: compressible, turbulent, multi-species, supersonic

```
Inlet=supersonicInlet(mdot=2.7 kg/s, p=202650 Pa, T=400 K, k=0.05 m*m/s/s,
                      omega=500.0 1/s, mixture=[H2=0.5, O2=0.5])
```

For compatibility with *Chem*, the option `prescribed=` is also allowed. With this option, one can specify a data file in which the velocity, pressure, temperature, mixture species mass fractions and turbulence quantities are specified at several control points. Values are computed at the centers of the faces on the boundary via a generalized interpolation of the data specified at the control points. For the `supersonicInlet` boundary condition, all entries from the data file are used since no information is extrapolated from the interior to the boundary for supersonic flow. The `prescribed=` option is used in a manner like the following:

Listing 33: prescribed BC

```
Inlet=supersonicInlet(prescribed="bc.dat")
```

Any name may be specified for the boundary condition file. If one specifies no name between the quotes, the default file name `bc.dat` is used.

11.3.10 Symmetry

This boundary condition is used where one desires to enforce a zero normal gradient condition at the boundary. This includes symmetry boundaries in 3-D simulations where the flow is symmetric about the boundary. For 2-D simulations, flow variables are only a function of two of the coordinate directions. Grid boundaries with normal vectors parallel to the third coordinate direction should be assigned as `symmetry` so the flow solution will not vary in this third direction. As there are no options required for this boundary condition, it is generally used as follows:

Listing 34: symmetry

```
Boundary=symmetry
```

11.3.11 totalPressureInlet

This boundary condition can be used for both incompressible and compressible flow and is designed to hold a single fixed value of total pressure at each of the faces on the boundary. The total pressure value is specified using the `p0=` option. Temperature can be specified either directly with the `T=` option or indirectly with a total temperature via the `T0=` option. If temperature is specified, any of the methods detailed *here* for scalar specification may be used. If total temperature is specified, one may only use a single constant value. For turbulent flow simulations, one must also specify the turbulence quantities with the `k=` and `omega=` options, again using any of the methods detailed *here* for scalar specification. If the simulation contains multiple species, one should also assign the species mass fraction values using the `mixture=` option. When using the `mixture=` option, one need not list values for which the species mass fraction is zero, as this will be automatically handled by the code.

Listing 35: compressible, turbulent, multi-species

```
Inlet=totalPressureInlet(p0=202650 Pa, T0=400 K, k=0.05 m*m/s/s, omega=500.0 1/s,
                      mixture=[H2=0.5, O2=0.5])
```

For compressible flows with real fluids, one must also use the option `incompressible`. This is currently required since the implementation of the total pressure inlet for compressible flow is based on equations which assume an ideal-gas equation of state. Use of the `incompressible` option instructs the code to revert to the incompressible form of the total pressure relations, which is a suitable approximation for low Mach number inlet flows.

Listing 36: compressible, turbulent, multi-species, real-fluid

```
Inlet=totalPressureInlet(p0=202650 Pa, T0=400 K, k=0.05 m*m/s/s, omega=500.0 1/s,
                         mixture=[H2=0.5, O2=0.5], incompressible)
```

If the simulation involves an incompressible flow, then one should **not** include the `incompressible` option, as the incompressible form of the total pressure relation is automatically invoked by the code.

References

11.4 Appendix: Inviscid Fluxes and Gradient Limiting

Numerical treatment of the convection terms in the governing equations is of primary concern regarding both accuracy and stability of the computation. Selection of the convection scheme is made using the run control file variable `inviscidFlux`.

```
inviscidFlux:  SOU
```

Three numerical treatments are available, a first-order scheme (`FOU`), a second-order scheme (`SOU`) and a second-order scheme intended for use in compressible flows with shock waves (`SLAU2`). The `FOU` and `SOU` schemes may be used for both incompressible and compressible simulations. The `SLAU2` scheme is only available for compressible simulations. These schemes as well as the associated limiters are briefly detailed in the following sections.

11.4.1 First-Order Upwinding

This scheme is selected by specifying the value `FOU` for the `inviscidFlux` variable. With first-order upwinding, face values for the dependent variables are obtained by a simple zeroth-order extrapolation from the upwind associated cell center. Overall, this scheme results in a calculation which is first-order accurate in space. In general, one should not use first-order upwinding for production runs where grid independent solutions are required as a prohibitive number of grid cells would be required to achieve grid-independence. The generally dissipative nature of this scheme can be advantageous for starting simulations from scratch with a poor initial condition if it is found that second-order schemes have difficulty at start-up due to their relatively less dissipative nature.

11.4.2 Second-Order Upwinding

This scheme is selected by specifying the value `SOU` for the `inviscidFlux` variable in the run control file. With second-order upwinding, face values for the dependent variables are obtained via a linear extrapolation from the upwind associated cell center using both the value and gradient of the dependent variable. The linear extrapolation can be limited for stability purposes so that no extrema are introduced by the extrapolation using the limiters discussed *here*. This is generally a requirement for numerical stability when simulating flows on engineering meshes. Overall, this scheme results in a calculation which is second-order accurate in space and is the preferred scheme for production simulations that do not involve the presence of shock waves.

11.4.3 SLAU/SLAU2

The Simple Low-dissipation Advection-Upstream-splitting-method (SLAU) scheme should be used in compressible simulations that contain shock waves. This scheme is selected by specifying the value `SLAU` or `SLAU2` for the `inviscidFlux` variable in the run control file. The scheme maintains formal second-order accuracy in regions away from the shocks and degenerates to lower order in the immediate vicinity of the shocks to maintain monotonicity in transitioning from the state on one side of the shock to the state on the other side of the shock. Similar to the second-order upwinding scheme, the SLAU scheme also involves extrapolation operations which require the use of the limiters discussed *here*. An example showing the specification of the SLAU scheme is shown below.

```
inviscidFlux:  SLAU
```

A hybrid SLAU scheme is also available in *Stream.* This hybrid scheme is a blend of SLAU or SLAU2 with a vanLeer Hanel Scheme. This allows for users to have more control of the numerical stability with simulations of flows that have strong shockwaves. Note: To activate the hybrid SLAU scheme, the user must still specify either `SLAU` or `SLAU2` for the `inviscidFlux` variable in the run control file. To activate this scheme, add the slau_hybrid variable to the run control file as shown below (default shown):

```
slau_hybrid:  0
```

The default value for this variable is 0 if `slau_hybrid` is not included in the run control file, this defaults to the non-hybrid SLAU scheme that was set by the `inviscidFlux` variable. Other options for the variable are `1` and `2`. Specifying `1` only hybridizes the momentum flux and specifying `2` hybridizes both the momentum and energy fluxes.

11.4.4 Inviscid Flux for Turbulence Equations

There may be instances in which one may suspect that the turbulence equations are the source of nonlinear instability within a time step. This could be due to any number of factors, including a poor initial condition or some flow feature that takes the calibration of the turbulence models far out of the range of their intended validity. In such circumstances, in order to achieve stability, one may be required to bring the turbulence equations down to first-order. This is accomplished using the run control file variable `turbulenceInviscidFlux` as shown below (default shown):

```
turbulenceInviscidFlux:  SOU
```

The default value for this variable is `SOU`. One should attempt to use `SOU` whenever possible, as the use of `FOU` results in excessively smeared shear layers, recirculation regions and separated flow features. Often it is sufficient just to use `FOU` to get past a bad initial condition, and then restart to `SOU`.

11.4.5 Limiters

Limiters are used in conjunction with the second-order convection schemes discussed above as well as throughout the code where projection of cell gradient information to the cell faces is required.

```
limiter:  venkatakrishnan
```

The `venkatakrishnan` (default) option can be specified for the Venkatakrishnan limiter [Venk1993] to be used. The `barth` option will activate the Barth limiter [BaJe1989]. If the value `none` is specified, no limiting will be performed. The `mlp` option can be specified to activate a limiter that is based on the multidimensional limiting process [Zhan2018], which is a limiter that is more robust than the Venkatakrishnan limiter for flows with shockwaves. A second, even more tuned limiter [Zhan2018] for flows with shockwaves can be specified by using `mlp_pw`. This is an MLP limiter that uses a pressure function to detect shockwaves and utilizes a larger stencil of cell and node information in order to determine an appropriate limiter value. It is important to note that at the current time the only variable that controls limiting is the limiter variable. Thus, the same limiter will be used for limiting the convection schemes as well as other terms in the governing equations. Often, a user may think that they can use the value `zero` to select first-order upwinding. In *Stream*, one **should not** use this option, as this turns off all limiting throughout the code. The proper way to select first-order upwinding is with the variable `inviscidFlux`.

The `venkatakrishnan`, `mlp`, and `mlp_pw` limiters utilize something called the Venkatakrishnan limiting function. This is a simple algebraic quantity that controls the sensitivity of the limiter. It has a parameter that can be set to tune the sensitivity of the limiters for regions of smooth flows. The run control file variable for setting this parameter is `Kl`, and an example of its specification is shown below (default shown).

```
Kl:  1.0
```

A rule of thumb for this variable is that if a limiter seems to be limiting the flow field in smooth regions and causing disturbances in the smooth field, to try and increase this value. For very larger values of this variable, say greater than 20, this essentially begins to turn off the limiter completely and will result in numerical instability and eventual crashing of the code. The optimal value is often problem-dependent, but the default value is a good starting choice most of the time.

Table 4: Limiters available in *Stream*

Option	Description	Default
venkatakrishnan	Standard Venkatakrishnan limiter	Y
barth	Barth-Jespersen limiter	N
NB	Nodal Barth-Jespersen limiter	N
none	No limiting on any equations	N
mlp	Multi-dimensional limiting process	N
mlp_pw	Multi-dimensional limiting process with pressure-weighting	N

References

11.5 Appendix: Equation Options

The run control file has variables that govern the solution of each governing equation that is to be solved in a simulation; these are the equation options variables. These variables are composed of an options list that generally contains parameters that give the user control over the way that each governing equation is solved. These sections below discuss the equation options for each governing equation in the base *Stream* code.

11.5.1 Momentum Equation

Options for controlling the solution of the momentum equations are specified using the `momentumEquationOptions` variable in the run control file.

Listing 37: Specified Momentum Equation Options

```
momentumEquationOptions: <linearSolver=SGS, relaxationFactor=0.5, maxIterations=5>
```

If one wishes to use the default values, this line could alternately be specified as follows:

Listing 38: Default Momentum Equation Options

```
momentumEquationOptions: <>
```

It is generally best to explicitly specify the governing parameters in the options list as shown in the first form so that one is actively aware of parameters involved in the simulation rather than having to remember the default values. This is generally true of the other governing equations as well. The momentum equation options variable must be specified for all simulations. The parameter `linearSolver` specifies the solver for the linearized momentum equation. All linear solvers are supported except for *HYPRE*. The parameter `relaxationFactor` governs how much of the solution is kept from the updated values after the solution of the linearized momentum equation at the current iteration. Valid values

are any number between zero (solution stays at previous iteration value) and one (solution set to values from the linear solve). Using the parameter `maxIterations`, one can specify the maximum number of iterations that are performed in the linear solver.

11.5.2 Pressure Correction Equation

Options for the pressure correction equation are set with the `pressureCorrectionEquationOptions` variable in the run control file.

Listing 39: Specified Pressure Correction Equation Options (default values shown)

```
pressureCorrectionEquationOptions: <linearSolver=HYPRE, relaxationFactor=0.1,␣
↪maxIterations=5>
```

The pressure correction equation options variable must be specified for all simulations. The parameter `linearSolver` specifies the solver for the pressure-correction equation. All linear solvers described in the previous section are supported. For simulations involving anything other than idealized model problems on high-quality 2-D meshes, one will generally want to use either the *PETSC* solver or the *HYPRE* solver. The parameter `relaxationFactor` governs how much of the solution is kept from the updated values after the solution of the pressure-correction equation at the current iteration. Valid values are any number between zero (pressure field not updated) and one (full pressure-correction value update to pressure field). Using the parameter `maxIterations`, one can specify the maximum number of iterations that are performed in the linear solver.

Two primary methods of solving the pressure correction equation are supported by *Stream*. The SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) and the SIMPLEC (Semi-Implicit Method for Pressure-Linked Equations-Consistent) algorithms are available. These methods can be selected by specifying either `SIMPLE` or `SIMPLEC` in the run control file using the `pressureBasedMethod` variable as shown below (default value shown):

```
pressureBasedMethod:  SIMPLE
```

11.5.3 Pressure Equation

When using the `SIMPLER` method in the iterative solver stream one must also specify options for the pressure equation using the variable `pressureEquationOptions` in the run control file in a manner like the following (default values shown):

Listing 40: Specified Pressure Equation Options

```
pressureEquationOptions: <linearSolver=HYPRE, relaxationFactor=0.1, maxIterations=5>
```

Options for the pressure equation are identical to those for the pressure-correction equation detailed above.

11.5.4 Energy Equation

Options for the energy equation are specified using the variable `energyEquationOptions` in the run control file in a manner like the following (default values shown):

Listing 41: Specified Energy Equation Options

```
energyEquationOptions: <linearSolver=SGS, relaxationFactor=0.5, maxIterations=5,
                        form=totalEnthalpy>
```

This variable need be specified only when running a compressible flow simulation. The `linearSolver` parameter specifies the solver for the energy equation. All linear solvers described in the previous section are supported. The next two options shown above are used in the same manner as described *above* for the momentum equation. The fourth option shown above specifies the form of the energy equation to use. In *Stream*, one may specify either `totalEnthalpy` for the total enthalpy form of the energy equation, `totalEnergy` for the total energy form of the energy equation, or `temperature` for the temperature form of the energy equation. For the `stream-piso` solver, the temperature form of the energy equation is not available.

11.5.5 Species Equation

For simulations involving multi-species mixtures, one must specify options for the species equations using the variable `speciesEquationOptions` in the run control file in a manner like the following (default values shown):

Listing 42: Specified Species Equation Options

```
speciesEquationOptions: <linearSolver=SGS, relaxationFactor=0.5, maxIterations=5>
```

The `linearSolver` parameter specifies the solver for the species equation. All linear solvers described in the previous section are supported. The remaining options shown above are used in the same manner as described *above* for the momentum equation.

11.5.6 Turbulence Equations

For turbulent simulations, one must specify options for the turbulence equations using the run control file variable `turbulenceEquationOptions` in a manner like the following (default values shown):

Listing 43: Specified Turbulence Equation Options

```
turbulenceEquationOptions: <linearSolver=SGS, relaxationFactor=0.5, maxIterations=5,
                            model=menterSST>
```

The `linearSolver` parameter specifies the solver for the turbulence equations. All linear solvers described in the previous section are supported. The next two options shown above are used in the same manner as described *above* for the momentum equation. Several different turbulence models can be chosen from using the `model` option. The section on *turbulence* covers all the valid options that can be selected for the `model` option.

During the non-linear solution process at any time step, it is possible for the turbulence equations to return non-physical values of the dependent variables k, the turbulent kinetic energy, and ω, the specific dissipation rate. Should these non-physical values not be limited, the simulation will immediately terminate due to floating-point exception. To prevent this from occurring one must specify physical fallback values that can be assigned for cells with non-physical values after the linear solve. The most robust way of continuing the simulation without causing un-necessary shock to the flow field has been found to simply assign free-stream values in place of the non-physical values that occur. This can be done using the run control file variables `kFreestream` and `omegaFreestream` as shown below (default values shown):

Listing 44: Freestream Values for Turbulence Equations

```
kFreestream: 1.0e-08
omegaFreestream: 10.0
```

While one may choose to accept the default values by not including these lines in the run control file, the preferred approach is to set these values to those used in the initial conditions, because free-stream values are commonly used in the initial condition for the variables k and ω and the location of the non-physical values of `k` and `omega` is sometimes in the vicinity of the edge of the boundary layer where the flow is transitioning back to free-stream.

11.6 Appendix: Linear Solvers

Linear solvers are used in the stream code to solve the linearized equations at each iteration in the solution process at a time step. Because the governing equations in both codes are solved in a sequential fashion (as opposed to a coupled fashion in most density-based methods), one can specify a different linear solver for each of the governing equations in the system of Navier-Stokes equations (momentum, pressure, energy, etc.) The five linear solvers which are currently available are shown in the table below. To select a linear solver for a given equation, one must use the `linearSolver` option in the corresponding equation options run control file variable. In order to facilitate the selection of linear solver parameters for all governing equations, three global run control file variables can be specified as follows (with default values shown):

Listing 45: Global Linear Solver Options

```
linearSolverTolerance: 1.0e-02
petscSolverName: GMRES
hypreSolverName: AMG
```

These global values are used as defaults for all governing equations, but they can be overridden. Below is an example of overriding the global default for the momentum equation.

Listing 46: Overriding the Global Linear Solver Options for the Momentum Equation

```
momentumEquationOptions: <linearSolver=SGS, relaxationFactor=0.5, maxIterations=5,
                          linearSolverTolerance=1.0e-03, petscSolverName=CG>
```

Local equation override is not currently supported for the `hypreSolverName` variable. As shown in the table below, only the `LSGS`, `PETSC` and `HYPRE` linear solvers use the specified linear solver tolerance to assess convergence. For these three solvers, the parameter `maxIterations` (specified for each governing equation) indicates the maximum number of iterations that will be performed before the solver kicks out. In general, the specified linear solver tolerance may be met before the maximum number of iterations is taken, in which case the solver terminates, declaring convergence. By contrast, the `SGS` and `FSGS` linear solvers will always perform the maximum number of iterations specified because the linear solver tolerance is not used.

Table 5: Available Linear Solvers

Solver	Description	Solver Names Supported	Solver Tolerance Used
SGS	Gauss-Seidel	N/A	No
FSGS	Cache-Optimized Gauss-Seidel	N/A	No
LSGS	Line Gauss-Seidel	N/A	Yes
PETSC	PETSc Library	GMRES, CG, BICG, CGS, BCGS	Yes
HYPRE	HYPRE Library	GMRES, AM	Yes

When using the HYPRE solver, the variable `hypreStrongThreshold` is used as follows (default value shown):

```
hypreStrongThreshold:  0.5
```

to set the value of the neighbor strong threshold, which has an important effect on the generation of the coarse multi-grid levels. One should generally use the highest value possible for this variable (range 0.25- 0.95) while still maintaining solution stability to substantially reduce CPU time. A starting value of 0.9 is recommended.

11.7 Appendix: Output Data

During the running of a simulation, there is a variety of information that is output by the code which can be used for monitoring the execution of the code as well as evaluating the flow solution. In the following sections, we give a detailed description of the information which is available.

11.7.1 Data Printed to Standard Output

When issuing the command to run the stream code, one should normally re-direct standard output and standard error to a log file so that this information is saved for both real-time monitoring and later analysis should some problem occur during execution of the code. One can do this by using a command like the following on the command line or in an execution script:

```
mpirun -np 1 stream -q solution caseName >& caseName.log &
```

Upon examination of the log file, one will see several types of information, each detailed in the following sections.

Residual Data

The primary means of evaluating whether the code is operating properly is to examine the residual data for the run. A residual line in the log file begins with the characters `R:` and is followed by the residual information. For the iterative solver *Stream*, the format of the residual line is as follows:

```
R: n it uRes vRes wRes ppRes eRes kRes omegaRes yRes
```

The table below provides the definitions of the residual variables shown above. Refer to the last column in the table for information which indicates the conditions under which the specific values will be output.

Table 6: Residual Variable Definitions

Variable	Description	Output
n	Time step number	Always
it	Iteration number	Always
uRes	Momentum eq. x-component residual	Always
vRes	Momentum eq. y-component residual	Always
wRes	Momentum eq. z-component residual	Always
ppRes	Pressure-correction residual	Always
eRes	Energy eq. residual	Compressible flow
kRes	Turbulent kinetic energy eq. residual	Turbulent flow
omegaRes	Specific dissipation rate eq. residual	Turbulent flow
yRes	Species eq. maximum residual	Multi-species mixture

For the solver, the residuals for all equations except pressure-correction are defined as the sum over all cells of the absolute value of the corresponding residual for each of the cells in the domain. The cell residual is defined as the right-hand side minus the left-hand side of the discretized cell equation. For the solver, the pressure correction residual

is computed as the sum over all cells of the absolute value of the discretized continuity equation for each of the cells in the domain.

By default, the residuals sent to standard output are not normalized. To normalize the residuals based on characteristic values pertinent to the problem at hand, one can use the run control file variable `referenceValue` as follows:

```
referenceValue:  <L=2.0, rho=10.0, v=3.0, h=1000.0, k=0.01,omega=2000.0>
```

This variable defines the pertinent reference values which are used to normalize the total solution residuals that are printed to standard output. Refer to the table below for a description of how these reference values are used. Note that there is no residual reference value required for species mass fraction since these values always remain in the range of zero to unity.

Table 7: Residual Normalization Reference Values

Residual Variable	Description
uRes, vRes, wRes	$\rho v^2 L^2$
ppRes	$\rho v L^2$
eRes	$\rho v h L^2$
kRes	$\rho v k L^2$
omegaRes	$\rho v \omega L^2$
yRes	$\rho v L^2$

Regardless of whether the residuals have been normalized or not, when monitoring the residual values to determine acceptable convergence of the simulation, use the following guidelines. For steady-flow simulations in which one is using `maxIterationsPerTimeStep=1`, **look for approximately three to four orders of magnitude drop in each of the residuals from their maximum value in the simulation** to indicate complete convergence. While running the simulation past this level of convergence will continue to refine the solution, changes in the solution after this point will generally be very minor. For unsteady simulations in which `maxIterationsPerTimeStep>1` one should also look for approximately three to four orders of magnitude drop in the residuals within each time step in order to eliminate the iterative (SIMPLE) error in the solution and give the full time-accurate behavior of the temporal differencing scheme.

Regarding complete convergence of a simulation, it is important to note that the above levels of residual drop can usually be achieved on model problems with ideal meshes. For non-model problems employing engineering meshes, it is often possible to achieve only two orders of magnitude residual drop, sometimes even less. This apparent lack of convergence is often caused by very localized problematic regions which may or may not impact a significant portion of the problem domain. To evaluate the suitability of the convergence, one will have to resort to more sophisticated measures such as a direct examination of the distribution of the cell residuals throughout the domain to determine the extent of non-convergence.

Integrated Data

During the simulation, a variety of integrated data is also redirected to standard output based on the value of the run control file variable `print_freq`, which is used as follows (default value shown):

```
print_freq:  100
```

If this line is not present in the run control file, then the default value will be used. Integrated data is output when the expression `(n mod print_freq)? = 0` evaluates to true, where n represents the time step number. There are two forms of data that are output, integrated volume data and integrated boundary data. Integrated volume data will appear as follows:

Listing 47: Integrated Volume Data

```
Integrated Volumetric Data (Complete Domain)
  total volume = 6.2100e-07 m^3
  total mass = 3.8311e-05 kg
  total energy = -1.6704e+01 J
  total enthalpy = -1.2421e+01 J
  total species masses: [H2=1.0000e-06,O2=3.0000e-06,H2O=3.4311e-05]
```

These values represent the total amount of the specified quantity that is currently in the complete domain and are computed based on a summation over all cells of the quantity in question. Total energy is defined as the total internal energy of the flow and does not include the kinetic energy of the bulk fluid motion. The same applies to total enthalpy. For incompressible flow simulations, these two quantities will be zero. In addition, for pure-fluid simulations, the total species masses information will not be output.

Integrated boundary data is provided for each individual boundary condition patch as well as boundary condition patch type. For example, for a two-dimensional incompressible lid-driven cavity flow simulation consisting of a box with a sliding lid and three no-slip walls, one might have the following boundaryCondition variable in the run control file:

Listing 48: Sample Boundary Condition Specification for Integrated Boundary Data

```
boundary_conditions:
<
BC_1=noslip, // left wall
BC_2=noslip, // right wall
BC_3=noslip, // bottom wall
BC_4=incompressibleInlet(v=-1.0 m/s), // lid
BC_5=symmetry, BC_6=symmetry // symmetry boundaries
>
```

In this case, one would see output for individual boundary condition patches such as the following:

Listing 49: Sample Specific Integrated Boundary Data

```
Integrated Boundary Data (BC_1)
  total area = 1.0000e+00 m^2
  mass transfer = 0.0000e+00 kg/s
  energy transfer = 0.0000e+00 W
  pressure force = -1.3380e-02 0.0000e+00 0.0000e+00 N
  viscous force = 2.7312e-06 1.6104e-03 -3.1117e-11 N
  total force = -1.3377e-02 1.6104e-03 -3.1117e-11 N
```

as well as for the collection of all no-slip boundaries such as the following:

Listing 50: Sample Boundary-Type Integrated Boundary Data

```
Integrated Boundary Data (noslip_BC)
  total area = 3.0000e+00 m^2
  mass transfer = 0.0000e+00 kg/s
  energy transfer = 0.0000e+00 W
  pressure force = -3.6021e-02 -3.2994e-06 0.0000e+00 N
  viscous force = 3.6243e-05 -9.7420e-04 2.4459e-11 N
  total force = -3.5985e-02 -9.7750e-04 2.4459e-11 N
```

It is important to note the conventions associated with these quantities. **Mass transfer is defined as the mass flow**

rate leaving the domain. Thus, for inlet boundaries, one should see a negative value for this quantity and a positive value for outlet boundaries (assuming no major re-entrant flow occurs at the outlet). **Energy transfer is defined as the sum of the convective and diffusive internal energy transfer rates and is positive for a net transfer leaving the domain through the boundary.** Thus, for no-slip boundaries, the energy transfer value amounts to simply the wall heat flux entering the boundary from the fluid. **The pressure, viscous and total forces are defined as forces by the fluid on the boundary.**

11.7.2 Field Data Written to Output Directory

During the simulation, nodal field data interpolated from the cell center and boundary face data is written to the output directory based on the value of the run control file variable `plot_freq`, which is used as follows:

```
plot_freq:  100
```

This variable is an optional variable in that there is no default value. If this variable is not included in the run control file, there will be no field data written to file. Field data is output when the expression `(n mod plot_freq)? = 0)` evaluates to true, where n represents the time step number. It is important to note that data is written at the beginning of the time step, so that the variables that are written represent the state of the simulation before the variables are updated at the new time step. The directory to which field data is written is called `/output` and can be found in the directory from which the simulation was initiated. This directory will be created if it is not already present. Field data is always written for default variables and optionally written for non-default variables based on user specification with the run control file variable `plot_output`, which is used in a manner like the following:

```
plot_output:  k, omega, viscosityRatio, laminarViscosity
```

The table below summarizes many of the variables which can be output, including their status as either a default or non-default variable. The default column in the table indicates variables that are automatically output when they applicable to a simulation. Default variables do not need to be specified in the `plot_output` line in the run control file.

Table 8: Common Field Variable Output

Variable	Description	Default
a	Speed of sound	Yes
cfl	CFL number	No
cp	Specific heat	No
hResidual	Energy eq. residual	No
hResidualTT	Energy eq. turn-over time	No
k	Turbulent kinetic energy	No
kclip	Turbulent kinetic energy clipped flag	No
kconduct	Thermal conductivity	No
kineticEnergy	Kinetic energy	No
kResidual	Turbulent kinetic energy eq. residual	No
kResidualTT	Turbulent kinetic energy eq. turn-over time	No
laminarViscosity	Material viscosity	No
omega	Specific dissipation rate	No
omegaclip	Specific dissipation rate clipped flag	No
omegaResidual	Specific dissipation rate eq. residual	No
omegaResidualTT	Specific dissipation rate eq. turn-over time	No
mix	Species mass fractions	Yes
pg	Gauge pressure	Yes
pPrime	Pressure-correction	No
pResidual	Pressure-correction eq. residual	No
pResidualTT	Pressure-correction eq. turn-over time	No
r	Density	Yes
t	Temperature	Yes
v	Velocity	Yes
viscosityRatio	Turbulent/laminar viscosity ratio	No
vort_mag	Vorticity magnitude	No
vResidual	Velocity eq. residual	No
vResidualTT	Velocity eq. turn-over time	No

Should one wish to prevent the output of default variables (to save disk space), one may use the run control file variable `plot_output_exclusive` in a manner like the following:

```
plot_output_exclusive:  r, v
```

In this example, the variables density and velocity are the only variables that are output. One final run control file variable of interest is the variable `plot_modulo`, which can be used to restrict the number of data files that are written to the `/output` directory. This variable is used as follows (default value shown):

```
plot_modulo:  0
```

Valid values include any integral numeric value greater than or equal to zero. When field data is scheduled to be written out according to the value of `plot_freq` described above, the following formats are used for the file names for scalar and vector variables.

Listing 51: Field Data File Name Formats

```
variable_sca.extension_casename
variable_vec.extension_casename
```

If `plot_modulo=0`, the extension is given the same value as the time step number. In this case, unique file names for each scalar and vector variable will be written to the `/output` directory for each time step for which data is requested. Thus, for example, if one wanted to save data every 1000th time step for the entire simulation, one would

set `plot_modulo=0` and `plot_freq=1000`. When the value of `plot_modulo` is greater than zero, the extension is computed by the expression (`extension=n mod plot_modulo`). Thus, if one wanted to save data every 100th time step for the most recent 1000 time steps, one would set `plot_modulo=1000` and `plot_freq=100`. In this case, the extension number would cycle between the values 100, 200, 300, 400, 500, 600, 700, 800, 900, 0 for the duration of the simulation.

11.7.3 Boundary Data Written to Output Directory

In addition to field data, certain limited nodal boundary data is written to the `/output` directory for no-slip boundaries. Consult the table below for a list of variables and their definitions. Boundary data output is also governed by the run control file variables `plot_freq` and `plot_modulo` which are used in the same manner as described in the section above for field data. Boundary data can also be written more frequently than field data by using the run control file variable boundary `plot_freq` in a manner like the following (default value shown):

```
boundary_plot_freq:  10000000
```

The large default value effectively prevents this variable from affecting boundary output unless the user specifies a more realistic value. As an example, assuming one had a value of `plot_freq=100` and set a value of `boundary_plot_freq=75`, boundary output would be scheduled both at every 100th time step as well as at every 75th time step. When boundary data is scheduled to be written out according to the value of `plot_output` and `boundary_plot_output`, the following formats are used for the file names for scalar and vector variables:

Listing 52: Boundary Data File Name Formats

```
variable_bnd.extension_casename
variable_bndvec.extension_casename
```

Computation of the extension is done in the same manner as described above for field data using the variable `plot_modulo`.

Table 9: Boundary Variable Output

Variable	Description	Type
`pw`	Wall pressure	Scalar
`qdot`	Heat flux to the wall	Scalar
`tau`	Wall shear stress	Vector
`tw`	Wall temperature	Scalar
`yplus`	Wall y^+ value	scalar

11.7.4 Probe Data

Solution information at selected points in the domain can be monitored through time by using the run control file variable `probe`. This variable allows one to place an arbitrary number of probes throughout the domain. The syntax for the usage of this variable is as follows:

Listing 53: Probe Variable Syntax

```
probe: <probe0=[0.0, 0.0, 0.0],
        probe1=[0.5 inch, 0.5 inch, 0.5 inch],
        probe2=[1.0 ft, 1.0 ft, 0.5 ft]>
```

where the vector location provided specifies the location of the probe. Units for the vector components can be optionally provided so the user does not have to do manual conversion if the grid is provided in dimensions other than meters. If

units for a probe are not provided, the default unit of meters is assumed for the probe. Units may be specified differently for each probe. Probe naming is up to the user, although the naming convention shown above is conventionally used. For each probe, a data file with the name of the probe as prefix and extension `.dat` will be created in the directory from which the simulation was initiated. In the above example, one would see files `probe0.dat`, `probe1.dat`, and `probe2.dat` when data first begins to be generated. The format for data entries written to the probe files is as follows:

```
n t T p rho a v pos dist y0 y1 ...
```

Refer to the table below for the definitions of the data entries. Probe data is not computed at the exact location of the probe via interpolation from nearby cells. The data at the center of the nearest cell to the probe is what is displayed in the probe file. The center of this cell is located at the value `pos`, and the distance from the cell center to the requested probe position is given by the value `dist`. While probe positions may be specified with arbitrary units, the values of `pos` and `dist` are always expressed in meters in the probe output files.

Table 10: Probe Variable Definitions

Variable	Description	Type	Units
`n`	Time step number	Scalar	–
`t`	Solution time	Scalar	s
`T`	Temperature	Scalar	K
`p`	Pressure	Scalar	Pa
`rho`	Density	Scalar	$\frac{kg}{m^3}$
`a`	Speed of sound	Scalar	$\frac{m}{s}$
`v`	Velocity	Vector	$\frac{m}{s}$
`pos`	Nearest cell center	Vector	m
`dist`	Distance from probe to cell center	Vector	m
`y0 y1 ...`	Species mass fractions	Scalar	–

The frequency of data output to the probe data files is controlled by the run control file variable `probe_freq`, which is used in a manner like the following:

```
probe_freq:  100
```

Probe data is output when the expression `(n mod probe_freq)? = 0` evaluates to true. With the value above, one would see data entries in the probe output files for time step 0 (initial condition) as well as time steps 100, 200, … . It is important to note that data values corresponding to a given entry represent the state of the solution at the beginning of the time step before the governing equations have been solved to update the solution. In other words, the data values correspond to the solution at the end of the previous time step.

11.7.5 Turnover Time

The Stream solver provides an output that assists users in assessing any convergence-related problems they may encounter while running cases. The turnover time metric provides the user with a cell-to-cell assessment of the convergence of each differential equation being solved.

The turnover time metric output variables use the following naming convention `<governingEquation>ResidualTT`. Examples of the variables that are available in the Stream solver are shown in the table below. The turnover time metric output variables are not output by default and must be specified in the `plot_output` variable.

Table 11: Turnover Time Metric Field Variable Output

Variable	Description	Module
pResidualTT`	Pressure-correction eq.	Main Code
vResidualTT	Momentum eq.	Main Code
tResidualTT	Temperature eq.	Main Code
hResidualTT	Total Enthalpy eq.	Main Code
eResidualTT	Total Energy eq.	Main Code
omegaResidualTT	Omega eq.	Main Code
kResidualTT	Turbulent Kinetic Energy eq.	Main Code
epsilonResidualTT	Epsilon eq.	Main Code

Consider the case of the turnover time scale metric for the pressure-correction equation.

$$\begin{aligned}\text{TurnoverTimeMetric} &= \log_{10}\left(\frac{\rho V_{\text{cell}}}{R_p}/\tau_{\text{ref}}\right)\\ &= \log_{10}\left(\frac{\tau_{\text{depletion}}}{\tau_{\text{ref}}}\right)\end{aligned}$$

Where V_{cell} is the cell volume, ρ is the cell density, R_p is the residual of the integrated differential equation over the cell (the subscript p simply denoting that it is the pressure-correction equation residual), and τ_{ref} is a characteristic time scale of the flow, such as the flow-through time. A flow-through time is simply an estimate of the time it would take for the fluid to flow through the domain. For many domains this can be estimated using the inlet velocity and the domain length scale. This characteristic time scale value must be provided by the user in the run control file using the `turnoverTimeScale` in the diagnostics variable as shown below. The units of the variable are assumed to be in seconds.

```
diagnostics:  <turnoverTimeScale=0.1>
```

The depletion timescale is defined as $\tau_{\text{depletion}} = \frac{\rho V_{\text{cell}}}{R_p}$ and is used here to simplify expressions.

The depletion time scale has units of seconds and can be thought of as the time it would take for the residual to deplete the cell of its contents for the governing equation. In this case, it would be the time for the residual to deplete the mass of the cell. The size of the residual is representative of the level of convergence that has been obtained for the governing equation in a particular cell. If a governing equation in a cell is converged to machine-accuracy at any time instant, the value of R_p will be essentially zero, implying an infinite depletion time. The ratio of the depletion time to the turnover reference timescale gives an intuitive sense of the level of convergence of the cell. The ratio should tend towards very large values in the case of a highly converged cell.

Stream outputs the logarithm of the ratio defined by the `TurnoverTimeMetric`. Based on this definition, the table below shows the correspondence between the convergence within a cell and the value of the turnover time metric output variable. This sometimes is not possible for flows with complex geometries, but for a converged simulation most of the cells in domain should have a value greater than zero.

Table 12: Rule-of-Thumb for Turnover Time Metric

Range	Assessment
`TurnoverTimeMetric` < -1	Poor convergence
-1 < `TurnoverTimeMetric` < 0	Fair convergence
0 < `TurnoverTimeMetric` < 1	Good convergence
`TurnoverTimeMetric` > 1	Excellent convergence

One caveat is to remember that this variable is provided as a quick metric to estimate the level of convergence within the simulation domain. In some instances, the characteristic reference timescale might vary within the domain such that

in certain regions the timescale is very large and in others it is very small. If you use the largest timescale to normalize the turnover time metric variable for a case like this, it can appear that the region with the much smaller timescale is not well converged when it is converged.

11.8 Appendix: Restarting Cases

When running a simulation, it is often useful to permanently save solution data at selected time steps so that, if necessary, the simulation may be re-initiated from one of these time steps. This data is referred to as restart data and consists of the discrete numerical solution for the primitive variables at all cell centers in the domain as well as the mass flux at all faces in the domain. Because all boundary condition data for the primitive variables is derived from cell-centered data, boundary data located at the center of faces on the boundaries for the primitive variables is not included in the restart data.

11.8.1 Restart Data

The directory where restart data is written is called `/restart` and can be found in the directory where the simulation was initiated. The frequency at which restart data is written to file is governed by the run control file variable `restart_freq`, which is used as follows:

```
restart_freq:  1000
```

This variable is an optional variable in that there is no default value. If this variable is not included in the run control file, there will be no restart data written. Restart data is output when the expression `(n mod restart f req)? = 0` evaluates to true, where `n` represents the time step number. As with output data, it is important to note that data is written at the beginning of the time step, so that the information that is written to the restart files represents the state of the simulation before the variables are updated at the new time step. No user input is required in terms of selecting which restart data is required to be written. This is automatically determined by the code based on the type of simulation that is being performed.

The run control file variable `restart_modulo` can be used to limit the amount of data that will accumulate in the restart directory. This variable is used as follows (default value shown):

```
restart_modulo:  0
```

Valid values include any integral numeric value greater than or equal to zero. When restart data is scheduled to be written out according to the value of `restart_freq` described above, the following formats are used for the file names:

Listing 54: Restart File Name Format

```
variable_hdf5.extension
```

If `restart_modulo=0`, the extension is given the same value as the time step number. In this case, unique file names for each variable will be written to the `/restart` directory for each time step for which data is requested. Thus, for example, if one wanted to save restart data for every 1000th time step for the entire simulation, one would set `restart_modulo=0` and `restart_freq=1000`. When the value of `restart_modulo` is greater than zero, the extension is computed by the expression `(extension = n mod restart_modulo)`. Thus, if one wanted to save data every 1000th time step for the most recent 10000 time steps, one would set `restart_modulo=10000` and `restart_freq=1000`. In this case, the extension number would cycle between the values 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 0 for the duration of the simulation. To perform a restart from saved data, place the restart time step number after the case name in a manner like the following:

```
mpirun -np 1 stream -q solution caseName 1000 >& caseName.log &
```

In this example, the code will use the restart data from the files `/restart/\* hdf5.1000` for the simulation initial condition rather than the initial condition information present in the run control file.

11.8.2 Restart Compressible Simulation from an Incompressible Simulation

There are certain situations in which it is sometimes helpful to run an incompressible simulation to establish a good flow field from a poor initial condition before proceeding on to the final compressible simulation. While this procedure is in general not necessary in practice, it can sometimes be used as a measure of last resort for getting a simulation started. Because the data in the `/restart` directory for an incompressible simulation does not include a file for temperature, one must use either the run control file variable `initialCondition` or `initialConditionRegions` to provide this information upon restart in compressible mode. It is important that one include all required entries in the selected initial condition variable to avoid code error handling (the code will complain that insufficient information has been provided for a compressible initial condition and terminate), however, only the temperature values will be obtained from the initial condition. All other values will be obtained from the incompressible simulation restart files. Upon restart, one should see the following message in the log file:

Listing 55: Restart Message

```
WARNING: No temperature restart file. Using temperature initial
condition value from .vars file.
```

It is important when using this restart technique that one uses a temperature initial condition that in conjunction with the pressure values in the pressure restart file results in a density value that is close to that of the incompressible simulation, to avoid restart shock. It should also be noted that this technique can only be used to restart from an incompressible simulation to a pure-fluid compressible simulation. Restart to compressible mixture material simulations is not currently supported.

11.8.3 Restarting a Turbulent Simulation from a Laminar Simulation

In some instances, if one suspects that the turbulence equations are having difficulty at start-up due to a poor initial condition, it may be helpful to establish a good flow field by running a laminar flow simulation before proceeding on to the final turbulent flow simulation. This can be accomplished by restarting in the normal manner, but because the restart files for the turbulence quantities are not available from the laminar simulation, one must provide this information from either the run control file variables `initialCondition` or `initialConditionRegions`. It is important to include all required entries in the selected initial condition variables to avoid code error handling features, however, only the turbulence quantities will be obtained from the initial condition. All other values will be obtained from the laminar simulation restart files. Upon restart, one should see the following messages:

Listing 56: Restart Message

```
WARNING: No k restart file. Using k initial condition value from
.vars file.

WARNING: No omega restart file. Using omega initial condition value
from .vars file.
```

11.8.4 Restarting a BDF2 Simulation from a BDF Simulation

In certain instances, one may be required to restart a second-order BDF2 simulation from a first-order BDF simulation. Such may be the case, for example, if one has a complex geometry where it is difficult to assign a good initial condition. Due to the poor initial condition, it may be impossible for the solution to stabilize when using second-order time integration. In this case, one would first run a certain number of time steps using first-order time integration, writing out restart data at an appropriate point, and then restart using second-order time integration. No active input by the user is required to execute this process, beyond the normal restart procedure, however, it is important to understand how the code handles this situation.

In a normal BDF2 run, when restart data is written to file, data is written for both the current time step and well as the previous time step, because this second-order time integration scheme employs two temporal solution levels. When a BDF2 restart is being made from a BDF simulation, data at the previous time level is no longer available. To circumvent this problem, the code assumes a zero temporal gradient condition upon restart and thus assigns the previous time level data from the current time level data which is available. Temporal accuracy is thus lost at the initiation of the restart.

11.9 Appendix: Thermodynamic Data

Thermodynamic properties for the fluid components involved in a simulation are specified in a chemistry model file. By convention, this file has the .mdl extension. To specify the model file for the simulation, use the run control file variable chemistry model in a manner:

```
chemistry_model:  air_5s17r
```

The code will look for the file air_5s17r.mdl. By default, the code first looks in the directory from which the simulation was initiated. If the file is not found in this location, the code will attempt to look in the location defined by the environment variable CHEMISTRY_DATABASE, if defined, which contains several chemistry models for commonly used mixtures. If the file is not found in this location, the code will terminate with an error condition. The chemistry model file is divided into two sections. The first section contains the thermodynamic data for each of the species involved in the simulation. The second section contains a list of the chemical reactions among the species. Information in the species section is always required, while the reaction section may be left empty for non-combusting flow simulations. In this section, we describe the contents of the species section in detail. Discussion of the reaction section of the model file is deferred to the section devoted to simulations with finite rate chemistry.

The species section contains the complete thermodynamic specification for each species in the simulation. In the following, we first discuss the basic structure of the species section, followed by a discussion of the thermodynamic models available for specifying the caloric equation of state for each species. The thermal equation of state for all species is the ideal-gas law. Non-ideal-gas equations of state can be used by loading the real-fluids module. This is discussed in the final section.

11.9.1 Species Definition

The definitions for all species in the simulation are contained within a single `species={};` block within the model file. An example of this section from the database model file `air_5s17r.mdl` looks as follows:

Listing 57: Example of a species section in a model file

```
species = {
   O2: <mf = 0.22> ;
   N2: <mf = 0.78> ;
   NO ;
   O ;
   N ;
   } ;
```

In this example, the file is declaring that there are five species in the simulation. Note that there is a minimum amount of information required to define the species in this case since most information for the species can be derived from pre-existing information in the species database. There are two distinct ways in which entries in the `species` section are interpreted by the code. If the `species` name is followed by an = character, then any information in the system database concerning this species is ignored.

The line below will specify the molecular mass, reference enthalpy, reference entropy, reference temperature, reference pressure and default mass fraction for the species H2, and over-ride any pre-existing information for this species in the database:

```
H2=<m=2.016, href=55749, sref=130751, Tref=300, Pref=101325, mf=1> ;
```

On the other hand, if the species name is followed by a `:` character, then the data specified is interpreted as augmenting the pre-existing data for that species. For example, the following line will augment the currently existing information for the species `H2` with a polynomial definition for the specific heat over the temperature range 75K to 300K:

Listing 58: Augmenting the existing information for the species H2

```
H2:<cp=[75.0,poly(40.4475, -3.01156e-01, 2.35251e-03, -7.42018e-06,
        8.35504e-09), 300.0]>
```

Multiple definition lines for a species can be present in the species section. For example, the complete definition for the species `H2` could look as follows:

Listing 59: Complete definition for the species H2

```
H2=<m=2.016> ;
H2:<href=55749, sref=130751, Tref=300, Pref=101325, mf=1> ;
H2:<cp=[75.0,poly(40.4475, -3.01156e-01, 2.35251e-03, -7.42018e-06,
        8.35504e-09), 300.0]>
```

Regardless of how the species are chosen to be defined, whether using the existing information in the species database and augmenting with additional data, or defining the species from scratch, it is important that all required thermodynamic variable are defined. The table below lists the variables that are used to specify the thermodynamic properties for each species.

Table 13: Variables for Species Thermodynamic Definition

Variable	Description	Units	Notes
m	Molecular Mass	amu	Automatically generated from database
n	Linear component of $e(T)$	–	Used in vibrational equilibrium model
theta_v	Vibrational Temperature	K	Used in vibrational equilibrium model
href	Reference Enthalpy	$\frac{J}{Kmol}$	Used to compute K_C
sref	Reference Entropy	$\frac{J}{(Kmol \cdot K)}$	Used to compute K_C
Tref	Reference Temperature	K	Used to compute various thermodynamic properties
Pref	Reference Pressure	Pa	Used to compute K_C
mf	Mass Fraction	–	Not used
cp	Curve Fit for Specific Heat	$\frac{J}{(Kmol \cdot K)}$	Vibration equilibrium used outside range

11.9.2 Creation of Model Files (`.mdl` files)

The chemistry model file (`.mdl` file) contains the complete thermodynamic specification for all the species in the simulation. In addition, if the flow is reacting, a complete specification of the reactions among the species must be provided. If one needs to create a new model file from scratch, the method of creating a model file from scratch is presented below.

A user does not need to manually create a `.mdl` file for every species. If you are working on a multi-species simulation and you need to create a `.mdl` file that contains many species, then you can use the following procedure to generate a `.mdl` file that can be edited to suit your specific needs.

1. Find an appropriate *Chemkin* formatted reaction mechanism that contains the species that you want to use. It can contain more species that what is needed, which is ok, because a subset of the output can be taken for use in a case without needed to use the entire output.
2. Have a set of *Chemkin* formatted thermodynamic and transport property database files that contain information about the species in your mechanism file.

The mechanisms are often paired with a set of thermodynamic and transport databases. These database files can vary in size depending on the complexity of the mechanism because reactions involving more species require more database entries. The utilities that *Stream* uses to create the species model file from the reaction mechanism, transport, and thermodynamic data files require those files to be in the *Chemkin* format. *Chemkin* is a proprietary software for modeling complex chemical kinetics, but the format is widely available on the internet.

We have a *Python* utility named `chemkin-converter.py` that can take a *Chemkin* formatted mechanism along with the thermodynamic and transport property database files and create a `.mdl` model file that includes species data and optionally reaction data. The utility can also take *Cantera* formatted YAML mechanism file and create a `.mdl` model file that includes species data and reaction data.

Below is a table of the arguments that can be passed to the script. If the script is run with no arguments, a list of the arguments and usage instructions will be printed to the screen.

Table 14: chemkin-converter.py arguments

Argument	Description	Optiona	Notes
casename	Desired name of `.mdl` file without the `.mdl` suffix	No	
mechanism	Full name of a *Cantera* `.yaml` mechanism file	Yes	Without this, only the species section of the `.mdl` file will be generated.
species_na	Text file containing species name remapping information	Yes	The left column is the species names in the `.mdl` file and the right columns ss the desired species name.
chemkin	A flag for whether to use the raw *Chemkin* input files to generate the species section of the `.mdl` file	Yes	If this flag isn't used, the mechanism argument becomes a required argument.
transport_tables	A flag for whether to generate a directory containing curve fits for species viscosity and thermal conductivity	Yes	If this isn't provided, the `.tran` file that the tool generates should be used by the user.

The following sections below detail the two methods that utilize *Chemkin* and *Cantera* to generate the model file.

Creation of Model Files (`.mdl`) directly from *Chemkin*

One mechanism resource that has been used frequently is the UCSD website: http://web.eng.ucsd.edu/mae/groups/combustion/mechanism.html. Download the *Chemkin* mechanism file and name it: `chem.inp`. Download the thermodynamic database file and name it: `therm.dat`. Finally download the transport property file and name it: `tran.dat`. In the *Stream* `bin` directory under the flamelet sub-directory(`/bin/flamelet/chemkin-converter`) run the `chemkin-converter.py` script in whichever directory you downloaded the files mentioned above and pass it the argument of the name that the script should give to the output `.mdl` file that gets generated.

For example, if you have a directory that has the `chem.inp`, `tran.dat`, and `therm.dat` files, you can generate an `.mdl` file that only has a species section by running the following command:

```
python <StreamInstallDirPath>/bin/flamelet/chemkin-converter/chemkin-converter.py
--casename case --chemkin
```

This command will generate a file called `case.mdl`. Inside this file will be a list of all the species that were in the mechanism in the standard model file form. The file will look like the image show below.

```
species = {
H2 = <mf=1.00000000e+00, m=2.01593995e+00, href=1.32815733e-02, sref=1.30678343e+05, Tref=2.98150000e+02, Pref=101325.0,
   cp=[2.00000000e+02,
      poly(1.94915660e+01,6.63527629e-02,-1.61947970e-04,1.67593892e-07,-6.13275497e-11),
      1.00000000e+03,
      poly(2.77472739e+01,-4.10748958e-04,4.15265347e-06,-1.49297605e-09,1.66499128e-13),
      3.50000000e+03]> ;
H = <m=1.00796998e+00, href=2.17993971e+08, sref=1.14715518e+05, Tref=2.98150000e+02, Pref=101325.0,
   cp=[2.00000000e+02,
      poly(2.07858500e+01,5.86437687e-12,-1.65947545e-14,1.91297692e-17,-7.71348204e-21),
      1.00000000e+03,
      poly(2.07858501e+01,-1.91930696e-10,1.34328097e-13,-3.93696666e-17,4.14218221e-21),
      3.50000000e+03]> ;
O = <m=1.59994001e+01, href=2.49169968e+08, sref=1.61057164e+05, Tref=2.98150000e+02, Pref=101325.0,
   cp=[2.00000000e+02,
      poly(2.63420499e+01,-2.72653718e-02,5.52326924e-05,-5.09508263e-08,1.75653711e-11),
      1.00000000e+03,
      poly(2.13630380e+01,-7.14818013e-04,3.48773750e-07,-8.32912281e-11,1.02128107e-14),
      3.50000000e+03]> ;
```

Fig. 1: Sample .mdl file showing the layout of the data contained the file.

The *Chemkin* formatted thermodynamic database files look like the following:

```
THERMO
   300.000  1000.000  5000.000
! GRI-Mech Version 3.0 Thermodynamics released 7/30/99
! NASA Polynomial format for CHEMKIN-II
! see README file for disclaimer
O                 L 1/90O   1               G   200.000  3500.000  1000.000    1
 2.56942078E+00-8.59741137E-05 4.19484589E-08-1.00177799E-11 1.22833691E-15    2
 2.92175791E+04 4.78433864E+00 3.16826710E+00-3.27931884E-03 6.64306396E-06    3
-6.12806624E-09 2.11265971E-12 2.91222592E+04 2.05193346E+00                   4
O2                TPIS89O   2               G   200.000  3500.000  1000.000    1
 3.28253784E+00 1.48308754E-03-7.57966669E-07 2.09470555E-10-2.16717794E-14    2
-1.08845772E+03 5.45323129E+00 3.78245636E+00-2.99673416E-03 9.84730201E-06    3
-9.68129509E-09 3.24372837E-12-1.06394356E+03 3.65767573E+00                   4
```

Fig. 2: Sample of a `.therm` file showing the general structure of data within it.

The *Chemkin* formatted transport property database files have the following structure:

```
AR                0   136.500     3.330     0.000     0.000     0.000
C                 0    71.400     3.298     0.000     0.000     0.000 ! *
C2                1    97.530     3.621     0.000     1.760     4.000
C2O               1   232.400     3.828     0.000     0.000     1.000 ! *
CN2               1   232.400     3.828     0.000     0.000     1.000 ! OIS
C2H               1   209.000     4.100     0.000     0.000     2.500
C2H2              1   209.000     4.100     0.000     0.000     2.500
C2H2OH            2   224.700     4.162     0.000     0.000     1.000 ! *
C2H3              2   209.000     4.100     0.000     0.000     1.000 ! *
```

Fig. 3: Sample of a `.tran` file showing the general structure of the data within it.

If you need a reaction section in the `.mdl` file, you need to pass a *Cantera* formatted YAML mechanism file to the script. If you have the `chem.inp`, `tran.dat` and `therm.dat` files already, you can generate a *Cantera* mechanism file via the following steps.

1. Install Cantera on your machine using: `pip install cantera`. This will install a version of the Cantera Python module on your machine. Additional tools are installed with this module, which you will now have access to.
2. Navigate to the directory containing the Chemkin files and run the following command: `ck2yaml --input chem.inp --transport tran.dat --thermo therm.dat --permissive` This will generate an output file called `chem.yaml`. This file will contain the species and reaction information that you will then provide to the `chemkin-converter.py` script to generate the `.mdl` file.
3. Call the `chemkin-converter.py` script with the `--mechanism` argument and pass it the name as follows: `python <StreamInstallDirPath>/bin/flamelet/chemkin-converter/chemkin-converter.py --casename case --mechanism chem.yaml --chemkin`

Creation of Model Files (`.mdl`) starting from *Cantera* YAML mechanism file

If you already have a *Cantera* formatted YAML mechanism file, you can generate a `.mdl` file that contains the species and reaction data using the following command:

```
python <StreamInstallDirPath>/bin/flamelet/chemkin-converter/chemkin-converter.py
--casename case --mechanism chem.yaml
```

The *Cantera* option only supports generating both the species and reaction sections of the `.mdl` file. If you only want the species section, you delete the reactions from the generated `.mdl` file.

11.9.3 Thermodynamic Models for Caloric Equation of State

The thermodynamic model for the caloric equation of state is selected using the run control file variable `thermodynamic_model`. One may choose any one of the following three specifications:

Listing 60: Example of a thermodynamic model specification

```
thermodynamic_model: vibrational
thermodynamic_model: curve_fit
thermodynamic_model: adaptive
```

If this variable is not present in the run control file, then the `adaptive` model is chosen by default. The `adaptive` model is not a model, but rather causes the code to use the `curve_fit` model if a C_P is specified or the `vibrational` model if a C_P is not specified.

The default caloric equation of state model is vibrational equilibrium model, which is the internal energy based on the assumption that the vibrational modes are in equilibrium. Assuming a harmonic oscillator for vibrational modes, the internal energy of the i[th] species in the mixture is given by the following equation:

$$e_i(T) = (h_f)_i + R_i \left[n_i T + \frac{\theta_{v,i}}{\theta_{v,i} \left(e^{\frac{\theta_{v,i}}{T}} - 1 \right)} \right]$$

The variables n_i and $\theta_{v,i}$ are obtained from the species specification variables `n` and `theta_v` respectively. The variable $(h_f)_i$ which represents the heat of formation, is computed from the species specification variables `href` and `Tref`.

The caloric equation of state can also be specified by providing curve-fit functions for C_P. The curve-fit is specified over temperature intervals using a list of temperatures with curve-fit functions specified between the temperature intervals. Currently two curve-fit functions are supported. The first is a fourth-degree polynomial given by the expression `poly(A,B,C,D,E)` where,

$$C_p = A + BT + CT^2 + DT^3 + ET^4$$

And the units are $\frac{J}{mol \cdot K}$. The following example shows the usage of this form in the definition of a C_P curve fit over three temperature intervals for hydrogen:

Listing 61: Polynomial curve-fit specification

```
H2: <cp=[75.0, poly(40.4475, -3.01156e-1, 2.35251e-3, -7.42018e-6, 8.35504e-9),
         300.0, poly(24.4709, 0.0289467, -6.46136e-05, 6.23552e-08, -2.09548e-11),
         1000.0, poly(25.4069, 0.004967, -1.39243e-08, -1.76658e-10, 2.09483e-14),
         5000.0]> ;
```

The second functional form, the Shomate form, is given by the expression `shomate(A,B,C,D,E)` where,

$$C_p = A + Bt + Ct^2 + Dt^3 + Et^{-2}$$

and where $T(K)/1000$ and C_P is in units of $\frac{J}{mol \cdot K}$. The following example shows the usage of this form in the definition of a C_P curve fit for H_2O from the NIST (National Institute of Standards and Technology) database, over two temperature intervals, one from 500-1700 Kelvin and another from 1700-6000 Kelvin.

Listing 62: Shomate curve-fit specification

```
H2O: <cp=[500.0, shomate(30.09200,6.832514,6.793435,-2.534480,0.082139),
         1700.0, shomate(41.96426,8.622053,-1.499781,0.098119,-11.15764),
         6000.0]> ;
```

It should be noted that in both of the above specifications where C_P is provided, by either the `poly()` or `shomate()` functional forms, the vibrational equilibrium model will be used at the temperatures below the lowest temperature interval where curve-fit data is not available, should it be necessary.

11.10 Appendix: Transport Properties

Transport properties are required for viscous flow simulations. These include both the laminar transport properties due purely to molecular motion as well as the turbulent transport properties due to the turbulent motions of a flow.

The run control file variables for specifying transport properties are the `transport_model` and the `diffusion_model` variables.

A high-level summary of the available options for the `transport_model` variable is given below:

Table 15: Transport Model Options

Option	Description
`none` or `module`	no transport model applied.
`sutherland`	Sutherland's law (with default properties for air).
`powerLaw`	A Power law dependence on temperature.
`const_viscosity`	Constant viscosity and thermal conductivity.
`chemkin`	Use a CHEMKIN formatted transport file (`.tran`) for multi-component flow viscosity and thermal conductivity properties.
`transportDB`	Use a set of data files that have curve fits of viscosity, thermal conductivity, and diffusion coefficients (`*_con.dat`, `*_vis.dat`, `*_.dif.dat`)
`database`	Species p,T database using Wilke's mixture rule.
`tabularEoS`	Use a database of files that contain a tabular fit of transport data from the `tabularEoS` module.

A high-level summary of the available options for the `diffusion_model` variable is given below:

Table 16: Diffusion Model Options

Option	Description
`const_diffusivity`	Constant diffusivity.
`chemkin`	Use a CHEMKIN formatted transport file (`.tran`) for multi-component flow diffusion coefficient properties.
`laminarSchmidt`	Diffusivity based on viscosity and a specified Schmidt number.
`unityLewisNumber`	Use the unity Lewis number assumption for computing the diffusion coefficient.
`transportDB`	Use a database of files that contain a tabular fit of transport data from the `tabularEoS` module.
`default`	Selects the diffusion model based on the transport model: `chemkin` when Chemkin is selected for `transport_model`, or `laminarSchmidt` otherwise.

The following sections go into more detail on the available options for the different specification types of transport properties.

11.10.1 Laminar Transport Properties

Laminar transport properties include the laminar viscosity, the thermal conductivity, and the mass diffusivity for each of the species for multi-species simulations. The run control file variable `transport_model` is used to specify the form of transport properties desired. One must always include `transport_model` in the run control file because there is no default value. Based on the value of this variable, other auxiliary variables may also be used. Several methods of specification are possible, as detailed in the following sub-sections.

Inviscid Flow

Inviscid flow is activated by setting the value of `transport_model` to none.

```
transport_model:  none
```

No other transport property variables are required with this specification.

Constant Properties

Constant transport properties for viscous flow simulations can be specified in the run control file.

Listing 63: Constant transport property specification

```
transport_model: none
mu: 1.0e-04
kcond: 0.1
```

The variable `mu` is used to set the laminar dynamic viscosity value (units $\frac{N \cdot s}{m^2}$). This variable has no default value and must be included in the run control file. The variable `kcond` is required only for compressible flows and is used to set the thermal conductivity value (units $\frac{W}{m \cdot K}$). This is also a required variable as there is no default value. For multi-species simulations with constant laminar viscosity and thermal conductivity, one should specify the laminar mass diffusivity using the laminar Schmidt number as follows:

```
laminarSchmidtNumber:  0.9
```

This variable need not be present in the run control file if the default value of `1.0` is desired.

Power Law Model

An option for specifying the viscosity and thermal conductivity variation with a power law dependence on temperature is available. The power law model is a simple model that is defined by the following relation.

$$\mu = \mu_{ref} \left(\frac{T}{T_{ref}} \right)^n$$

The required run control file variables for the power law model are as follows:

Table 17: Power Law Options

Variable	Description
`T_ref`	Reference temperature
`power`	The exponent in the power law model
`mu_ref`	Reference viscosity

Power law transport properties for viscous flow simulations can be specified in the run control file as follows.

Listing 64: Power law transport property specification

```
powerLawParam: <mu_ref=1.8e-5, T_ref=300, power=0.7>
```

11.10.2 Turbulent Transport Properties

The turbulent dynamic viscosity is computed directly from the turbulence models. Turbulent thermal conductivity and mass diffusivity are computed using turbulent Prandtl and Schmidt numbers, respectively. The default values are shown below along with the way to specify the variables in the run control file.

Listing 65: Default turbulent transport property values

```
turbulentPrandtlNumber: 0.7
turbulentSchmidtNumber: 0.95
```

These variables need not be present in the run control file if the default values are desired.

11.10.3 Multi-Species CHEMKIN Transport Properties

For multi-species simulations, a more detailed specification of the transport properties is required. A thermodynamic model file (`.mdl`) file is required and a dataset containing transport properties is required. The methods for generation of the thermodynamic model file as well as some of the transport properties databases is outlined here.

CHEMKIN Transport Properties

Specifying the `chemkin` option for the `transport_model` or `diffusion_model` run control file variables will result in transport property data being obtained from a CHEMKIN formatted transport file (called `*.tran`). The method for generating the CHEMKIN transport `.tran` file is outlined here.

Curve Fit Transport Properties

Specifying the `transportDB` option for the `transport_model` or `diffusion_model` run control file variables will result in transport property data being obtained from a set of data files that have curve fits of viscosity, thermal conductivity, and diffusion coefficients (`*_con.dat`, `*_vis.dat`, `*_.dif.dat`). These files are expected to be in the same directory as the run control file, and located in a folder named `transport`. The method for generating the curve fit transport data files is outlined here , specifically the section regarding the `--transport_tables` flag passed to the `chemkin-converter.py` utility.

11.11 Appendix: Velocity and Scalar Boundary Condition Specification

There are several ways of specifying velocity and scalar values for the inlet boundary conditions detailed in:ref:*Appendix C <appendix_c>*. The following sections on velocity and scalar boundary conditions apply specifically to the `incompressibleInlet`, `subsonicInlet`, `supersonicInlet` and `totalPressureInlet` boundary conditions.

11.11.1 Velocity Boundary Conditions

Specification of the velocity is accomplished using the `v=` option. In the following sections, we will use `incompressibleInlet`, although the forms shown apply identically to the other boundary condition types.

Constant Specification

With the constant velocity specification, the velocity for all grid faces on the boundary is assigned to the single specified value. The following example shows the three ways of assigning a constant velocity of 1 ft/s in the x-direction.

Listing 66: Types of constant velocity specification

```
BC_1=incompressibleInlet(v=0.3048 m/s)
BC_1=incompressibleInlet(v=[1.0 ft/s, 0.0, 0.0])
BC_1=incompressibleInlet(v=polar(1.0 ft/s, 0.0 deg, 0.0 deg)
```

Functional Specification

With the functional form of velocity specification, one can specify an arbitrary function of space and time for each of the velocity components. The velocity at each grid face on the boundary is assigned by evaluating the function at the coordinates of the face center.

Listing 67: Functional velocity specification

```
BC_1=incompressibleInlet(v=function(vX="-1.0+3.0*x+2.0*x*y+4.0*x*y/z-0.01*t",
                         vY="0.0", vZ="0.0"))
```

If any component (such as `vX`) is not specified, it is assumed to be zero. The mathematical functions are written using common programming symbols as shown above. Standard math functions such as `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`, `sinh()`, `cosh()`, `tanh()`, `exp()`, `sqrt()`, `log()`, and `log10()` may also be used in the function specification. At the current time, units are not supported, so values are assumed to be in `m/s`.

Cartesian Specification

With the Cartesian specification, one can specify a one-dimensional velocity profile with respect to a single coordinate direction (either x, y, or z). The following example shows how this form is used.

Listing 68: Cartesian velocity specification

```
BC_1=incompressibleInlet(v=cartesian(file="file.dat"))
```

The data file specified can be of any name. The format for this data file is shown in the figure below. In the figure, the variable `n` represents the number of data pairs in the file. A data pair consists of a xyz-coordinate and the components of a vector velocity value. The variable `f` represents a profile flag which tells which coordinate direction to interpolate with respect to. Use `f=0` to interpolate with respect to the x-coordinate direction. In this case, the y and z coordinate values in the file are ignored. Use `f=1` to interpolate with respect to y and f=2 to interpolate with respect to z. The velocity at each face on the boundary is assigned by interpolating within the file coordinate values and computing the associated interpolated velocity value. If the coordinate of a grid face center falls outside the range of the coordinate data in the file, then the limiting boundary values are used. For example, if `f=0` and $x_{\text{face}} < x_1$ then $v_{\text{face}} = v_1$ and if $x_{\text{face}} > x_n$ then $v_{\text{face}} = v_n$. Naturally, this implies that the coordinates of the points in the data file must be listed in

ascending order. Below is the format for cartesian velocity specification file.

$$
\begin{array}{cccccc}
n & f & & & & \\
x_1 & y_1 & z_1 & v_{x,1} & v_{y,1} & v_{z,1} \\
x_2 & y_2 & z_2 & v_{x,2} & v_{y,2} & v_{z,2} \\
x_3 & y_3 & z_3 & v_{x,3} & v_{y,3} & v_{z,3} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_n & y_n & z_n & v_{x,n} & v_{y,n} & v_{z,n}
\end{array}
$$

Axisymmetric Specification

Like the Cartesian specification, one can also specify a velocity profile with respect to a radial coordinate. The following example shows how this form is used.

Listing 69: Radial velocity specification

```
BC_1=incompressibleInlet(v=axisymmetric(file="file.dat"), referenceFrame=0)
```

As with the Cartesian specification *above*, the file specified can be of any name. The format for this axisymmetric velocity specification file is shown below.

$$
\begin{array}{cccc}
n & & & \\
r_1 & v_{x,1} & v_{y,1} & v_{z,1} \\
r_2 & v_{x,2} & v_{y,2} & v_{z,2} \\
r_3 & v_{x,3} & v_{y,3} & v_{z,3} \\
\vdots & \vdots & \vdots & \vdots \\
r_n & v_{x,n} & v_{y,n} & v_{z,n}
\end{array}
$$

In the figure above, the variable `n` represents the number of data pairs in the file. A data pair consists of a radial coordinate and the components of a vector velocity value. The velocity at each face on the boundary is assigned by interpolating within the file coordinate values and computing the associated interpolated velocity value. If the coordinate of a grid face center falls outside the range of the coordinate data in the file, then the limiting boundary values are used. For example, if $r_{\text{face}} < r_1$ then $v_{\text{face}} = v_1$ and if $r_{\text{face}} > r_n$ then $v_{\text{face}} = v_n$. It is important that the radial coordinates of the points in the data file be listed in ascending order. To compute the radial coordinate values of the boundary faces, an axis must be established that defines the location of `r=0`. At the current time, this is accomplished using the reference frame file. The format for this data file is shown below. To use the information in the reference frame file, one must first put the following line in the run control file:

```
referenceFrameFile:  file.dat
```

Note that any name may be used for the reference frame file and that there are no quotes surrounding the name as there are in the boundary condition specifications above. In the figure below, the variable `n` represents the number of reference frames defined in the file. Each reference frame includes an angular rotation rate Ω whose units are `rad/s` and the starting and ending coordinates of the axis. The angular rotation rate is not used in the specification of the axisymmetric profile but is used in other features of the code. The axis is defined as the directed line from the starting coordinate to the ending coordinate. The magnitude of the axis specified in the data file is not important, as the axis is internally normalized within the code. In addition, the reference frame numbering specified in the boundary condition starts from zero. So, if one wishes to use the second reference frame in the data file for the boundary condition, specify

`referenceFrame=1`. The format for the reference frame file is shown below.

$$
\begin{array}{cccccc}
n & & & & & \\
\Omega_1 & & & & & \\
X_{\text{start},1} & Y_{\text{start},1} & Z_{\text{start},1} & X_{\text{end},1} & Y_{\text{end},1} & Z_{\text{end},1} \\
\Omega_2 & & & & & \\
X_{\text{start},2} & Y_{\text{start},2} & Z_{\text{start},2} & X_{\text{end},2} & Y_{\text{end},2} & Z_{\text{end},2} \\
\vdots & & & & & \\
\vdots & & & & & \\
\Omega_n & & & & & \\
X_{\text{start},n} & Y_{\text{start},n} & Z_{\text{start},n} & X_{\text{end},n} & Y_{\text{end},n} & Z_{\text{end},n}
\end{array}
$$

11.11.2 Scalar Boundary Conditions

In this section we discuss the available methods for specifying scalar boundary condition values. Temperature will be used in the examples below, however, the same specifications apply directly all other scalar variables. Specification of the temperature is accomplished using the `T=` option. In the following examples, we will use `subsonicInlet`, although the forms shown apply identically to the other boundary condition types.

Constant Specification

With the constant temperature specification, the temperature for all grid faces on the boundary is assigned to the single specified value. The following example illustrates the use of this form:

```
BC_1=subsonicInlet(T=400 K)
```

Cartesian Specification

With the Cartesian specification, one can specify a one-dimensional temperature profile with respect to a single coordinate direction (either x, y, or z). The following example shows how this form is used.

Listing 70: Cartesian scalar specification

```
BC_1=subsonicInlet(T=cartesian("file.dat"))
```

See *the section above* for a detailed discussion of the file format and interpolation methods used. While the file format is like that for vector quantities, only a single scalar value for each interpolation data point is required, as shown below.

$$
\begin{array}{cccc}
n & f & & \\
x_1 & y_1 & z_1 & T_1 \\
x_2 & y_2 & z_2 & T_2 \\
x_3 & y_3 & z_3 & T_3 \\
\vdots & \vdots & \vdots & \vdots \\
x_n & y_n & z_n & T_n
\end{array}
$$

Axisymmetric Specification

Like the Cartesian specification, one can also specify a temperature profile with respect to a radial coordinate. The following example shows how this form is used.

Listing 71: Radial scalar specification

```
BC_1=subsonicInlet(T=axisymmetric("file.dat"), referenceFrame=0)
```

See *the section above* for a detailed discussion of the file format and interpolation methods used. While the file format is like that for vector quantities, only a single scalar value for each interpolation data point is required, as shown below. Specification of the reference frame is handled in an identical manner as described *earlier*.

$$
\begin{array}{ll}
n & \\
r_1 & T_1 \\
r_2 & T_2 \\
r_3 & T_3 \\
\vdots & \vdots \\
r_n & T_n
\end{array}
$$

11.12 Appendix: Solving the Pressure-Correction Equation

In the iterative solver stream, the most CPU intensive step in the solution algorithm is the solution of the pressure-correction equation. Following a successful run of the code, one can examine the debug files in `/debug` under the directory from which the simulation was initiated to see to top ten most expensive rules in the computation. For a single process run, the debug file is simply called `debug`. For multi-process runs, the debug files are named `debug.*`, where the wildcard represents the process number. For a typical run, especially multi-process runs, it is not uncommon to observe the pressure correction solution using as much as 50% to 75% of the entire simulation CPU time. While such usage may be justified for certain simulations, quite often one can reduce the computational expense of solving the pressure correction equation without incurring any detrimental effects on convergence within the time step. Under optimal conditions with the right selection of solver parameters, one may be able to reduce the expense of solving the pressure-correction equation to no more than 25% to 30% of the total CPU time, which can result in a substantial reduction is simulation turn-around time compared to using default solver parameters which are selected for robustness purposes.

In *Stream*, the pressure-correction equation is a highly approximate equation whose sole purpose is to nudge the flow solution back towards continuity satisfaction. Due to the approximate nature of the equation, it is often simply not worth over-solving the equation because the additional information may not prove to be of any value. In fact, over solution of the pressure-correction equation in the initial time steps of a simulation (when one has a bad initial condition) is a common cause of solution divergence, especially when using `SIMPLE` (as opposed to `SIMPLEC`). Thus, for several reasons, one should attempt to find the optimal balance where the pressure-correction equation is being solved "enough". To find this optimal point, one can use a series of sample runs on either the actual problem or a similar problem and vary the level of solution of the pressure-correction equation. Starting from the default solution parameters, if one reduces the level of solution of the pressure-correction equation and sees no detrimental effect on the level of convergence of the system of equations within the time step, then the new solution parameters are an improvement. One can repeat this process until it is evident that the convergence of the system of equations within the time step is being affected in a negative manner by the reduced work in the pressure-correction equation. At this point, one would have to increase the number of iterations to make up for the loss of convergence to offset the lack of convergence due to the under-solution of pressure-correction. This primary point is that there is a general trade-off between a larger number of cheap iterations and a smaller number of expensive iterations when using *Stream*. With experience, the optimal location in this trade-off will become evident.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Pope2000] S. B. Pope, Turbulent Flows, Cambridge University Press, 2000.

[Hart2016] J. Hart, "Comparison of Turbulence Modeling Approaches to the Simulation of a Dimpled Sphere," *Procedia Engineering,* vol. 147, pp. 68-73, 2016.

[Stre2001] M. Strelets, "Detached Eddy Simulation of Massively Separated Flows," in *39th AIAA Fluid Dynamics Conference and Exhibit*, Reno, 2001.

[MeKu2002] F. R. Menter and M. Kuntz, "Adaptation of Eddy-Viscosity Turbulence Models to Unsteady Separated Flow Behind Vehicles," in *Lecture Notes in Applied and Computational Mechanics*, Springer, Berlin, 2002.

[GGSM2012] M. S. Gritskevich, A. V. Garbaruk, J. Schütze and F. R. Menter, "Development of DDES and IDDES Formulations for the k-omega Shear Stress Transport Model," *Flow, Turbulence and Combustion,* vol. 88, pp. 431-449, 2012.

[LaSp1974] Launder, B.E.; Spalding, D.B. (March 1974). "The numerical computation of turbulent flows". Computer Methods in Applied Mechanics and Engineering. vol. 3, issue 2, pp. 269-289

[SLSY1995] Shih, T.-H.; Liou, W.W.; Shabbir, A.; Yang, Z. (1995). "A new k- eddy-viscosity model for high Reynolds number turbulent flows". Computers & Fluids. vol. 24, issue 3, pp. 227-238

[MeFB1998] C. L. Merkle, J. Z. Feng and P. E. O. Buelow, "Computational Modeling of the Dynamics of Sheet Cavitation," in *Proceedings of the 3rd International Symposium on Cavitation*, Grenoble, France, 1998.

[HoAU2007] A. Hosangadi, V. Ahuja and R. J. Ungewitter, "Analysis of Thermal Effects in Cavitating Liquid Hydrogen Inducers," *Journal of Propulsion and Power,* vol. 23, no. 6, pp. 1225-1234, 2007.

[GeZB2004] A. G. Gerber, P. J. Zwart and T. Belamri, "A Two-Phase Flow Model for Predicting Cavitation Dynamics," International Conference on Multiphase Flow, Yokohama, Japan, 2004.

[ScSS2008] G. H. Schnerr, I. H. Sezal and S. J. Schmidt, "Numerical investigation of three-dimensional cloud cavitation with special emphasis on collapse induced shock dynamics," *Physics of Fluids,* vol. 20, 2008.

[Pete2000] N. Peters, Turbulent Combustion, Cambridge University Press, 2000.

[ArCa1968] J. C. Armour, J. N. Cannon, Fluid flow through woven screens 1968.

[Cady1973] E. C. Cady, Study of Thermodynamic Vent and Screen Baffle Integration for Orbital Storage and Transfer of Liquid Hydrogen, 1973.

[NiNe2003] R. H. Nichols and C. C. Nelson, "Application of hybrid rans/les turbulence models," in *Technical Report*, 2003.

[Venk1993] V. Venkatakrishnan, "On the Accuracy of Limiters and Convergence to Steady-State Solutions," 31st Aerospace Sciences Meeting, Reno, 1993.

[BaJe1989] T. Barth and D. Jespersen, "The Design and Application of Upwind Schemes on Unstructured Meshes," 27th Aerospace Sciences Meeting, Reno, 1989.

[Zhan2018] J. L. B. C. Fan Zhang, "Modified multi-dimensional limiting process with enhanced shock stability on unstructured grids," *Computers & Fluids,* vol. 161, pp. 171-188, 2018.