



Glenn Flux Reconstruction (GFR) Version 2.0 User Manual

Seth C. Spiegel and Dennis A. Yoder
Glenn Research Center, Cleveland, Ohio

NASA STI Program Report Series

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- **TECHNICAL PUBLICATION.**
Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.**
Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain

minimal annotation. Does not contain extensive analysis.

- **CONTRACTOR REPORT.**
Scientific and technical findings by NASA-sponsored contractors and grantees.
- **CONFERENCE PUBLICATION.**
Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- **SPECIAL PUBLICATION.**
Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.**
English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at <http://www.sti.nasa.gov>

NASA/TM-20260006232



Glenn Flux Reconstruction (GFR) Version 2.0 User Manual

Seth C. Spiegel and Dennis A. Yoder
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

August 2026

Acknowledgments

This research was sponsored by NASA's Transformational Tools and Technologies (TTT) Project of the Transformative Aeronautics Concepts Program under the Aeronautics Research Mission Directorate. Resources supporting this work were provided by the NASA High-End Computing (HEC) Program through the NASA Advanced Supercomputing (NAS) Division at Ames Research Center. The authors would also like to thank our colleague H. T. Huynh who created of the flux reconstruction method, and we wish him well on his retirement after over 40 years at NASA Glenn Research Center.

Trade names and trademarks are used in this report for identification only. Their usage does not constitute an official endorsement, either expressed or implied, by the National Aeronautics and Space Administration.

Level of Review: This material has been technically reviewed by technical management.

This report is available in electronic form at <https://www.sti.nasa.gov/> and <https://ntrs.nasa.gov/>

NASA STI Program/Mail Stop 050
NASA Langley Research Center
Hampton, VA 23681-2199

Glenn Flux Reconstruction (GFR) Version 2.0 User Manual

Seth C. Spiegel
Dennis A. Yoder
NASA Glenn Research Center

Contents

1	Overview and Features	7
2	Introduction	9
2.1	Flux Reconstruction	9
2.1.1	1D Example	9
2.1.2	Solution Polynomial	12
2.1.3	Discontinuous Flux Function	13
2.1.4	Interface Common Flux Values	13
2.1.5	Left/Right Flux Jumps	14
2.1.6	Correction Functions	14
2.1.7	Continuous Flux Function	15
2.1.8	Derivative of the Continuous Flux Function	15
2.1.9	Time Advancement	15
2.2	Solution Points	16
2.3	Quadrature Points	17
2.4	Output Points	17
2.5	Solution Accuracy and Grid Refinement	17
3	Geometry and Mesh	19
3.1	Surface Curvature	19
3.2	Element Types	19
3.3	Grid Scaling	20
3.4	Mesh Resolution Requirements for Large-Eddy Simulation	20
4	Reference Frame Rotation	23
5	Physical Models	26
5.1	Euler and Navier-Stokes Equations	26
5.2	Equation of State	27
5.3	Viscosity	27
5.4	Synthetic Eddy Method (SEM)	28
5.4.1	SEM Description and Theory	28
5.4.2	SEM Usage	33
5.4.3	SEM Frequently Asked Questions	44
6	Reference Conditions	47
7	Boundary Conditions	49
7.1	How to Specify Boundary Conditions	49
7.1.1	Boundary Condition Values for <code>Set_BC_Value</code>	49

7.1.2	Fully Defining the Boundary Condition State	50
7.1.3	Situational Recommendations	51
7.1.4	Setting Boundary Condition through Name Match	52
7.2	Inflow Boundaries	53
7.2.1	General Inflow Boundary Condition	53
7.2.2	Subsonic Inflow Boundary Condition	54
7.2.3	Supersonic Inflow Boundary Condition	57
7.2.4	Synthetic Eddy Method (SEM) Boundary Condition	58
7.3	Outflow Boundaries	58
7.3.1	General Outflow Boundary Condition	58
7.3.2	Subsonic Outflow Boundary Condition	58
7.3.3	Supersonic Outflow Boundary Condition	59
7.4	Farfield Boundaries	59
7.4.1	Characteristic / Riemann-Invariant Boundary Condition	59
7.4.2	General Free-flow Boundary Condition	61
7.4.3	Freestream Farfield Boundary Condition	62
7.4.4	Fixed Farfield Boundary Condition	62
7.5	Wall Boundaries	62
7.5.1	Inviscid/Slip Wall	62
7.5.2	Adiabatic Viscous/No-Slip Wall	62
7.5.3	Isothermal Viscous/No-Slip Wall	64
7.6	Numerical Boundaries	64
7.6.1	Symmetry Boundary Condition	64
7.6.2	Periodic Coupled Boundaries	64
8	Time Stepping	71
8.1	Time Stepping Methods	71
8.1.1	Highly Recommended Runge-Kutta Methods	71
8.1.2	Other Runge-Kutta Methods	72
8.1.3	<i>NOT</i> Recommended Runge-Kutta Methods	73
8.2	Time Stepping Algorithms	73
8.2.1	CFL-based Time Stepping	73
8.2.2	Constant Global Time Stepping	75
8.2.3	Adaptive Time Stepping	75
8.3	Controlling the Time Step Size in GFR	76
8.3.1	Ramping	76
8.3.2	Factors that Indirectly Affect Maximum Stable Time Step Size	76
8.3.3	Finding a Near-Optimal Stable CFL	77
9	Numerics	79
9.1	Aliasing Error	79
9.2	Filtering	80
9.3	Troubled Cell Markers	82
9.3.1	Resolution Marker	82
9.3.2	Jump Marker	83
9.3.3	MLP Marker	83
9.3.4	Scaling the Marker Results to Identify Troubled Cells	84

10 Files	85
10.1 Namelist Input	85
10.2 Grid Input	85
10.2.1 CGNS	87
10.2.2 Gmsh	93
10.2.3 Plot3D	93
10.2.4 Interpolation Grid	95
10.3 Solution Input	95
10.4 Restart Input/Output	95
10.5 Display Output	96
10.6 Residual Output	96
10.7 Solution Output	96
10.8 Time-Averaging Output	97
10.9 Probes Output	98
10.10 Simulation Interrupt/Control Files	99
10.10.1 STOP_GFR	100
10.10.2 ADJUST_CFL	100
10.10.3 START_TIME_AVERAGING	101
10.10.4 DUMP_RESTART	101
10.10.5 DUMP_SOLUTION	101
11 User Input	103
11.1 nml_version	104
11.2 nml_grid	105
11.3 nml_reference	108
11.4 nml_physics	110
11.5 nml_initialization	111
11.6 nml_timestep	114
11.7 nml_bc	119
11.7.1 Boundary Condition Relaxation Inputs	123
11.7.2 Boundary Condition SEM Inputs	123
11.7.3 Boundary Condition Periodicity Inputs	124
11.8 nml_numerics	127
11.9 nml_limiter	132
11.10 nml_time_ave	138
11.11 nml_output	140
11.12 nml_output_variables	146
11.13 nml_sem	151
11.14 nml_chemistry	153
11.15 nml_testcase_channel	156
11.16 nml_testcase_pulse	158
11.17 nml_testcase_sod	159
11.18 nml_testcase_tcf	160
11.19 nml_testcase_vortex	162
11.20 nml_interpolation	164
11.21 nml_utility	165
11.22 nml_debug	167
11.23 Changes from Prior Version	169

12 Test Cases	179
12.1 1D Density Pulse Advection	179
12.2 1D Jiang-Shu Density Pulse Advection	180
12.3 1D Sod Shock Tube	180
12.4 1D Shu-Osher Problem	181
12.5 2D Inviscid Bump	182
12.6 2D Isentropic Vortex	183
12.7 2D Couette Flow	184
13 Limitations	185
13.1 Geometry and Mesh	185
13.1.1 Collapsed Elements	185
13.2 Synthetic Eddy Method	185
13.2.1 Inflow Plane	185
13.2.2 Repeatability of Random Seed	185
13.3 Boundary Conditions	185
13.3.1 Periodic Boundaries	185
A Polynomials	186
A.1 Legendre Polynomials	186
A.2 Radau Polynomials	187
References	188
Index	193

List of Tables

1	Degrees of freedom of supported element types.	20
2	Grid requirements for wall-resolved large-eddy simulation.	22
3	GFR boundary conditions.	49
4	Boundary condition name strings.	54
5	Recommended Runge-Kutta Methods.	72
6	Other Runge-Kutta Methods.	72
7	Not Recommended Runge-Kutta Methods.	73
8	Example maximum stable CFL relative to solution order.	77
9	Assumed mapping between CGNS and GFR boundary conditions.	92
10	Supported boundary condition names in Plot3D boundary condition file.	94
11	Time-averaged variables output to CGNS.	98
12	Permissible values for <code>Marker_Option</code>	135
13	Permissible values for <code>Marker_Variable</code>	136

List of Figures

1	Modal and nodal representation of a $\mathcal{P}3$ polynomial.	12
2	Left and Right Radau polynomials.	15
3	1D solution point distributions vs polynomial order, $\mathcal{P}1 - \mathcal{P}4$	16
4	2D triangle “Gauss” solution point distributions vs polynomial order, $\mathcal{P}1 - \mathcal{P}4$	16

5	Supported element types.	19
6	Energy spectra for a turbulent flow.	21
7	Reference Frames	23
8	Illustration of an inflow profile for a flat plate.	29
9	Illustration of the SEM virtual volume on flat plate inflow plane.	30
10	Illustration of the boundary layer section of the inflow profile.	31
11	Illustration of the SEM synthetic eddies.	31
12	Recommended SEM length scale and convective velocity.	44
13	Flow angles.	47
14	Example of aliasing error.	79
15	Exponential filter function, σ	80
16	Jump Marker Notation	83
17	Scaled Marker	84
18	Pointwise boundary condition settings for the spacecraft cabin ventilation fan grid.	88
19	Pointwise boundary condition settings for the THX3 grid.	90
20	Ignition source decay.	155
21	Density pulse test case.	179
22	Jiang-Shu density pulse test case.	180
23	Sod shock tube test case.	181
24	Shu-Osher test case.	181
25	Inviscid Gaussian bump test case.	182
26	Isentropic Euler vortex test case.	183
27	Couette flow test case.	184
28	Legendre polynomials and Gauss points.	187

Listings

1	Example GFR inputs for the spacecraft cabin ventilation fan grid.	89
2	Example GFR inputs for the THX3 grid.	91
3	Plot3D interface cut file example.	93
4	Plot3D boundary conditions file example.	94
5	Namelist cfl_adjustment.	100
6	Namelist nml_version.	104
7	Namelist nml_grid.	105
8	Namelist nml_reference.	108
9	Namelist nml_physics.	110
10	Namelist nml_initialization.	111
11	Namelist nml_timestep.	114
12	Namelist nml_bc.	119
13	Namelist nml_numerics.	127
14	Namelist nml_limiter.	132
15	Namelist nml_time_ave.	138
16	Namelist nml_output.	140
17	Namelist nml_output_variables.	146
18	Namelist nml_sem.	151
19	Namelist nml_chemistry.	153
20	Namelist nml_testcase_channel.	156
21	Namelist nml_testcase_pulse.	158

22	Namelist nml_testcase_sod.	159
23	Namelist nml_testcase_tcf.	160
24	Namelist nml_testcase_vortex.	162
25	Namelist nml_interpolation.	164
26	Namelist nml_utility.	165
27	Namelist nml_debug.	167

1 Overview and Features

The Glenn Flux Reconstruction (GFR) code is a high-order computational fluid dynamics (CFD) flow solver based on the flux reconstruction (FR) method.¹ The FR approach offers a simple, efficient, and easy-to-implement method for solving the differential form of the Navier-Stokes equations on unstructured grids and is accurate to an arbitrary order.¹⁻⁷ This makes it well-suited for scale-resolving simulations (SRS), e.g., large-eddy simulations (LES), of complex aeropropulsion flows. The code's performance demonstrates good parallel scaling, as well as an implementation of the FR method with a computational cost/degree-of-freedom/time-step that is essentially independent of the solution order of accuracy for structured geometries.⁸

Listed below are the primary features and capabilities of GFR. These are discussed in detail in the following sections.

- Element Types
 - * 1d (line)
 - * 2d (triangle, quadrilateral)
 - * 3d (tetrahedron, pyramid, prism, hexahedron)
 - * Collapsed 2d edges and 3d faces are not supported.
- Grid File Formats
 - * Plot3D
 - with support for high-order curved meshes through agglomeration
 - * CGNS
 - with native support for high-order curved meshes
 - * Gmsh
 - (ASCII Legacy version 2 only) with native support for high-order curved meshes
- Automatic domain decomposition via METIS
- Parallel processing via MPI
- Reference Frame
 - * Fixed or Rotating Frame
- Physical Models
 - * Euler or Navier-Stokes Equations
 - * Finite-Rate Chemistry
- Explicit Runge-Kutta Time Advancement
 - * Industry Standard Runge-Kutta Methods
 - Classic, Carpenter/Kennedy Low-Storage 5-stage/4th-order, SSP 3/3, SSP 5/4
 - * Optimized Runge-Kutta Methods⁹
 - Vortex-Optimized 11-stage/3rd-order, Vortex-Optimized 15-stage/4th-order, etc.
 - * Embedded-Pair Optimized Runge-Kutta Methods with Adaptive Time Stepping¹⁰
 - Optimized 9-stage/4th-order with embedded 3rd-order, etc.
- Numerical Stability Treatments and Shock-Capturing
 - * De-aliasing
 - * Filtering
 - * Limiting
- Boundary Conditions
 - * Inviscid Wall
 - * Adiabatic Viscous Wall

- * Isothermal Viscous Wall
- * Coupled Translated and/or Rotated Periodic Boundaries
- * Subsonic Inflow and Outflow
- * Supersonic Inflow and Outflow
- * Unsteady Turbulent Inflow using Synthetic Eddy Method
- Post-processing and Solution Visualization
- Time Averaging
 - * Parallel CGNS output of instantaneous and time-averaged solution data
 - Approximated high-order solutions using linear sub-cells (compatible with most visualization software supporting CGNS file format)
 - Native high-order solutions (limited compatibility with visualization software, e.g., Tecplot 360 version 2023 or newer)

2 Introduction

Accurate and efficient numerical methods for the simulation of complex fluid flows are needed to reduce the design and development costs of aerodynamic vehicles and aeropropulsion systems. One of the fundamental challenges is prediction of the turbulent motion within the flow and its effect on the overall motion of the fluid. Scale-resolving methods such as large-eddy simulation (LES) and direct numerical simulation (DNS) offer great promise in predicting turbulence. However, due to the high Reynolds numbers involved and resulting small turbulent scales, very fine spatial and temporal resolution is necessary.

For these simulations, the required cell count and computational cost of traditional second-order numerical schemes quickly become prohibitive. High-order numerical schemes can overcome this limitation. However, such schemes have historically only been found in structured grid methods with large stencils. These methods have numerous drawbacks, such as: difficulty in generating structured grids for complex configurations, high sensitivity to grid quality, complex boundary condition implementation due to the wide stencils, and poor scalability for parallel computing. These drawbacks severely limit the applicability of such methods to complex aeropropulsion configurations.

The flux reconstruction (FR) method provides a simple, efficient, and easy-to-implement method for solving the Navier-Stokes equations on unstructured grids and is accurate to an arbitrary order. By employing the high-order FR method with LES on unstructured grids, limitations related to structured grid generation, grid resolution, and computational efficiency can be overcome.

2.1 Flux Reconstruction

The flux reconstruction method is a simple and efficient framework for high-order numerical schemes of conservation laws. It uses a set of polynomial basis-functions to represent the solution's conserved variables within each computational cell or element of the fluid domain. This polynomial is used to compute the flux of the flow through the cell. Corrections are applied at each cell face to ensure conservation. The general FR framework can be used to express various other methods such as discontinuous Galerkin (DG), spectral difference, and spectral element methods. However, the specific FR implementation in GFR is most comparable to nodal DG methods.

Theoretical development of the flux reconstruction method can be found in Refs. 1–3,6. Details of the FR implementation in GFR and validation of the code are discussed in Refs. 8,11,12. The interested reader is directed there for full details of the method.

2.1.1 1D Example

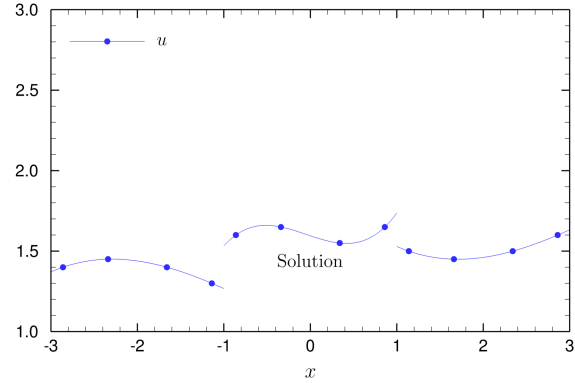
In order to provide the user with a basic understanding of how the FR method works, we will consider the one-dimensional advection equation:

$$u_t + f_x = 0 \qquad u = u(x, t) \qquad f = f(u) \qquad (1)$$

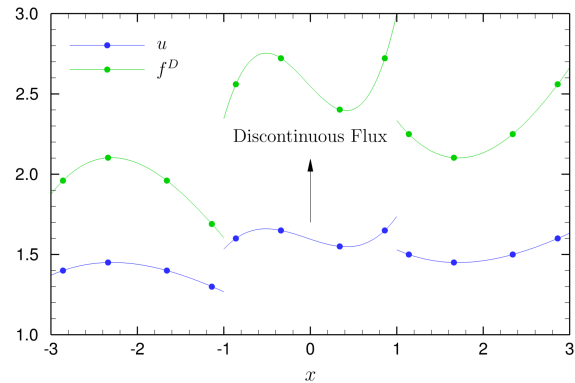
where the subscripts x and t denote partial differentiation with respect to space and time, respectively. The flux f is a (potentially nonlinear) function of the conserved solution variable(s) stored in u . The volume of the simulation domain is divided into an arbitrary number of nonoverlapping cells, and a mapping function, $r = \chi(x)$, is used to transform each cell from physical space to a common reference space/element. With r denoting the local coordinate variable that varies on the reference element $\mathbf{I} = [-1, 1]$, the inverse function $x = \chi^{-1}(r)$ similarly maps the reference element back to physical space.

The fundamental steps in the FR method can briefly be summarized as follows:

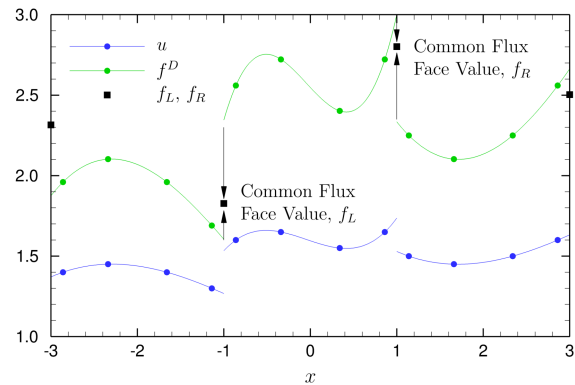
1. Represent the solution within each element as a polynomial of degree $\mathcal{P}$ defined by $\mathcal{P}+1$ solution points. The solution may be discontinuous across cell interfaces, even in the absence of shocks.



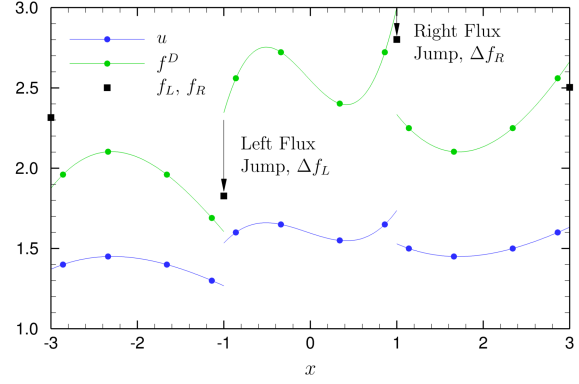
2. Compute the **discontinuous flux function** at the quadrature points. For this example, assume those are the same as the solution points.



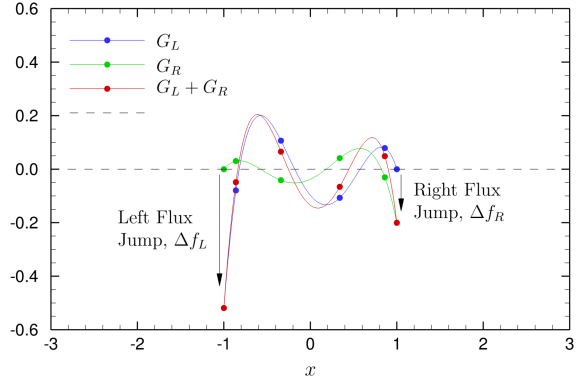
3. Compute a **common flux value** at the interfaces between cells using the solutions from the left and right cells, (u_L, u_R) .



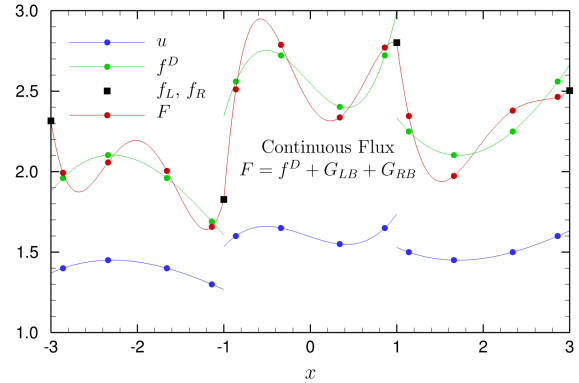
4. At each edge of the element, compute the difference between the **discontinuous flux function** and the interface **common flux value**. These are the left/right flux jumps.



5. Use the left/right flux jumps from the previous step to scale the **correction functions** of degree $\mathcal{P} + 1$; the scaled correction functions take the value of the flux jump at the respective left/right cell boundary while leaving the opposite cell boundary unchanged. Combining the two scaled correction functions results in the corrected flux function, which is a polynomial of degree $\mathcal{P} + 1$, takes the value of the flux jumps at their respective boundaries, and approximates the zero function in between. Note, however, that the correction functions generally have non-zero values at the interior points.

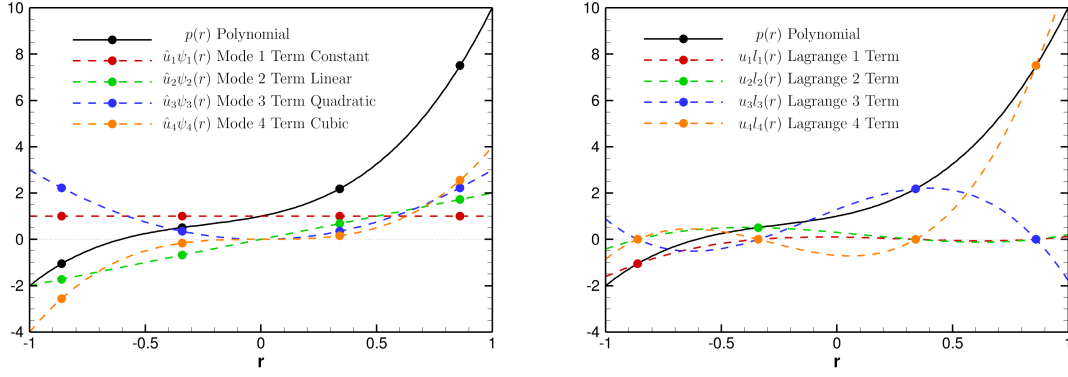


6. Add the **corrected flux function** to the **discontinuous flux function** to create a **continuous flux function** of degree $\mathcal{P} + 1$ that passes through the **common flux values**. Note that the flux at the interior solution points is also adjusted slightly to smoothly accommodate this change.



7. Compute the spatial **derivative of the continuous flux function**.
8. Use the derivative of the continuous flux function to advance the solution u to the next time level.

For the interested reader, we elaborate on each of these steps in the sections that follow.



(a) Modal Representation.

(b) Nodal Representation.

Figure 1: Modal and nodal representation of a $\mathcal{P}3$ polynomial.

2.1.2 Solution Polynomial

At a given time t , the exact solution to equation Eq. (1) is approximated locally within each cell as a polynomial of degree $\mathcal{P}$. In some instances we will use $\mathcal{P}_s$ to further clarify that this is the degree of the solution polynomial. This polynomial approximation provides a smooth variation of the solution within each cell. However, the polynomials between adjacent cells are generally discontinuous across cell interfaces even in the absence of shocks.

One way to represent the degree $\mathcal{P}$ solution polynomial is using a modal form

$$u(r) = \sum_{n=1}^{N_p} \hat{u}_n(t) \psi_n(r) \quad \psi_n(r) = r^{n-1} \quad (2)$$

where there are $N_p = \mathcal{P} + 1$ modes and $\hat{u}_n$ for $n = 1, 2, \dots, N_p$ represents the amplitude coefficients of the constant, linear, quadratic, etc. polynomial modes. Fig. 1a illustrates the modal decomposition of a third-order polynomial, $\mathcal{P}3$. Note that the various polynomial modes are symmetric/antisymmetric across the cell. Each mode, other than the constant mode, has the following characteristics: (1) its integral across the cell is zero, meaning it does not contribute to the average value within the cell, (2) the value in the center of the cell is zero, and (3) the value at the edge of the cell is directly related to the modal coefficient. One can clearly visualize the individual modes, but visualizing the solution at a point requires adding the contributions of all of the modes.

Another way to represent the solution is using a nodal form where the degree $\mathcal{P}$ polynomial is defined by $N_p = \mathcal{P} + 1$ solution points.

$$u(r) = \sum_{n=1}^{N_p} u_n(t) l_n(r) \quad l_n(r) = \prod_{\substack{i=1 \\ i \neq n}}^{N_p} \frac{r - \xi_i}{\xi_n - \xi_i} \quad (3)$$

The coordinate values of these solution points on $\mathbf{I}$ are given by ξ_n for $n = 1, 2, \dots, N_p$. For GFR, Gauss points are recommended. These are symmetrically, but not uniformly, distributed across the interior of the cell and do not lie on the edge of the cell. The Lagrange polynomial l_n is of degree $\mathcal{P}$ and smoothly interpolates between the $\mathcal{P} + 1$ nodal points, taking the value 1 at ξ_n and zero at all other points $\xi_{i \neq n}$. The solution value at each nodal point is given by $u_n = u(\xi_n)$ and is used to scale the amplitude of the corresponding Lagrange polynomial l_n . Fig. 1b illustrates the nodal

decomposition of the same $\mathcal{P}3$ polynomial shown previously. One can quickly identify the solution value at any nodal point, but visualizing the solution at any other location requires adding all the contributions together. In this form, one also cannot easily identify the various modes.

The modal and nodal forms each have their advantages. For example, applying a filter to remove high-frequency noise is most readily done using the modal form. However, setting boundary conditions at specific locations can be easier in nodal form. GFR's internal representation uses the nodal form, and when necessary switches to modal form through the use of a Vandermonde matrix $\mathcal{V}$.

$$\hat{u} = \mathcal{V}^{-1}u \quad (4)$$

After one (explicit Euler) timestep of size Δt is applied to Eq. (1), the solution at the new time level is given by

$$u^{\text{new}}(x) = u(x) - \Delta t \frac{\partial}{\partial x} [f(x)] \quad (5)$$

In order for u^{new} to be of the same polynomial order $\mathcal{P}$ as u , the flux function f must be one order higher.

Since each u_n is simply a coefficient in Eq. (3), the derivative of this polynomial is just a linear combination of the derivatives of the Lagrange polynomials.

$$\frac{d}{dr}u(r) = u_r(r) = \sum_{n=1}^{N_P} u_n \left[\frac{d}{dr}l_n(r) \right] \quad (6)$$

2.1.3 Discontinuous Flux Function

Using the nodal solution polynomial, we can evaluate the flux function at each solution point, i.e., $f_n^D = f(u_n)$, to obtain a polynomial of order $\mathcal{P}$. The superscript D is used to identify it as the *discontinuous* flux function because the resulting function only uses information inside a given cell and does not account for interaction with data from the neighboring cells. Since we know the value of the discontinuous flux at each solution point in a cell, we can compute its derivative using Eq. (6). However, if we employ the derivative of f^D to evaluate f_x , we obtain erroneous solutions since such a derivative does not include the interaction of data across cell interfaces. Therefore, we need to construct a so-called *continuous* flux function, denoted by $F(r)$, which accounts for this interaction between adjacent cells and approximates the discontinuous flux function in some sense, to which we can then calculate its derivative. The continuous flux function is obtained by adding a correction to f^D , the discontinuous one.

2.1.4 Interface Common Flux Values

One of the principles of a conservative scheme for steady-state flow is that the flux leaving one cell should be the same as that entering the next cell. This is not the case for the discontinuous flux described above. We therefore need to compute a common interface flux value that will be used by the two adjacent cells.

At each cell interface, let u_L and u_R be the values taken on by the solution polynomials to its left and right, respectively. An upwind flux value that is common for the two adjacent cells, denoted by $f^* = f^*(u_L, u_R)$, can then be defined via the flow direction. Here, for the case of the Euler or Navier-Stokes equations, we use Roe's flux¹³ with an entropy fix.¹⁴

2.1.5 Left/Right Flux Jumps

Let f_{LB}^* denote the common upwind flux at the ‘left boundary’ of the cell and f_{RB}^* denote the common upwind flux at the ‘right boundary’ of the cell. There is a discrepancy between the value of the discontinuous flux function evaluated at the left and right boundaries of the cell and the desired common upwind flux values. We denote these left/right flux jumps as $\Delta f_{\text{LB}} = f_{\text{LB}}^* - f^D(-1)$ and $\Delta f_{\text{RB}} = f_{\text{RB}}^* - f^D(1)$.

2.1.6 Correction Functions

Our goal is to construct a continuous flux function, $F(r)$, as a polynomial of degree $\mathcal{P}+1$ by requiring that it takes on the common upwind flux values at the two cell interfaces, and it approximates f^D within the interior of the cell via the $\mathcal{P} - 1$ remaining constraint conditions for the polynomial. However, instead of defining F directly, we define $G = F - f^D$, which represents the *corrected flux function* that must be added to the **discontinuous flux function** f^D in order to obtain the **continuous flux function** F .

At the two interfaces, $r = \pm 1$, the function G takes the following values which are the flux jumps described previously.

$$G(-1) = F(-1) - f^D(-1) = f_{\text{LB}}^* - f^D(-1) = \Delta f_{\text{LB}} \quad (7)$$

$$G(+1) = F(+1) - f^D(+1) = f_{\text{RB}}^* - f^D(+1) = \Delta f_{\text{RB}} \quad (8)$$

Therefore, $G(r)$ has prescribed left and right interface values, is of degree $\mathcal{P} + 1$, and approximates zero within the interior of the cell.

There are many ways one could prescribe how the correction approximates the zero function, and different choices result in different numerical schemes with different stability characteristics. Further discussion regarding the choices in correction functions can be found in Refs. 1,4,5,7. The GFR code uses Radau polynomials as the correction functions. This choice has been shown to recover a nodal discontinuous Galerkin (DG) method.

We separate the prescription of the jump at the left interface from that of the right. This separation plays a critical role in the approach.

$$G(r) = G_{\text{LB}}(r) + G_{\text{RB}}(r) \quad G_{\text{LB}}(r) = \Delta f_{\text{LB}} g_{\text{LB}}(r) \quad G_{\text{RB}}(r) = \Delta f_{\text{RB}} g_{\text{RB}}(r) \quad (9)$$

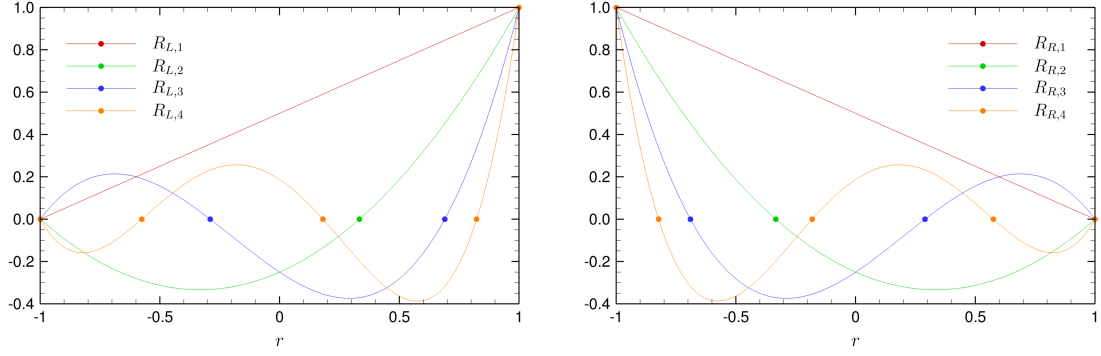
Let g_{LB} be the *correction function* for the left interface of the reference element defined by

$$g_{\text{LB}}(-1) = 1, \quad g_{\text{LB}}(1) = 0 \quad (10)$$

It is a polynomial of degree $\mathcal{P} + 1$ approximating the zero function in some sense. We always choose g_{RB} by reflection: $g_{\text{RB}}(r) = g_{\text{LB}}(-r)$. As a result, it is also a polynomial of degree $\mathcal{P} + 1$ with boundary values of

$$g_{\text{RB}}(-1) = 0, \quad g_{\text{RB}}(1) = 1 \quad (11)$$

In GFR, the left/right correction function of order $\mathcal{P} + 1$ is computed using the right/left Radau polynomial of that same order. Details of the Radau polynomial may be found in Appendix A.2. Fig. 2 illustrates the shape of different orders of the Radau polynomial. The value at one side is always zero, and the value at the other side is always one. The number of times the polynomial equals zero on $\mathbf{I}$ is equal to the order of the polynomial. This choice results in a correction that approximates the zero function in a least-squares sense, and the resulting method is equivalent to a nodal discontinuous Galerkin (DG) scheme.



(a) Left Radau Polynomial

(b) Right Radau Polynomial

Figure 2: Left and Right Radau polynomials.

2.1.7 Continuous Flux Function

The *continuous flux function* may now be computed by adding the **correction function** to the **discontinuous flux function**.

$$F = f^D + G \quad (12)$$

$$= f^D + G_{\text{LB}} + G_{\text{RB}} \quad (13)$$

$$= f^D + \Delta f_{\text{LB}} g_{\text{LB}} + \Delta f_{\text{RB}} g_{\text{RB}} \quad (14)$$

As will become apparent in the following two steps, it is not necessary to explicitly compute the values of the continuous flux function. We need only know how it is composed so that we can compute its derivative.

2.1.8 Derivative of the Continuous Flux Function

The derivative of the continuous flux function is given by:

$$\frac{\partial F}{\partial r} = \frac{\partial f^D}{\partial r} + \Delta f_{\text{LB}} g'_{\text{LB}} + \Delta f_{\text{RB}} g'_{\text{RB}} \quad (15)$$

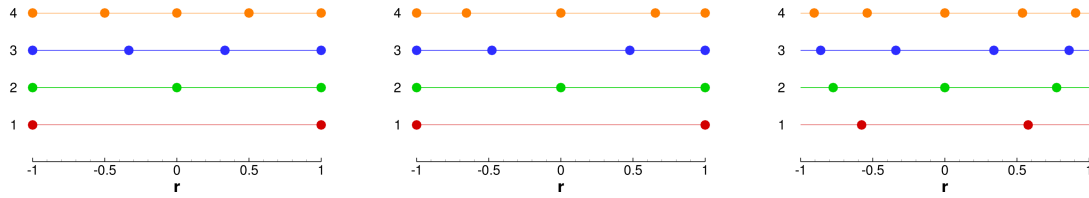
where $\partial f^D / \partial r$ can be computed in a manner similar to Eq. (6) and expressions for g'_{LB} and g'_{RB} follow from the derivatives of the Radau polynomials found in Appendix A.2. The inverse mapping function χ^{-1} can then be used to transform the derivative from reference space (r) to physical space (x), and that value for $\partial F / \partial x$ can be used to approximate $\partial f / \partial x$ at each solution point within the grid cell.

2.1.9 Time Advancement

After one (explicit Euler) timestep of size Δt is applied to Eq. (1), the solution at each point n at the new time level is given by

$$u_n^{\text{new}}(x) = u_n(x) - \Delta t \left. \frac{\partial}{\partial x} [F(x)] \right|_n \quad (16)$$

More advanced time stepping algorithms work similarly. Further details on time-stepping in GFR can be found in Section 8.

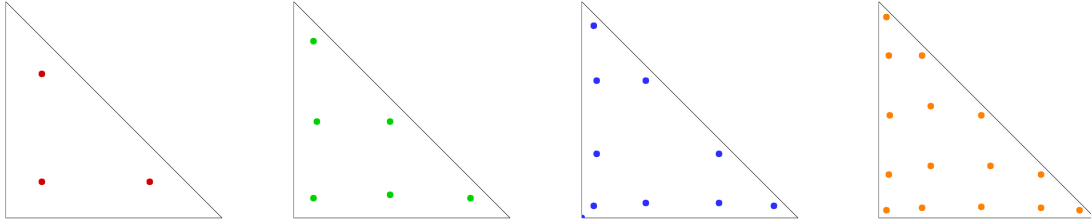


(a) Equidistant points.

(b) Lobatto points.

(c) Gauss points.

Figure 3: 1D solution point distributions vs polynomial order, $\mathcal{P}1 - \mathcal{P}4$.



(a) $\mathcal{P}1$

(b) $\mathcal{P}2$

(c) $\mathcal{P}3$

(d) $\mathcal{P}4$

Figure 4: 2D triangle “Gauss” solution point distributions vs polynomial order, $\mathcal{P}1 - \mathcal{P}4$.

2.2 Solution Points

As discussed in Section 2.1.2, the solution within each element is represented by a polynomial of order $\mathcal{P}$ defined by values at $\mathcal{P} + 1$ solution points. Equally-spaced (equidistant), Lobatto, or Gauss points may be used. Fig. 3 illustrates the distribution of these points within a standard element in one dimension. In each case, the distribution of points within the reference element is symmetric about the center of the cell. Equidistant points include points on both boundaries of the cell and uniform spacing is used for the interior points. Lobatto points also include points on both boundaries of the cell, but the interior spacing is non-uniform and biased toward the boundaries of the cell. Gauss points are distributed purely within the interior of the cell, and are also biased toward both boundaries of the cell.

Gauss points are generally preferred over Lobatto because the numerical quadrature associated with Gauss points has a slightly higher degree of accuracy than that for Lobatto points. Equidistant points should not be used in the FR method due to quadrature stability issues; they should only be used for post-processing/visualization purposes. Fig. 4 illustrates the distribution of “Gauss” points within a 2D triangle element. A tensor product is used to distribute solution points within 2D quadrilaterals, which is simply a 1D distribution in each direction. Similarly, a 3D prism and 3D hexahedron use a 1D distribution in the third direction. The distribution in a 3D tetrahedron and 3D pyramid are somewhat similar to the 2D triangle, but are difficult to visualize on a printed page.

See Also: [All_Orders](#) [All_Points](#) [Loc_Triangle_Pts](#)
[Solution_Order](#) [Loc_Solution_Pts](#) [Loc_Tetrahedra_Pts](#)
[Projection_Order](#) [Loc_Flux_Pts](#)
[Quadrature_Order](#) [Loc_Quadrature_Pts](#)
[Error_Order](#) [Loc_Error_Pts](#)

2.3 Quadrature Points

Instead of directly evaluating the nonlinear flux at the solution points, it can be more accurate to evaluate the flux using a higher quadrature order (i.e., a higher order polynomial and its corresponding points) and then projecting that onto a polynomial of the same order as the solution. Section 9.1 provides a more detailed explanation of this aliasing error and the steps involved in using quadrature points.

2.4 Output Points

GFR allows the user to output the solution at a different set of points (i.e., equidistant) for visualization. The reason for this is that since Gauss points do not extend to the boundaries of cells, many visualization tools will render the result with a small gap between each cell.

See Also: [All_Orders](#), [Output_Order](#), [Loc_Output_Pts](#)

2.5 Solution Accuracy and Grid Refinement

We start by introducing the term degree-of-freedom (DoF) which is simply a location at which we are solving the governing equations. Note, some refer to a DoF as an individual equation being solved at a point, so an individual solution point would contain 5 DoF for the 3D Euler/Navier-Stokes equations; here we refer to a DoF as the solution point itself. For traditional finite-difference and finite-volume methods, the governing equations are solved at each grid node (or cell for cell-based methods), so the total DoF is just the total number of nodes (or cells) in the grid. Within the context of high-order finite-/spectral-element methods, a single grid cell will solve the governing equations on a set of solution nodes (or modes for modal methods) which we refer to as solution points. Each of these solution points is a DoF in a simulation, and the total number of solution points in a cell depends on the element type and the polynomial order of the solution (Section 3.2 provides more detail on the DoF for each element type). Assuming ideal parallel scaling with minimal communication costs, the computational cost of a simulation is proportional to the total DoF within the entire computational domain. With finite-difference, finite-volume, and spectral-element methods being so vastly different, the fairest way to compare the computational costs of the different methods is only between simulations containing similar numbers of total DoF.

The computational mesh plays a critical role in any scale-resolving simulation (SRS), as the grid will filter out the scales of turbulent structures smaller than the grid resolution. As such, it is important to understand the effect that refinement—grid (h), order (p), or combination thereof (h/p)—will have on any given simulation. For finite-difference and finite-volume methods, h -refinement is only possible by changing the number of grid nodes/cells. True p -refinement is only possible by increasing the order of accuracy while keeping the grid resolution constant, e.g., comparing 2nd- and 4th-order finite-difference schemes on the same structured grid. For spectral-element methods like FR, increasing $\mathcal{P}$ on the same grid results in h/p -refinement because the number of DoF in each cell also increases with $\mathcal{P}$. Therefore, in the context of FR, p -refinement requires decreasing the

grid resolution such that the total DoF across all grid cells is equivalent between different values of $\mathcal{P}$.

Achieving quality wall-resolved LES (WRLES) results can be a tricky and time consuming process that is highly dependent on the grid quality and numerical set up. WRLES generally requires high-resolution meshes for both accuracy and stability, so much so that the maximum stable time step can be multiple orders of magnitude smaller compared to RANS. One of the major benefits of spectral-element methods is that the order of the method can be increased while simultaneously decreasing the size of the grid to maintain a certain level of accuracy of a simulation. Decreasing the size of the grid will generally increase the size of the smallest grid cells which will directly increase the maximum allowable stable time step; however, increasing $\mathcal{P}$ also reduces the maximum allowable stable time step since the time step scales as $1/\mathcal{P}^2$. Furthermore, increasing $\mathcal{P}$ can introduce numerical instabilities that require added numerical dissipation to prevent the simulation from diverging. Consequently, this numerical dissipation will also damp out the smallest turbulent scales within a simulation unless the dissipation can be precisely targeted to only the areas within the solution where it is absolutely needed. Ultimately, there is a back and forth required with grid and numerical setups to find an optimal combination to achieve accuracy goals within the computational time available for a set of simulations.

3 Geometry and Mesh

3.1 Surface Curvature

When constructing a mesh for use with GFR, it is highly recommended to include the high-order surface curvature information. To export a high-order mesh, your mesh generation software may require that every boundary surface be defined by a CAD or database surface, even those that are planar surfaces. Therefore, it is important to understand the requirements of your meshing software before you begin.

Without the high-order surface curvature information, GFR will be forced to construct the elements using straight line segments. This can cause faceting of the surface shape, and with a high-order code like GFR, this can significantly degrade the solution accuracy in the best case scenario, with a high likelihood of instabilities. For purely planar or rectangular geometries, which have no curvature, this is not a problem.

3.2 Element Types

The standard element types supported by GFR are shown in Fig. 5. These include: 1D line, 2D triangle, 2D quadrilateral, 3D tetrahedron, 3D pyramid, 3D prism, and 3D hexahedron. The number of degrees-of-freedom (DoF), i.e., solution points, within a given element type is a function of the order $\mathcal{P}$ of the solution polynomial used. Table 1 lists this function for each element type and gives example DoF counts for $\mathcal{P}.1\text{--}\mathcal{P}.4$. The last column of Table 1 also provides the maximum solution order that can be used for each element type.

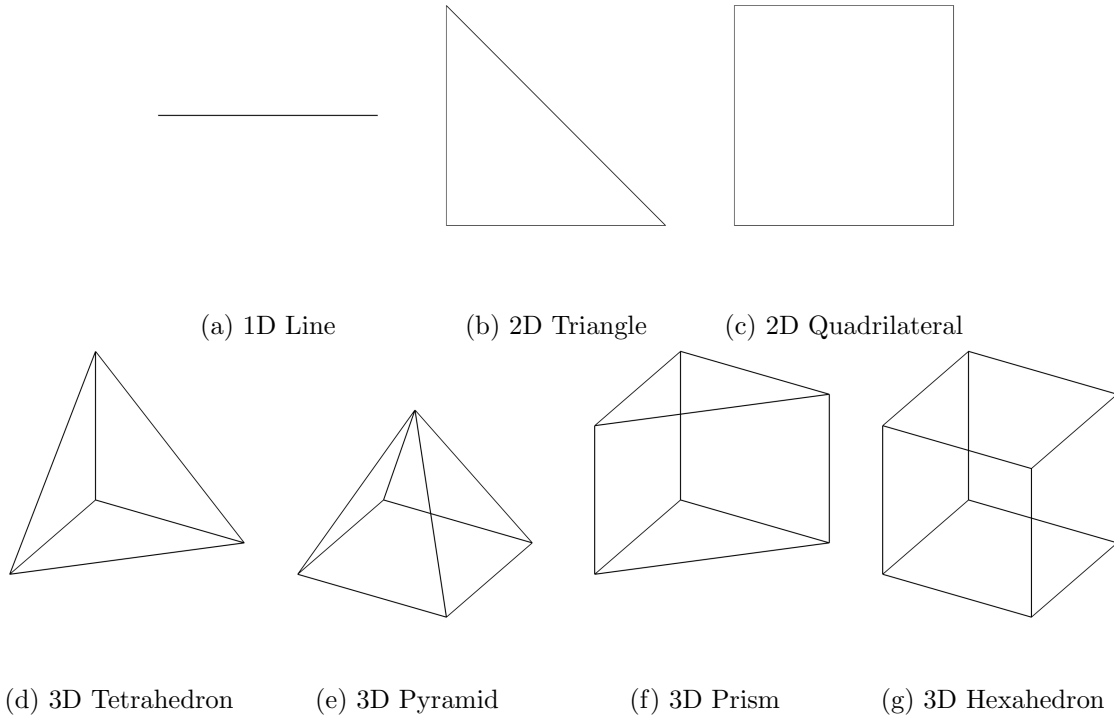


Figure 5: Supported element types.

Table 1: Degrees of freedom of supported element types.

Dim	Element Type	Degrees of Freedom	$\mathcal{P}1$	$\mathcal{P}2$	$\mathcal{P}3$	$\mathcal{P}4$	Maximum $\mathcal{P}$
1D	Line	$(\mathcal{P} + 1)$	2	3	4	5	unlimited
2D	Triangle	$(\mathcal{P} + 1)(\mathcal{P} + 2)/2$	3	6	10	15	8
2D	Quadrilateral	$(\mathcal{P} + 1)^2$	4	9	16	25	unlimited
3D	Tetrahedron	$(\mathcal{P} + 1)(\mathcal{P} + 2)(\mathcal{P} + 3)/6$	4	10	20	35	6
3D	Pyramid	$(\mathcal{P} + 1)(\mathcal{P} + 2)(2\mathcal{P} + 3)/6$	5	14	30	55	unlimited
3D	Prism	$(\mathcal{P} + 1)(\mathcal{P} + 1)(\mathcal{P} + 2)/2$	6	18	40	75	8
3D	Hexahedron	$(\mathcal{P} + 1)^3$	8	27	64	125	unlimited

3.3 Grid Scaling

GFR expects the mesh file to be in units of meters. However, in many instances the mesh file may be in another set of units or perhaps nondimensional. For these situations, the code allows the user to specify a multiplicative scaling factor that will convert the coordinates of the existing grid into units of meters.

NOTE: Throughout this document, we shall refer to the units of the mesh file as *grid units* and the units after applying the grid scaling as *grid scaled units* ($= [\text{grid units}] * \text{Grid_Scaling_Factor}$). For example, if the grid units are inches, the input file should include the line

```
Grid_Scaling_Factor = 0.0254    ! convert inches to meters
```

See Also: `Grid_Scaling_Factor`

3.4 Mesh Resolution Requirements for Large-Eddy Simulation

In order to comprehend the mesh resolution requirements for large-eddy simulation (LES), one must have some understanding of the turbulent eddies that are being simulated. Even though an instantaneous turbulent velocity field often appears chaotic, it is actually composed of a multitude of swirling three-dimensional turbulent eddies that are all intertwined. These eddies span a continuous range of varying sizes, from the (largest) integral length scale ($l = k/\epsilon$) to the (smallest) Kolmogorov scale ($\eta = (\nu^3/\epsilon)^{1/4}$), where k is the turbulent kinetic energy, ϵ is the turbulent dissipation rate ($-dk/dt$), and ν is the kinematic viscosity. The energy contained in each eddy is proportional to its size.

At any given point in the flow, we can perform a Fourier decomposition of the energy containing eddies to examine the spectral distribution in terms of wavenumber, $\kappa = 2\pi/(\text{eddy diameter})$. Doing so produces the energy spectral density, $E(\kappa)$, which is the distribution of energy as a function of wavenumber. As shown in Fig. 6, the horizontal axis represents (from left to right) the range of eddy sizes from the largest to the smallest. This range increases with increasing Reynolds number. The vertical axis represents the energy contained in those eddies, and the area under the curve represents the turbulent kinetic energy at that particular point in the flow. Instabilities in the largest structures generate turbulence, the intermediate structures transfer that energy, and the smallest structures dissipate the energy into heat.

The idea behind LES is that we need not simulate the full range of scales, only the largest ones that contain the predominant share of energy (e.g., 80 percent of the area under the curve starting from the left). Thus the grid should be fine enough to fully resolve (i.e., represent) eddies at the highest desired wavenumber, typically within the inertial subrange or outer part of the viscous subrange. The dissipation effect of the high wavenumber (small scale) turbulence that is

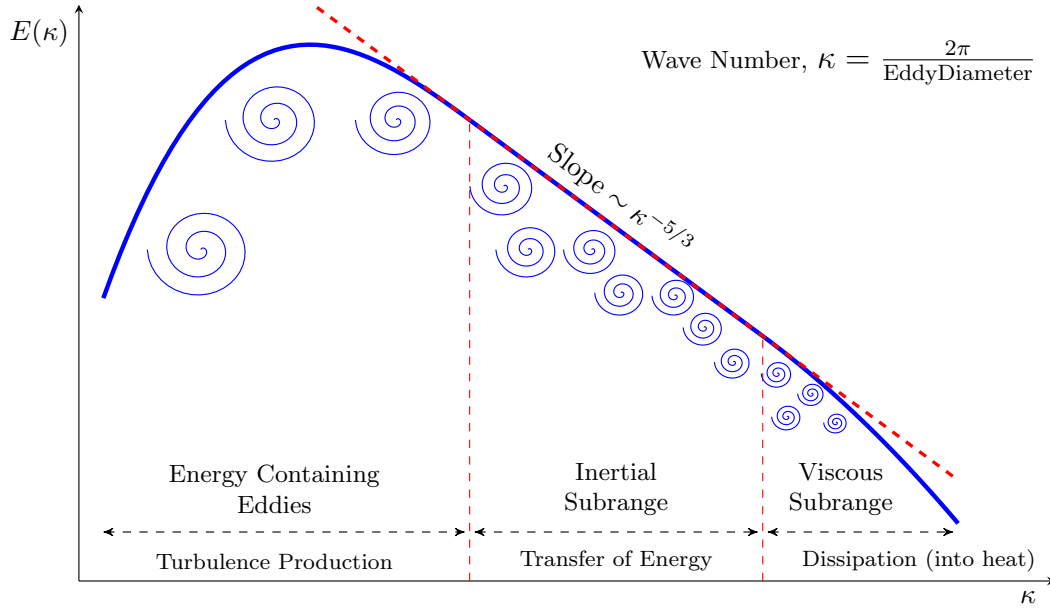


Figure 6: Energy spectra for a turbulent flow, log-log scales.

not properly resolved by the grid is sometimes represented using an explicit subgrid-scale model. Implicit large-eddy simulation (ILES) takes a different approach, using the inherent dissipation of the numerical scheme to act as the subgrid model. For boundary layer flows at high Reynolds numbers, it may not be feasible to perform a wall-resolved LES due to the extremely fine grid resolution required to resolve the small-scale turbulent structures. In such cases, some codes offer near-wall models or wall functions in what is referred to as wall-modeled LES.

In isotropic turbulence, the eddies are statistically uniform with no preferred direction, resulting in $\overline{u'u'} = \overline{v'v'} = \overline{w'w'}$. This situation might occur far away from boundaries. In boundary layer flows, the turbulent eddies are distorted by the mean shear and physical spatial limitation in the wall-normal direction, resulting in anisotropic turbulence that can exhibit long streamwise vortex filaments. The streamwise ($\overline{u'u'}$) turbulence is accentuated and the wall-normal ($\overline{v'v'}$) component is reduced. In free shear flows, such as jets and mixing layers, the mean shear again causes an increase in the streamwise ($\overline{u'u'}$) component and a decrease in the shear direction ($\overline{v'v'}$), but not to the same extent.

Table 2 summarizes the grid spacing requirements for wall-resolved LES from a number of sources. These are given in terms of near-wall coordinates (i.e., $\Delta x^+ = \Delta x u_\tau / \nu$, where u_τ is the friction velocity). Note that the wall-normal direction uses grid stretching with an initial wall spacing $\Delta y_{\min}^+ \leq 1$ and 3–40 points within $y^+ < 10$. The spanwise spacing (Δz^+) is typically 10–20, and the streamwise spacing (Δx^+) is often double or triple that. Ref 15 notes that using overly coarse values for Δx^+ or Δz^+ can lead to unphysical boundary layer streaks and error in the predicted skin friction. Based on these sources, for traditional finite-difference or finite-volume solvers, a good starting grid would have $(\Delta x^+, \Delta y_{\min}^+, \Delta z^+) = (20\text{--}30, 1, 15)$ with 10 points below $y^+ < 10$ (which corresponds to a geometric growth rate of 1.02). For free shear flows, 50 points or more are likely needed across the shear layer thickness, δ .

For GFR, the volume solution points are often defined using either Gauss or Lobatto points, which are non-uniformly distributed toward the boundaries of each cell (see Fig. 3). In addition, the number of points in each cell is determined by the chosen polynomial order for the solution, $\mathcal{P}$.

Table 2: Grid requirements for wall-resolved large-eddy simulation.

Δx^+	$\Delta y_{\min}^+$	Δz^+	$N_{pts}, y^+ < 10$	Reference
50–150	< 1	15–40	20–40	Georgiadis ¹⁶
50–150	1	10–40	3–5	Breuer ¹⁵
50–100	1	10–20	3	Sagaut ¹⁷
50	1	15		Sagaut ¹⁸
25	< 1	11	10	Schlatter ¹⁹
20–30	0.8–1.2	10–15		Debroeyer ²⁰
10–50	0.5–2.5	10–20		Ameri ²¹

When building a mesh, we therefore compare the above values to the average solution point spacing within a high-order cell (i.e., $\overline{\Delta x^+} = \Delta x_{\text{cell}}^+ / (\mathcal{P} + 1)$). From that, we obtain the recommended *cell spacings* of $\Delta x_{\text{cell}}^+ = (10\text{--}80)(\mathcal{P} + 1)$, $\Delta y_{\text{cell},\min}^+ = 2$, $\Delta z_{\text{cell}}^+ = (10\text{--}40)(\mathcal{P} + 1)$, and a wall-normal growth rate of 1.5 for boundary layer flows. For free shear flows, we recommend isotropic cells with $\Delta y_{\text{cell}} / \delta = (\mathcal{P} + 1) / 50$ across the shear layer thickness, δ .

4 Reference Frame Rotation

GFR supports a rotating reference frame in which all elements in the domain move at a constant rotation rate. This can be used to simulate flow within rotating channels or turbomachinery components. The default rotation rate is zero, which means the simulation is computed in a fixed reference frame.

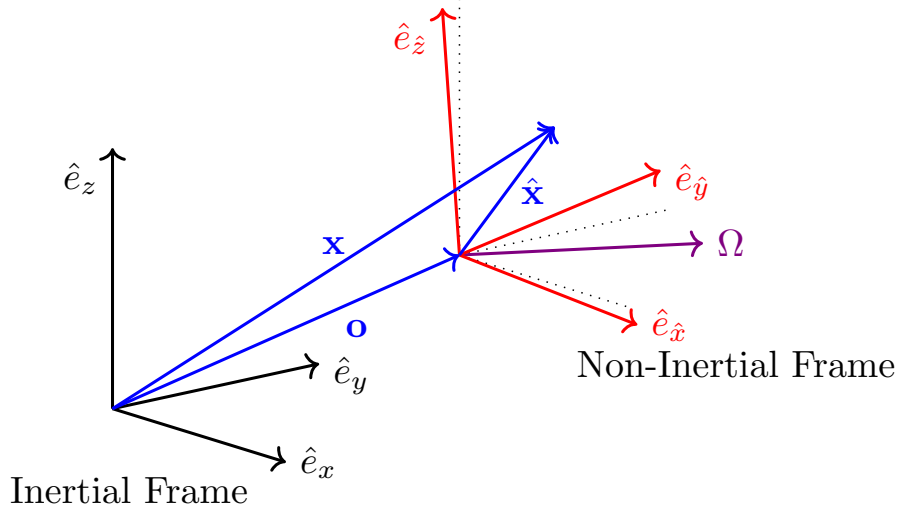


Figure 7: Reference Frames

Fig. 7 illustrates the relationship between the inertial (fixed) (x, y, z) frame and the non-inertial (rotating) $(\hat{x}, \hat{y}, \hat{z})$ frame, whose origin is shifted by $\vec{o}$ and whose axes may be rotated by the rotation rate vector $\vec{\omega}$. This spin vector $\vec{\omega}$ is defined by the rotation rate $\dot{\theta}$ of the reference frame about a unit axis $\vec{a}$.

$$\vec{a} = a_x \hat{e}_x + a_y \hat{e}_y + a_z \hat{e}_z \quad a = \|\vec{a}\| = \sqrt{a_k a_k} = 1 \quad (17)$$

$$\vec{\omega} = \omega_x \hat{e}_x + \omega_y \hat{e}_y + \omega_z \hat{e}_z = \omega \vec{a} \quad \omega = \|\vec{\omega}\| = \sqrt{\omega_k \omega_k} = \dot{\theta} \quad (18)$$

A rotation rate tensor $\mathbf{\Omega}$ can likewise be defined such that $\mathbf{\Omega} \cdot \vec{v} = \vec{\omega} \times \vec{v}$ for any vector $\vec{v}$. For clarity, we shall write this using the following annotated tensor matrix notation, where the associated basis vectors are indicated in blue to the left and top of the rows and columns of each matrix.

$$\Omega_{ij} = -\varepsilon_{ijk} \omega_k = \begin{matrix} & \hat{e}_x & \hat{e}_y & \hat{e}_z \\ \hat{e}_x & \begin{bmatrix} 0 & -\omega_z & \omega_y \end{bmatrix} \\ \hat{e}_y & \begin{bmatrix} \omega_z & 0 & -\omega_x \end{bmatrix} \\ \hat{e}_z & \begin{bmatrix} -\omega_y & \omega_x & 0 \end{bmatrix} \end{matrix} \quad \Omega = \|\Omega_{ij}\| = \sqrt{\Omega_{ij} \Omega_{ij} / 2} = \sqrt{\omega_k \omega_k} = \dot{\theta} \quad (19)$$

From this, a passive transformation $\mathbf{Q}$ can be constructed for converting the vector components from the inertial to the non-inertial unit vectors.

$$\mathbf{Q} = \begin{matrix} & \hat{e}_x & \hat{e}_y & \hat{e}_z \\ \hat{e}_{\hat{x}} & \begin{bmatrix} a_x^2 b + c & a_x a_y b + a_z s & a_x a_z b - a_y s \end{bmatrix} \\ \hat{e}_{\hat{y}} & \begin{bmatrix} a_x a_y b - a_z s & a_y^2 b + c & a_y a_z b + a_x s \end{bmatrix} \\ \hat{e}_{\hat{z}} & \begin{bmatrix} a_x a_z b + a_y s & a_y a_z b - a_x s & a_z^2 b + c \end{bmatrix} \end{matrix} \quad \begin{aligned} \theta &= \int_{t=0}^t \dot{\theta} dt \\ b &= 1 - \cos \theta \\ c &= \cos \theta \\ s &= \sin \theta \end{aligned} \quad (20)$$

Note that $\mathbf{Q}(t)$ is a time-dependent proper orthogonal rotational transformation tensor such that:

$$\mathbf{Q}\mathbf{Q}^T = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}, \quad \det \mathbf{Q} = 1 \quad (21)$$

The motion of this moving non-inertial reference frame $\vec{\hat{x}}$ can therefore be described in terms of the arbitrary time-dependent translation and rotation of the inertial reference frame $\vec{x}$ by the following relations:

$$\hat{t} = t + t_0 \quad t = \hat{t} - t_0 \quad (22)$$

$$\vec{\hat{x}} = \mathbf{Q}(t) (\vec{x} - \vec{o}(t)) \quad \vec{x} = \mathbf{Q}^T \vec{\hat{x}} + \vec{o} \quad (23)$$

$$\hat{x}_i = Q_{ij} (x_j - o_j) \quad x_i = Q_{ij}^T \hat{x}_j + o_i \quad (24)$$

It is important to remember that scalars, vectors, and tensors represent physical quantities that are defined *with respect to a particular reference frame* (i.e., as perceived by an observer in that frame). For example, to an observer standing in a room with stagnant air, the air is still. However, to another observer in the room riding a carousel, the air would seem to approach at a speed equal and opposite his motion.

Once the physical orientation of a vector or tensor has been defined (based on the chosen reference frame), it is still possible to represent that quantity using a different set of unit vectors than the default set for that frame. This includes other unit vectors within the same frame or even the unit vectors of another frame. For example, using the transformation $\mathbf{Q}$, it is possible to write any fixed frame vector or tensor using the unit vectors of the moving frame.

$$\dot{v}_i = Q_{im} v_m \quad (25)$$

$$\hat{T}_{ij} = Q_{im} T_{mn} Q_{nj}^T \quad (26)$$

Thus, $\vec{v}$ and $\vec{\hat{v}}$ represent the same physical vector (magnitude and direction) but are written here with separate symbols because their component values differ due to the different unit vectors. An *objective* scalar, vector, or tensor is one that maintains the same magnitude and orientation when observed from both the inertial and non-inertial frames. Such objective quantities transform according to:

$$\text{Scalar:} \quad \hat{s} = s \quad (27)$$

$$\text{Vector:} \quad \hat{v}_i = \dot{v}_i \quad (28)$$

$$\text{Tensor:} \quad \hat{T}_{ij} = \dot{T}_{ij} \quad (29)$$

In other words, the vector/tensor defined with respect to the moving frame perspective happens to match that observed from the fixed frame. The same unit vectors must be used when comparing these vector/tensor component values between frames, so the *transformed* fixed frame components must be used.

For these reasons, it is important to distinguish both the observer perspective and the unit vectors used to express the component form of vectors and tensors. To avoid confusion, we use the following convention: quantities without an accent ($\vec{v}$) are defined with respect to the inertial frame and that frame's unit vectors, quantities denoted with ($\vec{\hat{v}}$) are defined with respect to the non-inertial frame and that frame's unit vectors, and quantities denoted with ($\vec{\dot{v}}$) are defined with respect to the inertial frame but written in terms of the unit vectors in the non-inertial frame through the application of the transformation tensor $\mathbf{Q}$. Most of the scalars used herein are objective quantities, and we shall not distinguish between such scalars in different frames. However, because velocity is

the most commonly encountered non-objective vector quantity, this also means that kinetic energy and any other quantity computed from it (e.g., total conditions for energy, enthalpy, pressure, or temperature) are also non-objective scalar quantities.

When applying the transformation to the rotation axis, one finds (after substituting $\mathbf{Q}$ from Eq. (20) and doing some math):

$$\begin{matrix} & \mathbf{\hat{e}_{\hat{x}}} & \mathbf{\hat{e}_{\hat{y}}} & \mathbf{\hat{e}_{\hat{z}}} \\ \vec{a} \equiv & \begin{bmatrix} a_{\hat{x}} \\ a_{\hat{y}} \\ a_{\hat{z}} \end{bmatrix} \end{matrix} = \mathbf{Q} \vec{a} = \mathbf{Q} \cdot \begin{matrix} & \mathbf{\hat{e}_x} & \mathbf{\hat{e}_y} & \mathbf{\hat{e}_z} \\ \vec{a} \equiv & \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \end{matrix} = \begin{matrix} & \mathbf{\hat{e}_{\hat{x}}} & \mathbf{\hat{e}_{\hat{y}}} & \mathbf{\hat{e}_{\hat{z}}} \\ \vec{a} \equiv & \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \end{matrix} \quad (30)$$

Thus, the values of the components of $\vec{a} = (a_{\hat{x}}, a_{\hat{y}}, a_{\hat{z}})$ are identical to those of $\vec{a} = (a_x, a_y, a_z)$ even though they are written using two different sets of unit vectors. This occurs because as the frame rotates, the angle between the rotation axis and the unit vectors remains the same. Since $\vec{\omega} = \dot{\theta} \vec{a}$, a similar conclusion holds for the vector components of $\vec{\omega}$ and $\vec{\omega}$ as well as the tensor components of $\hat{\Omega}$ and Ω .

$$\vec{\omega} = \omega_x \mathbf{\hat{e}_{\hat{x}}} + \omega_y \mathbf{\hat{e}_{\hat{y}}} + \omega_z \mathbf{\hat{e}_{\hat{z}}} = \mathbf{Q} \vec{\omega} \quad (31)$$

$$\hat{\Omega}_{ij} = -\varepsilon_{ijk} \omega_k = \begin{matrix} & \mathbf{\hat{e}_{\hat{x}}} & \mathbf{\hat{e}_{\hat{y}}} & \mathbf{\hat{e}_{\hat{z}}} \\ \hat{\Omega}_{ij} = -\varepsilon_{ijk} \omega_k = & \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \end{matrix} = (\mathbf{Q} \Omega \mathbf{Q}^T)_{ij} \quad (32)$$

For additional information about the derivation of the equations of motion in a rotating reference frame, see Ref 22.

See Also: `nml_physics, Moving_Ref_Frame%Rotation, Moving_Ref_Frame%Translation`

5 Physical Models

5.1 Euler and Navier-Stokes Equations

GFR may be used to solve the compressible conservation form of the unsteady Navier-Stokes equations, in which all stress tensor and heat transfer terms are retained. It may also be used to solve the Euler equations, where the viscous terms are ignored. The fluid by default is modeled as a calorically-perfect gas, but GFR can also use one of the preliminary chemistry models included with the code for testing thermally-perfect gas mixtures with or without chemical reactions. In addition, the effects of a rotating reference frame can also be included.

Using the notation described in Section 4 for the rotating reference frame, we define the absolute velocity vector $\vec{u}$ with respect to the fixed inertial frame. To a stationary observer, a particle in the flow field would appear to move at velocity $\vec{u}$. On the other hand, to an observer moving with the rotating axes, the same fluid particle would appear to move at a different velocity $\vec{u}$. This is referred to as the relative velocity vector $\vec{u}$ with respect to the rotating frame. The difference between them is due to the motion of the second observer moving with the axes of the non-inertial frame. In the general case, the velocity vector of the non-inertial frame is $\vec{w} = \dot{\Omega}\vec{x} + \mathbf{Q}\dot{\vec{o}} = \vec{\omega} \times \vec{x} + \mathbf{Q}\dot{\vec{o}}$. (Note that $\vec{w}$ should not be confused with the spin vector $\vec{\omega}$. To avoid such ambiguity, we avoid the spin vector and instead use expressions involving the spin tensor $\dot{\Omega}$.) So, the relationship between these three velocity vectors is:

$$\vec{u} \equiv \mathbf{Q}\vec{u} = \vec{u} + \vec{w} \quad (33)$$

Note that $\mathbf{Q}\vec{u}$ represents the absolute velocity vector (i.e., the velocity observed from the fixed frame), but written in terms of the non-inertial unit vectors. To make the notation compact, we use the symbol $\vec{u}$. For non-inertial flows, thermodynamic properties like density, pressure, temperature, viscosity, and thermal conductivity are objective quantities meaning they are the same in both reference frames, i.e., $[\hat{\rho}, \hat{p}, \hat{T}, \hat{\mu}, \hat{\lambda}] = [\rho, p, T, \mu, \lambda]$. However, as noted in Section 4, kinetic energy and therefore total values are perceived differently in each frame.

The Navier-Stokes equations in the non-inertial frame may be written in terms of the absolute velocity as follows:

$$\frac{\partial Q}{\partial t} + \frac{\partial F_j}{\partial x_j} = \frac{\partial F_j^v}{\partial x_j} + S \quad (34)$$

$$Q = \begin{bmatrix} \rho \\ \rho u_i \\ \rho e_t \end{bmatrix} \quad F_j = \begin{bmatrix} \rho(u_j - w_j) \\ \rho u_i(u_j - w_j) + p\delta_{ij} \\ \rho e_t(u_j - w_j) + p u_j \end{bmatrix} \quad F_j^v = \begin{bmatrix} 0 \\ \hat{\tau}_{ij} \\ \hat{\tau}_{ij}u_j - \hat{q}_j \end{bmatrix} \quad S = \begin{bmatrix} 0 \\ -\rho \dot{\Omega}_{ij}u_j \\ 0 \end{bmatrix} \quad (35)$$

where the total energy is given by

$$\rho e_t = \frac{p}{\gamma - 1} + \frac{1}{2}\rho u_i u_i \quad (36)$$

For this form of the energy equation, e_t is the total energy in the fixed reference frame.

The viscous stress tensor is based on Stokes' hypothesis and given by the equation

$$\hat{\tau}_{ij} = 2\mu \hat{S}_{ij} - \frac{2}{3}\mu \hat{S}_{kk}\delta_{ij} \quad (37)$$

where the strain rate tensor is computed from

$$\hat{S}_{ij} = \frac{1}{2} \left(\frac{\partial \hat{u}_i}{\partial \hat{x}_j} + \frac{\partial \hat{u}_j}{\partial \hat{x}_i} \right) = \frac{1}{2} \left(\frac{\partial \acute{u}_i}{\partial \hat{x}_j} + \frac{\partial \acute{u}_j}{\partial \hat{x}_i} \right) \quad (38)$$

$$\frac{\partial \hat{u}_i}{\partial \hat{x}_j} = \frac{\partial}{\partial \hat{x}_j} (\acute{u}_i - \acute{w}_i) = \frac{\partial \acute{u}_i}{\partial \hat{x}_j} - \acute{\Omega}_{ij} \quad (39)$$

The heat flux vector, $\hat{q}_j$, is modeled by Fourier's law

$$\hat{q}_j = -\lambda \frac{\partial T}{\partial \hat{x}_j} \quad (40)$$

where λ is the thermal conductivity which is related to the specific heat at constant temperature, c_p , and Prandtl number, Pr , through the relation

$$\lambda = \frac{\mu c_p}{\text{Pr}} \quad (41)$$

For a perfect gas, $c_p = \gamma R_{\text{gas}} / (\gamma - 1)$.

See Also: `nml_physics`, `nml_reference`, `Moving_Ref_Frame%Rotation`, `Governing_Equations`, `Gamma`, `Gas_Constant`, `Prandtl`

5.2 Equation of State

For a thermally and calorically-perfect gas, the equation of state is given by:

$$p = \rho R_{\text{gas}} T \quad (42)$$

Note that density, temperature, and pressure are objective scalars that have the same values regardless of reference frame.

See Also: `Gas_Constant`

5.3 Viscosity

Sutherland's viscosity law is used to define the laminar viscosity as a function of local temperature.

$$\frac{\mu}{\mu_0} = \left(\frac{T}{T_0} \right)^{3/2} \frac{T_0 + S_0}{T + S_0} \quad (43)$$

The coefficients of the model are specified as part of the reference conditions described in Section 6, and are set by default to those for air: $\mu_0 = 1.716 \times 10^{-5} \text{ N s/m}^2$, $T_0 = 273 \text{ K}$, $S_0 = 110.4 \text{ K}$.

NOTE: If the `Reynolds` number is specified in the user input, then the μ_0 constant will be adjusted to obtain this Reynolds number.

See Also: `Suth_Muref`, `Suth_Tref`, `Suth_Sref`, `Reynolds`

5.4 Synthetic Eddy Method (SEM)

5.4.1 SEM Description and Theory

For turbulent flow, one rarely knows what precise unsteady values to specify at the inflow boundary. Often, however, time-averaged boundary-layer profile data exists or can be estimated based on Reynolds number. One way to provide an unsteady turbulent inflow is to add an upstream region of grid which has a steady uniform inflow, some numerical or physical tripping mechanism, and a long run of length for the boundary layer to grow to the desired thickness before entering the region of interest. Unfortunately, this method is computationally expensive due to the added upstream length and unsteady iterations needed to obtain time-averaged statistics. It also involves numerous parameters related to the trip and run length that likely require adjustment and repetition of the whole simulation. Furthermore, the method might not even provide an adequate boundary layer due to other effects such as upstream surface curvature or pressure gradient.

The synthetic eddy method (SEM)²³ allows the user to specify a time-averaged inflow profile for the mean flow and turbulent stresses. The algorithm will then use that data to artificially generate unsteady inflow values that, when averaged over a sufficient amount of time, will replicate the specified profiles. The way in which the SEM works can be summarized as follows.

1. The SEM profile file is read to obtain the mean flow and turbulent statistics to be imposed on the inflow boundary. The height of the SEM profile is determined from the furthest point specified in the file.
2. A grid-free virtual upstream volume is created that overlaps the physical inflow plane in the streamwise direction.
3. Synthetic eddies with random intensities are randomly seeded within this virtual volume.
4. Each grid point on the physical inflow boundary computes its own unique fluctuating velocity based on a weighted aggregate of the intensities of the synthetic eddies within the specified turbulent length scale.
5. The fluctuating velocity is then combined with the user-provided time-averaged profile for the mean flow and turbulent stresses to create an unsteady turbulent inflow boundary condition.
6. The eddies within the virtual volume are convected as particles using the specified convective velocity in the direction perpendicular to the physical inflow plane and updated each time step. Every time an eddy leaves the virtual volume, a new eddy, with a random intensity, is created at a random location on the inflow face of the virtual volume.
7. Repeat the process, starting from step 4.

The following sections, 5.4.1.1 – 5.4.1.7, provide additional details on the SEM implementation in GFR.

5.4.1.1 SEM Nomenclature

In this section, we define a number of variables that will be used quite often when describing the SEM. Since multiple SEM inflow boundaries can be specified in the `nml.bc` namelist input block, we will use *i* to represent a specific `BC_Input(i)%SEM` instance.

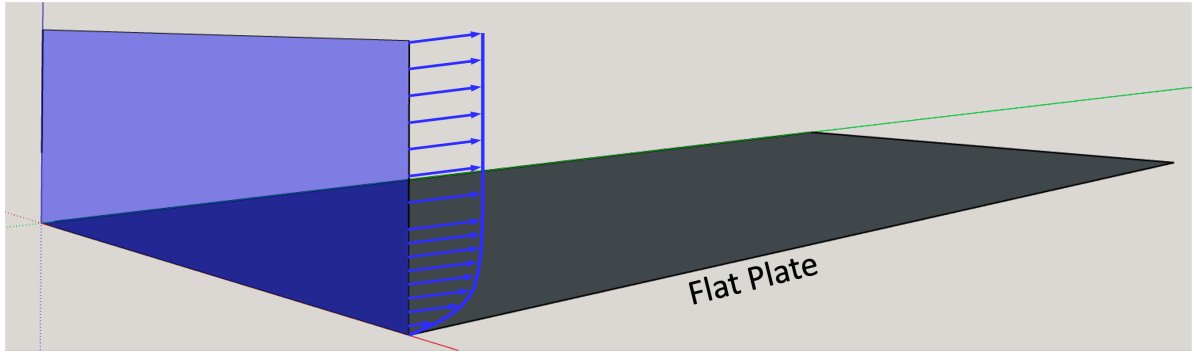


Figure 8: Illustration of an inflow profile for a flat plate.

- U_{BL} := The mean velocity at the freestream or edge of the boundary layer in the provided SEM profile file, `BC_Input(i)%SEM%Profile_File`. See Section 5.4.2.3 for details of the profile file.
- δ_{BL} := The height of the boundary layer within the SEM profile. It is important to note that δ_{BL} is not necessarily the same as h_{SEM} defined below. Ideally, the SEM profile should include the entirety of the boundary layer, so in general, $h_{SEM} \geq \delta_{BL}$. While δ_{BL} is used in our description of the boundary layer to be imposed, the SEM algorithm only uses h_{SEM} .
- U_{SEM} := The SEM convective velocity, which is the speed at which the synthetic eddies are propagated through the SEM box. It is specified using the input `BC_Input(i)%SEM%Convective_Velocity`.
- l_{SEM} := The SEM length scale, which determines the “zone of influence” or “sensory distance” of each synthetic eddy. It is specified using the input `BC_Input(i)%SEM%Length_Scale`.
- h_{SEM} := The height of the SEM profile. This is obtained from the furthest profile point specified in the SEM profile file.
- $\mathcal{V}_{SEM}$:= The volume of the SEM box, defined by Eq. (45).
- $\mathcal{V}_{eddy}$:= The volume of an eddy, l_{SEM}^3 for a cube or $\pi/6 l_{SEM}^3$ for a sphere, depending on `Use_Spherical_Eddy_Volume`.

5.4.1.2 Create a virtual box at SEM inflow

To summarize how the SEM works, we start with Fig. 8 that gives a conceptual illustration of a flat plate case. The no-slip wall is the bottom gray surface extending downstream, and the blue plane on the left is the inflow boundary surface. The blue arrows show an idealized flow profile over this flat plate including the boundary layer near the wall.

The SEM starts by creating a grid-free virtual volume/box that overlaps the physical inflow plane in the streamwise direction. In Fig. 9, the green rectangular box overlapping the blue inflow plane represents the virtual volume that is created for the SEM. In the streamwise direction (x in this example), assuming the inflow plane is located at a constant x value of x_{BC} , the SEM box is

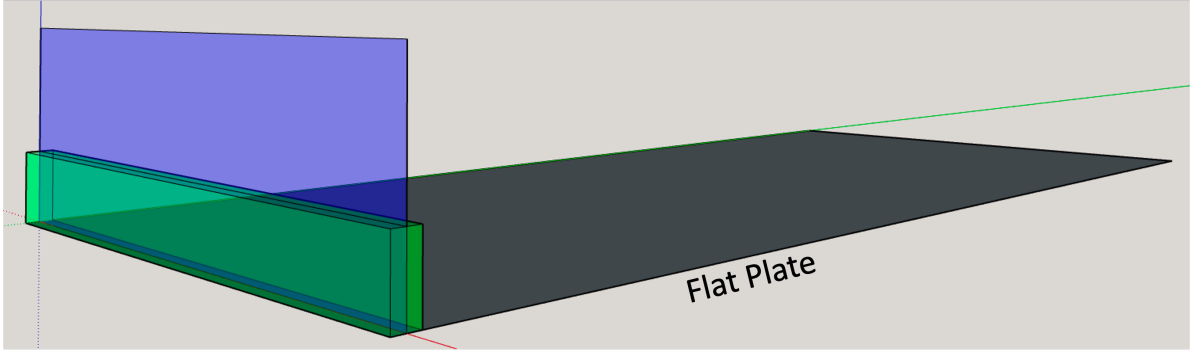


Figure 9: Illustration of the SEM virtual volume on flat plate inflow plane.

axially centered at x_{BC} and extends upstream and downstream of the inflow plane by a distance of the SEM length scale, l_{SEM} . In the spanwise direction (z in this example), the SEM box extends the full span of the inflow plane (say from z_{min} to z_{max}), with an extra l_{SEM} added to both ends. In the wall normal direction (y in this example), the SEM box extends from the wall surface, y_{wall} , to the height of the SEM profile, h_{SEM} , also with an extra l_{SEM} added to both ends. The final SEM box is the rectangular volume contained by

$$\begin{aligned} x_{BC} - l_{SEM} &\leq x \leq x_{BC} + l_{SEM} \\ y_{wall} - l_{SEM} &\leq y \leq y_{wall} + h_{SEM} + l_{SEM} \\ z_{min} - l_{SEM} &\leq z \leq z_{max} + l_{SEM} \end{aligned} \quad (44)$$

with volume of

$$\mathcal{V}_{SEM} = (2l_{SEM})(h_{SEM} + 2l_{SEM})(2l_{SEM}) \quad (45)$$

5.4.1.3 Define flow profile for SEM

The SEM requires a data file containing mean flow and turbulent stress variables for the boundary layer profile that the user wants to impose at the SEM inflow. Details regarding the required content and format of this data file are given in Section 5.4.2.3. Continuing with the previous flat plate example, Fig. 10 shows the boundary layer section of the idealized flow profile, identified by the green arrows, which will be imposed at the inflow. This flow profile will contain the mean flow variables

$$\bar{\mathbf{q}} = [\bar{\rho}, \bar{u}_i, \bar{T}]$$

at a set of locations within the boundary layer with an accompanying wall distance variable. The specific Reynolds stress tensor, $R_{ij} = \overline{u'_i u'_j}$, also needs to be provided in the data file to transform randomly generated numbers into velocity fluctuations that are physically correlated to the turbulent boundary layer profile.

The wall distance for each face point is used to interpolate the mean flow and Reynolds stress variables from the profile data to the face point. The Cholesky lower-triangular decomposition of the Reynolds stress tensor is performed resulting in the following matrix being defined at each face point

$$L_{ij} = \begin{bmatrix} \sqrt{R_{11}} & 0 & 0 \\ \frac{R_{21}}{L_{11}} & \sqrt{R_{22} - L_{21}^2} & 0 \\ \frac{R_{31}}{L_{11}} & \frac{R_{32} - L_{21}L_{31}}{L_{22}} & \sqrt{R_{33} - L_{31}^2 - L_{32}^2} \end{bmatrix}$$

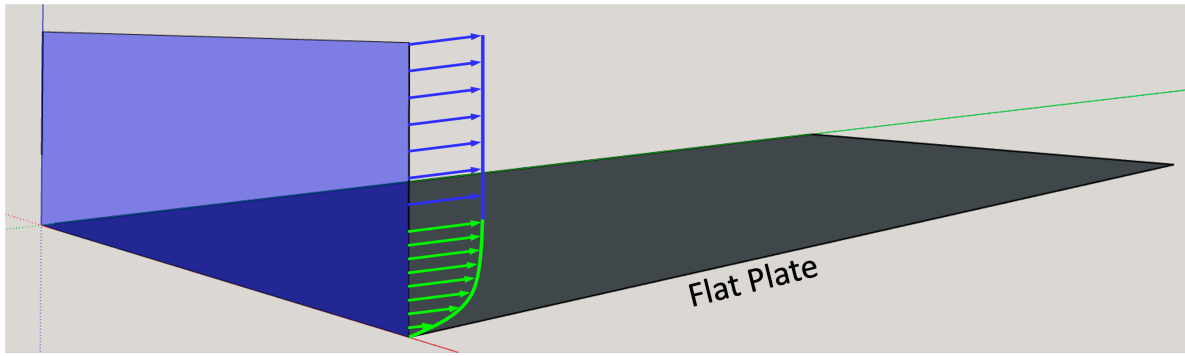


Figure 10: Illustration of the boundary layer section of the inflow profile, shown in green. The mean flow and turbulent stress variables defining this boundary layer are given in a data file that is input to the SEM boundary condition. The SEM will use this profile information to create an unsteady boundary condition with velocity fluctuations that should recreate this profile when averaged over time.

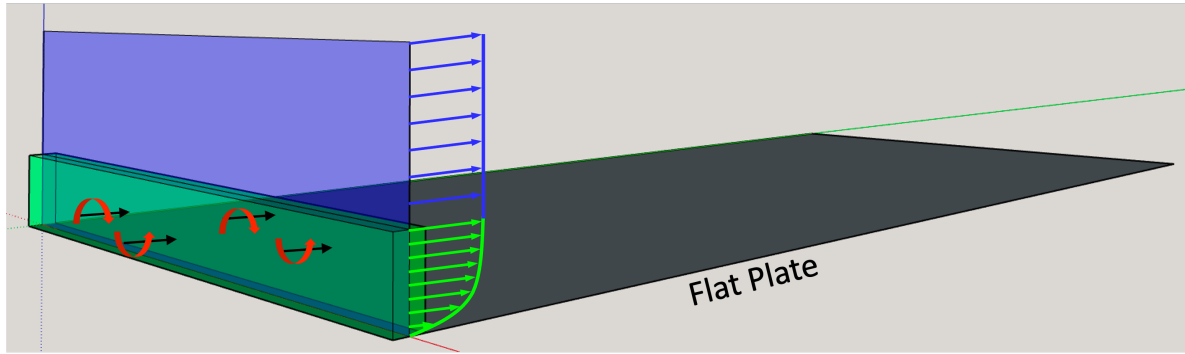


Figure 11: Illustration of the SEM synthetic eddies randomly placed within the SEM virtual box. The red arrows represent the random intensity/spin for the eddy in the streamwise direction.

5.4.1.4 Populate virtual box with randomly placed synthetic eddies

The SEM box is populated with randomly located synthetic eddies, with the general idea shown in Fig. 11. Each synthetic eddy is not truly an eddy in the physical sense. In reality, they are just source points that contain a randomized “intensity” in each coordinate direction. Using a random number generator that produces a value σ between 0 and 1, each intensity is simply

$$\epsilon_j = \begin{cases} -1 & \text{if } \sigma \leq 0.5 \\ +1 & \text{if } \sigma > 0.5 \end{cases}$$

A random number is generated separately for each coordinate direction to get the ± 1 intensity value in each direction. Since the term “intensity” often implies the scale of a variable instead of a binary value, it might be better to think of this as a positive/negative “spin” for the eddy in each direction.

The input value for the SEM length scale, l_{SEM} , determines the “zone of influence” for each eddy. This zone of influence is used in a distance-weighted shape function, described in Section 5.4.1.5, to determine how much an eddy affects nearby face points. Any face point that is beyond the distance l_{SEM} from the eddy will not be affected by the eddy at all.

The volume for an individual eddy, $\mathcal{V}_{\text{eddy}}$, can be thought of as the eddy’s total zone of influence.

By default, the zone of influence is applied separately for each coordinate direction, so the volume of an eddy is approximated as a cube, $\mathcal{V}_{\text{eddy}} = l_{\text{SEM}}^3$. The input `Use_Spherical_Eddy_Volume` can be used to treat the volume of the eddy as spherical, $\mathcal{V}_{\text{eddy}} = 4/3 \pi (l_{\text{SEM}}/2)^3 = 1/6 \pi l_{\text{SEM}}^3$, where l_{SEM} is the diameter of the sphere.

Finally, the number of eddies, N , placed in an individual SEM box is dependent on $\mathcal{V}_{\text{SEM}}$ and $\mathcal{V}_{\text{eddy}}$ using the equation

$$N = [\text{Box_Num_Eddies_Multiplier}] \times \frac{\mathcal{V}_{\text{SEM}}}{\mathcal{V}_{\text{eddy}}}$$

The multiplier is an input that can be used to globally increase or decrease the density of eddies within all of the SEM volumes.

5.4.1.5 Sum SEM fluctuations at each face point

Velocity fluctuations are imposed at each face point on the SEM boundary by summing the contributions from each synthetic eddy on that face point. Let $u_{j,k}^*$ be the unscaled fluctuating velocity in direction j for face point k . The contribution of the synthetic eddy n towards $u_{j,k}^*$ is given by

$$u_{j,k,n}^* = \epsilon_j f_{\wedge} \left(\frac{x_k - x_n}{l_{\text{SEM}}} \right) f_{\wedge} \left(\frac{y_k - y_n}{l_{\text{SEM}}} \right) f_{\wedge} \left(\frac{z_k - z_n}{l_{\text{SEM}}} \right)$$

where ϵ_j is the “intensity” or spin in the j -coordinate direction, and $f_{\wedge}(a)$ is a distance-weighted shape function. GFR uses a tent function (hence the $\wedge$) which is defined as

$$f_{\wedge}(a) = \begin{cases} \sqrt{\frac{3}{2}} (1 - |a|) & \text{if } |a| < 1 \\ 0 & \text{otherwise} \end{cases}$$

The total unscaled fluctuating velocity vector for face point k is obtained by summing the influence of all of the eddies on that point.

$$u_{j,k}^* = \sum_{n=1}^N u_{j,k,n}^*$$

5.4.1.6 Compute BC states using SEM fluctuations

Our ultimate goal is to compute the interior common fluxes from Section 2.1.4 for each face point on the SEM boundary. Since we already know the interior state at each face point (found by evaluating the solution polynomial from the interior cell at the face point), all that remains is to determine the boundary/exterior state for these points. Once the unscaled fluctuating velocity has been computed for face point k as shown in the previous section, it can be used with the SEM profile data that was interpolated to this face point in Section 5.4.1.3. The scaled fluctuating velocity in coordinate direction i , u'_i , is computed as the matrix multiplication

$$u'_{i,k} = L_{ij,k} u_{j,k}^*$$

Finally, the velocity for the boundary state, $u_{\text{BC},i}$, in the coordinate direction i is the sum of the mean and fluctuating parts

$$u_{\text{BC},i} = \bar{u}_i + u'_i$$

where each of these terms is assumed to be unique for the current face point k so the subscript k has been removed from each term for clarity.

If the namelist input variable `Include_Thermo_Fluctuations=.FALSE.` is used, then SEM fluctuations will not be applied to the thermodynamic variables from the SEM profile. The remaining two boundary state variables, ρ_{BC} and T_{BC} , are set to the value of the respective variable interpolated from the profile to the wall distance of the face point, i.e.,

$$\begin{aligned}\rho_{BC} &= \bar{\rho}_k \\ T_{BC} &= \bar{T}_k\end{aligned}$$

If `Include_Thermo_Fluctuations=.TRUE.`, then SEM fluctuations will be applied to the thermodynamic variables. These fluctuations are based on the Strong Reynolds Analogy following equations 38 and 39 from Toubert and Sandham.²⁴ After some simplifications, the temperature fluctuations are given by

$$T' = \frac{-u'\bar{u}}{c_p}$$

Here, u is the component of velocity in the streamwise direction and c_p is the specific heat at constant pressure. The boundary state temperature is just the sum of the mean and fluctuating temperatures

$$T_{BC} = \bar{T} + T'$$

and the boundary state density is then evaluated as

$$\rho_{BC} = \bar{\rho} \left(1 - \frac{T'}{\bar{T}} \right)$$

5.4.1.7 Convect eddies through the box

The last part of the SEM is convecting the eddies through the virtual box (although, technically, this is the first step of the SEM for every time step beyond the initialization). All eddies are convected through the SEM box based on the SEM convective velocity, U_{SEM} , specified in the input file. With $x_n(t)$ giving the streamwise location of eddy n at time t , and Δt giving the change in time between the current and previous time steps, the new streamwise location of the eddy is

$$x_n(t + \Delta t) = x_n(t) + U_{SEM}\Delta t$$

If the new location for any eddy takes it outside the box, i.e., $x_n > x_{BC} + l_{SEM}$, that respective eddy is deleted, and a new eddy is created at a random location on the inflow of the SEM box to take the place of the old eddy. Because it is highly probable that multiple eddies will leave the box each time step, a small randomized offset is added to the streamwise location of each new eddy to prevent “waves of eddies” that share the same axial location in the SEM box.

5.4.2 SEM Usage

Having described some of the basic theory and nomenclature of the synthetic eddy method, we will now discuss how it works within the GFR code.

5.4.2.1 Splitting off Subsonic BC for Boundary Faces Not in SEM Profile

In general, an SEM boundary condition is normally specified for the entire inflow boundary, but the height of the boundary layer defined in the SEM profile is usually only a small fraction of the overall inflow. Since the computational cost of the SEM scales with the number of boundary points involved in the SEM, any boundary points that are outside the height of the SEM profile, h_{SEM} , are split off into a new and completely separate subsonic inflow boundary condition that uses the standard

method for a subsonic inflow. In order to accomplish this, during the pre-processor, the wall distances for all boundary points on the SEM boundary are computed and then sorted to separate those boundary points with wall distance exceeding h_{SEM} into the new subsonic inflow boundary condition. The flow conditions for this new subsonic inflow boundary are a direct copy of the flow conditions specified in **BC_Input**(i) for the respective SEM boundary condition. This ensures that 1) the minimum number of boundary points are contained within the SEM boundary condition, and 2) the subsonic inflow outside the SEM is not overprescribed allowing for characteristic information to both enter and leave the computational domain through the boundary.

There are two details to keep in mind regarding this behavior of splitting off boundary points not within the SEM profile height into a new separate boundary condition.

- *Consistency between SEM and subsonic inflow boundary conditions.*

It is best to make sure that the flow conditions specified in **BC_Input**(i) for an SEM boundary condition are consistent with the flow conditions at the boundary layer edge of the SEM profile file. Stability issues can occur if there is a mismatch between the SEM profile and the corresponding subsonic inflow boundary condition. Since the subsonic inflow boundary conditions generally prescribe a fixed total temperature or total pressure, it is recommended to check that these two quantities are consistent between the specified boundary condition inputs and their respective values at the outer edge of the SEM profile. Other possible quantities to check are Mach number and velocity magnitude.

- *SEM profile height and efficiency.*

The profile data in the SEM profile file is not required to only contain the boundary layer part of the flow. The profile is allowed to contain as much of the main freestream flow as desired. There can be some inefficiencies from including part of the freestream flow in the SEM profile. The height of the SEM box is based on the height of the SEM profile, h_{SEM} ; and the number of synthetic eddies is based on the volume of the SEM box, V_{SEM} . Therefore, including parts of the freestream flow will increase the number of synthetic eddies in the SEM box. In the freestream, the turbulent stresses are generally near-zero. When the SEM algorithm computes a random fluctuation for a boundary point in the freestream section, the correlation converting this to a physical fluctuation will result in a non-fluctuating velocity profile since the turbulence is zero. All of this is to say that the freestream parts of the SEM profile will waste computational resources trying to compute velocity fluctuations where there are no fluctuations.

It is generally best to only specify the boundary layer profile up to, but not beyond, the boundary layer edge, i.e., $h_{\text{SEM}} = \delta_{\text{BL}}$. By specifying the minimum profile height, more boundary points will get separated to the new subsonic inflow boundary condition. The traditional subsonic inflow BC is more efficient than the SEM BC. Even with the highly efficient manner in which the SEM BC has been implemented, the traditional subsonic inflow BC is still more efficient; the difference between a full subsonic inflow BC and a full SEM inflow BC is approximately around 10-15% additional computational cost.

5.4.2.2 Restarting Simulations with SEM

When using one or more SEM boundary conditions, the state of each SEM is written to the restart file, where it is appended to the end after all the standard restart information is written. The SEM information written to the restart file includes:

- The random seed information to allow continuing the random number generator sequence from exactly where it left off the previous time step.

- The number of SEM boxes and the number of eddies in each box.
- For each individual eddy: its location, its random intensities in each direction, and its length scale in each direction.

There are three situations that can occur involving simulation restarts and SEM boundary conditions:

- *Restarting a simulation that previously used an SEM boundary condition and continuing use of the SEM.*

This is probably the most common of the three situations. The SEM information will be read from the restart file and a few sanity checks will be made to make sure the restart SEM is consistent with what is given in the input file. For example, the location of each eddy will be checked against the SEM boxes created from the information in the input file, and execution will be stopped with an error message if any eddy is located outside the SEM boxes.

This is also the only scenario in which the input variable `Use_Random_Seed_From_Restart` has any effect on a simulation. This variable controls whether or not the random seed information in the restart file is used when initializing the SEM algorithm at the start of the new/restarted simulation. If set to `.TRUE.`, the random seed from the restart will be used to reset the random number generator to exactly where it left off when the restart file was written, and the SEM will behave as if the simulation never stopped. The default value of `.FALSE.` will ignore the random seed in the restart file, and it will instead use the current date and time to create a new “random” seed for the random number generator. This might have a very small effect on the “randomness” of the locations and intensities generated for new eddies needed to replace eddies that have propagating completely through their SEM box. However, the total number of time steps needed for a simulation will almost certainly prevent this from having any kind of measurable effect on the simulation results.

NOTE: Very rarely (but it does happen), the SEM can randomly cause a simulation to abort execution due to an NaN or other floating point exception. Usually, the only way to get past one of these occasions without modifying the solution through limiting and/or filtering is to reset the random number generator down a different path by setting

`Use_Random_Seed_From_Restart=.FALSE.`

- *Restarting a simulation that previously used an SEM boundary condition and replacing it with a non-SEM boundary condition.*

This scenario is particularly useful when replacing the SEM with a periodic boundary condition. An example of this would be something like a turbulent channel flow case. Starting a streamwise periodic turbulent channel flow from a uniform solution would take an enormous amount of simulation (and computational) time to develop into a fully turbulent flow without any sort of tripping mechanism (which you would not want for a streamwise periodic flow). One way to speed this up is to use an SEM inflow with a subsonic outflow boundary condition and run the simulation for just over one flow-through-time to populate the channel with turbulent structures. Then the simulation could be restarted replacing the SEM inflow and subsonic-outflow boundary conditions with a streamwise periodic boundary condition. In this scenario, the SEM information is read from the restart file but it will be completely ignored since there are no SEM boundary conditions specified in the input file.

- *Adding an SEM boundary condition when restarting a simulation that did not use one.*

This scenario might occur when running a simulation without an SEM to get a stable flowfield as a pseudo-initialization and then add an SEM afterwards, or perhaps to add an SEM to the inflow to see how that might affect the overall solution. In this scenario, the SEM is basically set up in the exact same way as if you were starting a new simulation, with the difference being you maintain whatever time history information exists in the restart file.

5.4.2.3 SEM Profile File

The following three sections describe: the required format of the SEM profile file (5.4.2.3.1); the valid variables, and their respective identifier strings, that can be included in the SEM profile file (5.4.2.3.2); and how the SEM profile data is mapped onto the boundary face points using interpolation and a rotation transformation (5.4.2.3.3).

5.4.2.3.1 SEM Profile File Format

"V ₁ "	"V ₂ "	"V ₃ "	...	"V _n "
Np				
V ₁ (1)	V ₂ (1)	V ₃ (1)	...	V _n (1)
V ₁ (2)	V ₂ (2)	V ₃ (2)	...	V _n (2)
...				
V ₁ (Np)	V ₂ (Np)	V ₃ (Np)	...	V _n (Np)

In the above file format:

- Columns are separated by space(s). Multiple spaces are treated as one. Leading and trailing spaces on each line are ignored.
- n is the number of variables present in the file. It is also the number of columns in the file.
- Np is an integer giving the number of profile data locations, i.e., the number of rows of data.
- "V_{*i*}" is a character string specifying the variable name for the i^{th} column of data. Quotes are suggested but not required. Valid names are described in Section 5.4.2.3.2.
- V_{*i*}(*j*) is a floating point value for the j^{th} profile point of the i^{th} variable, where $j = 1, \dots, \text{Np}$ and $i = 1, \dots, n$. Data values are specified in SI units. The wall distance variable, d , should be in **grid scaled units** (i.e., meters). The profile will be imposed on the mesh points (after the **Grid.Scaling.Factor** has been applied to the mesh).
- The data rows should be sorted in the order of increasing distance from the wall.

5.4.2.3.2 SEM Profile Variable Names

Variable	Input Name	Description
d	Y R	wall distance
ρ	RHO DENSITY	density
u	U VX	axial velocity, aligned with the mean flow of the profile
v	V VY	vertical velocity, perpendicular to the wall
w	W VZ	span velocity, parallel to the wall and perpendicular to the mean flow
T	T TEMPERATURE	temperature
τ_{xx}	TAUXX TAU11 RXX R11 UU	$\overline{u'u'}$ normal stress
τ_{yy}	TAUYY TAU22 RYY R22 VV	$\overline{v'v'}$ normal stress
τ_{zz}	TAUZZ TAU33 RZZ R33 WW	$\overline{w'w'}$ normal stress
τ_{xy}	TAUXY TAU12 RXY R12 UV TAUYX TAU21 RYX R21 VU	$\overline{u'v'}$ shear stress ($= \tau_{yx}$)
τ_{xz}	TAUXZ TAU13 RXZ R13 UW TAUZX TAU31 RZX R21 WU	$\overline{u'w'}$ shear stress ($= \tau_{zx}$)
τ_{yz}	TAUYZ TAU23 RYZ R23 VW TAUZY TAU32 RZY R32 WV	$\overline{v'w'}$ shear stress ($= \tau_{zy}$)

A few things to note about these variables:

- These strings are case insensitive when read in by the SEM code.
- The wall distance variable, d , is the only required quantity in the SEM profile file. All other variables are optional.
 - If any of the mean flow variables (ρ, u, v, w, T) are not provided, the value of that variable in the profile will be set to a constant based on the conditions specified for the respective BC in **BC_Input(i)**. It should be noted that it is probably best that the axial velocity, u , also be specified since the intended use is for a boundary layer.
 - If any of the six stress terms are not provided, the value of that variable will be set to zero for the entire profile.
- The wall distance variable, d , should be in **grid scaled units** (i.e., meters). The profile will be imposed on the mesh points (after the **Grid_Scaling_Factor** has been applied to the mesh).
- The stress variables in the profile file are treated as $\tau_{ij} = \overline{u'_i u'_j}$ in units of m^2/s^2 , without the density or minus sign often used in the definition of turbulent stress. These would more correctly be referred to as single-point velocity correlation terms.
- Using the string “R” for wall distance tells the code to use a cylindrical box instead of a rectangular box. This has the same effect as adding **BC_Input(i)%SEM%Is_Cylindrical=.TRUE.** to the input file, where i is the index for the SEM BC of interest. This solely affects how the eddies are randomly generated to achieve a truly random distribution in the radial, r , direction. For a cylindrical SEM box, randomly generating r ($r_{\min} \leq r \leq r_{\max}$) and θ ($\theta_{\min} \leq \theta \leq \theta_{\max}$) coordinate values to place a new eddy will result in more eddies near the symmetry axis and

less eddies further away. In order to generate a truly uniform random distribution, we need to use the equation $r = \sqrt{\text{Random}() [r_{\max}^2 - r_{\min}^2] + r_{\min}^2}$ where $\text{Random}()$ is a function that generates a random number on the unit interval. This applies an area weighting to the radial distribution and results in a uniform distribution of eddies in the volume.

5.4.2.3.3 Mapping the SEM Profile onto Boundary Points

From the description of the SEM algorithm in sections 5.4.1.3 and 5.4.1.6, each boundary face point included in the SEM inflow must have a mean flow state $\overline{\mathbf{Q}}$ and Reynolds stress tensor R_{ij} prescribed from the SEM profile data. From Section 5.4.2.3.2, the profile data is defined as a 1D function of wall distance within a ‘wall-aligned coordinate system’. In order to get the prescribed conditions at a boundary point from this 1D data, two operations need to be performed on the data: interpolation and rotation.

Interpolation is needed because the boundary points are almost certainly not located at the exact wall distances given in the SEM profile, so the first step of interpolation is to compute the wall distance (i.e., distance from a given point to the boundary point closest to it with a wall boundary condition) for all SEM boundary points. Once the wall distance for each SEM boundary point is known, then the boundary layer properties can be obtained using a simple linear interpolation of the provided SEM profile file.

Because the profile data is given in a ‘wall-aligned coordinate system’, a rotation also needs to be applied to the velocity vector and stress tensor parts of the interpolated profile data such that it is aligned relative to how the wall normal of the nearest wall point is orientated within the Cartesian coordinate system. The remainder of this section consists of a list of three bullet points providing details on how the wall normal vector is used to create the transformation matrix needed for rotating the SEM profile data defined in a ‘wall-aligned coordinate system’ to the standard Cartesian coordinate system of the computational domain. This information is included here mostly for reference and/or code development purposes. Therefore, the general user only interested in how to specify and use an SEM boundary condition in a simulation can skip the remainder of this section.

- *Coordinate Transforms Using a Wall-Aligned Normal Vector*

In the description that follows, we are interested in mapping the SEM profile data onto the boundary points on the SEM inflow plane. The SEM data is provided as a 1D profile relative to wall distance and extending in the direction of the wall-normal vector. Therefore, our goal is to create a transformation matrix, $\mathbf{Q}$, that transforms the SEM vector/tensor data from the ‘wall-aligned coordinate system’ to the standard Cartesian coordinate system of the grid.

Let $(\hat{i}, \hat{j}, \hat{k})$ be the standard unit basis vectors for the Cartesian coordinate system in $\mathbb{R}^3$, and let $(\hat{e}_{\tilde{x}}, \hat{e}_{\tilde{y}}, \hat{e}_{\tilde{z}})$ be the unit basis vectors for the wall-aligned coordinate system also in $\mathbb{R}^3$. With $\tilde{\mathbf{v}} = v_{\tilde{x}}\hat{e}_{\tilde{x}} + v_{\tilde{y}}\hat{e}_{\tilde{y}} + v_{\tilde{z}}\hat{e}_{\tilde{z}}$ being an arbitrary vector defined by the wall-aligned basis vectors, our goal is to find $\mathbf{Q}$ such that

$$\vec{\mathbf{v}} = \mathbf{Q} \cdot \tilde{\mathbf{v}}$$

where $\vec{\mathbf{v}} = v_x\hat{i} + v_y\hat{j} + v_z\hat{k}$ is the same vector but defined by the standard Cartesian basis vectors.

NOTE: Our definition of $\mathbf{Q}$ may be the inverse of what is found in other sources, i.e., they might define it such that $\tilde{\mathbf{v}} = \tilde{\mathbf{Q}} \cdot \vec{\mathbf{v}}$. Since one of the properties of transformation matrices is $\mathbf{Q}^T = \mathbf{Q}^{-1}$, this just means our $\mathbf{Q}$ is equivalent to their $\tilde{\mathbf{Q}}^T$.

Before we continue to define $\mathcal{Q}$, we first need to define two surface normals for the relevant grid boundaries. Let

$$\hat{n} = \hat{n}_x \hat{i} + \hat{n}_y \hat{j} + \hat{n}_z \hat{k}$$

be the unit vector that is normal to the wall surface and points out of the wall and into the computational domain. Similarly, let

$$\hat{s} = \hat{s}_x \hat{i} + \hat{s}_y \hat{j} + \hat{s}_z \hat{k}$$

be a unit vector that represents some sort of “collective unit-normal” for the SEM inflow plane that points into the computational domain. A unit vector based on the direction of the inflow velocity vector could also be used here.

The general method for formulating a transformation matrix is

$$\mathcal{Q} = \begin{bmatrix} \cos(\hat{e}_{\bar{x}}, \hat{i}) & \cos(\hat{e}_{\bar{y}}, \hat{i}) & \cos(\hat{e}_{\bar{z}}, \hat{i}) \\ \cos(\hat{e}_{\bar{x}}, \hat{j}) & \cos(\hat{e}_{\bar{y}}, \hat{j}) & \cos(\hat{e}_{\bar{z}}, \hat{j}) \\ \cos(\hat{e}_{\bar{x}}, \hat{k}) & \cos(\hat{e}_{\bar{y}}, \hat{k}) & \cos(\hat{e}_{\bar{z}}, \hat{k}) \end{bmatrix}$$

where for any two vectors $\vec{u}$ and $\vec{v}$

$$\cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$$

One way to think of $\mathcal{Q}$ is that each column is a basis vector of the wall-aligned coordinate system expressed as a unit vector in the standard Cartesian coordinate system. We can then express this slightly differently as

$$\mathcal{Q} = \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix}$$

where we define the following three unit vectors as

$$\begin{aligned} \hat{a} &= \hat{e}_{\bar{x}} = a_x \hat{i} + a_y \hat{j} + a_z \hat{k} \\ \hat{b} &= \hat{e}_{\bar{y}} = b_x \hat{i} + b_y \hat{j} + b_z \hat{k} \\ \hat{c} &= \hat{e}_{\bar{z}} = c_x \hat{i} + c_y \hat{j} + c_z \hat{k} \end{aligned}$$

We can now start filling this transformation matrix. The wall-aligned coordinate system should be just that, so we define the $\hat{e}_{\bar{y}}$ basis vector to be aligned with the wall normal which is simply $\hat{b} = \hat{e}_{\bar{y}} = \hat{n}$ or $(b_x, b_y, b_z) = (\hat{n}_x, \hat{n}_y, \hat{n}_z)$. We know $\hat{c}$ is perpendicular to $\hat{b}$, so we just need to take the cross-product of some arbitrary vector with $\hat{b}$ to satisfy this requirement. Using $\hat{s}$ as this arbitrary vector, we can define $\hat{c}$ as

$$\begin{aligned} \hat{c} &= \frac{\hat{s} \times \hat{b}}{\|\hat{s} \times \hat{b}\|} \\ &= \frac{\hat{s} \times \hat{n}}{\|\hat{s} \times \hat{n}\|} \end{aligned}$$

Finally, knowing that the vector $\hat{a}$ must be perpendicular to both $\hat{b}$ and $\hat{c}$, we can define $\hat{a}$ as

$$\begin{aligned}\hat{a} &= \frac{\hat{b} \times \hat{c}}{\|\hat{b} \times \hat{c}\|} \\ &= \frac{\hat{n} \times (\hat{s} \times \hat{n})}{\|\hat{n} \times (\hat{s} \times \hat{n})\|}\end{aligned}$$

This can also be written in a slightly different way using the vector triple product identity

$$\hat{a} = \frac{\hat{s} - (\hat{n} \cdot \hat{s}) \hat{n}}{\|\hat{s} - (\hat{n} \cdot \hat{s}) \hat{n}\|}$$

Now that the three vectors $[\hat{a}, \hat{b}, \hat{c}]$ are defined, we can directly compute the matrix components of $\mathcal{Q}$, which are now entirely in terms of the two known vectors $\hat{n}$ and $\hat{s}$.

As an example, we will assume the inflow plane is constant x resulting in $\hat{s} = (1, 0, 0)$. After some simplifying, the resulting transformation matrix is

$$\mathcal{Q} = \begin{bmatrix} c & \hat{n}_x & 0 \\ \frac{-\hat{n}_x \hat{n}_y}{c} & \hat{n}_y & \frac{-\hat{n}_z}{c} \\ \frac{-\hat{n}_x \hat{n}_z}{c} & \hat{n}_z & \frac{\hat{n}_y}{c} \end{bmatrix}$$

where $c = \sqrt{\hat{n}_y^2 + \hat{n}_z^2}$.

- *Coordinate Transformation of a Vector*

As stated previously, our goal is to use this transformation matrix to map the velocity vector and Reynolds stress tensor in the SEM profile file onto the SEM inflow boundary points. For example, say we have a velocity vector $(\tilde{u}, \tilde{v}, \tilde{w})$ from the SEM profile defined in the wall-aligned coordinate system. In the case of a constant x inflow plane, transforming this vector to the standard Cartesian coordinate system is simply

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} c & \hat{n}_x & 0 \\ \frac{-\hat{n}_x \hat{n}_y}{c} & \hat{n}_y & \frac{-\hat{n}_z}{c} \\ \frac{-\hat{n}_x \hat{n}_z}{c} & \hat{n}_z & \frac{\hat{n}_y}{c} \end{bmatrix} \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix}$$

Expanding this out gives

$$\begin{aligned}u &= \tilde{u}c + \tilde{v}\hat{n}_x \\ v &= \frac{-\tilde{u}\hat{n}_x\hat{n}_y}{c} + \tilde{v}\hat{n}_y - \frac{\tilde{w}\hat{n}_z}{c} \\ w &= \frac{-\tilde{u}\hat{n}_x\hat{n}_z}{c} + \tilde{v}\hat{n}_z + \frac{\tilde{w}\hat{n}_y}{c}\end{aligned}$$

- *Coordinate Transformation of a Tensor*

Transforming a tensor in wall-aligned coordinates, $\tilde{\sigma}$, to reference coordinates, σ , is accomplished through

$$\sigma = \mathcal{Q} \cdot \tilde{\sigma} \cdot \mathcal{Q}^T$$

If the tensor $\tilde{\sigma}$ is symmetric, as in the case of the Reynolds stress tensor, we are only concerned about the terms $[\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{xz}, \sigma_{yz}]$. For a constant x inflow plane, expanding out the above equation and solving for these terms results in a set of six linear equations that can be rewritten in matrix form as

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{xz} \\ \sigma_{yz} \end{bmatrix} = \begin{bmatrix} c^2 & \hat{n}_x^2 & 0 & 2c\hat{n}_x & 0 & 0 \\ \frac{\hat{n}_x^2\hat{n}_y^2}{c^2} & \hat{n}_y^2 & \frac{\hat{n}_z^2}{c^2} & \frac{-2\hat{n}_x\hat{n}_y^2}{c} & \frac{2\hat{n}_x\hat{n}_y\hat{n}_z}{c^2} & \frac{-2\hat{n}_y\hat{n}_z}{c} \\ \frac{\hat{n}_x^2\hat{n}_z^2}{c^2} & \hat{n}_z^2 & \frac{\hat{n}_y^2}{c^2} & \frac{-2\hat{n}_x\hat{n}_z^2}{c} & \frac{-2\hat{n}_x\hat{n}_y\hat{n}_z}{c^2} & \frac{2\hat{n}_y\hat{n}_z}{c} \\ -\hat{n}_x\hat{n}_y & \hat{n}_x\hat{n}_y & 0 & c\hat{n}_y - \frac{\hat{n}_x^2\hat{n}_y}{c} & -\hat{n}_z & \frac{-\hat{n}_x\hat{n}_z}{c} \\ -\hat{n}_x\hat{n}_z & \hat{n}_x\hat{n}_z & 0 & c\hat{n}_z - \frac{\hat{n}_x^2\hat{n}_z}{c} & \hat{n}_y & \frac{\hat{n}_x\hat{n}_y}{c} \\ \frac{\hat{n}_x^2\hat{n}_y\hat{n}_z}{c^2} & \hat{n}_y\hat{n}_z & \frac{-\hat{n}_y\hat{n}_z}{c^2} & \frac{-2\hat{n}_x\hat{n}_y\hat{n}_z}{c} & \frac{\hat{n}_x(\hat{n}_z^2 - \hat{n}_y^2)}{c^2} & \frac{\hat{n}_y^2 - \hat{n}_z^2}{c} \end{bmatrix} \begin{bmatrix} \tilde{\sigma}_{xx} \\ \tilde{\sigma}_{yy} \\ \tilde{\sigma}_{zz} \\ \tilde{\sigma}_{xy} \\ \tilde{\sigma}_{xz} \\ \tilde{\sigma}_{yz} \end{bmatrix}$$

5.4.2.4 Inputs Controlling Global SEM Behavior

```
&nml_sem
  Use_Profile_For_Initialization = F
  Use_Random_Seed_From_Restart   = F
  Use_Spherical_Eddy_Volume     = F
  Include_Thermo_Fluctuations    = F
  Box_Num_Eddies_Multiplier     = 1.0
/
```

Inputs that are universally applied to all SEM boundary conditions are specified in the `nml_sem` input namelist block. Each one controls a different part of the SEM algorithm.

Use_Profile_For_Initialization

This is a logical variable that, if set to `.TRUE.`, will use the SEM profile to initialize the entire flow domain based on the y coordinate value and the profile height variable. This is very experimental and only really works for very simple geometries like a flat plate.

Use_Random_Seed_From_Restart

As discussed in Section 5.4.2.2, when using an SEM boundary condition, the SEM data will also be written to the restart files including the seed used for the random number generator.

This is a logical variable that, if `.TRUE.`, will read the seed for the random number generator from the restart file and use it to reset the random number generator to exactly where it left off when the restart file was written, as if the simulation never stopped.

The default value is `.FALSE.` which will not use the restart information, and will instead use the current date and time to create a new “random” seed for the random number generator. This might have the tiniest effect on the “randomness” of the strength and locations of the synthetic eddies in the SEM BC. However, the total number of time steps needed for a simulation will almost certainly prevent this from having any kind of measurable effect on the simulation results.

Very rarely (but it does happen), the SEM can randomly cause a simulation to die due to an NaN or other floating point exception. Setting this to false is the only way to get past one of these occasions without modifying the solution through limiting and/or filtering.

Use_Spherical_Eddy_Volume

This is a logical variable that determines if the volume of each synthetic eddy is treated as a box

(.FALSE.)[default] or as a sphere (.TRUE.). This input affects the number of eddies that will be used for an individual SEM boundary condition, since

$$[\# \text{ of eddies}] = [\text{Box_Num_Eddies_Multiplier}] \times \frac{[\text{Volume of SEM box}]}{[\text{Volume of synthetic eddy}]}$$

Include_Thermo_Fluctuations

This is a logical variable that determines if fluctuations are added to the density and temperature profiles when computing the SEM boundary states. This has the value .FALSE. by default meaning the density and temperature profiles are interpolated from the SEM profile with no fluctuations. If this is set to .TRUE., fluctuations are added to the density and temperature profiles based on Ref 24. Details on this are given in Section 5.4.1.6.

Box_Num_Eddies_Multiplier

This is a floating point variable that is used as a multiplier when computing the total number of synthetic eddies used for the SEM boundary conditions, i.e.,

$$[\# \text{ of eddies}] = [\text{Box_Num_Eddies_Multiplier}] \times \frac{[\text{Volume of SEM box}]}{[\text{Volume of synthetic eddy}]}$$

It should be noted that the number of eddies for an individual SEM BC are limited to be between a minimum of 5,000 and maximum of 500,000. These limits prevent (1) too few eddies for being present to adequately model the flow, and (2) too many eddies that add little to the modeling but increase computational cost.

5.4.2.5 Inputs Controlling Individual SEM Boundaries

```
&nml_bc
! Inputs for boundary group (1):
BC_Input(1)%SEM%Profile_File      = ""
BC_Input(1)%SEM%Is_Cylindrical    = F
BC_Input(1)%SEM%Length_Scale      = 0.0 ! [m]
BC_Input(1)%SEM%Convective_Velocity = 0.0 ! [m/s]
BC_Input(1)%SEM%Profile_Height    = 0.0 ! [m]
BC_Input(1)%SEM%Sub_In_BC_Grp     = 0
BC_Input(1)%SEM%Sub_In_BC_Idx     = 0
/
```

Inputs that are only applied to a specific SEM boundary are specified in the **nml_bc** input namelist block.

BC_Input(i)%SEM%Profile_File

This variable is a character string (max length of 150 characters) that specifies the path to the file containing the SEM profile used for the SEM boundary condition.

BC_Input(i)%SEM%Is_Cylindrical

This is a logical variable that is used to declare whether the SEM box is either rectangular or cylindrical. The default value of .FALSE. indicates the SEM box is rectangular, whereas a value of .TRUE. tells the code to use a cylindrical box for this SEM BC. See the last bullet in the list at the end of Section 5.4.2.3.2 for some additional technical details.

BC_Input(i)%SEM%Length_Scale

This is a floating point variable that gives the length scale (in meters) of the “zone of influence” of each synthetic eddy. See Section 5.4.2.6 for recommended values.

BC_Input(i)%SEM%Convective_Velocity

This is a floating point variable that gives the velocity (in m/s) at which the synthetic eddies are convected through the SEM box. See Section 5.4.2.6 for recommended values.

The following three variables are also part of the `BC_Input(i)%SEM` structure, but they are only used internally within the code:

```
BC_Input(i)%SEM%Profile_Height
BC_Input(i)%SEM%Sub_In_BC_Grp
BC_Input(i)%SEM%Sub_In_BC_Idx
```

Defining any of these three variables in the input file will be ignored. Their values will be determined after processing the SEM inputs, profile file, and respective boundary faces. This information will be written to the stdout display as part of the startup process. It can be used to verify that the profile data was properly interpreted.

5.4.2.6 Convective Velocity / Length Scale Recommendations

Previous testing seems to suggest a slower convective velocity should be used when using a smaller length scale, and a higher convective velocity should be used when using a larger length scale. Let U_{SEM} be the value of `BC_Input(i)%SEM%Convective_Velocity`. and let l_{SEM} be the value of `BC_Input(i)%SEM%Length_Scale`. Additionally, let U_{BL} be the axial velocity at the edge of the boundary layer in the SEM profile file, and let δ_{BL} be boundary layer height in the SEM profile. The following combinations all produce decent results in previous tests of a channel flow:

$$\begin{aligned} U_{\text{SEM}} &= U_{\text{BL}}/4 ; & l_{\text{SEM}} &= \delta_{\text{BL}}/5 \\ U_{\text{SEM}} &= U_{\text{BL}}/2 ; & l_{\text{SEM}} &= \delta_{\text{BL}}/5 \\ U_{\text{SEM}} &= U_{\text{BL}}/2 ; & l_{\text{SEM}} &= \delta_{\text{BL}}/4 \\ U_{\text{SEM}} &= U_{\text{BL}}/2 ; & l_{\text{SEM}} &= \delta_{\text{BL}}/3.75 \\ U_{\text{SEM}} &= U_{\text{BL}}/1 ; & l_{\text{SEM}} &= \delta_{\text{BL}}/3.75 \end{aligned}$$

and are shown in Fig. 12. From these results, recommended values for these two inputs are:

$$\begin{aligned} U_{\text{BL}}/4 &\leq U_{\text{SEM}} \leq U_{\text{BL}} \\ \delta_{\text{BL}}/5 &\leq l_{\text{SEM}} \leq \delta_{\text{BL}}/3.5 \end{aligned}$$

5.4.2.7 Example SEM Boundary Condition

```
&nml_bc
! Inputs for boundary group (1):
BC_Input(1)%Set_BC_Value      = 5           ! 5 = SEM Inflow BC
BC_Input(1)%Pressure          = 100000.0    ! [Pa]
BC_Input(1)%Temperature       = 300.0       ! [K]
BC_Input(1)%Mach              = 0.2
BC_Input(1)%SEM%Profile_File  = "[Path to profile file]"
BC_Input(1)%SEM%Is_Cylindrical = F
BC_Input(1)%SEM%Length_Scale  = 0.0001     ! [m]
BC_Input(1)%SEM%Convective_Velocity = 32.728 ! [m/s]
/
```

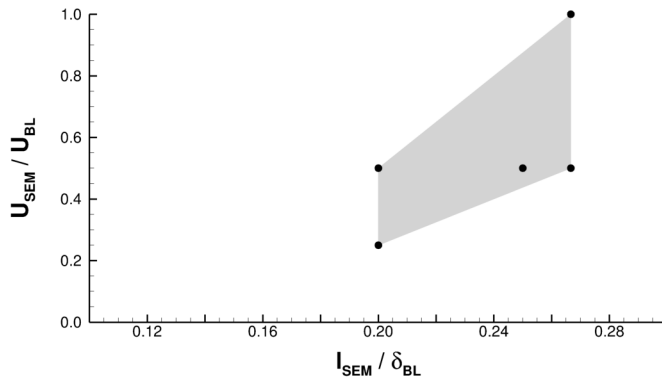


Figure 12: Recommended SEM length scale and convective velocity.

In this example, if temperature is not specified in the SEM profile file, the temperature for the entire SEM profile will be set to 300.0 K.

Additionally, all face points with a wall distance larger than the h_{SEM} determined from the profile file will be split off to a new subsonic inflow boundary condition as discussed in Section 5.4.2.1. The flow conditions for this new BC will be taken from (in this example) the Pressure, Temperature, and Mach components of `BC.Input(1)`.

The `Convective_Velocity` is chosen to provide $U_{SEM}/U_{BL}=0.5$.

5.4.3 SEM Frequently Asked Questions

- **Limitations on the physical inflow boundary: must it be planar?**

Yes, the inflow boundary must be planar. The algorithm that interpolates the SEM profile onto the inflow boundary points uses wall distance and the wall normal to rotate and interpolate the profile. The wall distance part would cause problems with a non-planar face.

- **Limitations on the physical inflow boundary: must it be constant- x ?**

Yes. In its current form, the SEM can only be used on a constant- x inflow boundary. One reason for this is that the synthetic eddies are only propagated in x .

- **Can the SEM be used to add freestream turbulence to a “farfield” subsonic inflow boundary, i.e., one that is not adjacent to a wall boundary?**

No, not at the moment. The SEM initialization algorithm is dependent on finding the nearest wall boundary point for each inflow boundary point. This is used to compute wall distance and get the wall normal at the wall point, which are both used to interpolate and rotate the SEM profile data to get the imposed flow profile at the inflow point. Attempting to use the SEM in this way would almost certainly cause either a crash via segmentation fault or (if it somehow got past the initialization) a badly imposed flow profile at the inflow. We are hoping to add this capability to the SEM in the near future.

- **Is the virtual volume extruded normal to the physical inflow boundary or along some velocity vector (based on inflow profile)?**

Neither. It takes the min/max of each coordinate direction over all the boundary points on the inflow plane and respectively subtracts/adds the SEM length scale to each value to get the extents of the virtual box.

- **Are the eddies convected normal to the physical inflow boundary or along some velocity vector (based on inflow profile)?**

The eddies are currently convected in only the x direction.

- **Is the sensory distance between an eddy and face point done in 3D (as a sensory spherical volume) or 2D (within the physical inflow plane)?**

A tent function is used to get a contribution in each direction, then all three contributions are multiplied together to get a total fluctuating value. There is an input to treat the sensory distance of each eddy as a sphere, `Use_Spherical_Eddy_Volume`, but this only affects how the eddy volume factors in to computing the total number of eddies within an individual SEM box and has no effect when computing the contribution of each eddy to the velocity fluctuations for the boundary condition.

- **What characteristic information feeds upstream?**

The SEM does not use any information from the interior to set the boundary states. However, as discussed in Section 5.4.2.1, any points on an SEM inflow boundary that are outside of the provided boundary layer profile are automatically separated into a standard subsonic inflow boundary that does permit outgoing characteristic information.

- **For subsonic flow, does SEM overprescribe the flow conditions?**

Yes, the SEM ends up prescribing all flow variables for the boundary state and does not use the interior state at all. This ultimately results in a “supersonic” or “fixed” inflow type boundary condition. However, this seems to cause little to no numerical issues, so we just tend to ignore this and take the win.

- **Should the entire inflow boundary be prescribed SEM or only some portion?**

It does not really matter. The parts of the inflow boundary outside the SEM profile height will be separated out to a new BC. If you prescribe the entire inflow boundary as SEM, it will result in two BC's: one for the SEM and one for a subsonic inflow BC for those faces outside the SEM profile height. Manually separating out the SEM and non-SEM parts of the inflow boundary will result in three BC's: the same two as before plus one for the manually separated out boundary faces. The difference in computational cost between these two should be negligible.

- **Is more than one SEM permitted?**

Yes, multiple SEM boundaries can be specified. It is possible, but unlikely, that the virtual boxes of two or more SEM BC's overlap slightly, but each SEM and its corresponding set of boundary faces are completely self-contained, meaning there is no influence from a different SEM.

- **How are corner flows handled?**

The distance variable in the SEM profile is height off the wall. The nearest wall point for each face point is found, and the SEM profile is interpolated to that face point using the normal at the wall point with the help of rotation transformation matrices to orient the flow velocities and stresses relative to the wall. For an inflow boundary containing a corner where two walls meet, this interpolation results in a sort of pinch-point of the flow profiles at the corner. This may not be the best 2D profile to specify at the inflow, but it seems to result in a reasonable corner flow by the time the unsteadiness transitions into a physical turbulent boundary layer.

- **How much streamwise distance is required to achieve realistic turbulence?**

If using an SEM convective velocity and length scale within the recommended values provided in Section 5.4.2.6, it could take anywhere from 10-15 δ_{BL} to develop realistic turbulence statistics in line with the SEM profile, with 12 δ_{BL} being a good starting estimate.

- **If the inflow profile represents the desired goal at some reference station, but the inflow boundary must be placed upstream for realistic turbulence to develop, should the profile height be scaled to account for boundary layer growth by the time it reaches the reference station?**

The height of the boundary layer once it has achieved realistic turbulence should be reasonably close to the SEM profile height. Therefore, based on the previous question, the inflow boundary should be placed 12-15 δ_{BL} upstream of the location you would like the SEM profile exist in the simulation flow field.

- **Is it best to only specify the boundary layer profile up to but not beyond the boundary layer edge?**

Yes. By providing a profile file with data that only extends to the edge of the boundary layer, more boundary points will be separated out to the new subsonic inflow boundary condition. The traditional subsonic inflow BC is more efficient than the SEM BC. However, the SEM BC is quite efficient, so the difference between a full subsonic inflow BC and a full SEM inflow BC is probably less than 10% computational cost.

- **How to specify which inflow(s) use SEM and which do not?**

An SEM boundary is specified by setting `BC_Input(i)%Set_BC_Value=5` in the `nml_bc` namelist input block.

- **What input specifies the file name for the SEM profile?**

`BC_Input(i)%SEM%Profile_File="[Path to profile file]"` in the `nml_bc` namelist input block.

- **Is the SEM profile 1D or 2D?**

The SEM profile is specified as a 1D profile as described in Section 5.4.2.3.

6 Reference Conditions

The user-specified reference conditions are used to initialize the volume solution and to compute the non-dimensional variables used within the flow solver. The reference conditions typically represent the freestream flow. These conditions include the gas properties: ratio of specific heats (γ), gas constant (R_{gas}), Prandtl number (Pr), Schmidt number (Sc), and coefficients to the Sutherland viscosity law. They also include the flow conditions: Mach number, Reynolds number, static pressure, temperature, density, as well as angle of attack α in the xy -plane and angle of sideslip β in the xz -plane. These flow angles are depicted in Fig. 13 and used to compute the velocity vector components as follows:

$$u = U \cos(\beta) \cos(\alpha) \quad (46)$$

$$v = U \cos(\beta) \sin(\alpha) \quad (47)$$

$$w = U \sin(\beta) \quad (48)$$

For positive angle of attack α between 0 and 90, the velocity will have a positive component in the y -direction. For positive angle of sideslip β between 0 and 90, the velocity will have a positive component in the z -direction.

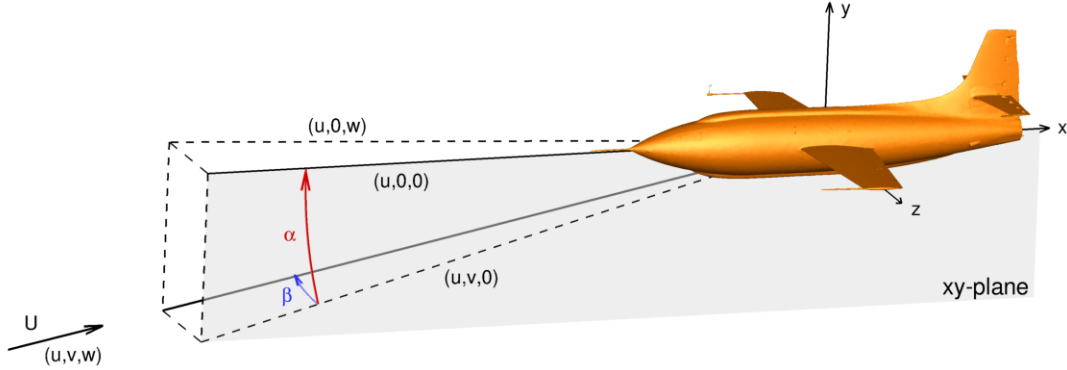


Figure 13: Flow angles.

Reference conditions are specified in the `nml.reference` input namelist block. Details regarding the velocity and length scale used to compute the Reynolds number can be found in the description of the `Reynolds` input variable.

Internally, GFR assumes $L_\infty = 1.0\text{m}$, $t_\infty = L_\infty/a_\infty$, and the following non-dimensionalization:

$$\begin{array}{lll} \tau^* = t/t_\infty & \rho^* = \rho/\rho_\infty & e_t^* = e_t/a_\infty^2 \\ x^* = x/L_\infty & p^* = p/\rho_\infty a_\infty^2 & \mu^* = \mu/\mu_\infty \\ u^* = u/a_\infty & T^* = T/T_\infty & \end{array}$$

NOTE:

The nondimensional time, τ^* , is only used in two places within GFR: the second column of the iteration data written to the GFR output and the input `Timestep` are both nondimensionalized the same as τ^* .

There is another nondimensional time that is used more frequently in GFR. This nondimensional time, t^* , uses the input `Time.Scaling.Factor` to scale the simulation time by something that may

be more useful for a given problem, e.g., flow-through-time for a channel flow or nozzle-diameter-time for a jet. These two nondimensional times are related through the equation

$$t^* = \mathcal{T}^* \cdot t_\infty \cdot [\text{Time_Scaling_Factor}] = t \cdot [\text{Time_Scaling_Factor}] \quad (49)$$

References to the term “tstar” in the inputs, output, source code, documentation, and elsewhere in the GFR universe are referring to the value t^* .

7 Boundary Conditions

7.1 How to Specify Boundary Conditions

7.1.1 Boundary Condition Values for Set_BC_Value

The primary method for specifying which boundary condition to use is by providing a value to the input `BC_Input(i)%Set_BC_Value`, where i is an integer for the current boundary condition. The supported values and their respective physical boundary conditions are listed in table 3.

Table 3: GFR boundary conditions.

Set_BC_Value :	Type of Boundary Condition
Inflow boundary conditions	
1 :	General inflow (7.2.1)
2 :	Subsonic inflow (7.2.2)
3 :	Supersonic inflow (7.2.3)
5 :	SEM (unsteady turbulent) inflow* (7.2.4)
Outflow boundary conditions	
11 :	General outflow (7.3.1)
12 :	Subsonic outflow (7.3.2)
13 :	Supersonic outflow (7.3.3)
Farfield boundary conditions	
21 :	Characteristic / Riemann-invariant inflow/outflow* (7.4.1)
22 :	General free-flow inflow/outflow (7.4.2)
23 :	Freestream (fixed) farfield (7.4.3)
24 :	Fixed farfield (7.4.4)
Wall boundary conditions	
50 :	Inviscid/slip wall (7.5.1)
52 :	Adiabatic viscous/no-slip wall (7.5.2)
53 :	Isothermal viscous/no-slip wall (7.5.3)
Numerical boundary conditions	
80 :	Symmetry boundary (7.6.1)
81 :	Periodic (translated and/or rotated) coupled boundaries (7.6.2)
Other internal BC values that may be reported.	
Note: Do <u>NOT</u> set Set_BC_Value to any of these values.	
0 :	Not a boundary condition
99 :	Partition/communication boundary condition
-9998 :	Unsupported boundary condition
-9999 :	Unknown boundary condition

* Not compatible with multi-component gas mixtures.

As shown in Section 2.1, the FR method needs to compute a common flux (2.1.4) at each interface flux point in order to create the continuous flux function (2.1.7–2.1.8) needed to update the solution at the end of each time step (2.1.9). The inviscid common flux is computed using

a Riemann solver (see [Invs_Flux_Method](#)) which requires as input a left and right state, Q_L and Q_R , respectively. Furthermore, the viscous common flux requires the gradient of the left and right states, ∇Q_L and ∇Q_R , respectively. See section *II.B.2* in Ref 8 for details on the discretization of the diffusion terms, and see inputs [Visc_Flux_Method](#) and [Visc_Variables_To_Ave](#) on usage options in GFR.

Due to the compact nature of FR/DG, each boundary condition really just determines how the boundary state is evaluated before computing the common flux function at the flux points of the respective boundary face. Let f_I^* be the inviscid common flux function — in this case some kind of upwind or Riemann flux function — defined as $f_I^* = f_I^*(Q_L, Q_R)$. Similarly, let f_V^* be the viscous common flux function defined as $f_V^* = f_V^*(Q_L, Q_R, \nabla Q_L, \nabla Q_R)$. For boundary faces, the left state is always from the interior cell and the right state is the exterior state associated with the boundary condition. Additionally, let Q_{BC} represent the boundary condition state, which is specified in the input file for the boundary condition; Q_{BC} is related — but not necessary equivalent — to Q_R . In general, the right state is a function of both the left/interior state and the boundary condition state, i.e., $Q_R = Q_R(Q_L, Q_{BC})$.

Sections [7.2–7.6](#) will provide details for each boundary condition option currently available in GFR. For each boundary condition type, this will cover what components of [BC_Input\(i\)](#) are required and/or optional to create the needed information in Q_{BC} , and how that information and the interior state, Q_L , are used to determine the right state, Q_R . For most of the boundary conditions, the right gradient state is just copied from the interior gradient state, i.e.,

$$\nabla Q_R = \nabla Q_L \quad (50)$$

so this will be assumed for each boundary condition below unless otherwise stated.

Finally, for a given boundary condition, if both [BC_Input\(i\)%Set_BC_Value](#) is not set and the grid file does not provide a boundary condition type, the name of the boundary condition — taken from [BC_Input\(i\)%Name](#) if provided, otherwise taken from the name assigned to the boundary condition in the grid file — is parsed for certain keywords. If a keyword match is found, a boundary condition will be assigned based on the keyword. Details on this are provided in Section [7.1.4](#).

7.1.2 Fully Defining the Boundary Condition State

All flow-through boundary conditions (i.e., inflow and outflow) require a minimum set of flow conditions to be defined in the input file in order to fully define the boundary condition state. There is one important detail to mention in how the boundary conditions are processed before continuing on about what inputs are needed to fully define the boundary condition state. The first thing GFR does when processing the boundary conditions specified in the input file is check if any individual boundary condition contains all default flow conditions. This check for default flow conditions checks all components of [BC_Input\(i\)](#) *except*:

```
BC_Input(i)%Name
BC_Input(i)%Set_BC_Value
BC_Input(i)%Periodic%...
```

If an individual boundary condition is found to be set to the default flow conditions, the flow state for that boundary condition will be set to the freestream reference conditions specified through the [nmlreference](#) namelist. If just *one* of *any* of the other components (or sub-components) of [BC_Input\(i\)](#) are *not* set to the default value, the remainder of this section on fully defining the boundary condition state applies to the respective boundary condition.

As discussed in Section [11.7](#), a variety of input variables are available to make specifying the boundary flow conditions as easy as possible. These input flow conditions include:

<code>BC_Input(i)%Pressure</code>	<code>BC_Input(i)%Total_Pressure</code>
<code>BC_Input(i)%Temperature</code>	<code>BC_Input(i)%Total_Temperature</code>
<code>BC_Input(i)%Density</code>	<code>BC_Input(i)%Total_Density</code>
<code>BC_Input(i)%Mach</code>	<code>BC_Input(i)%VelocityX</code>
<code>BC_Input(i)%AoA_Alpha</code>	<code>BC_Input(i)%VelocityY</code>
<code>BC_Input(i)%AoA_Beta</code>	<code>BC_Input(i)%VelocityZ</code>

As a general rule, four variables are sufficient to fully define the boundary condition state. If the velocity boundary condition has zero angle of attack, using the default value of zero for both angles of attack reduces the minimum number of variables to three. After reading in the specified variables for a boundary condition, GFR will iterate to define the full boundary state from the specified variables. This iteration will loop five times and use what variables have been previously set to define any remaining undefined state variables. After completing the five iterations, if any state variables remain undefined or are inconsistent with what was specified in the input file, GFR will abort execution and report an error including the values of all the input and computed state variables to help the user fix any errors in the input file.

One thing to note is that some boundary conditions may not actually need a fully defined boundary state to define the right state. An example of this would be a subsonic outflow boundary condition that only requires the exterior static pressure to compute the right state, and most CFD codes will only require a single input value for this quantity. However, at the moment, GFR will require additional information to fully define the boundary condition state, even though the boundary condition itself will completely ignore all the other state variables and only use static pressure to compute the right state.

An example of a minimal set of variables to define a boundary condition would be:

✓ Fully defined boundary condition

```
BC_Input(i)%Mach
BC_Input(i)%Temperature
BC_Input(i)%Total_Pressure
```

From this information, and using the default zero angle of attack, the velocity vector and remaining static/total density/pressure/temperature variables can be defined to fully define the boundary condition state. The following example would not constitute a minimal set of variables to fully define a boundary condition:

✗ Incomplete boundary condition

```
BC_Input(i)%Density
BC_Input(i)%Temperature
BC_Input(i)%Pressure
```

This boundary condition cannot be fully defined because no velocity condition can be deduced from the given quantities. However, this would be fully defined if one of these inputs were replaced with their respective total/stagnation value, since the Mach number for the boundary condition could be determined from having both the static and total conditions.

7.1.3 Situational Recommendations

This section includes some recommendations on which boundary conditions to use in certain situations.

- We recommend characteristic boundary conditions for farfield boundaries that are generally aligned with the mean flow. The next best option is the general free-flow boundary condition

to allow both inflow and outflow through the boundary. The farfield and fixed boundary conditions should best be avoided since they are essentially supersonic inflow boundary conditions, which could potentially cause problems if the interior flow propagates to the farfield boundary and needs to exit the computational domain.

- The characteristic boundary condition works fairly well as a general outflow boundary condition, but is not recommended for a subsonic inflow since it does not enforce total conditions. Maintaining total conditions is often necessary for internal or confined flow problems.
- If you are not entirely sure if an inflow or outflow boundary will be either fully subsonic or supersonic, then use the general versions of those boundary conditions since they are set up to use the “best” method of the respective boundary condition. However, if you do know that it will be entirely subsonic or supersonic, it is best to use that respective version of the boundary condition. This will slightly reduce the computational cost of determining the boundary condition states since it eliminates the need to evaluate the Mach number of the flow normal to the boundary face in order to determine which boundary condition to use. The overall computational savings may be small and only noticeable over tens of thousands of time steps.
- Symmetry boundary conditions should generally be avoided for LES since turbulent flows are never symmetric about a plane. However, a slip wall boundary condition — which is mathematically identical to a symmetry boundary conditions — can be used for LES provided its usage “makes physical sense”. As an example, the boundary conditions used for the THX3 simulations (see Fig. 19) used a slip wall for the bottom boundary which included all surfaces on the same plane as the flat plate behind the square nozzle. It might have been better to use a characteristic boundary condition to allow for downward spillage of the flow at the sides of the plate, but the slip walls seem to work sufficiently for this problem.
- Caution should be exercised when using periodic boundary conditions in the cross-stream directions for LES, since a narrow physical domain may limit the size of turbulent structures.

7.1.4 Setting Boundary Condition through Name Match

NOTE: The method described here for assigning a boundary condition type should be avoided. Instead, `BC_Input(i)%Set_BC_Value` should be used to specify the boundary condition type for all boundary conditions.

For a given boundary condition, if both `Set_BC_Value` is not set and a boundary condition type is not provided by the grid file, GFR will attempt to assign a boundary condition type based on the boundary condition name. This was the previous method used by GFR before `BC_Input(i)` was added to the code. It was retained for backwards compatibility mainly for use with Plot3D grid files, but was included as a double redundancy for other grid files.

Before continuing, it should be noted that it is *highly recommended* that all boundary condition types are specified in the input file using `Set_BC_Value`. If using a CGNS grid file, it is also good practice that each boundary condition is given a GFR-compatible boundary condition type in the grid file as backup. As stated at the beginning of this section, this method of using the boundary condition name to assign a boundary condition type should be avoided. It is only documented here to make the user aware that this behavior exists within the code.

The name for the boundary condition is taken from `BC_Input(i)%Name` if provided; otherwise, it is taken from the name assigned to the boundary condition in the grid file. The boundary

condition name is parsed for keywords which will assign certain boundary conditions depending on any keywords that might be found. Before checking for a name match, the boundary condition name is processed through the following steps to create a final character string for matching:

- All alphabet characters are converted to upper case.
- All spaces, underscores, and hyphens are removed.
- The substring "SUPERSONIC" is reduced to "SUP".
- The substrings "BOUNDARY", "BC", and "SONIC" are removed.

At this point, two special checks are attempted on the remaining string. If any substring within the remaining string matches "SEM" or "PERIODIC", `Set_BC_Value` will be set to an SEM boundary condition or a periodic boundary condition, respectively. If neither of those substrings exist, then the remaining strings from table 4 will be used to provide a positive name match and successfully assign a boundary condition type. The first column of table 4 lists the boundary condition value assigned to `Set_BC_Value` if, after manipulating `Name` using the above steps, the string that remains is equivalent to any of the strings in the last column. If no name match is made, `Set_BC_Value` is set to -9999 (Unknown boundary condition) and GFR will stop with an error message.

7.2 Inflow Boundaries

7.2.1 General Inflow Boundary Condition

This boundary condition combines subsonic and supersonic inflow boundary conditions into a single option. This boundary condition takes the velocity from the left/interior state, $\vec{u}_L$, and evaluates the magnitude of that velocity normal to the boundary face, $v_{n,L}$, as

$$v_{n,L} = \vec{u}_L \cdot \hat{n}$$

where $\hat{n}$ is the unit normal for the boundary face that points outside the computational domain. The interior static temperature T_L and $v_{n,L}$ can then be used to compute an interior Mach number normal to the inflow boundary, $M_{n,L}$, as

$$M_{n,L} = \frac{v_{n,L}}{\sqrt{\gamma R_{\text{gas}} T_L}}$$

If $M_{n,L} \leq -1$ (negative because the normal points outside the domain), the inflow is determined to be supersonic, and the right state is evaluated the same way as the supersonic inflow of Section 7.2.3 by just fixing it to the boundary condition state, i.e.,

$$Q_R = Q_{\text{BC}}$$

Otherwise, the inflow is subsonic, and the right state is evaluated based on the subsonic inflow boundary condition with `Sub_Inflow_Method` = 4 (7.2.2.4).

It should be noted that this boundary condition will always operate as if `Sub_Inflow_Method` = 4 when the interior Mach number is found to be subsonic. The value of the input `Sub_Inflow_Method` has no control over the subsonic inflow method used for this boundary condition and is therefore ignored.

Table 4: Boundary condition name strings.

Set_BC.Value	Boundary Condition	Section #	Matching Strings
1	General Inflow	(7.2.1)	"INFLOW" "INLET"
2	Subsonic Inflow	(7.2.2)	"SUBINFLOW" "SUBIN" "INFLOWSUB" "INSUB"
3	Supersonic Inflow	(7.2.3)	"SUPINFLOW" "SUPIN" "INFLOWSUP" "INSUP"
5	SEM Turbulent Inflow	(7.2.4)	"SEM"
11	General Outflow	(7.3.1)	"OUTFLOW" "EXIT"
12	Subsonic Outflow	(7.3.2)	"SUBOUTFLOW" "SUBOUT" "OUTFLOWSUB" "OUTSUB"
13	Supersonic Outflow	(7.3.3)	"SUPOUTFLOW" "SUPOUT" "OUTFLOWSUP" "OUTSUP" "EXTRAP" "EXTRAPOLATE"
21	Characteristic Inflow/Outflow	(7.4.1)	"CHARACTERISTIC" "CHAR" "CHARACTER"
22	General Inflow/Outflow	(7.4.2)	"GENERAL" "FREEFLOW" "FARFIELD"
23	Freestream (Fixed) Farfield	(7.4.2)	"FREESTREAM" "REFERENCE"
24	Fixed Farfield	(7.4.2)	"FIXED"
50	Slip Wall	(7.5.1)	"SWALL" "SLIP" "SLIPWALL" "WALLINVISCID" "EWALL" "EULER" "EULERWALL"
52	Adiabatic Wall	(7.5.2)	"AWALL" "ADIABATIC" "WALL" "WALLVISCOUS" "WALLVISCOUSHEATFLUX"
53	Isothermal Wall	(7.5.3)	"IWALL" "ISOTHERMAL" "WALLVISCOUSISOTHERMAL"
80	Symmetry Boundary	(7.6.1)	"SYMM" "SYMMETRY" "SYMMETRYPLANE"
81	Periodic Boundary	(7.6.2)	"PERIODIC"

7.2.2 Subsonic Inflow Boundary Condition

We first need to introduce notation for some quantities that will be utilized frequently when describing the different subsonic inflow methods. For the velocity vector $\vec{u}$, let $\mathcal{U}$ represent the total magnitude of the vector, i.e.,

$$\mathcal{U} = \|\vec{u}\|$$

We also introduce the following notation to represent the vector of the directional cosines for each respective component of a velocity vector

$$\hat{\beta} = \frac{\vec{u}}{\mathcal{U}} \quad (51)$$

where the accent $\hat{\cdot}$ is used here to denote this is a unit vector by definition. For most of the following subsonic inflow methods, the directional cosines of the prescribed inflow velocity are used to fix the angle of attack for the right state velocity, therefore $\hat{\beta}_R = \hat{\beta}_{BC}$.

Using these definitions, the velocity in the direction of the (outward pointing) surface normal

can be written as

$$v_n = \vec{u} \cdot \hat{n} = (\mathcal{U} \hat{\beta}) \cdot \hat{n} = \mathcal{U} (\hat{\beta} \cdot \hat{n}) = \mathcal{U} \mathcal{B} \quad (52)$$

where $\mathcal{U}$ is the magnitude of the velocity vector defined above and $\mathcal{B}$ represents the fraction of $\mathcal{U}$ in the direction of $\hat{n}$. For a flow that is perpendicular to the surface, $\mathcal{B} = -1$. We also note that $v_n < 0$ on an inflow surface and $|v_n| < a$ on a subsonic inflow.

The Riemann invariants and corresponding characteristic velocities are given by

$$R^+ = v_n + \frac{2a_L}{\gamma - 1} \quad \lambda_1 = v_n + a > 0 \quad \text{Outgoing} \quad (53)$$

$$R^- = v_n - \frac{2a_L}{\gamma - 1} \quad \lambda_2 = v_n - a < 0 \quad \text{Incoming} \quad (54)$$

It should be noted that for multi-component gas mixtures, the right state is determined in a manner similar to method 4 (7.2.2.4) unless method 2 (7.2.2.2) is specified via `Sub_Inflow_Method`.

7.2.2.1 Sub_Inflow_Method = 1

This method performs a Newton iteration on the total temperature to get the right state. It is based on the method found on pages 8–9 of Ref 25 from the subsection titled *Internal Flows* in Section 4.2.1. This method uses the specified boundary condition values $p_{t,BC}$, $T_{t,BC}$, and $\hat{\beta}_{BC}$ along with the outward propagating Riemann invariant, R^+ , to determine the right state.

We start with enforcing that the total temperature for the right state is equal to the prescribed total temperature

$$T_{t,R} = T_R \left(1 + \frac{\gamma - 1}{2} M_R^2 \right) = T_{t,BC}$$

Using the relations $M = \mathcal{U}/a$, $a^2 = \gamma R_{\text{gas}} T$, and $c_p = \gamma R_{\text{gas}} / (\gamma - 1)$, the total temperature can be rewritten as

$$T_{t,BC} = T_R + \frac{1}{2c_p} \mathcal{U}_R^2 \quad (55)$$

The next step is to set the outgoing Riemann invariant, R^+ , using the interior state. We use the subscript L to note that the invariant is computed from the left (interior) state.

$$R_L^+ = \vec{u}_L \cdot \hat{n} + \frac{2a_L}{\gamma - 1} \quad (56)$$

We want to find a right state normal velocity and speed of sound that keeps the outgoing invariant constant, or

$$\vec{u}_R \cdot \hat{n} + \frac{2a_R}{\gamma - 1} = R_L^+ \quad (57)$$

Using Eq. (52) to represent the velocity normal to the surface, we can rearrange Eq. (57) as

$$\mathcal{U}_R = \frac{1}{\mathcal{B}_{BC}} \left(R_L^+ - \frac{2a_R}{\gamma - 1} \right) \quad (58)$$

where the subscript BC has been added to $\mathcal{B}_{BC}$ to indicate that we are using the direction cosines from the boundary condition velocity vector to enforce the right state velocity to the prescribed angles of attack. Finally, we can combine Eq. (55) and Eq. (58) resulting in the equation

$$T_{t,BC} = T_R + \frac{1}{2c_p} \left[\frac{1}{\mathcal{B}_{BC}} \left(R_L^+ - \frac{2a_R}{\gamma - 1} \right) \right]^2$$

With $a_R = \sqrt{\gamma R_{\text{gas}} T_R}$, this equation becomes solely dependent on the right state temperature (T_R) with all the other variables being either constant or prescribed by the boundary condition state

$$f(T_R) = T_R + \frac{1}{2c_p} \left[\frac{1}{\mathcal{B}_{\text{BC}}} \left(R_L^+ - \frac{2a_R}{\gamma - 1} \right) \right]^2 - T_{t,\text{BC}} = 0 \quad (59)$$

We can solve this numerically using a Newton iteration

$$\begin{aligned} T_R^{n+1} &= T_R^n - \frac{f(T_R^n)}{f'(T_R^n)} \\ a_R^{n+1} &= \sqrt{\gamma R_{\text{gas}} T_R^{n+1}} \end{aligned}$$

where n is the current iteration of the Newton solver. All that remains is taking the derivative of $f(T_R)$, which condenses nicely to

$$f'(T_R) = 1 - \frac{\mathcal{U}_R}{\mathcal{B}_{\text{BC}} a_R}$$

After the Newton iteration converges to a solution for T_R and a_R , we can then continue on to compute the final right state variables needed for the boundary condition. The next step is to update $\mathcal{U}_R$ using Eq. (58) from which a Mach number, $M_R = \mathcal{U}_R/a_R$, can be evaluated. Finally, the primitive variables for the right state can be computed as

$$\vec{u}_R = \mathcal{U}_R \hat{\beta}_{\text{BC}} \quad (60)$$

$$p_R = p_{t,\text{BC}} \left(1 + \frac{\gamma - 1}{2} M_R^2 \right)^{-\gamma/(\gamma-1)} \quad (61)$$

$$\rho_R = \frac{p_R}{R_{\text{gas}} T_R} \quad (62)$$

7.2.2.2 Sub_Inflow_Method = 2

This method is exactly the opposite of the subsonic outflow method, using the density and velocity specified for the boundary condition and taking the static pressure from the interior

$$\begin{aligned} \rho_R &= \rho_{\text{BC}} \\ \vec{u}_R &= \vec{u}_{\text{BC}} \\ p_R &= p_L \end{aligned}$$

This method does not enforce the total temperature or pressure at the inflow, so caution should be taken with its use. This method is probably only suitable for a subsonic inflow that is far upstream from the body of interest. Even then, it is probably best to use one of the other subsonic inflow methods or the characteristic boundary condition (7.4.1) instead.

7.2.2.3 Sub_Inflow_Method = 3

This method uses the specified boundary condition values $p_{t,\text{BC}}$, $\hat{\beta}_{\text{BC}}$, and $H_{t,\text{BC}}$ (total enthalpy) along with the outward propagating Riemann invariant, R^+ , to determine the right state. It is very similar to Sub_Inflow_Method = 1 (7.2.2.1), but a quadratic equation is solved for a_R instead of using a Newton iteration for T_R . It is also equivalent to the method found in Section 2.7 of Ref 26 (pages 7–9), though somewhat different notation is used there and the flow is assumed to be normal to the boundary.

We start by rewriting Eq. (59) in terms of the speed of sound, with R^+ computed from the left (interior) state using Eq. (56),

$$\frac{a_R^2}{\gamma R_{\text{gas}}} + \frac{1}{2c_p} \left[\frac{1}{\mathcal{B}_{\text{BC}}} \left(R_L^+ - \frac{2a_R}{\gamma - 1} \right) \right]^2 - T_{t,\text{BC}} = 0 \quad (63)$$

and then rearranging the terms into a standard quadratic form with $H_{t,\text{BC}} = c_p T_{t,\text{BC}}$.

$$\left[\mathcal{B}_{\text{BC}}^2 + \frac{2}{\gamma - 1} \right] a_R^2 - [2R_L^+] a_R + \frac{\gamma - 1}{2} \left[(R_L^+)^2 - 2\mathcal{B}_{\text{BC}}^2 H_{t,\text{BC}} \right] = 0 \quad (64)$$

Using the quadratic formula to solve this equation results in two solutions for a_R . If for some reason both solutions are negative, the speed of sound from the interior state is used for the right state, i.e., $a_R = a_L$. Otherwise a_R is taken to be the maximum value between the two solutions.

The temperature can be computed from $T_R = a_R^2 / \gamma R_{\text{gas}}$, and the remaining primitive variables can be updated in the same way as in `Sub_Inflow_Method=1` (Section 7.2.2.1). $\mathcal{U}_R$ is updated using Eq. (58) from which a Mach number, $M_R = \mathcal{U}_R / a_R$, can be evaluated. Then $\vec{u}_R$, p_R , and ρ_R are computed from Eqs. (60)–(62).

7.2.2.4 Sub_Inflow_Method = 4

This is the default method used for the subsonic inflow boundary condition, because it is the simplest of the four available methods that also enforces the total conditions specified for the boundary condition. This method uses the specified boundary condition values $p_{t,\text{BC}}$, $T_{t,\text{BC}}$, and $\hat{\beta}_{\text{BC}}$ along with the interior Mach number at the boundary to determine the right state. Since both $p_{t,\text{BC}}$ and $T_{t,\text{BC}}$ are specified, this implies that $\rho_{t,\text{BC}}$ is also specified through the ideal gas law.

It starts by enforcing that the right state Mach number is set to the interior Mach number

$$M_R = M_L = \frac{\mathcal{U}_L}{a_L}$$

which then allows us to compute the static temperature for the right state based on the specified total temperature using

$$T_R = T_{t,\text{BC}} \left(1 + \frac{\gamma - 1}{2} M_R^2 \right)^{-1}$$

From here, the magnitude of the right state velocity can be computed as

$$\mathcal{U}_R = M_R a_R = M_R \sqrt{\gamma R_{\text{gas}} T_R}$$

which gives us everything that is needed to finally compute the primitive variables for the right state

$$\rho_R = \left(\frac{p_{t,\text{BC}}}{R_{\text{gas}} T_{t,\text{BC}}} \right) \left(1 + \frac{\gamma - 1}{2} M_R^2 \right)^{-1/(\gamma - 1)} \quad (65)$$

$$\vec{u}_R = \mathcal{U}_R \hat{\beta}_{\text{BC}} \quad (66)$$

$$p_R = \rho_R R_{\text{gas}} T_R \quad (67)$$

7.2.3 Supersonic Inflow Boundary Condition

This is the simplest of the inflow boundary conditions. Since the incoming flow is supersonic, no information from the interior of the computational domain can propagate upstream to the inflow boundary condition. Therefore, the right state is simply fixed to the boundary condition state, i.e.,

$$\mathbf{Q}_R = \mathbf{Q}_{\text{BC}}$$

7.2.4 Synthetic Eddy Method (SEM) Boundary Condition

The Synthetic Eddy Method (SEM) provides an unsteady turbulent inflow boundary condition that is crucial for accurate LES without requiring large run-up sections of computational grid for fully developing a turbulent boundary layer prior to the computational geometry of interest. The user provides an inflow profile with the desired time-averaged flow properties and turbulence statistics. At each time step, the SEM will then generate an unsteady inflow profile with turbulent fluctuations that, when time averaged, will reproduce the requested distribution. Section 5.4 provides extensive details on the synthetic eddy method: Section 5.4.1 provides the theory behind the method as well as an overview of how the method works; Section 5.4.2 provides details on how to use the SEM in GFR including the file format for the SEM profile, the relevant input variables with some recommendations, and other aspects unique to the SEM implementation within GFR; and Section 5.4.3 provides a collection of frequently asked questions.

7.3 Outflow Boundaries

7.3.1 General Outflow Boundary Condition

This boundary condition combines subsonic and supersonic outflow boundary conditions into a single option. This boundary condition takes the velocity from the left/interior state, $\vec{u}_L$, and evaluates the magnitude of that velocity normal to the boundary face, $v_{n,L}$, as

$$v_{n,L} = \vec{u}_L \cdot \hat{n}$$

where $\hat{n}$ is the unit normal for the boundary face that points outside the computational domain. The interior static temperature T_L and $v_{n,L}$ can then be used to compute an interior Mach number normal to the outflow boundary, $M_{n,L}$, as

$$M_{n,L} = \frac{v_{n,L}}{\sqrt{\gamma R_{\text{gas}} T_L}}$$

If $M_{n,L} \geq 1$, the outflow is supersonic, and the right state is evaluated the same way as Section 7.3.3 by fixing it to the interior state, i.e.,

$$Q_R = Q_L$$

Otherwise, the outflow is subsonic, and the right state is evaluated based on the subsonic outflow boundary condition (7.3.2), except there is no safe guard for the outflow being supersonic since that situation is already dealt with for this boundary condition; the safe guard on the flow becoming reversed does still exist, but again, only provides minimal protection against very minor violations of the assumed outflow.

7.3.2 Subsonic Outflow Boundary Condition

This boundary condition assumes the velocity normal to the boundary face is subsonic and leaving the computational domain. The right state is computed as

$$\vec{u}_R = \vec{u}_L$$

$$T_R = T_L$$

$$p_R = p_{BC}$$

This boundary condition does have some built-in safe guards in case the velocity at the boundary face becomes supersonic or reversed (i.e., inflow instead of outflow). However, these safe guards only provide minimal protection against minor violations of the assumed subsonic outflow.

- If you suspect your outflow boundary will likely contain large sections of both subsonic and supersonic outflow, you should consider using either the general outflow (7.3.1) or characteristic (7.4.1) boundary conditions.
- If you suspect there is a chance parts of your outflow boundary could reverse direction, at a minimum you should switch to using a characteristic (7.4.1) boundary condition instead of a subsonic outflow boundary condition. However, the better option is to modify the grid so that the outflow boundary is in a region that maintains a flow directed out of the domain and is not susceptible to reversing that direction.

7.3.3 Supersonic Outflow Boundary Condition

This is the reverse of the supersonic inflow boundary condition. Since the outgoing flow is supersonic, no information from the boundary condition can propagate upstream into the computational domain. Therefore, the right state is simply fixed to the interior/left state, i.e.,

$$Q_R = Q_L$$

If the outflow is adjacent to a no-slip wall, this boundary condition could result in stability problems within the subsonic region of the boundary layer. If stability issues occur, it is recommended to switch to either a characteristic (7.4.1) or general outflow (7.3.1) boundary condition, since both of these boundary conditions test the outgoing Mach number to determine whether to use a subsonic or supersonic boundary condition.

7.4 Farfield Boundaries

7.4.1 Characteristic / Riemann-Invariant Boundary Condition

The characteristic boundary condition is an inflow/outflow boundary condition that works for both subsonic and supersonic flows. This boundary condition uses the Riemann invariants to compute the right state variables based on what flow regime the boundary condition state is in, which is determined by the magnitude and direction normal to the boundary face of the interior velocity. Here are a few comments/warnings about using this boundary condition before continuing on with describing how it is formulated:

- When used as a subsonic inflow boundary condition, there is no guarantee that the total conditions at the inflow will be enforced. This should not be a problem if the inflow is significantly far upstream from the computational body of interest (e.g., external flows).
- This works well as a farfield boundary condition and has not shown any problems when the boundary condition velocity is aligned tangent to the boundary face.
- This works well for an outflow boundary condition for both internal and external flows, whether they are subsonic, supersonic, or mixed.
- If used for an outflow boundary condition, the prescribed velocity for the boundary condition is important for subsonic outflows, whereas the boundary condition velocity is ignored for the subsonic outflow boundary condition in Section 7.3.2. It should be noted that, similar to when used for subsonic inflows, this boundary condition applied to subsonic outflows does not guarantee that any specified boundary condition state will be held constant.

Recalling that $\hat{n}$ is the outward-facing unit normal for the boundary face, let $v_{n,L} = \vec{u}_L \cdot \hat{n}$ be the magnitude of the interior velocity normal to the boundary face, and $v_{n,BC} = \vec{u}_{BC} \cdot \hat{n}$ be the same

but for the prescribed boundary condition velocity. Similarly, let a_L and a_{BC} respectively be the speed of sound for the interior and boundary condition states. In order to compute the final right state, we need to compute four quantities: R^+ , the outgoing Riemann invariant; R^- , the incoming Riemann invariant; $\vec{v}_{t,R}$, the tangential velocity vector at the boundary face; and s_R , the entropy for the right state.

The outgoing Riemann invariant, R^+ , is evaluated as

$$R^+ = \begin{cases} v_{n,BC} + \frac{2a_{BC}}{\gamma - 1} & \text{if } \frac{v_{n,BC}}{a_{BC}} \leq -1 \quad (\text{supersonic inflow}) \\ v_{n,L} + \frac{2a_L}{\gamma - 1} & \text{otherwise} \end{cases}$$

The incoming Riemann invariant, R^- , is evaluated as

$$R^- = \begin{cases} v_{n,L} - \frac{2a_L}{\gamma - 1} & \text{if } \frac{v_{n,L}}{a_L} \geq 1 \quad (\text{supersonic outflow}) \\ v_{n,BC} - \frac{2a_{BC}}{\gamma - 1} & \text{otherwise} \end{cases}$$

The tangential velocity vector, $\vec{v}_{t,R}$, is evaluated as

$$\vec{v}_{t,R} = \begin{cases} \vec{u}_{BC} - (\vec{u}_{BC} \cdot \hat{n}) \hat{n} & \text{if } v_{n,L} \leq 0 \quad (\text{inflow}) \\ \vec{u}_L - (\vec{u}_L \cdot \hat{n}) \hat{n} & \text{if } v_{n,L} > 0 \quad (\text{outflow}) \end{cases}$$

Finally, the entropy, s_R , is evaluated as

$$s_R = \begin{cases} \frac{p_{BC}}{\rho_{BC}^\gamma} & \text{if } v_{n,L} \leq 0 \quad (\text{inflow}) \\ \frac{p_L}{\rho_L^\gamma} & \text{if } v_{n,L} > 0 \quad (\text{outflow}) \end{cases}$$

These four quantities are used to compute the right state in the following way. First, the characteristic normal velocity is computed by averaging the Riemann invariants

$$v_{n,R} = \frac{1}{2} (R^+ + R^-)$$

From there, the right state velocity can be determined by simply adding the normal and tangential components.

$$\vec{u}_R = v_{n,R} \hat{n} + \vec{v}_{t,R}$$

Next, the speed of sound is computed.

$$a_R = \frac{\gamma - 1}{4} (R^+ - R^-)$$

Finally, the right state density and pressure can be computed as

$$\rho_R = \left(\frac{a_R^2}{\gamma s_R} \right)^{\frac{1}{\gamma-1}}$$

$$p_R = \frac{a_R^2 \rho_R}{\gamma}$$

7.4.2 General Free-flow Boundary Condition

This boundary condition serves as a generalized inflow/outflow for both subsonic and supersonic flow regimes, essentially combining the general inflow (7.2.1) and general outflow (7.3.1) boundary conditions into a single option. This boundary condition takes the velocity from the left/interior state, $\vec{u}_L$, and evaluates the magnitude of that velocity normal to the boundary face, $v_{n,L}$, as

$$v_{n,L} = \vec{u}_L \cdot \hat{n}$$

where $\hat{n}$ is the unit normal for the boundary face that points outside the computational domain. The interior static temperature T_L and $v_{n,L}$ can then be used to compute an interior Mach number normal to the inflow boundary, $M_{n,L}$, as

$$M_{n,L} = \frac{v_{n,L}}{\sqrt{\gamma R_{\text{gas}} T_L}}$$

From here, there are four options based on the magnitude and sign of $M_{n,L}$:

- $M_{n,L} \geq +1$ (*Supersonic Outflow*)

The interior flow is supersonic and exiting the domain. The right state is evaluated the same way as the supersonic outflow boundary condition (7.3.3) by just fixing it to the interior state, i.e.,

$$Q_R = Q_L$$

- $M_{n,L} \leq -1$ (*Supersonic Inflow*)

The interior flow is supersonic because the magnitude of the Mach number is greater than or equal to 1. It is entering the domain since a negative value indicates the flow is aligned in the opposite direction as the outward facing normal. The right state is evaluated the same way as the supersonic inflow boundary condition (7.2.3) by just fixing it to the boundary condition state, i.e.,

$$Q_R = Q_{BC}$$

- $0 \leq M_{n,L} < 1$ (*Subsonic Outflow*)

The interior flow is subsonic and exiting the domain. The right state is evaluated based on the subsonic outflow boundary condition (7.3.2), except there are no safe guards in place for supersonic or reversed flows since the other three conditional statements of this bulleted list explicitly deal with those situations for this boundary condition.

- $-1 < M_{n,L} < 0$ (*Subsonic Inflow*)

The interior flow is subsonic because the magnitude of the Mach number is less than 1; it is entering the domain since a negative value indicates the flow is aligned in the opposite direction as the outward facing normal. The right state is evaluated based on the subsonic inflow boundary condition with `Sub_Inflow_Method = 4` (7.2.2.4). It should be noted that this boundary condition will always operate as if `Sub_Inflow_Method = 4` when the interior Mach number indicates a subsonic inflow condition; the value of the input `Sub_Inflow_Method` has no control over the subsonic inflow method used for this boundary condition and is therefore ignored.

7.4.3 Freestream Farfield Boundary Condition

This is a supersonic inflow boundary condition, with the only difference between this and the actual supersonic inflow boundary condition in Section 7.2.3 is this uses Q_∞ from the reference conditions in Section 6 as Q_{BC} .

7.4.4 Fixed Farfield Boundary Condition

This is exactly the same thing as using a supersonic inflow boundary condition. See Section 7.2.3.

7.5 Wall Boundaries

7.5.1 Inviscid/Slip Wall

The inviscid wall boundary condition, sometimes referred to as a slip wall, is used to model a frictionless solid-wall surface by enforcing the flow velocity and primitive variable gradients to be tangent to the boundary surface. The details provided here also apply to the symmetry boundary condition (7.6.1) because its formulation happens to be identical to a slip wall boundary condition.

For this boundary condition, there is no momentum or heat flux through the boundary so the thermodynamic properties at the wall are copied from the interior:

$$\begin{aligned}\rho_R &= \rho_L \\ T_R &= T_L\end{aligned}$$

To enforce tangency of the velocity at the wall, the wall velocity, $\vec{u}_R$, is computed by taking the interior velocity vector, $\vec{u}_L$, and subtracting out the normal components of the velocity vector, $(\vec{u}_L \cdot \hat{n})\hat{n}$, resulting in the tangential velocity at the surface:

$$\vec{u}_R = \vec{u}_L - (\vec{u}_L \cdot \hat{n})\hat{n} \quad (68)$$

For those interested, Eq. (68) can be “simplified” into a triple vector product as:

$$\vec{u}_R = \hat{n} \times (\vec{u}_L \times \hat{n})$$

For this boundary condition, the wall-normal component of the primitive variable gradients should be zero at the wall. To accomplish this, we repurpose Eq. (68) to compute tangential gradient vectors for each primitive variable at the surface given the interior gradient vector using

$$\vec{\varphi}_R = \vec{\varphi}_L - (\vec{\varphi}_L \cdot \hat{n})\hat{n}$$

where $\vec{\varphi}$ is a stand-in for the gradient of one of the primitive variables, i.e.,

$$\vec{\varphi} = \nabla\rho, \nabla u, \nabla v, \nabla w, \text{ or } \nabla T$$

7.5.2 Adiabatic Viscous/No-Slip Wall

The adiabatic viscous/no-slip wall boundary condition enforces a prescribed wall velocity at the boundary surface with zero heat flux through the boundary, i.e., $\partial T / \partial \hat{n} = 0$. Note, this boundary condition defaults to a stationary wall with zero wall velocity, but this can be overridden by specifying non-zero values for any of the boundary condition velocity components. The thermodynamic properties for this boundary condition are just copied from the interior:

$$\begin{aligned}\rho_R &= \rho_L \\ T_R &= T_L\end{aligned}$$

For most use cases, the prescribed wall velocity is simply zero, e.g., a stationary flat plate. However, there are certain situations where prescribing a non-zero wall velocity is needed; a canonical example would be the classic lid-driven cavity problem, and a more real-world example would be a spinning hub in a turbomachinery problem. If the prescribed wall velocity is not rotational, the following components of `BC_Input(i)` directly set the wall velocity for the boundary condition:

```
BC_Input(i)%VelocityX
BC_Input(i)%VelocityY
BC_Input(i)%VelocityZ
```

If the prescribed wall velocity is rotational, the input array

```
BC_Input(i)%Rotation_Rate(1:3)
```

is used to specify the rotation rate, $\vec{\Omega}$, (in RPMs) for all wall points in the boundary condition. The prescribed wall velocity at each wall point depends on the radius, $\vec{r}$, from the axis of rotation to the wall point. Since the axis of rotation for rotating boundaries originates from the specified `RotationCenter`, $\vec{x}_c$, we can also write this in terms of the position vector. The relation is given by (assuming implicit unit conversions):

$$\vec{u}_R = \vec{\Omega} \times \vec{r} = \vec{\Omega} \times (\vec{x} - \vec{x}_c)$$

Note, this is different from prescribing a non-rotational wall velocity where all wall points are set to the same velocity.

If using a moving reference frame, the input

```
BC_Input(i)%Is_Specified_In_MRF
```

can be used to simplify the boundary condition inputs for cases involving multiple solid wall boundaries with velocities dependent on which reference frame the boundary is viewed from. An example of this would be a turbomachinery fan containing a spinning hub and rotors — with a rotation rate of $\vec{\Omega}$ — inside a stationary case, which is how this problem is viewed from the inertial/fixed/absolute reference frame. However, when viewed from the rotating reference frame, the hub and rotor are stationary and the case is now rotating at $(-\vec{\Omega})$. Note, the prescribed wall velocity components are specified as the absolute velocity components.

To illustrate the use of `BC_Input(i)%Is_Specified_In_MRF`, let us continue with this turbomachinery example of a spinning hub/rotor surrounded by a stationary case, and our simulation is using a rotating reference frame with the same rotation rate as the spinning hub/rotor. Let `BC_Input(1)` be the boundary condition inputs for the casing, and `BC_Input(2)` be for the hub/rotor. Using the default values of zero for the three velocity components, the boundary conditions for this case can be provided as:

```
BC_Input(1)%Is_Specified_In_MRF = .FALSE.
BC_Input(2)%Is_Specified_In_MRF = .TRUE.
```

One benefit from this approach is that the boundary condition wall velocities will automatically adjust if the rotation rate of the rotating reference frame is changed.

For this boundary condition, the temperature gradient is the only right gradient state treated differently from the other flow-through boundary conditions (Eq. (50)). The adiabatic condition prescribing zero heat flux through the boundary requires the temperature gradient to be tangential to the boundary surface. This is exactly the same as the temperature gradient boundary condition for the slip wall in Section 7.5.1, which is

$$\nabla T_R = \nabla T_L - (\nabla T_L \cdot \hat{n}) \hat{n}$$

7.5.3 Isothermal Viscous/No-Slip Wall

The isothermal viscous/no-slip wall boundary condition enforces a prescribed wall velocity and temperature at the wall boundary surface. The isothermal no-slip wall boundary condition is very similar to the adiabatic no-slip wall in Section 7.5.2, with the only difference between them being how the wall temperature and temperature gradients are set. Therefore, the remainder of this section only provides details for what is done differently from the adiabatic wall boundary condition. Particularly, all of the details in Section 7.5.2 on how the prescribed wall velocity is determined are exactly the same for isothermal walls.

The backbone of the isothermal wall boundary condition is prescribing a fixed wall temperature for a solid boundary surface. The prescribed wall temperature for the boundary condition, T_{BC} , is specified using the input `BC_Input(i)%Wall.Temp`. For the rest of the thermodynamic gas properties, pressure is taken from the interior, and then the ideal gas law is used to compute the density at the wall, resulting in the following right state variables:

$$\begin{aligned}T_R &= T_{BC} \\p_R &= p_L \\ \rho_R &= \frac{p_R}{R_{gas}T_R}\end{aligned}$$

Finally, unlike for adiabatic walls, the isothermal wall temperature gradient is just copied from the interior,

$$\nabla T_R = \nabla T_L$$

Since this is also done for the gradients of all of the other variables, the right gradient states for the isothermal wall are just the standard “flow-through” gradient boundary condition from Eq. (50) used by most boundary conditions.

7.6 Numerical Boundaries

7.6.1 Symmetry Boundary Condition

This boundary condition is generally used to reduce the size of the overall computational domain by taking advantage of flow symmetry for a given problem. For example, instead of running a simulation of a full airplane, a symmetry boundary can be created along the centerline of the airplane geometry so the simulation is only performed on half of the airplane. Although intended for different uses, this boundary condition is equivalent to the inviscid/slip wall boundary condition from Section 7.5.1.

NOTE: It should be warned that using symmetry boundary conditions to take advantage of flow symmetry is generally not recommended for quality LES. The mean flow may be symmetric over long time-averages, but the instantaneous turbulent flow structures are *not* symmetric.

7.6.2 Periodic Coupled Boundaries

Periodic coupled boundaries permit the flow exiting one boundary of the computational domain to enter through a different boundary, and vice versa. These two boundaries form a coupled pair of periodic boundary conditions. Periodic boundaries fall into two categories of periodicity: translational and rotational. A pair of periodic boundaries can be either translational, rotational, or a combination of both. Translational periodicity is quite useful for problems where the flow in one or more directions can be treated as infinite, e.g., channel flow; details and examples are

discussed in Section 7.6.2.2.2. Rotational periodicity is useful for problems containing rotational symmetries, e.g., a single blade from a turbomachinery rotor; details and examples are discussed in Section 7.6.2.2.3.

We first start with some preliminary definitions and notation that will be used for periodic coupled boundary conditions. For all examples in this section, we will be working with a pair of periodic boundary conditions that are specified in the input file using `BC_Input(i)` and `BC_Input(j)`. For the remainder of this section, every instance of `BC_Input(i)` will have the following two specified values

```
BC_Input(i)%Name = "Face Side A"
```

```
BC_Input(i)%Set_BC_Value = 81
```

and similarly, `BC_Input(j)` will have the following two specified values

```
BC_Input(j)%Name = "Periodic-2"
```

```
BC_Input(j)%Set_BC_Value = 81
```

Note, the strings used here for the `Name` components of each boundary condition have just been randomly created to illustrate that there is no restriction that they be coordinated in any way. In practice, it is recommended that they are coordinated just to make things easier to understand, say something like "Periodic-1A" and "Periodic-1B" if there might be more than one pair of periodic boundary conditions.

7.6.2.1 Things to Consider when Using Periodic Boundary Conditions

- After all translation and rotation operations are performed on one side of a pair of coupled periodic boundaries, the linear grid components (linear cell vertices and faces) of the two boundaries need to result in a 1-to-1 match. The algorithm for finding all of the matching pairs of vertices between two periodic boundaries works by computing the distance from the current vertex from the left boundary to all the vertices on the right boundary that have not yet been matched to a left vertex. The matching right vertex is simply determined to be the one with the minimum distance to the current left vertex. If the minimum distance is greater than 0.001 **grid scaled units** (i.e., meters), an error will occur.

For a pair of matching linear cell faces on periodic boundaries, the geometries for both faces of the pair need to match (i.e., both triangles, or both quadrilaterals), and the set of grid vertices for both faces need to create a matching set. For example, let us say we have face *a* defined by vertices [1, 2, 3], and it was found to be a periodic match for face *b* defined by vertices [143, 22, 200]. For this to be a valid match, vertex 1 needs to match to one of the three vertices for face *b*; for this example we will say it matches to vertex 200. Vertex 2 then needs to be a match to either vertex 143 or 22; we will say 22 for this example. That means vertex 3 *must* be a match to vertex 143, or else this is not a valid pair of matching faces. Finally, the order of the matched vertices is not important here since after the faces are confirmed to be a valid match, the ordering will be used to find the relative rotation between the two faces.

- Provided the previous bullet is satisfied such that the linear grid components of a pair of periodic boundaries are found to be an exact 1-to-1 match, there are no other requirements on the topology, shape, or orientation of the boundaries. Periodic boundaries can be high-order, curved surfaces, and they are not required to be planar or aligned with any coordinate direction/axis. For example, a two-dimensional infinite turbine blade cascade could be simulated using only a single blade with translationally periodic boundaries along the curved geometric mid-passage lines above/below the blade.

- For LES, make sure the distance between a pair of coupled periodic boundaries are separated by a distance that is sufficient enough to allow for turbulent structures to breakdown/dissipate/cascade into finer turbulent length scales along the direction of the periodicity. If a pair of coupled periodic boundaries are too close to each other, the turbulent length scale will get artificially constrained in the periodic direction. This will cause energy to get trapped in the larger/macro vortical structures, ultimately preventing a truly three-dimensional turbulent flow.
- Similar to the previous bullet, be careful with over-prescribing a simulation with too many periodic boundaries in such a way that the periodic boundary conditions change the flow dynamics. The use of periodic boundary conditions is actually creating an infinite array of identical coupled simulations that are interacting with each other. If the effect of this coupling is not well thought out, it can have unforeseen consequences ranging from unstable simulations that can be hard to diagnose the source of the instabilities, to questions on the validity and/or accuracy of a simulation and any conclusions that may have resulted.

As mentioned in the previous bullet, having periodic boundaries too close to each other is an example of this, but another example is the fully periodic isentropic vortex problem.¹¹ For this problem, 1) the Euler equations provide no physical dissipation; 2) the high-order FR scheme provides minimal numerical dissipation; and 3) the fully periodic boundaries prevent both numerical and machine roundoff errors from escaping the system. With no means to dissipate the error or let it leave the system, all of the numerical and roundoff errors created by the simulation become trapped, and eventually these errors grow large enough to negatively effect the simulations with unstable/diverging solutions. Replacing the periodic boundaries with characteristic boundary conditions completely eliminates all instabilities allowing us to demonstrate that the FR method achieves super accuracy for this problem. Further details about this can be found in Ref 11.

7.6.2.2 Specifying Coupled Periodic Boundary Conditions

When specifying periodic boundary conditions, some input parameters are commonly used with all such boundaries, and others are specific to translational or rotational periodicity. We discuss each in the sections that follow.

7.6.2.2.1 Inputs Common to All Periodic Boundaries

This section discusses inputs that can be used with both translational or rotational periodic boundaries. The `Periodic` subcomponent of `BC_Input(i)` is used to provide the information that will allow GFR to group two boundary conditions into a matching pair of periodic boundaries, which in this example will be `BC_Input(i)` and `BC_Input(j)`. Both sides of a periodic pair must provide their respective information in the input file. As you will see, the information required by opposing sides is the reverse/negative of each other, and is used to aid error checking of the periodic boundary setup.

The two inputs used to define a matching periodic pair are

```
BC_Input(i)%Periodic%Matching_BC_Name
BC_Input(i)%Periodic%Matching_BC_Idx
```

The values for these two inputs need to provide the respective information from the opposing matching periodic boundary condition. Using boundary conditions i and j defined at the beginning of Section 7.6.2, the inputs for boundary condition i would look like

```
BC_Input(i)%Name = "Face Side A"
```

```
BC_Input(i)%Set_BC_Value = 81
BC_Input(i)%Periodic%Matching_BC_Name = "Periodic-2"
BC_Input(i)%Periodic%Matching_BC_Idx = j
```

and the inputs for boundary condition j would look like

```
BC_Input(j)%Name = "Periodic-2"
BC_Input(j)%Set_BC_Value = 81
BC_Input(j)%Periodic%Matching_BC_Name = "Face Side A"
BC_Input(j)%Periodic%Matching_BC_Idx = i
```

The remaining parts of `BC_Input(i)%Periodic` are used to define the physical orientation between the pair of periodic boundaries, and will be covered in the next two sections.

7.6.2.2.2 Translated Periodic Boundary Condition

Translated periodic boundary conditions are useful for problems where the flow in one or more directions can be treated as infinite. Examples primarily consist of two-dimensional geometries (e.g., an airfoil) that are converted to a three-dimensional body with an infinite span in the third dimension. Another example would be a 3D channel flow with translated periodic boundary conditions in both the span-wise *and* streamwise directions. The periodicity in the streamwise direction allows for the flow to recirculate indefinitely until it becomes a fully developed channel flow. However, the streamwise periodicity requires a body force to drive the streamwise bulk flow in the absence of a pressure differential generated by traditional inflow/outflow boundary conditions.

For each periodic boundary condition, the input vector

```
BC_Input(i)%Periodic%Translation(1:3)
```

provides the translation vector (in grid units) that transforms the current boundary surface to align it exactly with its corresponding periodic pair. For a given periodic pair with translational periodicity, the following relation must be satisfied:

```
BC_Input(i)%Periodic%Translation(:) = - BC_Input(j)%Periodic%Translation(:)
```

For example, say you have an S-duct that is 5.0 inches long and drops 1.0 inch vertically, and your grid file for this geometry is also provided in inches. If you wanted to make this problem streamwise periodic, your inputs for boundary condition i would look like

```
BC_Input(i)%Periodic%Translation(1:3) = [ 5.0, 1.0, 0.0]
```

and the inputs for boundary condition j would look like

```
BC_Input(j)%Periodic%Translation(1:3) = [-5.0, -1.0, 0.0]
```

Of course, an S-duct is generally not a problem that you would want to make streamwise periodic. It is merely used here to illustrate an example of the periodic translational vector having components in multiple directions.

The elements on the boundary faces must be identical in order to have one-to-one connectivity between cell faces. When using translated periodic boundary conditions for large-eddy simulations, one must make sure the extent of the periodic direction is long enough as to not artificially constrain the turbulent length scale in the periodic direction.

7.6.2.2.3 Rotated Periodic Boundary Condition

Rotated periodic boundary conditions are useful for problems containing rotational symmetry. This is particularly prevalent in turbomachinery flows where simulations of a single periodic rotor blade can save significant computational expense compared to simulations of the full 360-degree rotor wheel with dozens of rotor blades. There are three subcomponent arrays of `BC_Input(i)%Periodic`

that can be used to specify the rotational transformation between a pair of rotated periodic boundary conditions:

```
BC_Input(i)%Periodic%RotationCenter(1:3)
BC_Input(i)%Periodic%RotationAngle(1:3)
BC_Input(i)%Periodic%RotationAxis(1:4)
```

One may use either `RotationAngle` or `RotationAxis` to define rotated periodic boundary conditions, but not both. If both are specified, `RotationAxis` will be ignored and `RotationAngle` will be used. For both methods, the input vector `RotationCenter(1:3)` gives the coordinates for the rotation origin (in **grid units**) used for defining the rotation angles between rotated periodic boundary conditions.

The first way to specify the rotational transformation between a pair of rotated periodic boundary conditions is through the input vector `RotationAngle(1:3)`. This vector gives the rotation angles (in degrees) about each of the coordinate axes from the current periodic boundary interface to the connecting periodic boundary interface. If rotating about more than one axis, the rotation is performed first about the *x*-axis, then the *y*-axis, and finally the *z*-axis. For a given periodic pair with rotational periodicity, the following relation needs to be true:

```
BC_Input(i)%Periodic%RotationAngle(:) = - BC_Input(j)%Periodic%RotationAngle(:)
```

The second way to specify the rotational transformation between a pair of rotated periodic boundary conditions is through the input vector `RotationAxis(1:4)`. The first 3 values of this array define a vector representing the axis of rotation for this rotated periodic boundary condition, and the last value of the array defines the rotation angle (in degrees) about this axis. The magnitude of the vector defining the axis of rotation is not used; the sole purpose of this vector is to define the direction of the rotation axis. For a given periodic pair with rotational periodicity, one of the two following sets of relations needs to be true:

```
BC_Input(i)%Periodic%RotationAxis(1:3) = BC_Input(j)%Periodic%RotationAxis(1:3)
BC_Input(i)%Periodic%RotationAxis(4) = - BC_Input(j)%Periodic%RotationAxis(4)
```

or

```
BC_Input(i)%Periodic%RotationAxis(1:3) = - BC_Input(j)%Periodic%RotationAxis(1:3)
BC_Input(i)%Periodic%RotationAxis(4) = BC_Input(j)%Periodic%RotationAxis(4)
```

For example, say you have a rotor fan with 15 blades with the rotation axis aligned with the Cartesian *x*-axis, and you would like to run a simulation of just a single blade using rotational periodicity. Using `RotationAngle`, your inputs for boundary condition *i* would look like

```
BC_Input(i)%Periodic%RotationAngle(1:3) = [ 24.0, 0.0, 0.0]
```

and the inputs for boundary condition *j* would look like

```
BC_Input(j)%Periodic%RotationAngle(1:3) = [-24.0, 0.0, 0.0]
```

If you wanted to use `RotationAxis` instead, your inputs for boundary condition *i* would look like

```
BC_Input(i)%Periodic%RotationAxis(1:4) = [ 1.0, 0.0, 0.0, 24.0]
```

and the inputs for boundary condition *j* would look like

```
BC_Input(j)%Periodic%RotationAxis(1:4) = [ 1.0, 0.0, 0.0, -24.0]
```

7.6.2.3 Past Examples of Periodic Boundary Conditions

These are examples taken from GFR input files that have been used in the past to run simulations for problems including periodic boundaries. Note that the array indices used for `BC_Input` are simply copied over from the input files, meaning each index is relative to the other boundary

conditions that also existed in the input file it was taken from. Part of the intent behind keeping these array indices as they originally were in their respective input file is to demonstrate that the previously used variable indices of i and j can represent any array element of BC_Input provided that they link to each other when they form a pair of periodic boundaries.

- Streamwise and spanwise periodic channel flow

```
&nml_bc
  BC_Input(1)%Name = "INFLOW"
  BC_Input(1)%Set_BC_Value = 81
  BC_Input(1)%Periodic%Matching_BC_Name = "OUTFLOW"
  BC_Input(1)%Periodic%Matching_BC_Idx = 2
  BC_Input(1)%Periodic%Translation(1) = 0.036576

  BC_Input(2)%Name = "OUTFLOW"
  BC_Input(2)%Set_BC_Value = 81
  BC_Input(2)%Periodic%Matching_BC_Name = "INFLOW"
  BC_Input(2)%Periodic%Matching_BC_Idx = 1
  BC_Input(2)%Periodic%Translation(1) = -0.036576

  BC_Input(5)%Name = "ZMIN"
  BC_Input(5)%Set_BC_Value = 81
  BC_Input(5)%Periodic%Matching_BC_Name = "ZMAX"
  BC_Input(5)%Periodic%Matching_BC_Idx = 6
  BC_Input(5)%Periodic%Translation(3) = 0.018288

  BC_Input(6)%Name = "ZMAX"
  BC_Input(6)%Set_BC_Value = 81
  BC_Input(6)%Periodic%Matching_BC_Name = "ZMIN"
  BC_Input(6)%Periodic%Matching_BC_Idx = 5
  BC_Input(6)%Periodic%Translation(3) = -0.018288
/
```

- Single rotor blade from spacecraft cabin ventilation fan²⁷

- Rotor consists of nine blades spinning about the x -axis resulting in a rotationally periodic domain that is rotated 40° about the x -axis.
- This grid has two pairs of rotationally periodic boundaries: one for the primary computational domain, and a second for a sponge region appended onto the regular outflow boundary to damp out non-physical pressure waves from reflecting off the outflow boundary. The periodic boundary is split in this way for convenience, but could be combined.

```
&nml_bc
  BC_Input(8)%Name = "ThetaMin"
  BC_Input(8)%Set_BC_Value = 81
  BC_Input(8)%Periodic%Matching_BC_Name = "ThetaMax"
  BC_Input(8)%Periodic%Matching_BC_Idx = 9
  BC_Input(8)%Periodic%RotationAngle(1) = 40.0

  BC_Input(9)%Name = "ThetaMax"
  BC_Input(9)%Set_BC_Value = 81
  BC_Input(9)%Periodic%Matching_BC_Name = "ThetaMin"
  BC_Input(9)%Periodic%Matching_BC_Idx = 8
  BC_Input(9)%Periodic%RotationAngle(1) = -40.0

  BC_Input(10)%Name = "ThetaMinSponge"
  BC_Input(10)%Set_BC_Value = 81
  BC_Input(10)%Periodic%Matching_BC_Name = "ThetaMaxSponge"
  BC_Input(10)%Periodic%Matching_BC_Idx = 11
  BC_Input(10)%Periodic%RotationAngle(1) = 40.0

  BC_Input(11)%Name = "ThetaMaxSponge"
  BC_Input(11)%Set_BC_Value = 81
  BC_Input(11)%Periodic%Matching_BC_Name = "ThetaMinSponge"
  BC_Input(11)%Periodic%Matching_BC_Idx = 10
  BC_Input(11)%Periodic%RotationAngle(1) = -40.0
/
```

- Ceramic Matrix Composite 9% thickness (CMC9) blade cascade²⁸ with spanwise and vertical periodicity²⁹

```
&nml_bc
  BC_Input(3)%Name = "periodic_yp13"
  BC_Input(3)%Set_BC_Value = 81
  BC_Input(3)%Periodic%Matching_BC_Name = "periodic_ym"
  BC_Input(3)%Periodic%Matching_BC_Idx = 4
  BC_Input(3)%Periodic%Translation(2) = -0.13

  BC_Input(4)%Name = "periodic_ym"
  BC_Input(4)%Set_BC_Value = 81
  BC_Input(4)%Periodic%Matching_BC_Name = "periodic_yp13"
  BC_Input(4)%Periodic%Matching_BC_Idx = 3
  BC_Input(4)%Periodic%Translation(2) = 0.13

  BC_Input(5)%Name = "periodic_zp013"
  BC_Input(5)%Set_BC_Value = 81
  BC_Input(5)%Periodic%Matching_BC_Name = "periodic_z0"
  BC_Input(5)%Periodic%Matching_BC_Idx = 6
  BC_Input(5)%Periodic%Translation(3) = -0.052

  BC_Input(6)%Name = "periodic_z0"
  BC_Input(6)%Set_BC_Value = 81
  BC_Input(6)%Periodic%Matching_BC_Name = "periodic_zp013"
  BC_Input(6)%Periodic%Matching_BC_Idx = 5
  BC_Input(6)%Periodic%Translation(3) = 0.052
/
```

- Taylor-Couette flow³⁰ with streamwise periodicity

```
&nml_bc
  BC_Input(3)%Name = "Inflow"
  BC_Input(3)%Set_BC_Value = 81
  BC_Input(3)%Periodic%Matching_BC_Name = "Outflow"
  BC_Input(3)%Periodic%Matching_BC_Idx = 4
  BC_Input(3)%Periodic%Translation(1) = 6.283185307

  BC_Input(4)%Name = "Outflow"
  BC_Input(4)%Set_BC_Value = 81
  BC_Input(4)%Periodic%Matching_BC_Name = "Inflow"
  BC_Input(4)%Periodic%Matching_BC_Idx = 3
  BC_Input(4)%Periodic%Translation(1) = -6.283185307
/
```

8 Time Stepping

Acronyms used in this section:

RK : Runge-Kutta
ERK : Explicit Runge-Kutta
SSP : Strong stability preserving ERK
VOERK : Vortex-optimized ERK
AOERK : Advection-optimized ERK
FSAL : First-same-as-last
EPERK : Embedded-pair ERK with adaptive time stepping
PID : Proportional-integral-derivative

8.1 Time Stepping Methods

GFR exclusively employs explicit Runge-Kutta (ERK) time-stepping methods for advancing simulations in time. There are currently 28 different ERK methods available in GFR; 6 of these methods are recommended due to significantly greater performance compared to the other available methods. These 28 methods can be separated into two different types of ERK methods: traditional ERK methods that use the Courant-Friedrichs-Lewy (CFL) number to determine the optimal time step size that maintains stability, and embedded-pair ERK (EPERK) methods that use a Proportional-Integral-Derivative (PID) controller to automate the selection of an optimal time step size. Further details on the ERK methods in GFR can be found in Ref 27.

Each of the 28 ERK methods available in GFR are listed within one of the following three sections where the method is categorized as “Recommended”, “Other”, and “*NOT* Recommended”. The CFL-based and EPERK methods are listed separately within each category for easier identification. Each method listed includes: the **Runge_Kutta_Scheme** input value to activate it; the number of Runge-Kutta stages, order, and description of the scheme; and the speedup relative to the 3-stage/3rd-order SSP ERK method (SSP3) which is used as a baseline for performance. It should be noted that the speedup referenced here is in terms of time-to-solution, or the physical/real-world time taken to complete just the solver part of a simulation. Additionally, care was taken to ensure that simulations used to evaluate speedup were identical aside from the ERK method used to advance the simulations in time, e.g., all simulations were run on the same system with the same parallel settings with the only difference in the input file being the specified ERK method. Speedup data is not available for every method.

8.1.1 Highly Recommended Runge-Kutta Methods

The methods in table 5 are highly recommended due to significantly greater performance compared to the baseline SSP3 method. Although these methods require the evaluation of $1.7 \times - 5 \times$ as many right-hand-side evaluations (i.e., number of stages), the maximum stable CFL allowed by each of these methods compared to that of the baseline SSP3 method more than offsets this additional computational cost resulting in significant reductions in time-to-solution.

Table 5: Recommended Runge-Kutta Methods.

Runge_Kutta_Scheme	Stages / Order	Description	Speedup*
CFL-Based Methods			
113	11-stage / 3 rd -order	VOERK ⁹	1.33x
154	15-stage / 4 th -order	VOERK ⁹	1.33x
4	5-stage / 4 th -order	SSP ERK ³¹	1.22x
165	16-stage / 5 th -order	VOERK ⁹	1.18x
Adaptive Time Stepping Methods			
943	9-stage / 4 th (3 rd)-order	EPERK FSAL ¹⁰	1.22x
532	5-stage / 3 rd (2 nd)-order	EPERK FSAL ¹⁰	1.09x

* Relative to Runge_Kutta_Scheme=3, the 3-stage/3rd-order SSP ERK method.

8.1.2 Other Runge-Kutta Methods

The methods in table 6 are categorized as “other” because they are neither high- nor low-performing with similar performance to the baseline SSP3 method. This section contains the most popular ERK methods used in CFD, particularly the Classic RK family of methods, SSP2, the 5-stage/4th-order Carpenter-Kennedy low-storage RK method, and, probably the most popular of all ERK methods especially with FR/DG, SSP3.

Table 6: Other Runge-Kutta Methods.

Runge_Kutta_Scheme	Stages / Order	Description	Speedup*
CFL-Based Methods			
-2	2-stage / 2 nd -order	Classic RK	n/a
-3	3-stage / 3 rd -order	Classic RK	n/a
-4	4-stage / 4 th -order	Classic RK	n/a
2	2-stage / 2 nd -order	SSP ERK ³²	n/a
3	3-stage / 3 rd -order	SSP ERK ³²	1.00x
5	5-stage / 4 th -order	Carpenter-Kennedy low storage RK ³³	0.99x
53	5-stage / 3 rd -order	VOERK ⁹	1.08x
64	6-stage / 4 th -order	VOERK ⁹	1.02x
85	8-stage / 5 th -order	VOERK ⁹	1.05x
Adaptive Time Stepping Methods			
-943	9-stage / 4 th (3 rd)-order	EPERK non-FSAL ¹⁰	n/a
-532	5-stage / 3 rd (2 nd)-order	EPERK non-FSAL ¹⁰	n/a

* Relative to Runge_Kutta_Scheme=3, the 3-stage/3rd-order SSP ERK method.

8.1.3 *NOT* Recommended Runge-Kutta Methods

The methods in table 7 are not recommended mostly due to significantly worse performance compared to the baseline SSP3 method. The one exception is the 1-stage/1st-order Classic RK method (i.e., Forward Euler method), which is not recommended due to its low accuracy and small CFL limit. Although some of the AOERK methods achieve similar time-to-solution performance compared to SSP3, they have all been listed here as not recommended because every one of these methods is worse than their respective VOERK method counterpart.

Table 7: Not Recommended Runge-Kutta Methods.

Runge_Kutta_Scheme	Stages / Order	Description	Speedup*
CFL-Based Methods			
± 1	1-stage / 1 st -order	Forward Euler method	n/a
−53	5-stage / 3 rd -order	AOERK ⁹	n/a
−113	11-stage / 3 rd -order	AOERK ⁹	n/a
−64	6-stage / 4 th -order	AOERK ⁹	n/a
−154	15-stage / 4 th -order	AOERK ⁹	n/a
−85	8-stage / 5 th -order	AOERK ⁹	n/a
−165	16-stage / 5 th -order	AOERK ⁹	0.98x
Adaptive Time Stepping Methods			
54	6-stage / 5 th (4 th)-order	Dormand-Prince EPERK ³⁴	0.64x
853	12-stage / 8 th (5 th)-order	Dormand-Prince EPERK ³⁵	0.61x
1054	10-stage / 5 th (4 th)-order	EPERK FSAL ¹⁰	0.85x
−1054	10-stage / 5 th (4 th)-order	EPERK non-FSAL ¹⁰	n/a

* Relative to Runge_Kutta_Scheme=3, the 3-stage/3rd-order SSP ERK method.

8.2 Time Stepping Algorithms

As established in Section 8.1, GFR exclusively employs explicit Runge-Kutta (ERK) time-stepping methods for advancing simulations in time. In this section, we focus on the algorithmic details of the ERK methods and other aspects of time stepping in GFR. There are essentially four different time stepping options available in GFR: constant global time step, global CFL-based time step, local CFL-based time step, and adaptive global time step.

8.2.1 CFL-based Time Stepping

The majority of the Runge-Kutta methods in GFR use a time step that is loosely based on the CFL condition. With $CFL_{\max}$ being the maximum stable CFL value for a given method, the maximum time step size, Δt , allowed for the method is

$$\Delta t = \frac{h}{|\lambda|_{\max}} CFL_{\max}$$

where h is a characteristic length scale for the cell and $|\lambda|_{\max}$ is the magnitude of the maximum eigenvalue for the governing equations, also sometimes referred to as the spectral radius. For traditional ERK methods, $CFL_{\max}$ is somewhere in the range of 0.2–2.0.

Algorithm 1 Low storage implementation (3S*)

```
1:  $S_1 \leftarrow u^n; S_2 \leftarrow 0; S_3 \leftarrow u^n$ 
2: for  $i = 1 : s$  do
3:    $\tau \leftarrow t^n + c_i \Delta t$ 
4:    $S_2 \leftarrow S_2 + \delta_i S_1$ 
5:    $S_1 \leftarrow \gamma_{1,i} S_1 + \gamma_{2,i} S_2 + \gamma_{3,i} S_3 + \beta_i \Delta t \mathbf{R}(\tau, S_1)$ 
6: end for
7:  $u^{n+1} \leftarrow S_1$ 
```

Algorithm 2 Embedded-pair low storage implementation (3S*+)

```
1:  $S_1 \leftarrow u^n; S_2 \leftarrow 0; S_3 \leftarrow u^n; S_4 \leftarrow u^n$ 
2: for  $i = 1 : s$  do
3:    $\tau \leftarrow t^n + c_i \Delta t$ 
4:    $S_2 \leftarrow S_2 + \delta_i S_1$ 
5:    $R_i \leftarrow \mathbf{R}(\tau, S_1)$ 
6:    $S_1 \leftarrow \gamma_{1,i} S_1 + \gamma_{2,i} S_2 + \gamma_{3,i} S_3 + \beta_i \Delta t R_i$ 
7:    $S_4 \leftarrow S_4 + \hat{b}_i \Delta t R_i$ 
8: end for
9:  $u^{n+1} \leftarrow S_1$ 
10:  $\hat{u}^{n+1} \leftarrow S_4$ 
11: if  $\hat{b}_{s+1} \neq 0$  then
12:    $\hat{u}^{n+1} \leftarrow \hat{u}^{n+1} + \hat{b}_{s+1} \Delta t \mathbf{R}(t^{n+1}, u^{n+1})$ 
13: end if
```

GFR uses an approximation to compute the ratio $h/|\lambda|_{\max}$, which means $\text{CFL}_{\max}$ — which is a property of a given ERK method that can be analytically determined — no longer applies. The **CFL** used by GFR is analogous to the true $\text{CFL}_{\max}$ in that it scales the GFR approximation of $h/|\lambda|_{\max}$ to determine Δt . The GFR **CFL** is generally on the same order as $\text{CFL}_{\max}$, with stable values ranging between 0.5–10.0 depending on the ERK method being used.

Most ERK methods implemented in GFR have been implemented using a low-storage algorithm that requires three registers per stage (3S*).³⁶ The low-storage 3S* algorithm for s stages is shown in algorithm 1. A register is essentially an array that consumes the same amount of memory as the solution array, i.e., all the conserved variables at all the solution points. The three registers required are: 1) the actual solution array for the current Runge-Kutta stage, S_1 ; 2) the solution array from the previous iteration, S_3 , and 3) an auxiliary array, S_2 . The function $\mathbf{R}(t^n, u^n)$ represents the evaluation of the right-hand-side residual of the system of equations, where t^n and u^n are respectively the time and solution at iteration n , and Δt is the change in time for the current iteration. The low-storage 3S* coefficients $[\gamma_{1,i}, \gamma_{2,i}, \gamma_{3,i}, \delta_i, \beta_i, c_i]$ can be found in the references for the ERK schemes.

There are two different methods in GFR for computing an approximation to $|\lambda|_{\max}$. When **|Timestep_Type|=1**, $|\lambda|_{\max}$ is computed in each cell by taking the maximum value of the spectral radius evaluated at each face point of the cell. This is the recommended method to use. When **|Timestep_Type|=2**, $|\lambda|_{\max}$ is computed in each cell by taking the maximum value of the inviscid eigenvalue evaluated at each solution point of the cell. This second method is experimental and not recommended.

8.2.1.1 Global CFL-based Time Stepping

Global time stepping is required for time-accurate simulations like LES. When using CFL-based time stepping, a global time step is used when **Timestep_Type** > 0.

At the beginning of each solver iteration, each grid cell computes its own unique time step size, and the minimum time step size across all grid cells is taken to be the global time step size that all cells use to update their solutions in time. Using the same time step size for all cells ensures that the solution across the entire computational domain is synchronized in time, and taking the minimum time step size ensures stability for all cells. This of course leads to the inefficiency in that

the cell with the smallest time step size “slows” all the other cells down. This makes global time stepping inefficient for steady-state problems.

8.2.1.2 Local CFL-based Time Stepping

A local time step is used when `Timestep_Type` < 0. For CFL-based local time stepping, each cell uses its own time step size to advance the simulation in time. At the beginning of each solver iteration, each grid cell computes its own unique time step size that is used to update the solution within the cell at the end of each Runge-Kutta stage. Small cells with strong gradients tend to have the smallest local time step. Since all cells are updated with different time step sizes, this can only be used for steady-state simulations and cannot be used for time-accurate simulations (e.g., LES) which require all cells to be synchronized in time.

8.2.2 Constant Global Time Stepping

Constant global time stepping is exactly as it sounds: a constant time step size, taken from the input `Timestep`, is globally used by all cells to update their solutions in time. This method is used by setting the input `Timestep_Type`=0. In general, the time step size is constant in that the value of `Timestep` is used for every solver iteration with a CFL of 1.0. However, if CFL ramping is being used, the time step size will change based on the ramping schedule. Once the ramping schedule finishes, CFL will revert to a value of 1.0.

Being a global time stepping method, this option is suitable for time-accurate simulations like LES. The difficulty in using a constant global time step is that the maximum allowable time step for a simulation is problem dependent, so it can only really be used when you have an idea of what time step size is needed for a problem. The primary use for constant time stepping would be for running a followup simulation using the restart file of a simulation that used a CFL-based global time step in which all initial transients have been removed from the flow. However, continuing the use of CFL-based global time stepping for this followup situation is just as good as, if not better than, switching to a constant time step size, since the CFL-based approach can reduce the time step size if something develops within the simulation that requires a smaller time step.

8.2.3 Adaptive Time Stepping

A second set of ERK methods exist within GFR which belong to the family of embedded-pair explicit Runge-Kutta (EPERK) methods. EPERK methods contain an additional Runge-Kutta method of lower order than the primary method that is simultaneously evaluated with minimal additional cost, thus an embedded pair of methods. At the end of a given time step, the difference between these two methods can be viewed as the error in the primary time-stepping method. This error can be used as the input to a PID controller to automate the selection of an optimal time step size for the following iteration.^{10,37,38}

The implementation and usage of the optimized EPERK schemes follows the work of Ref 10. The EPERK methods use the low-storage $3S^*_+$ algorithm shown in algorithm 2 that requires the same three registers as the ERK $3S^*$ low-storage algorithm plus a fourth register, S_4 , for the error estimate. The quantity $\hat{u}^{n+1}$ represents the solution as updated by the embedded lower-order scheme using the additional coefficients $\hat{b}_i$. The time stepping error from the embedded RK pair is given by

$$\varepsilon_{n+1} = \frac{1}{w_{n+1}}, \quad w_{n+1} = \frac{1}{m} \sum_{i=1}^m \left(\frac{u_i^{n+1} - \hat{u}_i^{n+1}}{\text{atol} + \text{rtol} \cdot \max \{ |u_i^{n+1}|, |\hat{u}_i^{n+1}| \}} \right)$$

where m is the total degrees-of-freedom, and `atol` and `rtol` are the respective absolute and relative error tolerances. Finally, the time stepping error is used with a PID controller to determine the time step size for the next iteration

$$\Delta t^{n+1} = \varepsilon_{n+1}^{\beta_1/k} \varepsilon_n^{\beta_2/k} \varepsilon_{n-1}^{\beta_3/k} \Delta t^n$$

where k is the order of the EPERK method, and β_i are the controller parameters. The error tolerances and controller parameters used in this work are those recommended in Ref 10.

NOTE: Adaptive time stepping cannot be used with limiting/filtering. The limiting/filtering step, which modifies the solution variables, occurs immediately after the completion of a Runge-Kutta stage. This means the right-hand-side that was used by the EPERK method to update the solution variables is no longer consistent with the limited/filtered solution variables. This inconsistency also applies to the embedded ERK method which ultimately causes the adaptive time stepping algorithm to fail when using limiting/filtering. Being a global time stepping method and its incompatibility with limiting/filtering, this option is suitable for time-accurate simulations of low-speed flows. Further details about EPERK methods and adaptive time stepping can be found in Ref 27.

As with the CFL-based global time stepping, a unique time step size is computed for each cell, and the minimum time step size across all grid cells is taken to be the global time step size that all cells use to update their solutions in time. When it works as intended, this method is the easiest to use because the adaptive algorithm produces a near-optimal time step size with zero input from the user. However, testing has found that it tends to not work more than it works as intended.

8.3 Controlling the Time Step Size in GFR

8.3.1 Ramping

After solution initialization, the flow may have very strong gradients as the boundary conditions are applied. This may necessitate a small timestep to keep the solution stable during these initial transients. However, as the flow progresses, a larger timestep may be permissible.

GFR has a ramping capability that applies a multiplicative factor to either the requested constant CFL or constant timestep. This factor linearly varies between specified beginning and ending values over a certain period of iterations.

See Also: `CFL_BEG`, `CFL_END`, `CFL_CYCLES`

8.3.2 Factors that Indirectly Affect Maximum Stable Time Step Size

The following are some considerations to be mindful of with how they affect the maximum stable time step.

8.3.2.1 Effect of Limiting/Filtering on Time Step Size

Limiting/filtering will generally increase the maximum stable time step size, since both methods essentially damp the higher order polynomial modes. This decreases the solution accuracy, but permits the solution to advance further in time without those higher order modes amplifying into a numerical instability. Note, however, that a larger time step size will likely cause some additional time-step induced instabilities, which may cause more cells to be marked and limited/filtered to maintain stability. Selecting the optimal combination of limiter/filter and time step settings can be a tedious process.

Table 8: Maximum stable CFL and resulting physical time step size, Δt , for simulations of the spacecraft cabin ventilation fan relative to the solution order, $\mathcal{P}$.

$\mathcal{P}$	CFL	Δt [s]
1	1.5	9.2×10^{-9}
2	3.5	6.0×10^{-9}
3	5.5	4.3×10^{-9}
4	8.0	3.1×10^{-9}

8.3.2.2 Effect of Solution Order on Time Step Size

For high-order flux reconstruction, discontinuous Galerkin, and spectral-element methods, it has been shown that the maximum stable time step scales with the solution polynomial order as $1/\mathcal{P}^2$.^{6,39} Thus, for higher-order solutions, expect substantially smaller time step sizes. Further details including a rigorous mathematical discussion can be found in Section 4.7 of Ref 39.

8.3.2.3 Effect of Solution Order on CFL

This is most likely a GFR-specific behavior that may be related to how the CFL is computed, but the solution order affects the maximum stable CFL inversely to how it affects the maximum stable time step size. That is for higher-order solutions, larger CFL values may be possible. As an example, take the spacecraft cabin ventilation fan shown in Fig. 18. Table 8 shows the maximum CFL used, and resulting physical time step size, for a single configuration of the fan run at $\mathcal{P}1$ – $\mathcal{P}4$, i.e., the same grid and flow conditions were used and the only thing different between the simulations was the solution order. All of these simulations used the 11-stage/3rd-order VOERK scheme (i.e., `Runge_Kutta_Scheme=113`). This example clearly shows how the maximum CFL that keeps GFR stable increases with $\mathcal{P}$, but the corresponding time step size still decreases with increasing $\mathcal{P}$ due to the $1/\mathcal{P}^2$ scaling from Section 8.3.2.2.

8.3.2.4 Effect of Transients on Time Step Size

When starting a new simulation or restarting a simulation with significant changes to the settings (e.g., changing the solution order or adjusting the pressure for a boundary condition), transients are created in the flow field that generally contain larger and sharper gradients than would normally exist in the well-established, stable flow field. These disturbances may require the use of a smaller time step size to prevent them from growing into numerical instabilities. Once the disturbances have subsided or convected out of the domain, it may be possible to increase the time step size.

8.3.3 Finding a Near-Optimal Stable CFL

This section details one method for trying to find a maximum stable CFL/time step size for a given simulation.

After creating the grid, the first step of running a simulation is creating the input file and running some initial tests to verify there are no issues with the grid or input file. Perform tests with a smaller CFL value that is almost certain to be stable, say 0.1-0.5, and have them run for 100 to 10,000 time steps.

Once it seems that there are no major issues, this is when to begin the second step of trying to run the simulation for a large number of time steps. During this phase, initial transients will dominate the solution and this is the time to play with the filter/limiting settings in order to find

a combination that keeps the simulation stable without damping out too much of the smallest resolved scales.

After the initial transients in the simulation are mostly eliminated and the solution appears to be stable, the next step is to push the CFL in order to find the maximum stable time step. If the simulation is running, the interrupt file `ADJUST_CFL` (Section 10.10.2) can be used to modify the CFL input variables; otherwise, the CFL inputs can simply be adjusted in the input file for the next simulation. Utilize the CFL ramping inputs to slowly increase the CFL with the actual goal of causing the simulation to diverge and crash. If the previous simulation was using a CFL of 0.5, one could set up the next simulation with the inputs

```
CFL_Ramping_Beg = 0.5
CFL_Ramping_End = 15.0
CFL_Ramping_Iterations = 14500
```

The final CFL value of 15.0 is chosen because it is assumed that this would almost certainly be too large for the simulation to remain stable. This new simulation is run and say it crashes once it reaches a CFL of 10.2. One should generally reduce the final CFL number by 10–15%, and use this value as the new CFL going forward. If you want to push the CFL to the highest possible value, you can also search through the output file `residual.dat` in order to find the approximate iteration at which the simulation residual begins to diverge. However, it is generally less of a headache to trim a few extra percent off just to make sure the simulation remains stable.

In this example, the number of CFL ramping iterations is chosen to be 14,500 so that each integer increment of the CFL number takes 1,000 iterations. This offers a moderate level of resolution in terms of what CFL value will maintain stability. For example, if the CFL ramping is over 1,000 iterations, narrowing down the optimum CFL value would be less accurate.

It is also important to remember that the maximum stable time step is something that can change over the course of a simulation since it is affected by many different aspects. If a simulation appears to be stable at a given CFL value and then suddenly crashes, try restarting from a recent restart file (but not too recent in case the instability already exists in the restart file) with a smaller CFL (say 50% smaller). If the simulation crashes at nearly the same simulation time, you might want to try an even smaller time step with an even earlier restart file, but most likely the instability is due to something that will need to be treated with numerical methods like de-aliasing or stronger filtering/limiting.

With enough practice and experience (and patience), various aspects of the steps provided in this section can get blended together to speed up this process even more. Unfortunately, a good understanding and feel for how the time step size responds to the factors listed in Section 8.3.2 is really needed to blend these steps effectively.

9 Numerics

Nearly all numerical methods with greater than first-order accuracy will exhibit some sort of numerical instability when used to solve nonlinear differential equations such as the compressible Navier-Stokes equations. This is especially true for high-order methods (i.e., greater than second-order) which can generate errors when any of the modes arising from the nonlinear terms are not adequately represented by the numerical scheme. Within a given element, this representation is defined by the stencil size for finite-difference methods or the set of basis functions (i.e., polynomial order or modes) for DG or FR methods.

In these cases, the energy from the unresolved modes are aliased onto the lower modes producing aliasing errors. The effect these errors can have on a given simulation is highly unpredictable, and many researchers have shown aliasing errors can produce severe instabilities within a problem.^{39–47} Preventing aliasing and other numerical instabilities is usually accomplished through the addition of some type of numerical dissipation. In the following sections, we discuss several important topics related to numerical stability:

- Aliasing Error
- Filtering
- Troubled Cell Markers

9.1 Aliasing Error

Aliasing error occurs when the polynomial basis functions are insufficient to represent the nonlinear terms. Consider the simplified example shown in Fig. 14, in which the exact solution u is given by a fourth order polynomial and the exact flux u^2 is given by an eighth order polynomial. In this case, a fourth order polynomial for the numerical solution can be used to exactly represent the physical solution. Evaluating the flux at the solution points will also result in a fourth order polynomial that matches the exact flux at those points. However, at any other location, the fourth order polynomial flux is unable to accurately approximate the eighth order exact flux. The area between the exact flux and polynomial flux curves represents the aliasing error.

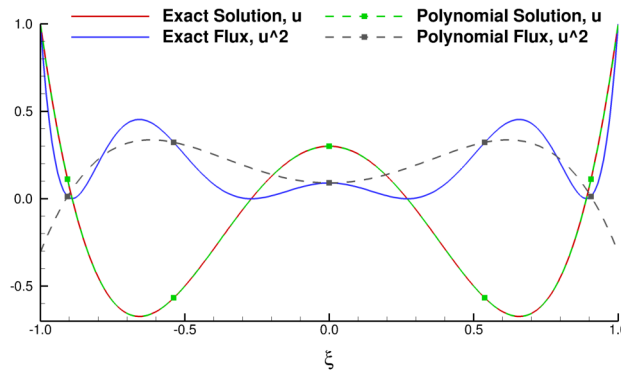


Figure 14: Example of aliasing error.

Ref 12 explored one means of reducing aliasing error using an early version of GFR. Instead of directly computing the flux using the same polynomial order and Gauss points as the solution,

the flux is evaluated using a higher quadrature order (i.e., a higher order polynomial and its corresponding Gauss points). This technique of over-integration by over-sampling at the quadrature points requires that the solution polynomial for the conserved variables be evaluated (i.e., interpolated) at the additional set of quadrature points in order to use those values in computing the flux. Then that higher order quadrature polynomial is projected (in a least squares sense) onto a flux polynomial of the same order as the solution. This reduces the *overall error* between the exact flux and polynomial flux curves, but in doing so the *specific error* between the exact flux and polynomial flux curves evaluated at the solution Gauss points will now be non-zero.

The main disadvantage to reducing aliasing error in this manner is the additional cost involved in supersampling at the additional quadrature points and then projecting that polynomial down to the solution order. For that reason, subsequent development of GFR (i.e., the viscous terms) has not incorporated the quadrature evaluation.

9.2 Filtering

In spectral-element methods, the solution within an individual element can be transformed into a hierarchical set of modal coefficients which are representative of the energy contained in each mode. A well-resolved numerical solution will have minimal energy contained within the higher modes, and the modal coefficients should have an exponential decay rate from lowest to highest modes. For this reason, an exponential filter is used as a source of numerical dissipation for stabilizing the simulations.

This diagonal exponential filter matrix is given by³⁹

$$\sigma(n) = \begin{cases} 1 & 0 \leq \eta \leq \eta_c \\ \exp \left[-\alpha \left(\frac{\eta - \eta_c}{1 - \eta_c} \right)^s \right] & \eta_c \leq \eta \leq 1 \end{cases} \quad \begin{array}{l} \eta = \text{modal fraction, } (n - 1)/\mathcal{P} \\ \eta_c = \text{modal fraction cutoff} \\ \alpha = \text{filter strength} \\ s = \text{filter order} \end{array} \quad (69)$$

where $n = 1, \dots, \mathcal{P} + 1$.

Three user-specified parameters, $[\eta_c, \alpha, s]$, control the aggressiveness of the filter, and their effects are illustrated in Fig. 15.

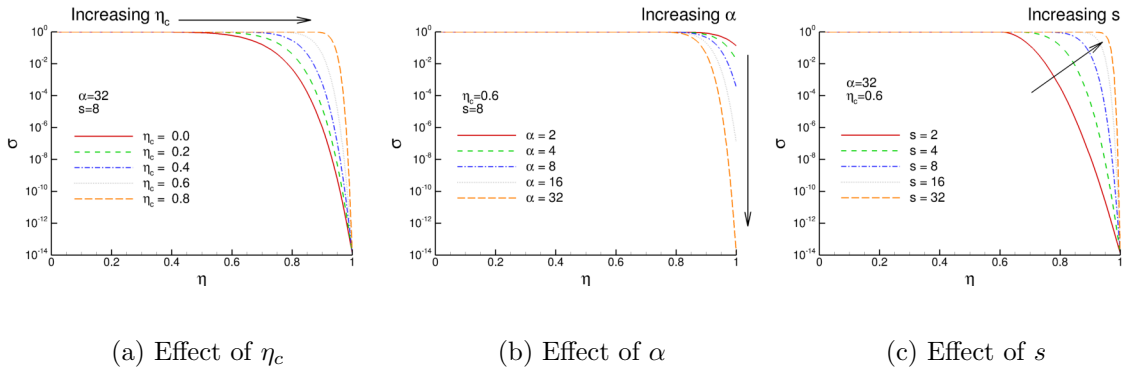


Figure 15: Exponential filter function, σ .

- η_c := This parameter specifies the modal fraction cutoff above which filtering will be applied. Everything below that modal fraction cutoff will be entirely preserved. Most importantly, the lowest mode (which defines the average value in the cell) remains unchanged in order to ensure conservation of mass, momentum, and energy. Valid input values for this parameter are real values on the interval $[0.0, 1.0]$, although a value of 1.0 is the same as using no filter at all because it will preserve all modes of the solution. A value of 0.0 will apply some amount of filtering to all modes except the lowest/constant mode.
- α := This parameter specifies the strength of the filter, i.e., the highest mode will be multiplied by the value $\exp(-\alpha)$. As α increases, the filter value at the highest mode(s) becomes smaller, which means that more damping is applied. Valid input values for this parameter are real values between 0.0 and roughly 36.0, where a value of 36.0 is nearly equivalent to multiplying the highest mode by 64-bit machine epsilon, i.e., $\ln(2^{-52}) \approx -36.0$.
- s := This parameter specifies the filter order which controls the exponential decay rate (i.e., how parabolic the filter function is on a log plot). As s increases, filtering has substantially less of an effect on the lower modes. Valid input values for this parameter are positive integers greater than or equal to 2; however, anything greater than 32 will essentially result in a filter that multiplies the highest mode by $\exp(-\alpha)$ applies negligible filtering to any lower modes that are larger than the cutoff mode. In practice, commonly used values are $[2, 4, 8, 12, 16, 32]$.

GFR uses a three-pass filtering approach. First, a *sponge filter* can be applied to the downstream region starting from a user-specified x-location. This filter region is often used to damp pressure fluctuations so they do not reflect from the outflow boundary and contaminate the upstream solution. Second, a *local filter* can be applied to those cells identified by the **troubled cell marker** in Section 9.3. Usually these cells require some significant level of filtering due to a shock or some local instability. Third, a weak *global filter* can be applied to all cells in the domain to remove any high-frequency noise in the solution; NOTE, use of the global filter should be avoided if possible since it filters *everything* including cells that do not need filtering.

It is recommended to check for troubled cells and perform filtering operations after each Runge-Kutta subiteration. This is the highest filtering frequency possible, with other researchers opting to filter only after the final Runge-Kutta stage or after a certain number of time steps. Filtering at such a high frequency allows for the use of a larger time step, but it can reduce the accuracy of the solution. This decrease in accuracy is usually outweighed by the increased time step size, which is often the most limiting computational factor in time-to-solution performance.

See Also:

<code>Global_Filter_Interval,</code>	<code>Local_Limiter_Interval,</code>	<code>Sponge_Filter_Interval,</code>
<code>Global_Filter_Option,</code>	<code>Local_Filter_Order,</code>	<code>Sponge_Filter_Option,</code>
<code>Global_Filter_Order,</code>	<code>Local_Filter_Alpha,</code>	<code>Sponge_Filter_Order,</code>
<code>Global_Filter_Alpha,</code>	<code>Local_Filter_Etac,</code>	<code>Sponge_Filter_Alpha,</code>
<code>Global_Filter_Etac,</code>		<code>Sponge_Filter_Etac,</code>
		<code>Sponge_Filter_X_Start</code>

9.3 Troubled Cell Markers

Instabilities within a given simulation are usually highly-localized to very small regions of the grid domain. Frequently, just a single cell out of perhaps millions of cells will entirely contain the instability at a particular time step of a simulation. Indiscriminately applying numerical dissipation to all cells in a simulation for stabilization purposes will add unwanted dissipation in places where it is not needed, severely affecting the overall accuracy of the solution. Therefore, a method is needed to identify only those cells whose solutions show signs of becoming unstable, so they can be specifically targeted with additional numerical dissipation. A few different types of troubled cell markers are available for use and are described below.

9.3.1 Resolution Marker

As discussed in Section 9.2, numerical instabilities often appear in the highest polynomial mode(s). An easy way to distinguish between well-resolved and possibly unstable cells is to examine the highest mode of the solution within each cell.

Recall that the modal form of the solution polynomial for a given flow variable is given by Eq. (2)

$$u(r) = \sum_{n=1}^{N_p} \hat{u}_n(t) \psi_n(r) \quad \psi_n(r) = r^{n-1}$$

where $\hat{u}_n$ represents the modal coefficients, with $\hat{u}_1$ being the coefficient for the lowest/constant mode and $\hat{u}_{N_p}$ being the coefficient for the highest/degree- $\mathcal{P}$ mode. A diagonal filter matrix can be applied to remove the highest mode of the solution.

$$\underline{\hat{u}} = \sigma \hat{u} \quad \sigma_n = \begin{cases} 0 & n = N_p \\ 1 & \text{otherwise} \end{cases} \quad (70)$$

A comparison of this filtered solution to the actual solution will provide a test for the possibility of numerical instabilities in the cell. Namely, in a well-resolved solution, $\hat{u} \simeq \underline{\hat{u}}$. Any cells in which this comparison shows a difference that is beyond a certain threshold are marked as “troubled” cells.

To do this, we compute the local resolution marker as^{48–50}

$$\mathcal{M}(\hat{u}, \underline{\hat{u}}) = \log_{10} \left[\frac{(\hat{u} - \underline{\hat{u}}) \cdot (\hat{u} - \underline{\hat{u}})}{\hat{u} \cdot \hat{u}} \right] \quad (71)$$

where $\hat{u}$ is the user-specified solution variable to check (density is recommended). If all of the conserved variables are specified, then Eq. (71) is computed for each one and the maximum value is used. Note that the numerator represents the norm of the highest mode and the denominator represents the norm of all the modes combined. The marker will have values in the range $\mathcal{M} = [-\infty, 0]$ where the lower value indicates that nothing is contained in the highest mode and the upper value indicates that everything is contained in the highest mode.

Next, we compute the local scaled marker as described in Section 9.3.4.

See Also: `MarkerOption`, `MarkerVariable`

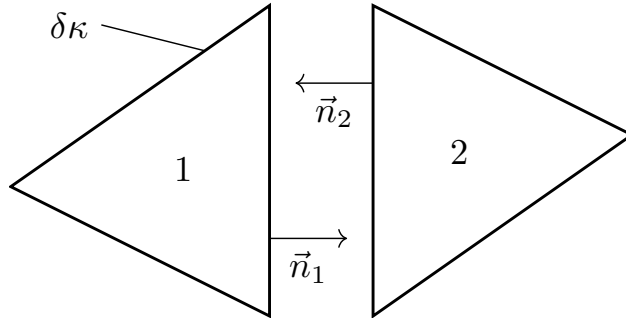


Figure 16: Jump Marker Notation

9.3.2 Jump Marker

For flows with shocks or other sudden discontinuities, it is useful to mark those cells that experience a sudden jump in pressure or density. The jump marker^{49–53} is a method for identifying such sudden changes across adjacent cells. Using the notation in Fig. 16, the jump indicator can be defined as:

$$J_{\kappa} = \frac{1}{\|\partial\kappa\|} \int_{\partial\kappa} \left| \frac{[[\psi]] \cdot \vec{n}}{\{\psi\}} \right| ds \quad \begin{aligned} [[\psi]] &= \psi_1 \vec{n}_1 + \psi_2 \vec{n}_2 = (\psi_1 - \psi_2) \vec{n}_1 \\ \{\psi\} &= (\psi_1 + \psi_2) / 2 \end{aligned} \quad (72)$$

$$= \frac{1}{\|\partial\kappa\|} \int_{\partial\kappa} \left| \frac{(\psi_1 - \psi_2)}{(\psi_1 + \psi_2) / 2} \right| ds = [0 : 2] \quad (73)$$

Here, $[[\psi]]$ represents the jump across adjacent face values and $\{\psi\}$ represents the average of those values. The jump indicator J_{κ} can vary between 0 (no jump) and 2 (infinite jump).

The jump marker is obtained by taking the logarithm of the value.

$$\mathcal{M}(J_{\kappa}) = \log_{10} [\max(J_{\kappa}, 2^{-23})] = [-6.9 : 0.3] \quad (74)$$

The jump marker $\mathcal{M}$ can vary between -6.9 (no jump) and 0.3 (infinite jump). The scaled marker is then computed as described in Section 9.3.4.

See Also: `Marker.Option`, `Marker.Variable`

9.3.3 MLP Marker

The MLP marker is normally computed as part of the Multidimensional Limiting Process.^{54–56} It first identifies troubled cells using the $\mathcal{P}1$ -projected MLP condition, which compares the cell vertex (corner) values to those of the adjacent cells. Then the MLP extrema detector is applied to identify local smooth extrema near the vertex points and unmark those cells.

When GFR outputs the MLP marker, the value $\mathcal{M}$ represents the polynomial order of the limited solution, with possible real values of $[0:1,2,3,...,\mathcal{P}]$. Values less than $\mathcal{P}$ indicate that the polynomial in the cell has been reduced to that value to maintain stability. Values between $[0:1]$ indicate that all of the higher modes have been removed and the linear mode was limited by that fraction in order to remain stable.

See Also: `Marker.Option`, `Marker.Variable`

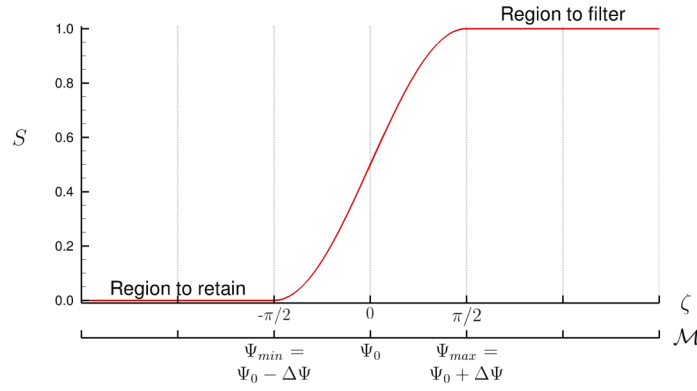


Figure 17: Scaled Marker

9.3.4 Scaling the Marker Results to Identify Troubled Cells

From the local marker, a local *scaled* marker can be computed

$$\mathcal{S} = \begin{cases} 0 & \zeta \leq -\pi/2 \\ (1 + \sin \zeta) / 2 & |\zeta| \leq \pi/2 \\ 1 & \zeta \geq \pi/2 \end{cases} \quad \zeta = \frac{\pi}{2} \frac{\mathcal{M} - \Psi_0}{\Delta\Psi} \quad (75)$$

where

$$\Psi_0 = -[C_1 + C_2 \log_{10}(\mathcal{P})] \quad \Delta\Psi = 0.5 \quad (76)$$

As shown in Fig. 17, the scaled marker varies between 0 (region to retain) and 1 (region to filter). Other works sometimes use this smooth variation to modify the strength of the dissipation applied. In GFR, this smoothly varying scaled marker can be output for visualization purposes. However, troubled cells are simply identified by $\mathcal{S} > 1.0 \times 10^{-8}$. This approach is more aggressive at identifying troubled cells.

The default values of the (C_1, C_2) coefficients are (4.0, 4.25) when using the resolution marker and (2.25, 3.0) when using the jump marker. GFR will use $(C_1, C_2, \Delta\Psi)$ to compute $\Psi_{min} = \Psi_0 - \Delta\Psi$ and $\Psi_{max} = \Psi_0 + \Delta\Psi$ unless the inputs for **(Marker_Min, Marker_Max)** = (Ψ_{min}, Ψ_{max}) are used instead.

When using the MLP marker, the scaled marker is simply computed as a binary flag to indicate a troubled cell, i.e., any cell in which the multidimensional limiting process would reduce the solution polynomial order.

$$\mathcal{S} = \begin{cases} 0 & \text{if } \mathcal{M} = \mathcal{P} \\ 1 & \text{otherwise} \end{cases} \quad (77)$$

See Also: **Marker_Dpsi**, **Marker_Min**, **Marker_Max**

10 Files

This section describes the various files that are read and written during a simulation and the input parameters associated with their use:

- **Namelist Input**
- **Grid Input**
- **Solution Input**
- **Restart Input/Output**
- **Display Output**
- **Residual Output**
- **Solution Output**
- **Time-Averaging Output**
- **Probes Output**
- **Simulation Interrupt/Control Files**

10.1 Namelist Input

GFR uses a Fortran namelist file for specifying user inputs. GFR is executed on the command line with the following syntax:

```
[MPI commands] gfr [optional args]
```

where [MPI commands] specifies the MPI executable and MPI arguments if running in parallel, **gfr** is the GFR executable, and [optional args] are the optional arguments that will be processed by GFR. The logic for detecting and verifying the GFR input file is as follows:

1. If the number of optional arguments is 1, it tests to see if the argument represents an existing file containing GFR namelists.
2. If the number of optional arguments is greater than 1, it searches through the arguments to find one equal to “-f”. If the input flag “-f” is found as one of the arguments, it tests to see if the very next argument following the flag represents an existing file containing GFR namelists.
3. If neither of the above finds an input file, GFR will check the current run directory for the file **gfr.in** and use that file if it exists.

If any of these steps finds what appears to be a valid GFR input file, it will proceed with reading that input file to run a simulation. If no input file is provided, the code will output the file **gfr.in.default** which contains all of the available input variables and their current default values.

The contents of the user input file are described in Section 11. The user may include every input variable in the input file. However, the only *required* entries are those inputs that the user wants to have non-default values.

Input variables may be added, removed, or renamed between versions. However, the code will attempt to convert an older input file to the current format.

10.2 Grid Input

For specific test cases with trivial geometry, the flow solver will internally generate a grid. In general, though, the user must provide the mesh to be used for the simulation. Grids that are two- or three-dimensional, structured or unstructured, can be read from CGNS, Plot3D, or Gmsh (very limited support) files. The GFR code will always assume the grid coordinates are in units

of meters. However, sometimes it can be easier or more useful to generate a grid in other units — we refer to these as *grid units* which could be in anything from millimeters, to inches, or even nondimensional — and the user-specified input `Grid_Scaling_Factor` provides a way to scale these grid units to meters as assumed by the solver. See Section 3.3 and `Grid_Scaling_Factor` for more information.

If any of the surface geometries contain curvature, it is *highly* recommended to use a high-order curved grid with at least $\mathcal{P}2$ grid cells to accurately represent the surfaces with curvature. This recommendation is simply so that the grid represents the curved boundary surfaces with at least some amount of curvature and not linear segments. Increasing the grid order to $\mathcal{P}3+$ can help improve the simulation accuracy, but how much of an improvement depends on the flow conditions, the amount of curvature in individual cells (think coarse cells with high curvature), etc. One example scenario that can show improved accuracy and stability from increasing the grid order to $\mathcal{P}3+$ is when significant regions of the geometry surfaces are defined by circles, i.e., cylindrical/axisymmetric or spherical geometries, simply because a $\mathcal{P}3+$ polynomial provides a markedly better approximation of a circle compared to a $\mathcal{P}2$ polynomial approximation. Note, the grid order and solution order are completely independent. The grid order does not need to match or exceed the solution order, or vice versa.

Grid files are currently subject to the following limitations:

- GFR does not support structured zones in CGNS grid files.
- Gmsh grid files should be avoided since GFR only supports Gmsh files in the older ASCII Legacy Version 2 format, and the Gmsh grid reader has been minimally tested.
- Multizone grids must have contiguous, point-matched, non-overlapping faces.
- Only the element types described in Section 3.2 are permitted.
- GFR does not support elements with collapsed faces or edges. For example, this can happen if trying to create a 3D axisymmetric unstructured grid by rotating a 2D unstructured grid about a pole boundary, and the 2D grid contains one or more quad cells with only a single node on the pole boundary. This will create an invalid hexahedral element with a single collapsed edge at the pole boundary. In this particular situation, it is generally recommended to create the 2D grid such that it does not go all the way to the pole boundary /rotation axis, create the 3D grid by rotating the 2D grid about the rotation axis, and finally create a separate 3D grid within the void that is created along the rotation axis.

The flow solver hides much of the complicated grid manipulation from the user. During the flow solution process, structured grids are converted to unstructured quadrilaterals or hexahedra, multizone grids are merged, and redundant faces are removed. The required number of solution points will then be inserted into the cell volume for the specified solution order. The coordinates of these points are determined using linear interpolation between the cell vertices and the chosen Gauss or Lobatto locations. The domain is then partitioned to make efficient use of the computational resources available. Before output files are generated, the domain is recombined to simplify flow visualization.

See Also: `Grid_File`, `Grid_Format`, `Grid_Scaling_Factor`, `Initialization`, `Loc_Solution_Pts`, `Solution_Order`

10.2.1 CGNS

CGNS is the recommended format to use for grid files with GFR. The primary reason for this is because CGNS is one of the few commonly used CFD grid formats that supports high-order curved elements. The Gmsh file format also supports high-order curved elements, but the GFR Gmsh grid reader is severely limited and out-of-date. The CGNS file format is very flexible in allowing for multiple ways to store the same computational mesh, which can make it difficult to create a universal CGNS grid reader that never fails. All of the CGNS grids used thus far with GFR have been generated by the Cadence Fidelity Pointwise software. Therefore, grids generated by Pointwise should have the highest success at being read and successfully processed by GFR. CGNS files generated by other grid generation software have not been tested for compatibility with GFR.

When the solid body surfaces in a flow contain nonlinear curvature, using a tessellation of linear segments to approximate this curvature — as is done with linear grids used for traditional 2nd-order finite-volume methods — results in a misrepresentation of the surface for high-order finite-element-type methods, e.g., FR/DG. Without any curvature information, these high-order methods will treat the linear tessellation of the curved surface as a series of flat plates, with each new linear segment of the surface resulting in a new compression or expansion corner. This linear approximation is sufficient for 2nd-order finite-volume methods because 1) the compression/expansion angle between linear segments is generally smaller for these methods since they generally require higher-resolution grids compared to higher-order methods, and 2) the error from the numerical discretization generally provides enough numerical dissipation to damp out any non-physical waves that might emanate from the intersection of two linear segments. All of this is to say that high-order curved meshes are critical for high-order methods to achieve their accuracy potential for flows involving solid body surfaces with nonlinear curvature.

We have used Pointwise to generate grids for GFR because it is one of the few available grid generation tools that supports unstructured, high-order curved meshes. Pointwise provides both written documentation and online videos demonstrating how to use the software to generate high-order meshes, so that information will not be covered here. The one important thing is to make sure any curved boundary surfaces are accurately represented as a Database surface in Pointwise, and that the Domain meshes on these surfaces have been projected to their respective Databases. The remainder of this section will cover the steps in the Pointwise grid generation process that affects the use of the grid within GFR.

10.2.1.1 Setting Boundary Conditions in Pointwise

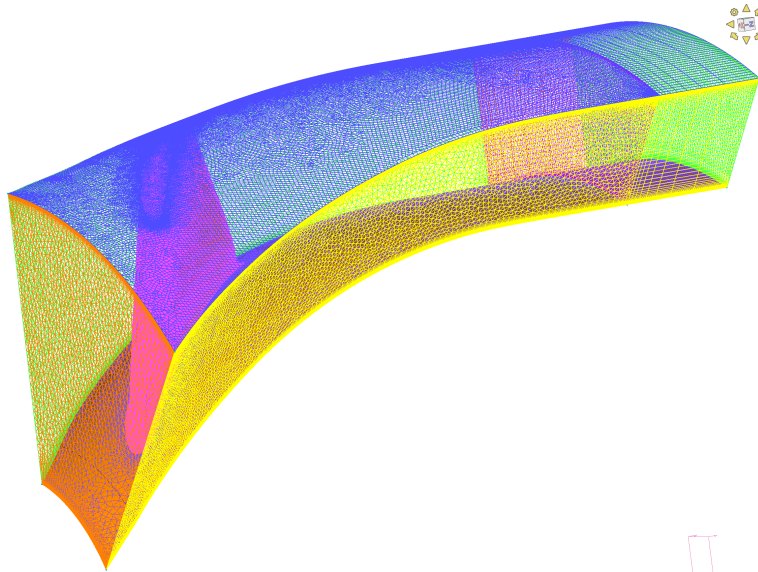
The primary task required to effectively use a CGNS grid exported from Pointwise with GFR is setting the boundary conditions for the grid. Setting the boundary conditions in Pointwise starts with accessing the "Set BC" dialog box by selecting "Set Boundary Conditions..." under the "CAE" menu. Figures 18a and 19a show examples of the "Set BC" dialog box for two different cases previously run with GFR. Images of the respective grids are also shown alongside the set of boundary conditions in figures 18b and 19b as a reference.

In the "Set BC" dialog box, the third column titled "Name" and the fourth column titled "CAE Type" are the user inputs that are used to configure the boundary conditions for use with GFR. The "Name" column allows you to input a string to give each boundary condition a name. The string provided in Pointwise as the name for a boundary condition is used to match against one of the boundary conditions in the GFR input file, specifically against the inputs `BC.Input(i)%Name`. The "CAE Type" column provides a drop down list of different boundary condition types supported by the CGNS standard. Note that certain boundary condition types supported by the CGNS standard

are not supported by GFR, but GFR also supports boundary condition types not supported by the CGNS standard. The boundary condition options in the drop down list are given in the first column of table 9. Ideally, the GFR input file will also provide the respective $\text{BC_Input}(i)\%\text{Set_BC_Value}$ to specify the type of boundary condition to use for the boundary. If Set_BC_Value is not specified, the option specified under "CAE Type" is used to set the type of boundary condition to use for the boundary, and table 9 lists the GFR boundary condition assigned based on the "CAE Type".

Panels						
List		Layers	Defaults	Set BC		
Set	#	Name	CAE Type	ID	Tri Count	Quad Count
☐	2	Connection	Connection	Connection	0	0
☐	0	Unspecified	Unspecified	Unspecified	0	0
☐	1	Inflow	Inflow	1	3,292	1,640
☐	1	ThetaMax	Farfield	3	7,542	5,360
☐	1	ThetaMin	Farfield	4	7,542	5,360
☐	1	WallCase	Wall Viscous	5	10,876	13,682
☐	1	WallHub	Wall Viscous	6	5,166	3,076
☐	5	WallBlade	Wall Viscous	7	52,284	0
☐	1	ThetaMinSponge	Farfield	8	0	536
☐	1	ThetaMaxSponge	Farfield	9	0	536
☐	1	Outflow	Outflow	10	4,502	2,783
☐	1	WallCaseSponge	Wall Viscous	11	0	592
☐	1	WallHubSponge	Wall Viscous	12	0	592

(a) Pointwise "Set BC" dialog box



(b) Computational geometry

Figure 18: Pointwise boundary condition settings for the spacecraft cabin ventilation fan grid. The string in the third column titled "Name" is used to match against $\text{BC_Input}(i)\%\text{Name}$ in the GFR input file. The selection in the fourth column titled "CAE Type" is used to set the boundary condition value based on table 9 if the input $\text{BC_Input}(i)\%\text{Set_BC_Value}$ is not specified for a boundary condition with the respective name.

```

&nml_bc
BC_Input(1)%Name           = "Inflow"
BC_Input(1)%Set_BC_Value   = 2           ! Subsonic inflow
BC_Input(1)%Total_Pressure = 102070.0    ! [Pa] (101992.2 static)
BC_Input(1)%Temperature    = 294.3       ! [K]
BC_Input(1)%Mach           = 0.033       ! Mach number

BC_Input(2)%Name           = "Outflow"
BC_Input(2)%Set_BC_Value   = 12          ! Subsonic outflow
BC_Input(2)%Pressure       = 102553.0    ! [Pa] (1.0055 * Inflow Pressure)
BC_Input(2)%Temperature    = 294.3       ! [K]
BC_Input(2)%Mach           = 0.033       ! Mach number

BC_Input(3)%Name           = "WallHub"
BC_Input(3)%Set_BC_Value   = 52          ! Adiabatic wall
BC_Input(3)%Is_Specified_In_MRF = T      ! Moving reference frame
BC_Input(3)%Rotation_Rate(1) = 0.0       ! [RPM]

BC_Input(4)%Name           = "WallHubSponge"
BC_Input(4)%Set_BC_Value   = 52          ! Adiabatic wall
BC_Input(4)%Is_Specified_In_MRF = T      ! Moving reference frame
BC_Input(4)%Rotation_Rate(1) = 0.0       ! [RPM]

BC_Input(5)%Name           = "WallCaseSponge"
BC_Input(5)%Set_BC_Value   = 52          ! Adiabatic wall
BC_Input(5)%Is_Specified_In_MRF = F      ! Fixed reference frame
BC_Input(5)%Rotation_Rate(1) = 0.0       ! [RPM]

BC_Input(6)%Name           = "WallCase"
BC_Input(6)%Set_BC_Value   = 52          ! Adiabatic wall
BC_Input(6)%Is_Specified_In_MRF = F      ! Fixed reference frame
BC_Input(6)%Rotation_Rate(1) = 0.0       ! [RPM]

BC_Input(7)%Name           = "WallBlade"
BC_Input(7)%Set_BC_Value   = 52          ! Adiabatic wall
BC_Input(7)%Is_Specified_In_MRF = T      ! Moving reference frame
BC_Input(7)%Rotation_Rate(1) = 0.0       ! [RPM]

BC_Input(8)%Name           = "ThetaMin"
BC_Input(8)%Set_BC_Value   = 81          ! Periodic BC
BC_Input(8)%Periodic%Matching_BC_Name = "ThetaMax"
BC_Input(8)%Periodic%Matching_BC_Idx = 9
BC_Input(8)%Periodic%RotationAngle(1) = 40.0 ! [deg]

BC_Input(9)%Name           = "ThetaMax"
BC_Input(9)%Set_BC_Value   = 81          ! Periodic BC
BC_Input(9)%Periodic%Matching_BC_Name = "ThetaMin"
BC_Input(9)%Periodic%Matching_BC_Idx = 8
BC_Input(9)%Periodic%RotationAngle(1) = -40.0 ! [deg]

BC_Input(10)%Name          = "ThetaMinSponge"
BC_Input(10)%Set_BC_Value  = 81          ! Periodic BC
BC_Input(10)%Periodic%Matching_BC_Name = "ThetaMaxSponge"
BC_Input(10)%Periodic%Matching_BC_Idx = 11
BC_Input(10)%Periodic%RotationAngle(1) = 40.0 ! [deg]

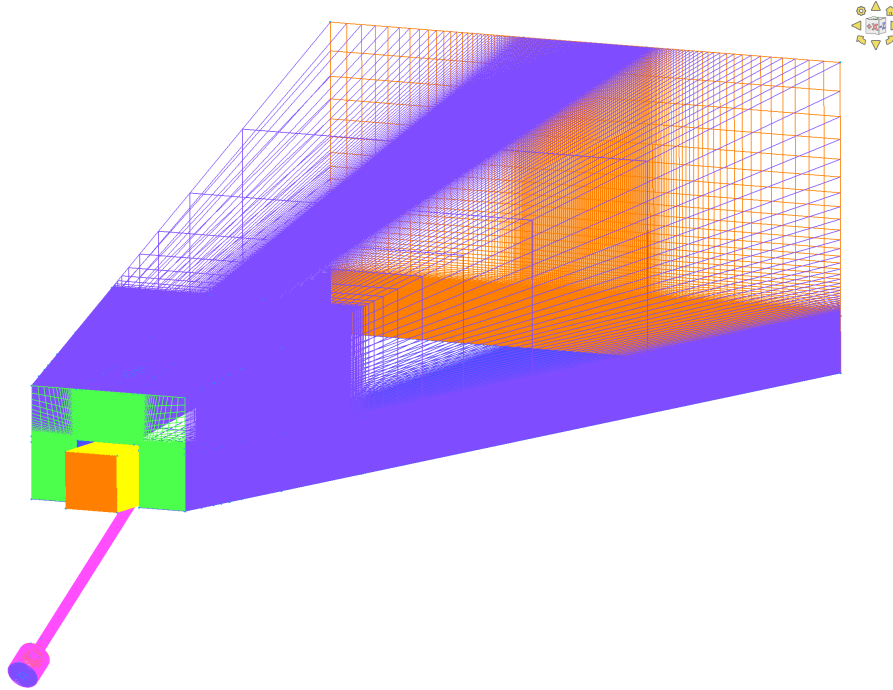
BC_Input(11)%Name          = "ThetaMaxSponge"
BC_Input(11)%Set_BC_Value  = 81          ! Periodic BC
BC_Input(11)%Periodic%Matching_BC_Name = "ThetaMinSponge"
BC_Input(11)%Periodic%Matching_BC_Idx = 10
BC_Input(11)%Periodic%RotationAngle(1) = -40.0 ! [deg]
/

```

Listing 1: Example GFR inputs that correspond to the boundary condition configuration shown in Fig. 18a for the spacecraft cabin ventilation fan grid.

Panels						
List		Layers	Defaults	Set BC		
Set	#	Name	CAE Type	ID	Tri Count	Quad Count
☐	0 (488)	Connection	Connection	Connection	0	0
☐	0 (66)	Unspecified	Unspecified	Unspecified	0	0
☐	1	InflowNozzle	Inflow Subsonic	1	0	14,400
☐	4	WallNozzleInterior	Wall Viscous	2	0	45,600
☐	6	WallNozzleLip	Wall Viscous	3	0	6,272
☐	14	WallNozzleExterior	Wall Viscous	4	0	11,872
☐	6	InflowPlenum	Inflow Subsonic	5	0	3,072
☐	12	WallPipe	Wall Viscous	6	0	29,088
☐	6	Outflow	Outflow	7	0	32,000
☐	24	WallPlate	Wall Viscous	8	0	37,812
☐	9	InflowExterior	Inflow Subsonic	9	0	11,328
☐	9	Bottom	Wall Inviscid	10	0	28,544
☐	44	Farfield	Farfield	11	0	184,724
☐	1	WallPlenumSide	Wall Viscous	12	0	1,440
☐	1	WallPlenumEnd	Wall Viscous	13	0	960

(a) Pointwise "Set BC" dialog box



(b) Computational geometry

Figure 19: Pointwise boundary condition settings for the THX3 grid. The string in the third column titled "Name" is used to match against `BC_Input(i)%Name` in the GFR input file. The selection in the fourth column titled "CAE Type" is used to set the boundary condition value based on table 9 if the input `BC_Input(i)%Set_BC_Value` is not specified for a boundary condition with the respective name.

```

&nml_bc
BC_Input(1)%Name           = "InflowNozzle"
BC_Input(1)%Set_BC_Value   = 5           ! SEM inflow
BC_Input(1)%Total_Pressure = 108869.053  ! [Pa] (15.79 psi)
BC_Input(1)%Total_Temperature = 538.97    ! [K] (970.146 R)
BC_Input(1)%Mach           = 0.3732     ! Inflow Mach number
BC_Input(1)%SEM%Convective_Velocity = 84.723    ! [m/s] 1/2 of  $U_{SEM}$ 
BC_Input(1)%SEM%Length_Scale = 0.00030038 ! [m] 1/5 of  $\delta_{BL}$ 
BC_Input(1)%SEM%Profile_File = "nozzle_exit.m-k-g-s.match_P0-T0.dat"

BC_Input(2)%Name           = "InflowPlenum"
BC_Input(2)%Set_BC_Value   = 2           ! Subsonic inflow
BC_Input(2)%Total_Pressure = 110500.0    ! [Pa] (16.03 psi)
BC_Input(2)%Pressure       = 98595.03    ! [Pa] (14.3 psi)
BC_Input(2)%Total_Temperature = 294.293   ! [K] (529.7274 R)
BC_Input(2)%AoA_Alpha      = 30.0        ! Inflow angle of attack

BC_Input(3)%Name           = "InflowExterior"
BC_Input(3)%Set_BC_Value   = 2           ! Subsonic Inflow
BC_Input(3)%Total_Pressure = 98767.68    ! [Pa] (14.325 psi)
BC_Input(3)%Total_Temperature = 294.591222 ! [K] (530.2642 R)
BC_Input(3)%Mach           = 0.05        ! Inflow Mach number

BC_Input(4)%Name           = "Farfield"
BC_Input(4)%Set_BC_Value   = 21          ! Characteristic in/outflow
BC_Input(4)%Total_Pressure = 98767.68    ! [Pa] (14.325 psi)
BC_Input(4)%Total_Temperature = 294.591222 ! [K] (530.2642 R)
BC_Input(4)%Mach           = 0.05        ! Inflow Mach number

BC_Input(5)%Name           = "Outflow"
BC_Input(5)%Set_BC_Value   = 12          ! Subsonic outflow
BC_Input(5)%Pressure       = 98595.03    ! [Pa] (14.3 psi)
BC_Input(5)%Temperature    = 294.444     ! [K] (530.0 R)
BC_Input(5)%VelocityX      = 169.4460    ! [m/s]  $\delta_{BL}$ 

BC_Input(6)%Name           = "WallPlate"
BC_Input(6)%Set_BC_Value   = 53          ! Isothermal wall
BC_Input(6)%Wall_Temp      = 427.888889  ! [K] (770.2 R)

BC_Input(7)%Name           = "WallNozzleInterior"
BC_Input(7)%Set_BC_Value   = 52          ! Adiabatic wall

BC_Input(8)%Name           = "WallNozzleLip"
BC_Input(8)%Set_BC_Value   = 52          ! Adiabatic wall

BC_Input(9)%Name           = "WallNozzleExterior"
BC_Input(9)%Set_BC_Value   = 52          ! Adiabatic wall

BC_Input(10)%Name          = "WallPipe"
BC_Input(10)%Set_BC_Value  = 52          ! Adiabatic wall

BC_Input(11)%Name          = "WallPlenumSide"
BC_Input(11)%Set_BC_Value  = 52          ! Adiabatic wall

BC_Input(12)%Name          = "WallPlenumEnd"
BC_Input(12)%Set_BC_Value  = 52          ! Adiabatic wall

BC_Input(13)%Name          = "Bottom"
BC_Input(13)%Set_BC_Value  = 50          ! Slip wall
/

```

Listing 2: Example GFR inputs that correspond to the boundary condition configuration shown in Fig. 19a for the THX3 grid.

Table 9: Assumed mapping of CGNS and GFR boundary conditions if `BC_Input(i)%Set_BC_Value` is not used. Unsupported CGNS boundary conditions will attempt to set the GFR boundary condition type based on the “name match” method described in Section 7.1.4 and table 4.

CGNS BC Option (PW "CAE Type")	GFR BC Value	Boundary Condition Description	Section #
Unspecified	-9999	Name match is attempted	(7.1.4)
User Defined	-9999	Name match is attempted	(7.1.4)
Axisymmetric Wedge	-9999	Name match is attempted	(7.1.4)
Degenerate Line	-9999	Name match is attempted	(7.1.4)
Degenerate Point	-9999	Name match is attempted	(7.1.4)
Dirichlet	-9999	Name match is attempted	(7.1.4)
Extrapolate	13	Supersonic outflow	(7.3.3)
Farfield	21	Characteristic inflow/outflow	(7.4.1)
General	22	General free-flow inflow/outflow	(7.4.2)
Inflow	1	General Inflow	(7.2.1)
Inflow Subsonic	2	Subsonic Inflow	(7.2.2)
Inflow Supersonic	3	Supersonic Inflow	(7.2.3)
Neumann	-9999	Name match is attempted	(7.1.4)
Outflow	11	General outflow	(7.3.1)
Outflow Subsonic	12	Subsonic outflow	(7.3.2)
Outflow Supersonic	13	Supersonic outflow	(7.3.3)
Symmetry Plane	80	Symmetry boundary	(7.6.1)
Symmetry Polar	-9999	Name match is attempted	(7.1.4)
Tunnel Inflow	-9999	Name match is attempted	(7.1.4)
Tunnel Outflow	-9999	Name match is attempted	(7.1.4)
Wall	52	Adiabatic wall	(7.5.2)
Wall Inviscid	50	Inviscid/slip wall	(7.5.1)
Wall Viscous	52	Adiabatic Wall	(7.5.2)
Wall Viscous Heat Flux	52	Adiabatic Wall	(7.5.2)
Wall Viscous Isothermal	53	Isothermal Wall	(7.5.3)

10.2.1.2 Limitations

Currently, there is no mechanism to read structured (i.e., i, j, k ordered) blocks/zones from CGNS grid files, so GFR only supports CGNS grid files with unstructured zones. Structured blocks that are written to a CGNS grid file as quad/hex-only unstructured zones are fully supported. When using Pointwise, converting structured blocks to quad/hex-only unstructured zones is possible by choosing the option to combine all blocks when using the regular (i.e., linear) CGNS export, which should result in an unstructured grid file that can be used by GFR. CGNS grids exported from Pointwise using the high-order option will always be written as a single unstructured zone. Already existing structured CGNS grid files must first be converted to unstructured CGNS format using third-party software to use with GFR.

GFR requires CGNS files to be HDF5 formatted. ADF formatted CGNS files can be converted to HDF5 format using the `cgnsconvert` utility that is included with the CGNS library.

10.2.2 Gmsh

The only Gmsh files supported by GFR are ASCII Legacy version 2 formatted files. The Gmsh grid reader in GFR is not very robust and should be avoided if possible.

10.2.3 Plot3D

When using Plot3D grid files, additional controls are available to individually strip points in each of the computational directions. The technique is often referred to as adjusting the grid stride or grid sequencing. The resulting coarser mesh appears as simple hexahedral elements formed from the remaining grid nodes and the omitted nodes provide no contribution. For a higher-order method like flux reconstruction, this can have an adverse effect along boundaries, where the discrete representation of the surface curvature becomes reduced. The coarser faceting of the geometry surface can introduce spurious waves into the solution.

After completing the optional coarsening of the input grid, the flow solver can be instructed to combine a certain multiple of the remaining adjacent cells in all directions to form agglomerated cells. These agglomerated cells are not simple hexes with planar faces. Rather the faces are formed by polynomials defined by the nodes of the intermediate cells that were combined together. This is a somewhat crude way of constructing higher-order cells that preserve surface curvature. In order for agglomeration to work, the number of nodes in each direction of the grid must be an appropriate multiple of the desired agglomeration order. For example, a third-order agglomeration requires merging three cells in each direction.

NOTE: The agglomeration order for the geometry is independent of the solution order. The volume solution points and face flux points will be interpolated from the curved geometry information.

See Also: [Plot3d_Agglomerate_Order](#), [Plot3d_Stride_All](#), [Plot3d_Stride_I](#), [Plot3d_Stride_J](#), [Plot3d_Stride_K](#), [Solution_Order](#)

10.2.3.1 Plot3D Interface Cut File

For multiblock Plot3D grids, block connectivity is specified via a text-based interface cut file. An example of this cut file is shown in listing 3. The first line is a required header intended to assist the user, and is otherwise ignored. Subsequent lines define each named cut and the block information on either side of the interface. This information consists of the block number, face direction and plane, and the subregion of that face. To assist the user, each line is parsed based on whitespace rather than assuming a fixed format. Comment lines beginning with “!” can be used throughout the file and blank lines are ignored. In order to perform the matching, each cut name must be unique. Face and subregion index directions are specified as I, J, or K. Face and subregion locations are specified as MIN, MAX, or as an integer value. For two-dimensional flows, all K index values should be set to MIN. Checks are performed to ensure that the specified interfaces are complete and consistent.

CUT_NAME	BLK	FACE	PLACE	IND1	BEG	END	IND2	BEG	END
!=====	===	====	=====	=====	===	===	=====	===	===
ONE	1	I	MAX	J	MIN	MAX	K	MIN	MAX
ONE	2	I	MIN	J	MIN	MAX	K	MIN	MAX

Listing 3: Plot3D interface cut file example.

See Also: [Plot3d_Cutfile](#)

Table 10: Supported boundary condition names in Plot3D boundary condition file.

BC_NAME	Description
INFLOW	Generic Inflow
SUBINFLOW	Subsonic Inflow
SUPINFLOW	Supersonic Inflow
OUTFLOW	Generic Outflow
SUBOUTFLOW	Subsonic Outflow
SUPOUTFLOW EXTRAP	Supersonic Outflow (Extrapolation)
FARFIELD	Generic farfield
CHARACTERISTIC	Characteristic Boundary
FREESTREAM	Freestream
FIXED	Fixed
SLIP	Inviscid Wall
AWALL WALL	Adiabatic Viscous Wall
IWALL	Isothermal Viscous Wall
SYMMETRY	Symmetry
PERIODIC	Periodic

10.2.3.2 Plot3D Boundary Condition File

For Plot3D grids, boundary conditions are specified via a separate text-based boundary condition file. An example of this file is shown in listing 4. Once again, the first line is a header intended to assist the user, and is otherwise ignored. Subsequent lines define each named boundary condition and the block information for where that condition applies. This information consists of the block number, face direction and plane, and the subregion of that face. This information is parsed based on whitespace rather than assuming a fixed format. Comment lines beginning with “!” can be used throughout the file and blank lines are ignored. Face and subregion index directions are specified as I, J, or K. Face and subregion locations are specified as MIN, MAX, or as an integer value. For two-dimensional flows, all K index values should be set to MIN. The supported boundary condition names are listed in table 10 and are used to specify the type of boundary condition to apply to a particular face region. The group name is a user-specified string that is used to specify a named set of flow values to apply in conjunction with the boundary condition type. The group name should correspond to a **BC_Input(i)%Name** entry in the namelist input file. Numerical values for the flow conditions will then be taken from the other **BC_Input(i)** properties.

BC_NAME	BLK	FACE	PLACE	IND1	BEG	END	IND2	BEG	END	GROUPNAME
!=====	===	=====	=====	=====	===	===	=====	===	===	=====
SUBINFLOW	1	I	MIN	J	MIN	MAX	K	MIN	MAX	INFLOW
AWALL	1	J	MIN	I	MIN	MAX	K	MIN	MAX	BOT_WALL
AWALL	1	J	MAX	I	MIN	MAX	K	MIN	MAX	TOP_WALL
!										
EXTRAP	2	I	MAX	J	MIN	MAX	K	MIN	MAX	OUTFLOW
AWALL	2	J	MIN	I	MIN	MAX	K	MIN	MAX	BOT_WALL
AWALL	2	J	MAX	I	MIN	MAX	K	MIN	MAX	TOP_WALL

Listing 4: Plot3D boundary conditions file example.

See Also: [Plot3d_BCSfile](#)

10.2.3.3 Limitations

GFR does not support Plot3D iblanking. This feature is often used to cut holes in structured grids and omit those points from the flow domain, or to define donor interpolation information for chimera overset meshes. Depending on the nature of the iblanking, it may be possible to decompose the mesh into multiple zones that no longer require iblanking.

10.2.4 Interpolation Grid

It is sometimes desirable to interpolate the solution from one grid file onto another grid file. For example, after making modifications to an existing grid or when performing a formal grid sensitivity study. Initializing the starting solution in this way can alleviate some of the startup transients that typically occur and can reduce the overall simulation cost. To perform this task, read the existing grid and solution (or restart) files as one normally would. In addition, set the various inputs in the `nml_interpolation` namelist block (Section 11.20) to read the recipient grid and establish the proper number of solution points on that grid. These inputs are analogous to and support the full range of grid formats used during normal grid import.

After the interpolation is complete, the global filter will be applied to the entire solution if it is enabled in the input file. This can help “soften” the result of the interpolation by removing unwanted oscillations that might have been added to the solution. The final step of the interpolation utility is to write restart and output files, after which the program will stop without performing any iterations. The user should examine the output files using flow visualization software to look for obvious defects which may have been introduced into the new solution file. If the interpolated solution is satisfactory, then modify the namelist input file to read the new grid and restart files, and not perform interpolation again.

The interpolation utility is one of the few parts of the GFR code that has OpenMP parallelization. Enabling OpenMP in the `system_variables.mk` file can really help improve the performance of the interpolation algorithm, especially for larger simulations.

See Also: [nml_interpolation](#) (Section 11.20)

10.3 Solution Input

Besides the GFR restart file, a solution can also be read from a Plot3D q-file. It is assumed that the q-file is non-dimensionalized using the same reference values provided to GFR. Note that initialization from a Plot3D q-file is very experimental and only works if using a matching Plot3D grid file.

See Also: [Initial_Solution_File](#), [Initial_Solution_Format](#), [nml_reference](#)

10.4 Restart Input/Output

At the beginning of a run job, the solution will be read from the restart file if requested by the user in the input file. During a run job, a restart file can be output at a specified iteration interval and/or time frequency. It is important to distinguish that this update is based upon the current *aggregate* iteration or time (including that from prior run jobs), and not the *elapsed* value since the start of the current run job or the last restart output. Any intermediate output can be disabled by setting these inputs to 0. However, a restart file will always be written at the end of a run job.

If time averaging is activated, then an additional time-averaged restart file will be processed whenever the instantaneous restart file is read/written.

See Also: `Load.Restart.File`, `Restart.File`, `Restart.Frequency`, `Restart.Interval`, `Restart.Output.Disabled`, `Compute.Time.Averaging`, `Input.Time.Ave.File`

10.5 Display Output

During a run job, if the current aggregate iteration is a multiple of `Display.Interval`, convergence information for that particular iteration is written to the display screen. This information provides a high-level overview of the solution convergence. Operating system commands can be used to capture or redirect the display screen output to a file for later use.

See Also: `Display.Interval`

10.6 Residual Output

During a run job, if the current iteration is a multiple of `Residual.Interval`, convergence information for that iteration is written to the `residual.dat` file. This file is written in Tecplot ASCII format, with columns of data and intuitive headers. It contains detailed convergence information for each of the transport equations being solved. The residual measures the change in a solution variable over the given time step and is computed as the L2 sum over all of the solution points.

See Also: `Residual.Interval`

10.7 Solution Output

At the start of every run job, CGNS grid and solution files are output. During a run job, if the current iteration is a multiple of `Output.Interval` and/or the current simulation time is a multiple of `Output.Frequency`, an updated CGNS solution file is output. This update is based upon the current *aggregate* iteration or time (including that from prior run jobs), and not the *elapsed* value since the start of the current run job or the last CGNS output. The iteration number will be part of the CGNS solution file name.

If the code terminates prematurely due to a detected NaN value, a CGNS solution file is written for the previous time step. The file name will include “NaN” to designate the solution was unstable. This file can be extremely helpful to help diagnose where the solution might have diverged.

At the end of every run job, an updated CGNS solution file is output for that iteration. A special CGNS file called `solution.time.cgns` is also output. This file contains CGNS links to each of the instantaneous CGNS solution files generated during the run, and can be used with flow visualization software to easily read and examine all the time strand solutions. Beware that if writing to an existing directory, `solution.time.cgns` will be overwritten with links to only those solution files generated during the current run.

Every CGNS solution file contains an internal CGNS link to the corresponding grid file. For most plotting software, the user will not need to explicitly load the corresponding grid file. However, the grid file cannot be renamed without breaking the internal CGNS link, and *the grid file must be located within the same directory as the solution file*.

NOTE: When running in parallel, the internal format of the output CGNS grid and solution files may depend on the number of processors used. If writing to an existing directory, then

`solution.grid.cgns` will be overwritten. This can cause problems if the number of processors has changed between successive run jobs, because each previous solution file has an internal link to `solution.grid.cgns` and the internal format of that file may no longer be compatible with the old solution file. Therefore, it is recommended that each successive run job use a different output directory. This way, the `solution.grid.cgns` and `solution.time.cgns` files for each run will be preserved.

See Also: `Output_Dir`, `Output_Interval`, `Output_Frequency`

10.8 Time-Averaging Output

When performing large-eddy simulations, time-averaged variables are often needed to aid in post-processing. Storing all of the instantaneous flow variables at every sampling time would permit one to compute any time-averaged quantity at the end of the simulation by re-processing the data, but would also require a prohibitive amount of storage. Instead, these time-averaged variables can be computed on-the-fly during each iteration of the solution process and output as a separate set of restart and CGNS files whenever those are written.

The list of variables that are computed and saved in the time-averaged restart file is as follows.

$\bar{\rho}$	$\overline{\rho^2}$		
$\bar{T}$	$\overline{T^2}$	$\overline{T_t}$	
$\bar{p}$	$\overline{p^2}$	$\overline{p_t}$	
$\overline{\rho u}$	$\bar{u}$		
$\overline{\rho v}$	$\bar{v}$	$\overline{\rho u_r}$	$\overline{u_r}$
$\overline{\rho w}$	$\bar{w}$	$\overline{\rho u_\theta}$	$\overline{u_\theta}$
$\overline{\rho e_t}$	$\overline{\rho T}$		
$\overline{u_x u_x}$	$\overline{\rho u_x u_x}$	$\overline{u_x T}$	$\overline{\rho u_x T}$
$\overline{u_r u_r}$	$\overline{\rho u_r u_r}$	$\overline{u_r T}$	$\overline{\rho u_r T}$
$\overline{u_\theta u_\theta}$	$\overline{\rho u_\theta u_\theta}$	$\overline{u_\theta T}$	$\overline{\rho u_\theta T}$
$\overline{u_x u_r}$	$\overline{\rho u_x u_r}$		
$\overline{u_x u_\theta}$	$\overline{\rho u_x u_\theta}$		
$\overline{u_r u_\theta}$	$\overline{\rho u_r u_\theta}$		

Note that these are the time averages of instantaneous flow variables, and the higher order terms do not directly represent the turbulent fluctuations. However, this set of terms provides the ability during post-processing to compute conservative and primitive mean-flow variables, turbulent stresses, and turbulent heat fluxes in either the Reynolds-averaged or Favre-averaged form. For example,

$$\begin{aligned}
 -\overline{\rho u'_x u'_\theta} &= -\bar{\rho} (\overline{u_x u_\theta} - \bar{u}_x \bar{u}_\theta) & -\overline{\rho u''_x u''_\theta} &= -(\overline{\rho u_x u_\theta} - \bar{\rho} \tilde{u}_x \tilde{u}_\theta) & \tilde{u}_x &= \overline{\rho u_x} / \bar{\rho} \\
 \overline{\rho u'_x h'} &= \bar{\rho} c_p (\overline{u_x T} - \bar{u}_x \bar{T}) & \overline{\rho u''_x h''} &= c_p (\overline{\rho u_x T} - \bar{\rho} \tilde{u}_x \tilde{T}) & \tilde{u}_\theta &= \overline{\rho u_\theta} / \bar{\rho} \\
 & & & & \tilde{T} &= \overline{\rho T} / \bar{\rho}
 \end{aligned} \tag{78}$$

If the input `Time_Ave_Vel_Is_Axisymm` is false, then the variables (u_x, u_r, u_θ) in the statistics above are replaced with (u, v, w) and the Cartesian stresses and heat fluxes are retained instead. If a rotating reference frame is used, then the output will also be with respect to that reference frame.

Even though all of the previously mentioned variables are computed and saved in the time-averaged restart file, most users are primarily interested in those needed to compute the Reynolds-averaged quantities. Therefore only the subset of variables in table 11 is output to the time-averaged CGNS file. Changing the output list requires modifying subroutine `get_cgns_time_ave_variables` and recompiling the code. In the CGNS variable names, $(\mathbf{r}, \mathbf{f})$ represent (u_r, u_θ) . So, the values for $\mathbf{rf} = \overline{u_r u_\theta}$ and $\mathbf{rT} = \overline{u_r T}$ can then be used to compute the turbulent stress and heat flux similar to Eq. (78).

Table 11: Time-averaged variables output to CGNS based on `Time_Ave_Vel_Is_Axisymm`.

Output Mode		Variables	CGNS Variable Names
XYZ	Axi		
✓	✓	$\bar{\rho}, \bar{p}, \bar{T}, \overline{T'T'}$	Density, Pressure, Temperature, TT
✓	✓	$\bar{T}_t, \bar{p}_t$	StagnationTemperature, StagnationPressure
✓	✓	$\bar{u}, \bar{v}, \bar{w}$	VelocityX, VelocityY, VelocityZ
	✓	$\bar{u}_r, \bar{u}_\theta$	$\mathbf{r}, \mathbf{f}$
✓		$\overline{u'u'}, \overline{v'v'}, \overline{w'w'}$	uu, vv, ww
	✓	$\overline{u'u'}, \overline{u'_r u'_r}, \overline{u'_\theta u'_\theta}$	uu, rr, ff
✓		$\overline{u'v'}, \overline{u'w'}, \overline{v'w'}$	uv, uw, vw
	✓	$\overline{u'u'_r}, \overline{u'u'_\theta}, \overline{u'_r u'_\theta}$	ur, uf, rf
✓		$\overline{u'T'}, \overline{v'T'}, \overline{w'T'}$	uT, vT, wT
	✓	$\overline{u'T'}, \overline{u'_r T'}, \overline{u'_\theta T'}$	uT, rT, fT

See Also: `nml.time_ave`

10.9 Probes Output

With time accurate simulations, it is impractical to store the entire unsteady solution history for every grid point. This is why the GFR code primarily outputs instantaneous and/or time-averaged files at either fixed iteration or time intervals. However, it is sometimes convenient to have the entire unsteady flow history at a limited subset of points for later analysis. The GFR code provides the ability to output this unsteady history using point probes. Probes can either be specified as an individual probe at a specific coordinate point using the input `Point_Probes` or as a rake of probes that are evenly spaced along a line between two coordinate points using the input `Rake_Probes`. Note that the solution point closest to each requested probe location will be used for the source of the probe data. This is done to reduce the performance overhead that otherwise would be incurred from having to interpolate the solution to the requested probe location if it was not located exactly at a solution point.

Each point probe is written to a separate Tecplot ASCII formatted file within the directory `Probe_files`. Time-history files for probes specified as individual point probes will be named `point-probe.i.tec`, where i corresponds to the index of `Point_Probes` from the input file. Time-history files for probes within a rake of probes will be named `rake-probe.i-j.tec`, where i corresponds to the index of `Rake_Probes` from the input file, and j corresponds to the index of the

individual probe within rake i . Each file contains a simple header and comments to identify the requested probe location from the input, the actual probe location, and flow variables.

The data for all probes is sampled at an iteration interval that is specified by the input `Probes.Collection.Interval` and saved to a buffer. The input variable `Probes.Write.Interval` specifies how many data collections to perform until writing out the buffer to the individual probe time-history files.

The time-history probe files are appended each time the code is executed. If the number or order of probes changes between two consecutive simulations, the user should remove the existing files before proceeding.

If time-history data at an evenly spaced time interval is required, then a constant time step should be used.

See Also: `Probes.Collection.Interval`, `Probes.Write.Interval`, `Point.Probes`, `Rake.Probes`

10.10 Simulation Interrupt/Control Files

There are several temporary files that can be used to interrupt a running simulation to request a certain action or modify certain parameters while the simulation continues to run. This is particularly useful on large HPC systems where the alternative — stopping the simulation, editing the input file, then resubmitting the job to the system queue, and waiting for it to start — could take days or even weeks for large jobs needing thousands of MPI processes. After completion of every N^{th} time step, GFR checks the current run directory for the presence of a file with one of the following case-sensitive file names:

- `STOP_GFR`
- `ADJUST_CFL`
- `START_TIME_AVERAGING`
- `DUMP_RESTART`
- `DUMP_SOLUTION`

with the description of each of the files provided in the following subsections.

The value of N depends on how GFR is compiled, and takes the following values:

- $N = 1$ when compiled in Debug mode
- $N = 50$ when compiled in Release/Production mode and *disabling* code that accesses the PBS environment
- $N = 100$ when compiled in Release/Production mode and *enabling* code that accesses the PBS environment

These files are temporary in that they will automatically be deleted immediately after each file is found, read, and processed. This behavior is necessary because otherwise each file would continuously be found and processed every N time steps if not deleted.

10.10.1 STOP_GFR

This interrupt file allows the user to stop the currently running simulation, either immediately or at a future iteration specified inside the file. If this file is found, the first line of the file is read, and then the file is closed and deleted. If the value read from the first line is an integer, this value is saved as the final iteration number for the running simulation. Note that this value is treated as an *aggregate* iteration number, i.e., it directly corresponds to the first column of the iteration data written to display/stdout. This will cause the simulation to complete the standard shutdown sequence after completing the final iteration, including writing the final restart and solution output files. If 1) the **STOP_GFR** file is empty; 2) the read operation on the first line fails due to an invalid (i.e., non-integer) quantity; or 3) the final iteration value read from the file is less than the current iteration number; then GFR will immediately exit the solver loop and begin the standard shutdown sequence.

Using this interrupt file is as simple as running a command like

```
echo 148234 > STOP_GFR
```

on the command line in the simulation run directory. This example would cause GFR to stop the simulation after completing the 148,234th time step.

10.10.2 ADJUST_CFL

This interrupt file allows the user to modify the current CFL schedule of a running simulation. This file is assumed to contain the namelist **cfl_adjustment** defined as:

```
&cfl_adjustment
  CFL                      = [current simulation value]
  CFL_Ramping_Beg          = [current simulation value]
  CFL_Ramping_End          = [current simulation value]
  CFL_Ramping_Iterations   = [current simulation value]
  CFL_Ramping_Start_Iter   = [current simulation value]
/
```

Listing 5: Namelist cfl_adjustment.

The inputs in this namelist are identical to the respective inputs in Section 11.6. Since this file is only read after a simulation has already started, each of these five variables will have already been defined in the preprocessor when the simulation was starting. Hence, this is why the “default value” for each input is given as = [current simulation value].

If the **ADJUST_GFR** file is found, GFR will attempt to read this namelist from the file. If any of these inputs are provided in the namelist, they will directly replace the corresponding value within the simulation. Once the current iteration number reaches the value specified in the input **CFL_Ramping_Start_Iter**, the CFL ramping schedule is reset so that the CFL is linearly increased from **CFL_Ramping_Beg** to **CFL_Ramping_End** between iterations (**CFL_Ramping_Start_Iter** + 1) and (**CFL_Ramping_Start_Iter** + **CFL_Ramping_Iterations**). If the value of **CFL_Ramping_Start_Iter** read from the interrupt file is less than the current iteration number, the CFL ramping schedule is reset immediately.

Whenever the CFL ramping schedule is changed by the inputs in this file, a new restart file will be written just before the new schedule begins in case it is too aggressive and causes the simulation to fail.

This interrupt file can be very helpful to push the limits of finding the largest CFL that can be used while also maintaining a stable solution. One method to find this limit is to slowly keep pushing the CFL higher until the simulation fails. In past experience, plotting the data in `residual.dat` from the failed simulation can identify an approximate iteration number where the solution began to diverge. This iteration number can be used with the iteration statistics written to the display/stdout to identify the CFL value at that iteration. From there, you should be able to run a stable simulation with a “near optimal” CFL by using a CFL that is 85%–95% of the CFL value when the previous failed simulation began to diverge.

10.10.3 START_TIME_AVERAGING

The simple presence of this interrupt file requests GFR to begin time averaging the current simulation. If the simulation is already time averaging, this interrupt file will be deleted and ignored. If the simulation is not time averaging, GFR will:

- Begin collecting time averages of various variable quantities (see Section 10.8).
- Start writing time-averaged restart files (`Restart_files/out.#.ave`) each time an instantaneous restart file is written.
- Start writing time-averaged solution output files (`CGNS_files/solution.time_ave.#.cgns`) each time an instantaneous solution output file is written.
- Reset the solution time written to the display/stdout to zero. From this point on, this quantity will report the total amount of time that the solution has been time averaged. Note, the actual simulation time is not reset within GFR and will continue accumulating in the background. Only the time quantity that is written to display is reset. The iteration counter is also not reset.
- Reset the maximum L2 residual quantities used to determine convergence, and begin reporting the L2 residual of the *time-averaged mean flow* variables to the `residual.dat` file instead of the *instantaneous* variables.

This file requires no contents, and any contents will be ignored. Using this interrupt file is as simple as running the following command on the command line in the simulation run directory:

```
touch START_TIME_AVERAGING
```

10.10.4 DUMP_RESTART

The simple presence of this interrupt file requests GFR to output an instantaneous restart file, as well as a time-averaged restart file if currently time-averaging. This file requires no contents, and any contents will be ignored. Using this interrupt file is as simple as running the following command on the command line in the simulation run directory:

```
touch DUMP_RESTART
```

10.10.5 DUMP_SOLUTION

The simple presence of this interrupt file requests GFR to output an instantaneous solution output file, as well as a time-averaged output file if currently time-averaging. This file requires no contents, and any contents will be ignored. Using this interrupt file is as simple as running the following

command on the command line in the simulation run directory:

```
touch DUMP_SOLUTION
```

11 User Input

As described in Section 10.1, GFR uses a Fortran namelist file for specifying user inputs, and can output a default input file for you to modify. The name of the input file to read should be added to the end of the MPI command line used to execute the code. This input file contains a number of namelist input blocks, which are described in the sections that follow.

- `nml_version`
- `nml_grid`
- `nml_reference`
- `nml_physics`
- `nml_initialization`
- `nml_timestep`
- `nml_bc`
- `nml_numerics`
- `nml_limiter`
- `nml_time_ave`
- `nml_output`
- `nml_output_variables`
- `nml_sem`
- `nml_chemistry`
- `nml_testcase_channel`
- `nml_testcase_pulse`
- `nml_testcase_sod`
- `nml_testcase_tcf`
- `nml_testcase_vortex`
- `nml_interpolation`
- `nml_utility`
- `nml_debug`

The order of the namelist blocks in the input file does not matter, nor does the order of the variables within a namelist block. The user may include every namelist block and every input variable in the input file. However, the only *required* entries are those inputs that the user wants to have non-default values. It is highly recommended to include the `nml_version` namelist block so that future versions of the code can properly interpret and convert your input file. Note that a “!” character in a Fortran namelist file begins a comment that extends to the end of the line. This can be useful for documenting your input file.

In each of the namelist input blocks described in the following sections, we first identify all the variables in the namelist block and then provide a detailed explanation of each input. Comments in the listings are used to document the expected units of the input variables (a good practice), and the values shown represent the default settings.

11.1 nml_version

This **namelist block** is used to inform GFR which version of the namelist to expect when reading through the input file. This namelist block is provided as a means to allow the use of older input files without having to manually convert them to the current format.

```
&nml_version  
  Namelist_Version = 2  
/
```

Listing 6: Namelist nml_version.

```
Namelist_Version = 2
```

The format of the namelist input file. Valid options are: 1, 2. Input variables may be added, removed, or renamed between versions. However, the code will attempt to convert an older input file to the current format. This report documents the format of namelist version 2. Users should add the **nml_version** block shown in listing 6 to the top of their input files so that future versions of the code will recognize it.

11.2 nml_grid

This **namelist block** is used to specify parameters related to the grid input file.

```
&nml_grid
  Grid_Scaling_Factor      = 1.0
  Grid_Format              = -1
  Grid_File                = ""
  Plot3d_Cutfile           = ""
  Plot3d_BCSfile           = ""
  Plot3d_Stride_I          = -1
  Plot3d_Stride_J          = -1
  Plot3d_Stride_K          = -1
  Plot3d_Stride_All        = -1
  Plot3d_Agglomerate_Order = -1
  Correct_CGNS_Grid_Points = F
  Internal_Grid_DoF        = 0
  Internal_Grid_Order      = -1
  Metis_Option             = 3
  Geom_Wts(1)%Tria        = 1
  Geom_Wts(1)%Quad        = 1
  Geom_Wts(1)%Tetr        = 1
  Geom_Wts(1)%Pyra        = 1
  Geom_Wts(1)%Pris        = 1
  Geom_Wts(1)%Hexa        = 1
/
```

Listing 7: Namelist nml_grid.

Grid_Scaling_Factor = 1.0

Multiplication factor (> 0.0) to scale the input grid file coordinates to the desired length units in meters. For example, use a value of 0.0254 to convert a grid containing coordinates in inches to meters. See the discussion of **grid units** and **grid scaled units** in Section 3.3.

Grid_Format = -1

Specifies the format of the input grid file.

- 1 = Internally generated grid (only supported with certain **Initialization** options)
- 1 = Gambit Neutral (very limited support, high-order curved elements not supported)
- 2 = Gmsh (limited support)
- 3 = Plot3D
- 4 = CGNS (recommended)

Grid_File = ""

File path to the input grid file relative to the current directory.

Plot3d_Cutfile = ""

File path to the input file that specifies the cut conditions for a multiblock Plot3D grid file.

```
Plot3d_BCSfile = ""
```

File path to the input file that specifies the boundary conditions for a Plot3D grid file.

```
Plot3d_Stride_All = -1
Plot3d_Stride_I   = -1
Plot3d_Stride_J   = -1
Plot3d_Stride_K   = -1
```

If > 0 , applies a coarsening stride to the specified direction of all Plot3D blocks.

`Plot3d_Stride_All` is applied first to all directions. Then the individual *i*, *j*, *k* options can be used to override the stride setting in that specific direction.

```
Plot3d_Agglomerate_Order = -1
```

If > 0 , agglomerates linear Plot3D cells into high-order grid cells of this polynomial order. For this to be successful, the number of *cells* in each direction for every block must be divisible by this value.

```
Correct_CGNS_Grid_Points = F
```

Experimental code that tries to correct the high-order points given in a CGNS grid file created by Pointwise. The high-order points should be equally spaced, but some software such as Pointwise does not require this.

```
Internal_Grid_DoF = 0
```

This specifies the grid resolution to use for an **internally generated grid**.

- < 0 User specified number of cells.
- $= 0$ 512 degrees of freedom.
- > 0 User specified degrees of freedom.

If the number of degrees of freedom are specified, then the number of cells needed to achieve the specified degrees of freedom in each direction will be computed from the solution order as:

$$[\# \text{ of cells}] = \text{nint}(\text{Internal_Grid_DoF}/(\text{Solution_Order} + 1)).$$

NOTE: An internally generated grid will only be created for specific **Initialization** conditions, as indicated on page 111. For the following cases, one should instead use the grid inputs in the associated namelist block.

Initialization	Description	Namelist Block
2	Channel flow	nml_testcase_channel
21	Taylor-Couette flow	nml_testcase_tcf
22	Freestream Taylor-Couette flow	nml_testcase_tcf

```
Internal_Grid_Order = -1
```

Polynomial order used by the internal grid generator to create a high-order grid. This only applies to internally generated grids with curved geometries (i.e., Taylor-Couette flow, channel, and annular pipe channel test cases). All other internally generated grids have linear geometries.

```
Metis_Option = 3
```

Specifies the METIS algorithm to use for partitioning the global grid. If the specified option fails, it will loop through all of the other options to try to find a successful partitioning.

- 1 = Create non-weighted partitions based on a multilevel recursive bisection while minimizing the edge cut.
- 2 = Create non-weighted partitions based on a multilevel k -way algorithm while minimizing the edge cut.
- 3 = Create non-weighted partitions based on a multilevel k -way algorithm while minimizing the total communication volume.
- 4 = Create weighted partitions based on a multilevel k -way algorithm while minimizing the total communication volume. This is equivalent to option 3 but also includes cell weights (based on geometry type).
- 5 = Create weighted partitions based on a multilevel k -way algorithm while minimizing the edge cut. This is equivalent to option 2 but also includes cell weights (based on geometry type).
- 6 = Create weighted partitions based on a multilevel recursive bisection while minimizing the edge cut. This is equivalent to option 1 but also includes cell weights (based on geometry type).

Note that the last three options include geometry-dependent computational cost into load balancing the partitions. Weighting for each geometry type and order is specified through the input **Geom_Wts**.

```
Geom_Wts(1)%Tria = 1  
Geom_Wts(1)%Quad = 1  
Geom_Wts(1)%Tetr = 1  
Geom_Wts(1)%Pyra = 1  
Geom_Wts(1)%Pris = 1  
Geom_Wts(1)%Hexa = 1
```

These are the geometry weights used by the METIS partitioning algorithm to split the mesh between processors. Valid values are integers in the range [0:100], with larger values indicating those geometries must perform more computational work relative to geometries with lower values. Note that the **Tria** and **Quad** inputs are for element volumes for 2D simulations, and thus are not used for 3D simulations. See the **Metis_Option** input for additional details.

Preliminary testing of the geometry weights indicates that improvements in performance may be highly dependent upon the number and distribution of each cell type throughout the domain and how many partitions are requested. We therefore offer no specific weighting recommendations.

11.3 nml_reference

This **namelist block** is used to specify the reference conditions for the simulation. These conditions are used within the flow solver to nondimensionalize the internal variables, and in most cases to initialize the volume solution. They are also written to the screen during code execution.

```
&nml_reference
  Mach           = 0.4
  Reynolds       = -1.0      ! rho*U*L/mu, L=1.0[m]
  Gamma         = 1.4
  Gas_Constant   = 287.05    ! [J/(kg*K)] = [m^2/(s^2*K)]
  Density        = -1.0      ! [kg/m^3]
  Temperature    = 300.0     ! [K]
  Pressure       = 100000.0  ! [Pa]
  AoA_Alpha      = 0.0       ! [deg]
  AoA_Beta       = 0.0       ! [deg]
  Prandtl        = 0.72
  Schmidt        = 0.72
  Suth_Muref     = 1.716e-5  ! [Pa*s] = [N*s/m^2]
  Suth_Tref      = 273.0     ! [K]
  Suth_Sref      = 110.4     ! [K]
/
```

Listing 8: Namelist nml_reference.

NOTE: Certain initialization options replace the reference conditions by preset values to ensure repeatability of these benchmark cases. See the **Initialization** input for details.

Mach = 0.4

Reference Mach number. Must be non-negative.

Reynolds = -1.0 ! rho*U*L/mu, L=1.0[m]

Reference Reynolds number, $Re = \rho UL/\mu$, based on reference conditions. Here, ρ is the reference density; U is the velocity magnitude computed from the reference Mach number and speed of sound at the reference temperature; L is the reference length of 1.0 meter; and μ is the viscosity computed from Sutherland's viscosity law using Eq. (43) at the reference temperature.

How the Reynolds number is used depends on the input value:

- ≤ 0 The Reynolds number will be computed from the other reference flow properties. This yields 8.7319×10^6 for the default values shown above, but will change if you modify any of the other reference conditions.
- > 0 The Reynolds number will be used with the other reference conditions you specify to recalibrate the μ_0 constant (**Suth_Muref**) in **Sutherland's viscosity law**, see Eq. (43). This can be used to prescribe an artificially lower Reynolds number that permits better resolution of turbulent scales with large-eddy simulation. It cannot be used to compute other unspecified conditions such as density, pressure, or temperature.

Gamma = 1.4

Reference γ (ratio of specific heats). If the value specified is negative, it will be reset to the default of 1.4 (the typical value for air).

Gas_Constant = 287.05 ! [J/(kg*K)] = [m^2/(s^2*K)]

Reference specific gas constant (units of J/(kg K)). If the value specified is negative, it will be reset to the default of 287.05 (the typical value for air).

NOTE: In the code, the default value actually has more precision since it is computed as $\text{Gas_Constant} = R_{\text{univ}}/M_{\text{air}}$, where R_{univ} is the universal gas constant that is hard-coded as a Fortran constant with the value 8314.5 J/(K kmol), and M_{air} is the molar mass of air which is also a hard-coded Fortran constant with the value 28.9651159 kg/kmol.

Density = -1.0 ! [kg/m^3]
Temperature = 300.0 ! [K]
Pressure = 100000.0 ! [Pa]

Reference static density ρ (kg/m³), temperature T (K), and pressure p (Pa).

NOTE: If all three inputs are positive, a consistency check will be performed. If one of these three inputs is negative, it will be computed from the other two using the equation of state, Eq. (42). So, for the default values shown above, density will evaluate to 1.160833 kg/m³.

AoA_Alpha = 0.0 ! [deg]
AoA_Beta = 0.0 ! [deg]

Reference flow angles (in degrees): α is the **angle of attack** measured in the x - y plane, and β is the **angle of sideslip** measured in the x - z plane. These flow angles are defined by Eqs. (46)–(48) and Fig. 13 on page 47.

Prandtl = 0.72

Prandtl number of the flow.

Schmidt = 0.72

Schmidt number of the flow.

Suth_Muref = 1.716e-5 ! [Pa*s] = [N*s/m^2]
Suth_Tref = 273.0 ! [K]
Suth_Sref = 110.4 ! [K]

These inputs correspond to the constants μ_0 , T_0 , and S_0 in **Sutherland's viscosity law**, Eq. (43).

NOTE: If the reference Reynolds number specified is positive, it will be used to recalibrate the μ_0 constant. See **Reynolds** for details.

11.4 nml_physics

This **namelist block** is used to specify the physical modeling options for the simulation.

```
&nml_physics
  Governing_Equations           = 2
  Moving_Ref_Frame%Rotation(1:3) = [0.0,0.0,0.0] ! [RPM]
  Moving_Ref_Frame%Translation(1:3) = [0.0,0.0,0.0] ! [m/s]
  Moving_Ref_Frame%Spinup%Iter   = 0
  Moving_Ref_Frame%Spinup%Tstar  = 0.0
/
```

Listing 9: Namelist nml_physics.

Governing_Equations = 2

Specifies the governing equations to use.

1 = Euler equations

2 = Navier-Stokes equations

Moving_Ref_Frame%Rotation(1:3) = [0.0,0.0,0.0] ! [RPM]

This specifies the rotation rate (in RPMs) of the moving reference frame about each of the three Cartesian axes.

Moving_Ref_Frame%Translation(1:3) = [0.0,0.0,0.0] ! [m/s]

This specifies the translation speed (in m/s) of the moving reference frame in each of the three Cartesian directions.

Moving_Ref_Frame%Spinup%Iter = 0

Number of time steps to spin up the rotation of the moving reference frame from 0 to max RPMs.

The spin-up period used will be the slower between this and **Moving_Ref_Frame%Spinup%Tstar**.

Moving_Ref_Frame%Spinup%Tstar = 0.0

If > 0 , this value specifies the non-dimensional time (t^*) interval, Eq. (80), to spin up the rotation of the moving reference frame from 0 to max RPMs. The spin-up period used will be the slower between this and **Moving_Ref_Frame%Spinup%Iter**.

11.5 nml_initialization

This **namelist block** is used to specify solution initialization.

```
&nml_initialization
  Initialization              = 1
  Jet_Init_Func              = 2
  Load_Restart_File          = F
  Restart_File                = "out.rst"
  Initial_Solution_File       = ""
  Initial_Solution_Format     = -99      ! -99 = Unknown
/
```

Listing 10: Namelist nml_initialization.

The initialization sequence is defined with the following precedence:

- If the file specified by **Initial_Solution_File** exists and **Initial_Solution_Format** is set to a valid value, then the solution data read from the specified file will be used as the initialization for a new simulation. Note that **Initial_Solution_File** and **Initial_Solution_Format** can be set to read an externally generated solution or a GFR restart file. The iteration and solution time will be set to zero.
- If the file specified by **Restart_File** exists and **Load_Restart_File**=T, then the solution will be initialized using the data read from the specified restart file. The iteration and solution time will continue from that in the restart file.
- Otherwise, the initial solution will be defined based on the **Initialization** option. The iteration and solution time will be set to zero.

NOTE: To continue a simulation from a restart file while preserving the iteration and solution time, one should set **Initial_Solution_File**="", **Load_Restart_File**=T, and **Restart_File** to the name of the input restart file. Many users forget to make these changes, and their simulations start over from the previously specified initialization rather than continuing the simulation.

Initialization = 1

Specifies the test-case/problem to solve, primarily affecting the initialization and post-processing of the simulation.

Value	IG	RC	PP	Description
1	.	.	.	Generic flow, initialize to reference conditions.
2	Y	.	.	Channel flow.
3	.	.	Y	Inviscid Gaussian bump.
4	.	.	Y	Laminar subsonic boundary layer.
5	Y	Y	.	Shu isentropic Euler vortex, propagating diagonally.
-5	Y	Y	.	Shu isentropic Euler vortex, stationary.
55	Y	Y	.	Shu isentropic Euler vortex, propagating diagonally, 4 total vortices.
-55	Y	Y	.	Shu isentropic Euler vortex, stationary, 4 total vortices.

Value	IG	RC	PP	Description
6	Y	.	.	Isentropic Euler vortex propagating in AoA Alpha direction.
-6	Y	Y	.	Isentropic Euler vortex propagating only in the <i>y</i> -direction.
7	Y	Y	.	Linear advection equation advecting a density profile.
8	Y	Y	.	Linear advection equation advecting an entropy profile.
9	Y	Y	.	Taylor-Green vortex problem.
10	.	.	.	Subsonic jet cases using the SMC000 nozzle.
-10	.	.	.	Subsonic jet cases using the ARN2 nozzle.
11	.	.	.	Supersonic jet cases.
12	.	.	.	Infinite cylinder.
13	.	.	.	Freestream preservation.
14	.	.	.	Reacting backstep.
15	.	.	.	2D diffusion flame fueled with hydrogen.
16	.	.	.	2D diffusion flame fueled with methane.
17	.	.	.	Nitrogen/Oxygen diffusion.
18	.	.	.	Compression corner.
19	Y	.	.	Methane flame speed test.
20	.	.	Y	THX Phase 3.
21	Y	Y	.	Taylor-Couette flow.
22	Y	.	.	Freestream Taylor-Couette flow.
23	Y	Y	.	1D Sod shocktube problem.
24	Y	.	.	1D Density pulse test.
25	Y	.	.	Annular pipe channel flow.
26	Y	Y	.	1D Shu-Osher problem.
27	Y	Y	.	1D Jiang-Shu density pulse test.

In the list above, the following designations are made:

- IG This input option supports an internally generated grid by setting **GridFormat** = -1 (internal grid), but external grids can be used as well. Grid resolution can be controlled as described under the **Internal_Grid_DoF** input.
- RC This input option overrides any user-specified reference conditions in **nml_reference** with preset values inside the code, which will be written to the screen during code execution.
- PP This input option only affects certain post-processed quantities, such as the screen output, and the flow will be initialized the same as option 1 (generic flow).

NOTE: This initialization is superseded when either loading a restart file by specifying **Load_Restart_File**=T or using the input **Initial_Solution_File** to initializing a simulation from a valid solution file. Instead, the flowfield will be read from the corresponding file. See the description of the **initialization sequence** on 111.

Jet_Init_Func = 2

Sometimes when initializing a jet simulation from freestream conditions, early transients within the nozzle can be numerically unstable, and a better initial solution is desirable.

If **Initialization** is ± 10 or 11, **Jet_Init_Func** can be used to specify different error function distributions to initialize the solution for the ARN2, SMC000, and the supersonic jet geometries. The jet geometry (e.g., ARN2, SMC000⁵⁷) used for the initialization function depends on the value of **Initialization**.

- 1 = Constant radial blending.
 - 2 = Blending in both x and r with a constant radial spreading.
 - 3 = Blending in both x and r with a downstream radial spreading.
- For any other value, the volume will be initialized to the freestream reference conditions.

```
Load_Restart_File = F
```

Specifies whether or not to read a restart file.

T = Read and initialize from the restart file given by **Restart_File**.

F = Do not read a restart file, but instead initialize according to **Initialization**.

NOTE: This will be superseded by **Initial_Solution_File** if the specified value is an existing file and **Initial_Solution_Format** is set to a valid value. The only difference is that this option keeps the iteration number and simulation time from the restart file, whereas **Initial_Solution_File** resets both to 0. See the description of the **initialization sequence** on 111.

```
Restart_File = "out.rst"
```

File path and name to the solution restart file that will be *read* if **Load_Restart_File** is true. This supersedes **Initialization**, but will be superseded by **Initial_Solution_File**. See the description of the **initialization sequence** on 111. The solution will continue from the iteration count and simulation time of the restart file.

NOTE: This has no effect on the restart files that are *written*, typically to:
Restart_files/out.<iteration>.rst.

```
Initial_Solution_File = ""
```

Path to a file that can be *read* for initializing the solution. This can be an externally generated solution or a GFR restart file. This is essentially the same as loading a restart file, except the *iteration count and simulation time are reset to 0*. See **initialization sequence** on page 111 for important details. When specifying **Initial_Solution_File**, one must also specify a valid value for **Initial_Solution_Format** as its default value will prevent this file from being read.

NOTE: This has no effect on the solution files that are *written*.

NOTE: After initialization is complete, set **Initial_Solution_File=""** so that subsequent simulations will continue from the previous restart file rather than starting over again. See page 111.

```
Initial_Solution_Format = -99 ! -99 = Unknown
```

Format of the file specified by input **Initial_Solution_File**.

0 = Restart file format.

3 = Plot3D file format (experimental).

NOTE: Any other value will prevent the file specified by **Initial_Solution_File** from being read, and another means will be used to initialize the solution.

11.6 nml_timestep

This **namelist block** is used to specify time stepping features.

```
&nml_timestep
  Runge_Kutta_Scheme      =    113
  Num_Timesteps_Min       =    100
  Num_Timesteps           =   25000
  Final_Time              =   -10.0   ! non-dim time, t*
  Timestep_Type           =         1
  Timestep                 =  1.0e-6   ! t[s] * ( a[m/s] / L=1.0[m])
  CFL                     =         1.0
  CFL_Ramping_Beg         =         0.1
  CFL_Ramping_End         =         1.0
  CFL_Ramping_Iterations  =         1
  CFL_Ramping_Start_Iter  =         0
  Convergence_Order_Abs_Res =   -16.0
  Convergence_Order_Max_Res =   -16.0
/
```

Listing 11: Namelist nml_timestep.

```
Runge_Kutta_Scheme = 113
```

Specifies which Runge-Kutta (RK) method to use. See Section 8.1 for additional details. Speedup is relative to the 3-stage/3rd-order SSP ERK method.

Option	Description	Speedup
Recommended options.		
113 =	11-stage/3 rd -order VOERK ⁹	1.33x
154 =	15-stage/4 th -order VOERK ⁹	1.33x
4 =	5-stage/4 th -order SSP ERK ³¹	1.22x
165 =	16-stage/5 th -order VOERK ⁹	1.18x
943 =	9-stage/4 th (3 rd)-order EPERK FSAL ¹⁰	1.22x
532 =	5-stage/3 rd (2 nd)-order EPERK FSAL ¹⁰	1.09x
Other options that work well.		
-2 =	2-stage/2 nd -order Classic RK	n/a
-3 =	3-stage/3 rd -order Classic RK	n/a
-4 =	4-stage/4 th -order Classic RK	n/a
2 =	2-stage/2 nd -order SSP ERK ³²	n/a
3 =	3-stage/3 rd -order SSP ERK ³²	1.00x
5 =	5-stage/4 th -order Carpenter-Kennedy low storage RK ³³	0.99x
53 =	5-stage/3 rd -order VOERK ⁹	1.08x
64 =	6-stage/4 th -order VOERK ⁹	1.02x
85 =	8-stage/5 th -order VOERK ⁹	1.05x
-943 =	9-stage/4 th (3 rd)-order EPERK non-FSAL ¹⁰	n/a
-532 =	5-stage/3 rd (2 nd)-order EPERK non-FSAL ¹⁰	n/a

Option	Description	Speedup
Options that do not perform as well.		
± 1	= 1-stage/1 st -order Forward Euler method	n/a
-53	= 5-stage/3 rd -order AOERK ⁹	n/a
-113	= 11-stage/3 rd -order AOERK ⁹	n/a
-64	= 6-stage/4 th -order AOERK ⁹	n/a
-154	= 15-stage/4 th -order AOERK ⁹	n/a
-85	= 8-stage/5 th -order AOERK ⁹	n/a
-165	= 16-stage/5 th -order AOERK ⁹	0.98x
54	= 6-stage/5 th (4 th)-order Dormand-Prince EPERK ³⁴	0.64x
853	= 12-stage/8 th (5 th)-order Dormand-Prince EPERK ³⁵	0.61x
1054	= 10-stage/5 th (4 th)-order EPERK FSAL ¹⁰	0.85x
-1054	= 10-stage/5 th (4 th)-order EPERK non-FSAL ¹⁰	n/a

Acronyms used in the description above:

SSP : Strong stability preserving

VOERK : Vortex-optimized explicit Runge-Kutta (ERK)

AOERK : Advection-optimized ERK

FSAL : First-same-as-last

EPERK : Embedded-pair ERK with adaptive time stepping

NOTE: EPERK methods do not work with limiting or filtering.

NOTE: The value in parenthesis for EPERK methods is the order of the embedded method. The difference between the actual method and embedded method represents the time-error from the time step. This is used by the adaptive time stepping proportional–integral–derivative (PID) controller to compute the size of the next time step.

Num_Timesteps_Min = 100

The minimum number of time steps to complete before checking whether any of the convergence criteria have been met.

NOTE: When initializing a simulation to a uniform flow, the very small changes that occur during the first few time steps can trigger a false positive convergence check if this is set to a small number, e.g., ≤ 5 .

Num_Timesteps = 25000

Number of time steps for the simulation to run, unless **Final.Time** is reached before completing all time steps.

Final_Time = -10.0 ! non-dim time, t*

Stops the simulation when the time within the simulation reaches this final time value. This value is specified in nondimensional units (t^*), Eq. (80), based on the input provided for **Time.Scaling.Factor**. For the isentropic vortex problem (**Initialization**=5 or 55), a negative value specifies the total number of periods to run the simulation. For all other cases, a negative value disables the use of **Final.Time** to stop a simulation.

NOTE: The nondimensional simulation time (t^*), Eq. (80), is written to standard output with the convergence information and to the residual file.

`Timestep_Type = 1`

This determines the method used to compute the time step size and the type of time stepping to use for the simulation (unless using one of the EPERK methods for adaptive time stepping, see note below).

- 2 = Local time step from the inviscid eigenvalue. (Experimental)
- 1 = Local time step from the spectral radius.
- 0 = Global time step from user specified time step.
- 1 = Global time step from the spectral radius.
- 2 = Global time step from the inviscid eigenvalue. (Experimental)

Local time stepping:

At the beginning of each solver iteration, each grid cell computes its own unique time step size that is used to update the solution within the cell at the end of each Runge-Kutta stage. Since all cells are updated with different time step sizes, this can only be used for steady-state simulations and cannot be used for time-accurate simulations (e.g., LES) which require all cells to be synchronized in time.

Global time stepping:

At the beginning of each solver iteration, the time step for each cell is computed in the same way as with local time stepping. However, the minimum time step size across all grid cells is taken to be the global time step size that all cells use to update their solutions in time. Global time stepping is required for time-accurate simulation like LES, but is generally inefficient for steady-state problems.

Inviscid eigenvalue:

The maximum inviscid eigenvalue is used to compute a time step size at each solution point within a cell, and the minimum of these values is taken to be the time step for the cell.

Spectral radius:

The time step within each cell is taken as the minimum value of the spectral radius evaluated at every face/flux point of the cell.

User specified time step:

A constant global time step is obtained from the user input for **Timestep**. **NOTE:** The input **Timestep** is specified as a nondimensional value. It is the physical time step nondimensionalized by the reference speed of sound and a length of 1 meter.

NOTE: For each of the methods above, once the time step is determined, the effective Δt will be computed as

$$\Delta t = CFL \times \text{time step} \quad (79)$$

where CFL is obtained from **CFL** or from **CFL_Ramping_Beg/End** if CFL ramping is being used. If **Timestep_Type**=0 (user specified time step), then **CFL** will be set to 1.0, however ramping may still be used. See **Timestep**.

NOTE: If using one of the EPERK methods for adaptive time stepping, the CFL column written to **stdout** for each iteration will contain an approximate CFL value, CFL_{EPERK} , for the adaptive time step size relative to the time step size that would have been computed for one of the CFL-based ERK methods, i.e.,

$$CFL_{\text{EPERK}} \approx \frac{\Delta t_{\text{EPERK}}}{\Delta t_{\text{ERK}}}$$

where Δt_{EPERK} is the time step size computed by the PID controller for the EPERK method, and Δt_{ERK} is the time step size that would have been computed using the method specified by `Timestep_Type` for a CFL-based ERK method with $\text{CFL} = 1.0$.

```
Timestep = 1.0e-6 ! t[s] * (a[m/s] / L=1.0[m])
```

Specifies the size of the time step that is used when `Timestep_Type=0` (constant global time step). The value specified by this input is nondimensionalized by L/a_∞ , the reference length (1 m) and speed of sound. Thus, the dimensional time step size will be `Timestep`* L/a_∞ seconds.

When a simulation is running, the nondimensional time step size is the second column written to stdout, so the value of this input is identical to that output under the conditions `Timestep_Type=0` and `CFL=1.0`. When using `Timestep_Type=0` (constant global time step), the value used for `CFL` is 1.0 unless CFL ramping is being used. If CFL ramping is being used, the ramping parameters will be used to evaluate the value of `CFL` to be used for each time step, but `CFL` will revert to a value of 1.0 once the ramping completes.

NOTE: This nondimensional time step is written to standard output with the convergence information. The `Timestep` is nondimensionalized by the reference length (1 m) and speed of sound, whereas the solution time that is written to standard output and `Final_Time` are both nondimensionalized by `Time_Scaling_Factor` via Eq. (80).

```
CFL = 1.0
```

Analogous to the CFL condition, this is used to scale the time step size to get a maximum stable time step. The maximum value this can be while also keeping a simulation stable is very dependent on the type of simulation, flow conditions, and input settings. Experimentation is generally required to find a good value. A lower value is more likely to keep a simulation stable, but will require more time steps to reach either convergence or a given simulation time. Typical values are $0.1 \leq \text{CFL} \leq 2$, but past experience has found simulations that were stable with values up to 10.0 for certain Runge-Kutta schemes.

If CFL ramping is actively being used, that computed value will be applied. Once CFL ramping has completed, the value of this input will be used.

```
CFL_Ramping_Beg = 0.1
CFL_Ramping_End = 1.0
```

Beginning and ending CFL values to use with the linear CFL ramping that is active when `CFL_Ramping_Iterations` ≥ 2 .

See also Section 10.10.2 for adjusting the CFL while the solver is running.

```
CFL_Ramping_Iterations = 1
```

The number of time steps used to linearly ramp the CFL value, starting at `CFL_Ramping_Beg` and ending at `CFL_Ramping_End`. `CFL_Ramping_Iterations` ≥ 2 activates CFL ramping.

NOTE: A new restart file will be written just before the ramping schedule begins in case it is too aggressive and causes the simulation to fail.

See also Section 10.10.2 for adjusting the CFL while the solver is running.

```
CFL_Ramping_Start_Iter = 0
```

Specifies the time step iteration at which to begin the CFL ramping schedule specified by `CFL_Ramping_Beg`, `CFL_Ramping_End`, and `CFL_Ramping_Iterations`. This input is primarily

used with the ADJUST_CFL control file (Section 10.10.2) that can be used to modify the CFL ramping schedule of a running simulation.

NOTE: A new restart file will be written just before the ramping schedule begins in case it is too aggressive and causes the simulation to fail.

See also Section 10.10.2 for adjusting the CFL while the solver is running.

<code>Convergence_Order_Abs_Res = -16.0</code>
--

This is the convergence goal for the orders of magnitude of the absolute residual.

<code>Convergence_Order_Max_Res = -16.0</code>
--

This is the convergence goal for the orders of magnitude reduction of the residual relative to the maximum value of the residual during the simulation.

11.7 nml_bc

This **namelist block** is used to specify boundary conditions.

```
&nml_bc
Walls_Are_Exact                = T
Sub_Inflow_Method              = 4
Use_BC_Conflicts               = T
BC_Input(1)%Name               = ""
BC_Input(1)%Set_BC_Value       = -9999
BC_Input(1)%Temperature        = -1.0           ! [K]
BC_Input(1)%Pressure           = -1.0           ! [Pa]
BC_Input(1)%Density            = -1.0           ! [kg/m^3]
BC_Input(1)%Mach               = -1.0
BC_Input(1)%VelocityX          = 0.0           ! [m/s]
BC_Input(1)%VelocityY          = 0.0           ! [m/s]
BC_Input(1)%VelocityZ          = 0.0           ! [m/s]
BC_Input(1)%Total_Temperature  = -1.0           ! [K]
BC_Input(1)%Total_Pressure     = -1.0           ! [Pa]
BC_Input(1)%Total_Density      = -1.0           ! [kg/m^3]
BC_Input(1)%AoA_Alpha          = 0.0           ! [deg]
BC_Input(1)%AoA_Beta           = 0.0           ! [deg]
BC_Input(1)%Wall_Temp          = -1.0           ! [K]
BC_Input(1)%Rotation_Rate(1:3) = [0.0,0.0,0.0]   ! [rpm]
BC_Input(1)%Is_Specified_In_MRF = F
BC_Input(1)%Mass_Frac(1:7)     = [7*0.0]
BC_Input(1)%Species_Name(1:7) = [7*"" ]
BC_Input(1)%Relax%Value        = 1.0
BC_Input(1)%Relax%Time         = 0.0           ! non-dim
BC_Input(1)%Relax%Iter         = 0
BC_Input(1)%SEM%Profile_File   = ""
BC_Input(1)%SEM%Is_Cylindrical = F
BC_Input(1)%SEM%Length_Scale   = 0.0           ! [m]
BC_Input(1)%SEM%Convective_Velocity = 0.0       ! [m/s]
BC_Input(1)%SEM%Profile_Height = 0.0           ! [m]
BC_Input(1)%SEM%Sub_In_BC_Grp  = 0
BC_Input(1)%SEM%Sub_In_BC_Idx  = 0
BC_Input(1)%Periodic%Matching_BC_Name = ""
BC_Input(1)%Periodic%Matching_BC_Idx = 0
BC_Input(1)%Periodic%RotationCenter(1:3) = [0.0,0.0,0.0] ! grid units
BC_Input(1)%Periodic%RotationAngle(1:3) = [0.0,0.0,0.0] ! [deg]
BC_Input(1)%Periodic%RotationAxis(1:4) = [0.0,0.0,0.0,0.0]
BC_Input(1)%Periodic%Translation(1:3) = [0.0,0.0,0.0]   ! grid units
/
```

Listing 12: Namelist nml_bc.

```
Walls_Are_Exact = T
```

Specifies whether the wall boundary conditions are set directly using the exact flow conditions or indirectly using the ghost state that provides the exact flow conditions when averaged with the interior state.

T = The wall boundary solution is the exact wall boundary condition,
e.g., $[u_{\text{wall}}, v_{\text{wall}}, w_{\text{wall}}] = 0$ for a no-slip wall.

F = The average of the wall boundary solution and the interior solution
on the boundary face recovers the exact wall boundary condition,
e.g., $[u_{\text{wall}}, v_{\text{wall}}, w_{\text{wall}}] = -[u_{\text{wall}}, v_{\text{wall}}, w_{\text{wall}}]$ for a no-slip wall.

NOTE: Hartmann and Leicht⁵⁸ report (Remark 6.13 on page 83) that using exact wall boundary conditions seems to be more accurate on coarse grids and for lower values of `Solution_Order`, whereas using the averaging method is more stable.

```
Sub_Inflow_Method = 4
```

Specifies which subsonic inflow boundary condition algorithm to use.

1 = Hold the total pressure and total temperature constant at the inflow, use the outgoing characteristic from the interior, and perform a Newton iteration to get the static temperature, speed of sound, and normal velocity magnitude for the exterior state.

2 = Use the BC static density and velocity, and the interior static pressure.

3 = Use the method specified in Ref 26. The interior static pressure, total enthalpy, and outward propagating Riemann invariant are extrapolated from the interior to the boundary. These values with the BC total pressure and total temperature are used to compute the BC speed of sound by solving a quadratic equation based on adiabatic and isentropic assumptions/relations.

4 = Use the BC total pressure, total temperature, angle of attack, and the interior state to get the interior Mach number at the boundary.

See Section 7.2.2 for a detailed description of how each of these methods is implemented.

NOTE: It is recommended to use options 4, 3, or 1 (in that order).

```
Use_BC_Conflicts = T
```

Logical flag for determining whether or not to overwrite boundary solutions in adjacent cells if co-located boundary points have different boundary conditions. The boundary preference hierarchy can be found in `kind_types_mod.f90` in the array `bc_hierarchy`.

NOTE: This is only relevant if using Legendre-Gauss-Lobatto points for solution points (`Loc_Solution_Pts = 2`).

```
BC_Input(1)%Name = ""
```

Character string that matches the name of a boundary group in the grid file. See Section 10.2.1.1 for using Pointwise to set boundary conditions for CGNS files. This input is the means of connecting boundary faces in the grid file with a boundary condition provided in the input file to specify the boundary type and flow conditions to use.

NOTE: The character string is treated as case insensitive.

```
BC_Input(1)%Set_BC_Value = -9999
```

This overrides the boundary condition type specified in the grid file, using this boundary condition type for the boundary group given by `BC_Input(1)%Name`. Valid values for boundary condition types are:

Inflow boundary conditions

- 1 Generic inflow, subsonic/supersonic
- 2 Subsonic inflow
- 3 Supersonic inflow
- 5 SEM (unsteady turbulent) inflow*

Outflow boundary conditions

- 11 Generic outflow, subsonic/supersonic
- 12 Subsonic outflow
- 13 Supersonic outflow

Farfield boundary conditions

- 21 Characteristic inflow/outflow based on Riemann invariants, subsonic/supersonic*
- 22 Generic inflow/outflow, subsonic/supersonic
- 23 Fixed to freestream/reference flow conditions
- 24 Fixed flow conditions (equivalent to supersonic inflow)

Wall boundary conditions

- 50 Slip wall
- 52 Adiabatic no-slip wall
- 53 Isothermal no-slip wall

Numerical boundary conditions

- 80 Symmetry boundary condition (equivalent to slip wall)
- 81 Periodic (translated and/or rotated) coupled boundaries

Other boundary conditions

- 9999 Do not override the boundary condition type from the grid file.

* Not compatible with multi-component gas mixtures.

```
BC_Input(1)%Temperature = -1.0 ! [K]
```

Static temperature (units of K).

```
BC_Input(1)%Pressure = -1.0 ! [Pa]
```

Static pressure (units of Pa).

```
BC_Input(1)%Density = -1.0 ! [kg/m^3]
```

Static density (units of kg/m³).

```
BC_Input(1)%Mach = -1.0
```

Mach number (dimensionless).

```
BC_Input(1)%VelocityX = 0.0 ! [m/s]
```

Velocity in the x coordinate direction (units of m/s).

```
BC_Input(1)%VelocityY = 0.0 ! [m/s]
```

Velocity in the y coordinate direction (units of m/s).

```
BC_Input(1)%VelocityZ = 0.0 ! [m/s]
```

Velocity in the z coordinate direction (units of m/s).

```
BC_Input(1)%Total_Temperature = -1.0 ! [K]
```

Total temperature (units of K).

```
BC_Input(1)%Total_Pressure = -1.0 ! [Pa]
```

Total pressure (units of Pa).

```
BC_Input(1)%Total_Density = -1.0 ! [kg/m^3]
```

Total density (units of kg/m³).

```
BC_Input(1)%AoA_Alpha = 0.0 ! [deg]
BC_Input(1)%AoA_Beta = 0.0 ! [deg]
```

Flow angles (in degrees): α is the **angle of attack** measured in the x - y plane, and β is the **angle of sideslip** measured in the x - z plane.

These flow angles are defined by Eqs. (46)–(48) and Fig. 13 on page 47.

```
BC_Input(1)%Wall_Temp = -1.0 ! [K]
```

Temperature for isothermal wall (units of K).

```
BC_Input(1)%Rotation_Rate(1:3) = [0.0,0.0,0.0] ! [rpm] @(x,y,z)-axes
```

This specifies the rotation rate vector of a moving boundary surface. The components are defined with respect to the Cartesian axes, and the magnitude is the rotation rate (in RPMs).

```
BC_Input(1)%Is_Specified_In_MRF = F
```

Setting this to true indicates that this boundary condition is specified in the moving reference frame. If the boundary is moving/rotating at the same speed as the moving reference frame, this works as a shortcut to apply the same motion to this boundary.

```
BC_Input(1)%Mass_Frac(1:7) = [7*0.0]
```

`BC_Input(i)%Mass_Frac(n)` specifies the mass fraction of the species given in `BC_Input(i)%Species_Name(n)`.

```
BC_Input(1)%Species_Name(1:7) = [7*""]
```

Specifies the chemical name/formula for each of the species.

11.7.1 Boundary Condition Relaxation Inputs

```
BC_Input(1)%Relax%Value = 1.0
BC_Input(1)%Relax%Time   = 0.0 ! non-dim, t[s] * (a[m/s] / L=1.0[m])
BC_Input(1)%Relax%Iter   = 0
```

Typically the volume solution is initialized from the reference conditions specified in the `nml.reference` namelist. If boundary conditions are immediately applied, the sudden change can introduce numerical instabilities. These relaxation parameters provide a way to gradually apply each boundary condition.

`BC_Input(i)%Relax%Value` specifies the relaxation value (between 0.0 and 1.0) to limit the difference between the interior and exterior states. The default value of 1.0 does not impose any limitation.

There are two means for specifying when to end relaxation: one based on solution time, the other based on solution iterations. Relaxation will end when either criteria is met.

If `BC_Input(i)%Relax%Time > 0.0`, it specifies the total accumulated solution time (nondimensionalized by the reference length of 1.0 meter and speed of sound) to end relaxation.

If `BC_Input(i)%Relax%Iter > 0`, it specifies the total accumulated time step iteration to end relaxation.

11.7.2 Boundary Condition SEM Inputs

```
BC_Input(1)%SEM%Profile_File = "" ! 150 char max
```

This gives the path to the file containing the flow profile used for the synthetic eddy method (SEM) boundary condition. See Section 5.4.2.3 for a detailed description of the required file format and flow variables needed for the SEM profile.

```
BC_Input(1)%SEM%Is_Cylindrical = F
```

This is used to declare if the synthetic eddy method (SEM) box is cylindrical. This solely affects how the eddies are randomly generated to achieve a truly random distribution in the radial, r , direction. For a cylindrical SEM box, randomly generating r ($r_{\min} \leq r \leq r_{\max}$) and θ ($\theta_{\min} \leq \theta \leq \theta_{\max}$) coordinate values to place a new eddy will result in more eddies near the symmetry axis and less eddies further away. In order to generate a truly uniform random distribution, we need to use the equation $r = \sqrt{\text{Random}() [r_{\max}^2 - r_{\min}^2] + r_{\min}^2}$ where `Random()` is a function that generates a random number on the unit interval. This applies an area weighting to the radial distribution and results in a uniform distribution of eddies in the volume.

```
BC_Input(1)%SEM%Length_Scale = 0.0 ! [m]
```

This gives the length scale (in meters) of the “zone of influence” of each synthetic eddy. Recommended values are between $(1/3.5)\times$ and $(1/5)\times$ the height of the boundary layer profile from the SEM profile file. See Section 5.4.2.6 for a more detailed description of this input and how it relates to `SEM%Convective_Velocity`.

NOTE: Previous testing seems to suggest that a slower convective velocity should be used when using a smaller length scale, and a higher convective velocity should be used when using a larger length scale.

```
BC_Input(1)%SEM%Convective_Velocity = 0.0 ! [m/s]
```

This gives the velocity (in m/s) at which the synthetic eddies are convected through the SEM box. Recommended values are between $0.25\times$ and $1\times$ the freestream velocity for this BC. See Section 5.4.2.6 for a more detailed description of this input and how it relates to `SEM%Length_Scale`.

NOTE: Previous testing seems to suggest that a slower convective velocity should be used when using a smaller length scale, and a higher convective velocity should be used when using a larger length scale.

```
BC_Input(1)%SEM%Profile_Height = 0.0 ! [m]
```

NOTE: *This variable is not an input.* It will be calculated based on the profile data read from `SEM%Profile_File`, and will be reported with the boundary condition information written to the display output. This output can be used to verify that the profile data was properly interpreted.

```
BC_Input(1)%SEM%Sub_In_BC_Grp = 0  
BC_Input(1)%SEM%Sub_In_BC_Idx = 0
```

NOTE: *Theses variables are not inputs.* They will be determined after setting up the SEM boundary condition and will be reported with the boundary condition information written to the display output. This output can be used to verify that the profile data was properly interpreted.

If the boundary layer profile specified in the `SEM%Profile_File` does not fully span the boundary domain, then a new inflow BC will automatically be created for the boundary points that are outside the SEM profile. `Sub_In_BC_Grp` identifies the *index of the BC group* to which the new inflow BC created for this SEM belongs. `Sub_In_BC_Idx` identifies the *index for the new inflow BC* created for this SEM.

NOTE: Inflow boundaries are automatically split in this way because the synthetic eddy method of imposing inflow turbulence actually overprescribes flow conditions on subsonic boundaries. Separating points outside the boundary layer profile into a standard inflow boundary allows those points to properly communicate information upstream as needed. Therefore, we recommend that the SEM profile file not extend much beyond the edge of the boundary layer. See Section 5.4.2.1 for a more detailed explanation.

11.7.3 Boundary Condition Periodicity Inputs

```
BC_Input(1)%Periodic%Matching_BC_Name = ""
```

This specifies the name of the other boundary condition that is a matching periodic pair with this boundary condition. If BC's i and j are a periodic pair, then the following two relations need to be true:

`BC_Input(i)%Periodic%Matching_BC_Name = BC_Input(j)%Name`, and
`BC_Input(j)%Periodic%Matching_BC_Name = BC_Input(i)%Name`.

NOTE: This and `Periodic%Matching_BC_Idx` are redundant, but together they serve as a means of error checking the inputs for periodic pairs.

```
BC_Input(1)%Periodic%Matching_BC_Idx = 0
```

This specifies the index of the other boundary condition that is a matching periodic pair with this boundary condition. If BC's i and j are a periodic pair, then the following two relations need to be true:

BC_Input(i)%Periodic%Matching_BC_Idx = j , and
BC_Input(j)%Periodic%Matching_BC_Idx = i .

NOTE: This and **Periodic%Matching_BC_Name** are redundant, but together they serve as a means of error checking the inputs for periodic pairs.

```
BC_Input(1)%Periodic%RotationCenter(1:3) = [0.0,0.0,0.0] ! [grid units]
```

This array gives the coordinate origin (in **grid units**) used for defining the rotation angles between rotated periodic boundary conditions.

NOTE: This input should be the same for a matching pair of rotated periodic boundary conditions.

```
BC_Input(1)%Periodic%RotationAngle(1:3) = [0.0,0.0,0.0] ! [deg]
```

This array gives the rotation angles (in degrees) about each of the coordinate axes from the current periodic boundary interface to the connecting periodic boundary interface. If rotating about more than one axis, the rotation is performed first about the x -axis, then the y -axis, and finally the z -axis.

NOTE: One may use either **Periodic%RotationAngle** or **Periodic%RotationAxis** to define rotated periodic boundary conditions, but not both. If both are specified, **RotationAxis** will be ignored and **RotationAngle** will be used.

NOTE: If BC's i and j are a periodic pair, the following relation needs to be true:

BC_Input(i)%Periodic%RotationAngle(:) = -BC_Input(j)%Periodic%RotationAngle(:)

```
BC_Input(1)%Periodic%RotationAxis(1:4) = [0.0,0.0,0.0,0.0]
```

The first 3 values of this array define a vector representing the axis of rotation for this rotated periodic boundary condition, and the last value of the array defines the rotation angle (in degrees) about this axis.

NOTE: The coordinate origin is used as the base of the vector defining the axis of rotation, NOT **Periodic%RotationCenter**.

NOTE: The magnitude of the vector defining the axis of rotation is not used. The sole purpose of this vector is to define the *direction* of the rotation axis.

NOTE: One may use either **Periodic%RotationAngle** or **Periodic%RotationAxis** to define rotated periodic boundary conditions, but not both. If both are specified, **RotationAxis** will be ignored and **RotationAngle** will be used.

NOTE: If BC's i and j are a periodic pair, one of the two following sets of relations needs to be true:

BC_Input(i)%Periodic%RotationAxis(1:3) = BC_Input(j)%Periodic%RotationAxis(1:3)
BC_Input(i)%Periodic%RotationAxis(4) = -BC_Input(j)%Periodic%RotationAxis(4)

or

BC_Input(i)%Periodic%RotationAxis(1:3) = -BC_Input(j)%Periodic%RotationAxis(1:3)
BC_Input(i)%Periodic%RotationAxis(4) = BC_Input(j)%Periodic%RotationAxis(4)

In other words, the opposing periodic face must either negate the axis direction or the rotation angle, but not both.

```
BC_Input(1)%Periodic%Translation(1:3) = [0.0,0.0,0.0] ! [grid units]
```

This array gives the translation vector (in **grid units**) from the current periodic boundary interface to the connecting periodic boundary interface.

NOTE: If BC's i and j are a periodic pair, the following relation needs to be true:

```
BC_Input( $i$ )%Periodic%Translation(:) = -BC_Input( $j$ )%Periodic%Translation(:)
```

In other words, the opposing periodic face must negate the translation vector.

11.8 nml_numerics

This **namelist block** is used to specify parameters for the numerical method.

```
&nml_numerics
  All_Orders              = -3
  Solution_Order          = 3
  Projection_Order        = -3
  Quadrature_Order        = -3
  Error_Order             = -3
  All_Points              = -1
  Loc_Solution_Pts        = 1
  Loc_Flux_Pts            = 1
  Loc_Quadrature_Pts      = 1
  Loc_Error_Pts           = 1
  Loc_Triangle_Pts        = 1
  Loc_Tetrahedra_Pts      = 1
  Invs_Flux_Method        = 1
  Visc_Flux_Method        = 2
  Visc_Variables_To_Ave   = 1
  Combine_Common_Flux_Projections = 0
  Correction_Function      = 1
/
```

Listing 13: Namelist nml_numerics.

All_Orders = -3

If > 0 , sets **Solution_Order**, **Projection_Order**, and **Quadrature_Order** all to this value.

NOTE: This overrides the values given for these three input parameters.

Solution_Order = 3

Specifies the degree of the polynomial space, $\mathcal{P}_S$, to use for the solution. The order-of-accuracy is approximately $\mathcal{P}_S + 1$.

NOTE: If **All_Orders** > 0 , then that value will override this input.

Projection_Order = -3

Specifies the degree of the polynomial space, $\mathcal{P}_P$, to project the residual onto when using over-integration by residual projection. The residual will be computed as a polynomial of degree $\mathcal{P}_S$ and projected down to a polynomial of degree $\mathcal{P}_P$. This can potentially improve stability if aliasing errors are suspected of destabilizing a simulation. If **Projection_Order** ≤ 0 , this will be set to the same value as **Solution_Order**.

NOTE: If **All_Orders** > 0 , then that value will override this input.

NOTE: This is only used when $0 < \text{Projection_Order} < \text{Solution_Order}$, and using this means the order-of-accuracy of the simulation is now approximately $\mathcal{P}_P + 1$ instead of $\mathcal{P}_S + 1$.

Quadrature_Order = -3

Specifies the degree of the quadrature polynomial space, $\mathcal{P}_Q$, to use when using over-integration by over-sampling at the quadrature points. If `Quadrature_Order` ≤ 0 , this will be set to the same value as `Solution_Order`.

Certain operations during a given residual evaluation (e.g., evaluating the common/upwind interface fluxes) produce either polynomials that are not within the solution polynomial space (e.g., of degree greater than the solution order) or functions that are not polynomials at all (e.g., pressure). By setting `Quadrature_Order` $>$ `Solution_Order`, these functions will be over-sampled at the quadrature points to produce approximations of these functions as polynomials of degree $\mathcal{P}_Q$, which are in turn projected down to polynomials of degree $\mathcal{P}_S$ before adding their respective contribution to the residual. This polynomial representation of degree $\mathcal{P}_S$ is generally more accurate than directly evaluating the function at the $\mathcal{P}_S$ polynomial points, and can potentially improve stability if aliasing errors are suspected of destabilizing a simulation. See Section 9.1 for an example of how aliasing error arises and the process of using over-integration by over-sampling at the quadrature points.

NOTE: If `All_Orders` > 0 , then that value will have first priority.

NOTE: If `Solution_Order` $>$ `Quadrature_Order`, then `Quadrature_Order` will be set to `Solution_Order`.

NOTE: This quadrature option is only used when `Quadrature_Order` $>$ `Solution_Order` or `Loc_Quadrature_Pts` $\neq$ `Loc_Solution_Pts`, otherwise the quadrature is evaluated directly at the solution points without the need for projection. In all cases, `Quadrature_Order` $\geq$ `Solution_Order` because using a lower quadrature order provides poor results. Unlike over-integration by residual projection, the order-of-accuracy of the simulation remains $\mathcal{P}_S + 1$.

NOTE: This quadrature option has not been working since the viscous and unstructured capability was added to the code. It should be fixed in an upcoming version.

Error_Order = -3

Specifies the degree of the polynomial space to use when computing integrated error quantities, e.g., the L^2 -norm of the residual. If `Error_Order` $<$ `Solution_Order`, this will be set to the same value as `Solution_Order`.

NOTE: This input is only used when `Error_Order` $>$ `Solution_Order` or `Loc_Error_Pts` $\neq$ `Loc_Solution_Pts`.

All_Points = -1

Valid values for this input are the same as those specified in `Loc_Solution_Pts`. If set to one of those values, this will set `Loc_Solution_Pts`, `Loc_Flux_Pts`, and `Loc_Quadrature_Pts` all to this value; otherwise, this input will be ignored and the three mentioned input parameters are unchanged.

NOTE: This will override the values given for the three mentioned input parameters set to a valid value.

Loc_Solution_Pts = 1

Specifies the location of the 1D solution points that are used in tensor products to create higher dimensional sets of solution points (used for edge, quad, pyramid, prism, and hex cell geometries):

- 0 = Equidistant points (not recommended)
- 1 = Legendre-Gauss (Gauss) points (recommended)
- 2 = Legendre-Gauss-Lobatto (Lobatto) points

See Fig. 3 in Section 2.2 to better understand the differences in these point locations.

NOTE:

For structured elements (quadrilaterals and hexahedra), a tensor product of these 1D solution points is used.

For pyramid elements, a special tensor product of these solution points is used. The 1D solution point locations are used to space the points vertically from the pyramid base to the peak. At each of these vertical locations, a 2D tensor product of solution points is used, with the base being a tensor product of degree $\mathcal{P}$, the next vertical location $\mathcal{P} - 1$, with the peak being $\mathcal{P}0$ or a single point.

For prism elements, a tensor product of these 1D solution points and the specified triangle points is used.

Loc_Flux_Pts = 1

Specifies the location of the flux points for edge and quadrilateral faces.

- 0 = Equidistant points (not recommended)
- 1 = Legendre-Gauss (Gauss) points (recommended)
- 2 = Legendre-Gauss-Lobatto (Lobatto) points

For triangle faces, **Loc_Triangle_Pts** is used for the flux points.

NOTE: Using different types of points for the solution and flux points is untested; therefore, it is recommended to use the input **All_Points** to use the same set of points for solution, flux, and quadrature points.

Loc_Quadrature_Pts = 1

Specifies the location of the quadrature points used for over-integration by over-sampling at the quadrature points. This input is treated identically to **Loc_Solution_Pts** except it uses **Quadrature_Order** to determine the number of quadrature points for each geometry type.

Loc_Error_Pts = 1

Specifies the location of the error points used for computing integrated error quantities. This input is treated identically to **Loc_Solution_Pts** except it uses **Error_Order** to determine the number of error points for each geometry type.

Loc_Triangle_Pts = 1

Specifies the location of the solution points used for triangles.

- 2 = Hesthaven-Warburton α -optimized points³⁹
- 0 = Equally spaced points (not recommended)
- 1 = Williams-Shunn points⁴⁶ (recommended)
- 2 = Barycentric Lobatto points

Loc_Tetrahedra_Pts = 1

Specifies the location of the solution points used for tetrahedra.

- 2 = Hesthaven-Warburton α -optimized points³⁹
- 0 = Equally spaced points (not recommended)
- 1 = Shunn-Ham points⁵⁹ (recommended)
- 2 = Barycentric Lobatto points

Invs_Flux_Method = 1

Specifies the approximate Riemann solver used to compute the common inviscid fluxes at the interfaces.

- 1 = Roe flux¹³ with entropy fix¹⁴
- 2 = HLLC flux⁶⁰
- 3 = LDFSS flux⁶¹
- 4 = Rotated-Roe-HLL flux⁶² (2D and 3D only)
- 5 = AUSM+ flux⁶³
- 6 = Rusanov flux⁶⁴

NOTE: Only the Roe flux with entropy fix or the Rusanov flux are supported when using a rotating reference frame. If any other option is selected with a rotating reference frame, the inviscid flux method will be reset to the Roe flux.

Visc_Flux_Method = 2

Specifies the method used to compute the common viscous fluxes at the interfaces.

- 1 = Bassi-Rebay 1 (BR1)⁶⁵ (high chances of stability issues)
- 2 = Bassi-Rebay 2 (BR2)⁶⁶
- 3 = Local DG (LDG)⁶⁷ (smaller CFL limit than BR2)

Visc_Variables_To_Ave = 1

Determines which variables are averaged to the interfaces when computing the common viscous fluxes.

- 1 = Average the interface fluxes. Compute the common viscous flux by averaging the left and right viscous interface fluxes which are computed using the left and right interface solutions and gradients.
- 2 = Average the interface solutions and gradients. Compute common solutions and gradients by averaging the left and right interface solutions and gradients, and then use the common interface values to compute the common viscous flux.

Combine_Common_Flux_Projections = 0

Specifies if the **common flux** projections are combined when using over-integration by over-sampling at the quadrature points.

- 0 = All projections of the over-sampled common fluxes are performed separately.
- 1 = The projections of the over-sampled interior-face common fluxes are performed separately from the projection of the over-sampled boundary-face common fluxes.
- 2 = All of the over-sampled common fluxes are computed and combined before projecting.

<code>Correction_Function = 1</code>

Specifies the **correction function** (Section 2.1.6) to use when computing the **continuous flux function**.

1 = g_{DG} correction function (recovers nodal DG method)

2 = g_2 correction function (only works for 1D)

3 = g_{Ga} correction function (only works for 1D)

11.9 nml_limiter

This **namelist block** is used to specify limiter, filter, and marker controls.

```
&nml_limiter
  Global_Filter_Interval      = 0
  Global_Filter_Option       = 0
  Global_Filter_Order        = 16
  Global_Filter_Alpha        = 36.0
  Global_Filter_Etac         = 0.0
  Sponge_Filter_Interval     = 0
  Sponge_Filter_Option       = 0
  Sponge_Filter_Order        = 8
  Sponge_Filter_Alpha        = 36.0
  Sponge_Filter_Etac         = 0.0
  Sponge_Filter_X_Start      = huge(0.0) ! [input grid units]
  Local_Limiter_Interval     = 0
  Local_Limiter_Option       = 0
  Local_Filter_Order         = 16
  Local_Filter_Alpha         = 36.0
  Local_Filter_Etac          = 0.0
  Marker_Option              = 0
  Marker_Variable            = 0
  Marker_Min                 = 1.0
  Marker_Max                 = 1.0
  Marker_Dpsi                = -1.0
  Marker_Stored_Interval    = 1
  Thermofix%Method          = 0
  Thermofix%Iterations       = 10
  Thermofix%Check_Cells      = T
  Thermofix%Check_Faces      = T
  Thermofix%Check_Nodes      = T
  Thermofix%Verbose          = T
  Thermofix%Scale_Factor     = 0.5
  Thermofix%Min_Pressure     = 10000. ! [Pa]
  Thermofix%Min_Temperature  = 100. ! [K]
  Thermofix%Min_Density      = 0.001 ! [kg/m^3]
/
```

Listing 14: Namelist nml_limiter.

Global_Filter_Interval = 0

Specifies the frequency to apply the global filter.

≤ 0 Apply the global filter after each Runge-Kutta stage.

> 0 Apply the global filter after the last Runge-Kutta stage if
 $\text{mod}(\text{current_iteration}, \text{Global_Filter_Interval}) = 0$.

The *current_iteration* is the total aggregate iteration number including previous jobs.

Global_Filter_Option = 0

Specifies whether the global filter is used for stabilization. This filter is denoted as a global filter because it is used on all cells regardless of the quality of the solution in each cell. The available options are as follows:

0 = Disable the global filter.

1 = Use the exponential filter, Eq. (69), where the polynomial order is given by $\mathcal{P} = \text{Solution_Order}$ for 1D, 2D, and 3D.

GFR uses a three-pass filtering approach involving global, local, and sponge filters. Each of them and their sequence of application is described in Section 9.2. Generally, we recommend that the global filter not be used, because it can introduce significant dissipation. When it is absolutely necessary to use the global filter, the filter settings should be set as weak as possible.

Global_Filter_Order = 16

This is an integer value between ($2 \leq s \leq 32$) that specifies the exponent/order of the global exponential filter, Eq. (69). As discussed in Section 9.2 and shown in Fig. 15c, the filter order controls how rapidly the filter decays. Larger values act to concentrate the filtering toward the highest mode.

Because the global filter is applied to all cells regardless of the solution quality in each cell, one generally wants to use as large a value as possible so as not to affect the lower modes. Starting with **Global_Filter_Order** of 32 or 16 and halving if necessary is a good approach.

Global_Filter_Alpha = 36.0

This is a real value between ($0. < \alpha \leq 36.$) that specifies the damping parameter of the global exponential filter, Eq. (69). As discussed in Section 9.2, and shown in Fig. 15b, this parameter controls the level of damping applied to the highest mode; larger values apply more damping. A value of 36.0 effectively filters the highest mode to machine zero.

Because the global filter is applied to all cells regardless of the solution quality in each cell, one ideally wants to use as small a value as possible so as not to introduce unwanted dissipation. However, since numerical error often first appears in the highest modes it may be necessary to aggressively filter those modes before it contaminates the lower modes. Starting with **Global_Filter_Alpha** of 2 or 4 and doubling if necessary is a good approach.

In the special case of $\mathcal{P} = 1$, the $\mathcal{P}1$ mode will be multiplied by $e^{-\alpha}$.

Global_Filter_Etac = 0.0

This is a real value between ($0. \leq \eta_c \leq 1.$) that specifies the modal cutoff fraction for the global exponential filter, Eq. (69). As discussed in Section 9.2 and shown in Fig. 15a, modes at or below this cutoff fraction will be entirely preserved. A value of 1.0 applies no filter and wastes computational resources.

For most simulations it is desirable to preserve the linear mode, and so a good approach is to set this input value to $\eta_c = 1/\mathcal{P}$ where $\mathcal{P} = \text{Solution_Order}$. However, if $\mathcal{P} \leq 2$ it may be best to use a value of 0, and if $\mathcal{P} > 4$ it may be possible to use a larger fraction.

In the special case of $\mathcal{P} = 1$, any value < 1.0 will just multiply the $\mathcal{P}1$ mode by $e^{-\alpha}$.

```

Sponge_Filter_Interval = 0
Sponge_Filter_Option   = 0
Sponge_Filter_Order    = 8
Sponge_Filter_Alpha    = 36.0
Sponge_Filter_Etac     = 0.0
Sponge_Filter_X_Start  = huge(0.0) ! [input grid units]

```

GFR uses a three-pass filtering approach involving global, local, and sponge filters. Each of them and their sequence of application is described in Section 9.2. Setting `Sponge_Filter_Option=1` activates the sponge filter, which is ideally used as an extra-strong exponential filter to damp non-physical pressure reflections near the outflow boundary. It is applied to any cells where all the x -coordinate values (in **grid units**) are larger than that specified by `Sponge_Filter_X_Start`.

The details for these inputs are the same as for the similarly named `Global_Filter` options. The user must manually specify the starting x -location (in **grid units**) of the sponge region via `Sponge_Filter_X_Start`, since the default value of `huge(0.0)` will be beyond any current grid location.

NOTE: If `AoA_Alpha` = $\pm 90^\circ$ and `AoA_Beta` = 0° , the y -axis will be assumed as the axial coordinate direction, and this input will be applied to the y -coordinate. If `AoA_Alpha` = 0° and `AoA_Beta` = $\pm 90^\circ$, the z -axis will be assumed as the axial coordinate direction, and this input will be applied to the z -coordinate.

```

Local_Limiter_Interval = 0

```

Specifies the frequency to apply the local limiter.

- ≤ 0 Apply the limiter after each Runge-Kutta stage.
- > 0 Apply the limiter after the last Runge-Kutta stage if
 $\text{mod}(\text{current_iteration}, \text{Local_Limiter_Interval}) = 0$.

For example, if `Local_Limiter_Interval` = 1, the local limiter will be applied at the end of every time step. The *current_iteration* is the total aggregate iteration number including previous jobs.

```

Local_Limiter_Option = 0

```

GFR uses a three-pass filtering approach involving global, local, and sponge filters. Each of them and their sequence of application is described in Section 9.2. `Local_Limiter_Option` specifies the type of limiter to apply to locally marked cells.

- 0 = Do not apply any kind of limiter.
- 1 = Apply filtering.
- 2 = Apply scaled filtering. The filter value is scaled by the magnitude of the scaled marker.
- 3 = Apply the multi-dimensional limiting process (MLP).
- 9001 = Apply recursive projection. For any troubled cell that is marked, project the solution down one order; recheck that cell to see if it is still troubled; keep repeating this process until the cell is no longer marked or has been reduced to a $\mathcal{P}1$ solution.
- 9002 = Apply scaled projection. For any troubled cell that is marked, project the solution down to a order that is dynamically chosen by the value of the local marker. Cells with higher marker values will be projected to lower orders, whereas cells with lower marker values will be projected down only an order or two from the solution order.

NOTE: The locally marked cells are determined from the inputs for **Marker_Option** and **Marker_Variable**. The local filter settings are defined by input values for **Local_Filter_Order**, **Local_Filter_Alpha**, and **Local_Filter_Etac**.

Local_Filter_Order	= 16
Local_Filter_Alpha	= 36.0
Local_Filter_Etac	= 0.0

As discussed in Section 9.2, the local filter is applied to those cells identified by the troubled cell marker. A good approach is to use the local options to apply a stronger filter to only those cells that have been marked as a troubled. This will help prevent the oscillatory solutions in these cells from causing the simulation to diverge. Then, if necessary, a weaker global filter can be applied to all cells to damp any other disturbances.

The details for these inputs are the same as for the similarly named **Global_Filter** options, but one likely wants to apply a stronger local filter setting. Use **Local_Filter_Order**=16 and halve if necessary, and **Local_Filter_Alpha**=36 and halve if possible. For **Local_Filter_Etac**, use the same recommendations as for **Global_Filter_Etac**.

Marker_Stored_Interval	= 1
-------------------------------	-----

This determines if the marker values written to the CGNS solution files are from the final Runge-Kutta stage or from each Runge-Kutta stage.

- < 0 Store markers at the same frequency of **Local_Limiter_Interval**. This can be used to synchronize the marker storage and limiter application.
- = 0 Store markers after each Runge-Kutta stage.
- > 0 Store markers after the last Runge-Kutta stage.

NOTE: Storing markers each stage increases memory and file size (if **Marker_Variables** are being output).

Marker_Option	= 0
----------------------	-----

Specifies which type of local marker to use.

- 0 = Use the default marker option for the chosen **Local_Limiter_Option**.
- 1 = Use the resolution marker described in Section 9.3.1.
- 2 = Use the jump marker described in Section 9.3.2.
- 3 = Use the MLP marker described in Section 9.3.3.

Table 12 lists the permissible and default values for **Marker_Option**, depending on **Local_Limiter_Option**.

Table 12: Permissible values for **Marker_Option**.

	Local_Limiter_Option	Marker_Option
0	None	1*, 2, 3
1	Filtering	1*, 2, 3
2	Scaled Filtering	1*, 2, 3
3	MLP	3*
9001	Recursive Projection	1*, 2
9002	Scaled Projection	1*, 2

* indicates default value.

Marker_Variable = 0

This determines which variable is used to evaluate the local marker.

-1 = Use pressure.

0 = Use the default marker variable for the chosen **Marker_Option**.

1 = Use density.

2 = Use x -momentum.

3 = Use y -momentum.

4 = Use z -momentum.

5 = Use total energy.

6 = Use the max value after evaluating the marker for each conserved variable.

Table 13 lists the permissible and default values for **Marker_Variable**, depending on **Marker_Option**.

Table 13: Permissible values for **Marker_Variable**.

	Marker_Option	Marker_Variable
1	Resolution	1*, 2, 3, 4, 5, 6
2	Jump	-1*, 1, 2, 3, 4, 5
3	MLP	1*, 2, 3, 4, 5

* indicates default value.

NOTE: Using all conserved variables (option 6) will result in marking the most number of cells. This will increase stability, but has the tendency to add unnecessary dissipation where it is not needed. The default of using only density seems to result in the best mix of stability with minimal unnecessary dissipation.

Marker_Min = 1.0

This gives the value for $\Psi_{min} = \Psi_0 - \Delta\Psi$, which is the bottom-end of the s -curve used to scale the marker value from the interval $(-\infty, 0]$ to the interval $[0, 1]$. Any cell with a marker value less than the value of this input will have a scaled marker value of 0, which means that cell will receive no limiting. See the discussion of the scaled marker in Section 9.3.4 for a more information

Marker_Max = 1.0

This gives the value for $\Psi_{max} = \Psi_0 + \Delta\Psi$, which is the top-end of the s -curve used to scale the marker value from the interval $(-\infty, 0]$ to the interval $[0, 1]$. Any cell with a marker value greater than the value of this input will have a scaled marker value of 1, which means that cell will receive the maximum strength of the limiting method being employed. See the discussion of the scaled marker in Section 9.3.4 for a more information

Marker_Dpsi = -1.0

This gives the value for the constant $\Delta\Psi$ that is used to scale the marker value from the interval $(-\infty, 0]$ to the interval $[0, 1]$. See the discussion of the scaled marker in Section 9.3.4 for a more information

<code>Thermofix%Method</code>	<code>= 0</code>	
<code>Thermofix%Iterations</code>	<code>= 10</code>	
<code>Thermofix%Check_Cells</code>	<code>= T</code>	
<code>Thermofix%Check_Faces</code>	<code>= T</code>	
<code>Thermofix%Check_Nodes</code>	<code>= T</code>	
<code>Thermofix%Verbose</code>	<code>= T</code>	
<code>Thermofix%Scale_Factor</code>	<code>= 0.5</code>	
<code>Thermofix%Min_Pressure</code>	<code>= 10000.</code>	<code>! [Pa]</code>
<code>Thermofix%Min_Temperature</code>	<code>= 100.</code>	<code>! [K]</code>
<code>Thermofix%Min_Density</code>	<code>= 0.001</code>	<code>! [kg/m^3]</code>

Sometimes during the course of a simulation, the solution will encounter a negative thermodynamic property (pressure, temperature, or density), which can cause a NaN (e.g., when computing speed of sound). The `Thermofix` inputs are intended to help remedy this situation. While they provide additional stability when needed, they also introduce additional dissipation, and should be avoided if possible.

`Method` specifies the method used to repair cells with bad thermodynamic properties. Checks are performed during each Runge-Kutta stage.

0 = None

1 = Limit the high-order modes using the exponential sponge filter.

2 = Scale the high-order modes by `Scale_Factor`.

`Iterations` specifies how many passes to make before abandoning further repair efforts.

`Scale_Factor` specifies the scaling factor to apply to the high-order modes when using `Method=2`.

`Check_Cells`, `Check_Faces`, and `Check_Nodes` specifies whether to check the thermodynamic properties at points within the cell volume, the cell faces, or the cell nodes (corners). If any of these checks identifies a troubled point, the solution within the cell volume is treated and re-interpolated to the faces and nodes.

`Verbose` is used to output information about the bad cells to the display output.

`Min_Pressure`, `Min_Temperature`, and `Min_Density` are used to specify threshold values below which action should be taken. These limits are imposed individually, so the three specified values need not satisfy the equation of state. Using small yet positive values is intended to identify problem regions before the solution has deteriorated too severely. The actual values for these parameter may need to be adjusted for each flow problem. These values should be small enough to not be physically encountered within the flow but large enough that the solution can still be recovered before fully diverging.

11.10 nml_time_ave

This **namelist block** controls time averaging of solution variables.

```
&nml_time_ave
  Compute_Time_Averaging      = F
  Num_Saved_Time_Ave_Restarts = 2
  Input_Time_Ave_File         = "out.ave"
  Time_Ave_Vel_Is_Axisymm     = F
  Start_Averaging_Timestep    = -1
  Start_Averaging_Tstar       = -1.0
/
```

Listing 15: Namelist nml_time_ave.

Compute_Time_Averaging = F

This flag enables or disables the accumulation of time-averaged flow variables. If enabled, time-averaged restart and CGNS solution files will be written whenever the standard restart and CGNS solution files are written.

The time-averaged restart file will be written to the “**Output_Dir**/Restart_files/” directory. In order to save disk space, the name of the time-averaged restart file will cycle between the following:

“**Output_Dir**/Restart_files/out.1.ave”

“**Output_Dir**/Restart_files/out.2.ave”

...

“**Output_Dir**/Restart_files/out.N.ave”

where $N = \text{Num_Saved_Time_Ave_Restarts}$. After a new time-averaged restart file is successfully written, a symbolic link will be created that points to the latest time-averaged restart file. The name of this link will be:

“**Output_Dir**/Restart_files/out.current.ave”

Num_Saved_Time_Ave_Restarts = 2

This is used to change the number of time-averaged restart files (out.*.ave) that are saved to the Restart_files directory. The default is 2, so output will cycle between out.1.ave and out.2.ave.

Through the setting of **Restart_Interval**, **Restart_Frequency**, and this input, one should ensure there is a viable fallback solution to restart from should an NaN occur.

Input_Time_Ave_File = "out.ave"

File path to the time-averaged solution restart file that will be read during initialization if the following criteria are met:

- **Load_Restart_File** = T
- One of the following is met:
 - **Compute_Time_Averaging** = T
 - **Start_Averaging_Tstar** > 0 and **Start_Averaging_Tstar** ≤ t^* , where t^* is the non-dimensional simulation time, Eq. (80), of the data that was read in from **Restart_File**.

- `Start_Averaging_Timestep` > 0 and `Start_Averaging_Timestep` ≤ *iteration*, where *iteration* is the iteration number of the data that was read in from `Restart_File`.

The code will read the time-averaged flow variables of the previous simulation from which the new simulation was restarted, and then add to those statistics during the current run job.

NOTE: If the above criteria are met, but the file specified by `Input_Time_Ave_File` does not exist, a completely new time averaging will be started. This is the case even if you were intending to restart a previous time averaging, and the reason the file could not be found was due to a small typo in the file name.

NOTE: This file is only for *reading* during initialization and is *not an output file*.

`Time_Ave_Vel_Is_Axisymm = F`

This option is useful for recording statistics for axisymmetric flows. It specifies whether axisymmetric velocity components should be included in the time-averaged flow variables. Note, regardless of the value for this input variable, the Cartesian momentum and velocity components ($[\rho v_x, \rho v_y, \rho v_z]$ and $[v_x, v_y, v_z]$) will always be averaged and saved to the time-averaged restart file.

- T The geometry is assumed to be axisymmetric about the x -axis. The axisymmetric momentum and velocity components in the radial ($\rho v_r, v_r$) and circumferential ($\rho v_\theta, v_\theta$) directions will also be averaged and saved to the time-averaged restart file. The stress and heat flux data will be for the axisymmetric terms, *not* the Cartesian terms (i.e., the axisymmetric velocity components are used in the time-averaged terms). For example, the averaged term $\overline{v_y T}$ will be replaced with $\overline{v_r T}$ and $\overline{\rho v_x v_z}$ will be replaced with $\overline{\rho v_x v_\theta}$.
- F The stress and heat flux data will be for the Cartesian terms (i.e., the Cartesian velocity components are used in the time-averaged terms).

NOTE: See Section 10.8 for additional details.

`Start_Averaging_Timestep = -1`

Iteration number to begin time averaging.

`Start_Averaging_Tstar = -1.0`

Non-dimensional simulation time (t^*), Eq. (80), to begin time averaging.

11.11 nml_output

This **namelist block** is used to control various output files.

```
&nml_output
  Time_Scaling_Factor      = 1.0
  Display_Interval         = 1
  Residual_Interval        = 0
  Restart_Interval         = 0
  Restart_Frequency        = -1.0 ! tstar
  Restart_Output_Uses_Host_Roots = T
  Restart_Output_Disabled  = F
  Output_Dir               = "."
  Output_Interval          = -5
  Output_Frequency         = -1.0 ! tstar
  Output_Order             = -3
  Output_Summary_File      = F
  Dmat_Grad_Temp_Order     = -1
  Loc_Output_Pts           = 0
  CGNS_High_Order_Output   = F
  CGNS_Mixed_Output        = T
  CGNS_Output_Disabled     = F
  Probes_Collection_Interval = 0
  Probes_Write_Interval    = 100
  Point_Probes(1)%xyz(1:3) = [-999.,-999.,-999.] ! grid units
  Rake_Probes(1)%num_probes = 0
  Rake_Probes(1)%xyz_beg(1:3) = [-999.,-999.,-999.] ! grid units
  Rake_Probes(1)%xyz_end(1:3) = [-999.,-999.,-999.] ! grid units
  Profile_IO_All           = F
  Profile_IO_Restart       = F
  Profile_IO_CGNS          = F
  Profile_IO_CGNS_Subst   = F
  Lustre_Stripe_Count      = 64
  Lustre_Stripe_Size       = "4m"
/
```

Listing 16: Namelist nml_output.

```
Time_Scaling_Factor = 1.0 ! [1/s]
```

This scaling factor is used to define a non-dimensional time (t^*)

$$t^* = (t - t_0) \cdot \text{Time_Scaling_Factor} \quad (80)$$

where t is the dimensional time (in seconds). If time-averaging has been activated, then t_0 is the dimensional time (in seconds) when time-averaging began. Otherwise, $t_0 = 0$. This non-dimensional time (t^*) is what is written to standard output and the residual file. It is also used by the following input variables to determine when certain events should occur:

- **Moving_Ref_Frame%Spinup%Tstar** for the moving reference frame startup,

- **Start_Averaging_Tstar** for time averaging startup,
- **Restart_Frequency** of restart files,
- **Output_Frequency** of CGNS files, and
- **Final_Time** for when to stop a simulation.

For example, many simulations are based on a number of flow-through periods. In this case, the **Time_Scaling_Factor** should be set to the value of U/L , where U is the convective velocity and L is the flow-through length of the domain.

Display_Interval = 1

Number of time steps between writing the time-stepping, residual, and convergence statistics to standard output. The input is treated as follows:

- < 0 Reset to **Num_Timesteps**/**|Display_Interval|**.
- = 0 Reset to 1.
- > 0 Use as-is.

NOTE: When running the Taylor-Green vortex problem or the Taylor-Couette problem, this input is also used as the interval for writing integrated kinetic energy dissipation rate (KEDR) statistics to the file “TaylorGreen_KEDR.dat” or “TaylorCouette_KEDR.dat”, respectively.

Residual_Interval = 0

Number of time steps between writing the residual and convergence statistics for all conserved and primitive variables to the residual file “residual.dat”. The input is treated as follows:

- < 0 Reset to **Num_Timesteps**/**|Residual_Interval|**.
- = 0 Reset to **Num_Timesteps**/500.
- > 0 Use as-is.

Restart_Interval = 0

Specifies the number of time steps between writing restart files. Note that a restart file is always written before the first (non-restarted) time step and after completing the last time step. With the default value of 0, no intermediate files will be written, only the restarts before the first time step and after the final time step. After successfully writing a new restart file, a symbolic link will be created that points to the latest restart file. The name of this link will be:

“**Output_Dir**/Restart_files/out.current.rst”

Restart_Frequency = -1.0 ! tstar

If > 0 , this value specifies the non-dimensional time (t^*) interval, Eq. (80), to write restart files. This is complimentary to **Restart_Interval** and will write additional restart files. After successfully writing a new restart file, a symbolic link will be created that points to the latest restart file. The name of this link will be:

“**Output_Dir**/Restart_files/out.current.rst”

Restart_Output_Uses_Host_Roots = T

Indicates which MPI processes are involved in writing to the restart file. If true, then only the root processes on each host node will preform the output. Otherwise, all MPI processes will be involved in the output.

NOTE: This is only applicable for parallel simulations run across multiple physical computers, e.g., multiple nodes on the NASA HECC supercomputers. Only change this if you encounter MPI errors that occur when trying to write restart files. Changing this to false will most likely result in higher I/O file contention and possibly increased memory usage.

`Restart_Output_Disabled = F`

If set to true, this option will prevent any instantaneous or time-averaged restart files from being written. This is intended for simulations that will complete with the allotted resources and will not need to be restarted later.

`Output_Dir = "."`

File path to the directory used for writing output files.

NOTE: If one is available, it is recommended that the output directory be located on a Lustre file system in order to get the best I/O performance possible.

`Output_Interval = -5`

This controls the number of time steps between writing CGNS solution files. The input value is interpreted as follows:

- < 0 For most situations, every `Num_Timesteps`/`|Output_Interval|` time steps.
- = 0 For most situations, every `Num_Timesteps`/50 time steps.
- > 0 For all situations, every `Output_Interval` time steps.

The first two options result in a fixed number of output files each time the code is run, assuming the code does not terminate early.

NOTE: For the following special cases, the input value is interpreted differently:

- If either `Output_Frequency` > 0 or `CGNS_Output_Disabled` = T, the action associated with those inputs will take precedence and this input will not be used.
- If running any of the transport problems (isentropic vortex, density/entropy advection), this input value is used to determine when a CGNS solution file will be written as follows:
 - < 0 At the completion of every `|Output_Interval|` flow-through periods.
 - = 0 At the completion of every flow-through period.
- If running either the Taylor-Green vortex problem or the Taylor-Couette problem, CGNS solution files will automatically be written at certain specific values of non-dimensional time (t^*) that are of interest to each specific problem. If `Output_Interval` = 0, only these problem specific CGNS solution files will be written; otherwise (`Output_interval` $\neq$ 0) this behaves the same as “most situations” in addition to writing the problem-specific CGNS solution files.

`Output_Frequency = -1.0 ! tstar`

If > 0, this value specifies the non-dimensional time (t^*) interval, Eq. (80), to write CGNS files. `Output_Interval` will be disabled.

`Output_Order = -3`

Specifies the degree of the polynomial space used to represent the solution in the CGNS solution files. This allows the ability to over-sample the solution polynomial within each cell which provides a smoother solution when visualizing within your favorite post-processing software.

NOTE: If using `CGNS_High_Order_Output`, the output order will be limited to a maximum of 4, since that is the maximum order supported by the CGNS library.

NOTE: If `Output_Order < Solution_Order`, the solution order will be used instead.

`Output_Summary_File = F`

Setting this to true will create the file “summary.log” when the simulation successfully stops. This file contains summarized information about the simulation like:

- computational cost
- solution order
- number of cells
- degrees of freedom
- total time or t^* elapsed during this simulation
- final time or t^* at the end of the simulation
(i.e., combining the elapsed value with that from the restart file)
- the current length of time averaging (if applicable)
- and more

When running the channel flow test case using the internal grid generator, it will also contain detailed grid information such as the number of grid cells and degrees of freedom in each direction, as well as their respective spacings.

`Dmat_Grad_Temp_Order = -1`

If outputting the derivative of temperature, this is the quadrature order used for computing temperature before projecting down to the solution order and computing the derivative.

NOTE: If `Dmat_Grad_Temp_Order < 0`, `Solution_Order` will be used instead.

`Loc_Output_Pts = 0`

Specifies the location of the output nodal points within the grid cells when writing the CGNS solution files.

0 = Equidistant points nodes

1 = Legendre-Gauss (Gauss) points nodes

2 = Legendre-Gauss-Lobatto (Lobatto) points nodes

NOTE: Using Legendre-Gauss nodes for the CGNS solution output files will result in empty space between all cells because those points do not extend all the way to the cell edge and there is no connectivity information between cells. This only affects your ability to visualize the solution and does not affect the solution accuracy in any way.

`CGNS_High_Order_Output = F`

Setting this to true will write the CGNS grid file (solution.grid.cgns) using high-order cells. If not enabled (the default), the CGNS grid file breaks each cell into linear sub-cells.

NOTE: Visualizing high-order CGNS solution files requires specialized software, such as Tecplot 2023 or newer.

`CGNS_Mixed_Output = T`

This determines how cell connectivity is handled in high-order CGNS grid files. If true, all grid cells will be combined into a single zone regardless of cell geometry or order. If false,

each individual combination of cell geometry and order will be written to its own zone. Some post-processing software may prefer the latter format.

```
CGNS_Output_Disabled = F
```

Completely disables CGNS solutions from being output at the output interval.

On some systems with a large number of processors, the output of CGNS files can be slow. It can be advantageous to instead output restart files, and later use **Convert_Restart_To_CGNS** in the `nml_utility` namelist block of Section 11.21 to have GFR batch convert those restart files to CGNS. This post-processing step can be done using a minimal number of processors, because most of the work is simple file I/O.

```
Probes_Collection_Interval = 0
```

If > 0 , this value specifies the iteration interval used to collect time-history data for each of the probes. The information collected at this interval is saved to a buffer and only written out after completing the number of collections specified by the input **Probes_Write_Interval**. With the default value of 0, no time-history probe data will be collected or output.

```
Probes_Write_Interval = 100
```

This value specifies how often to write out the buffer used to collect time-history data for the probes.

NOTE: This interval is relative to the number of collections completed, and is not relative to the simulation iteration.

```
Point_Probes(1)%xyz(1:3) = [-999.,-999.,-999.] ! input grid units
```

This specifies the location of a single point **probe** to collect time-history data. The values specified for this input are assumed to be in the units of the input mesh file (i.e., *grid units* as discussed in Section 3.3). Up to 200 point probes may be used. Probes set to the default location are ignored. For more information on the point probe files, see Section 10.9.

NOTE: The solution point closest to these coordinates will be used as the source for the probe data.

```
Rake_Probes(1)%Num_Probes    = 0
Rake_Probes(1)%xyz_beg(1:3)  = [-999.,-999.,-999.] ! input grid units
Rake_Probes(1)%xyz_end(1:3)  = [-999.,-999.,-999.] ! input grid units
```

These inputs provide the details necessary to create a rake of point probes. The number of probes to create within rake i is specified using the input **Rake_Probes(i)%Num_Probes**. **Rake_Probes(i)%xyz_beg(1:3)** and **Rake_Probes(i)%xyz_end(1:3)** specify the coordinates for the start and end points of the rake, with both of these coordinate inputs assumed to be in the units of the mesh file (i.e., *grid units* as discussed in Section 3.3).

In the case that **Num_Probes** = 1, the single probe created by the rake will be located halfway between the start and end points for the rake. Otherwise, probes will be located at both the start and end points for the rake, and **Num_Probes** – 2 probes will be equally spaced along the line between the start and end points.

Up to 5 rake probes may be used, with no limit on the number of probes per rake. Rakes with **Num_Probes** = 0 or **xyz_beg** = **xyz_end** are ignored, and no time-history probe data will be collected or output for the rake. For more information on the rake probe files, see Section 10.9.

NOTE: For each individual rake probe, the solution point closest to the probe coordinates will be used as the source for the probe data.

NOTE: It should be warned that too many probes (1000+) may have a large negative impact on code and file system performance.

`Profile_IO_All = F`

If true, sets both `Profile_IO_CGNS` and `Profile_IO_Restart` to true. If false, it has no effect on the other input parameters.

`Profile_IO_Restart = F`

Activates timers to measure how long it takes to write each restart or time-averaged restart file, and outputs the results to standard output.

`Profile_IO_CGNS = F`

Activates timers to measure how long it takes to write each CGNS solution file, and outputs the results to standard output.

`Profile_IO_CGNS_SubS = F`

Activates timers to measure how long it takes to complete each individual call to a function from the CGNS library, and outputs the results to standard output.

`Lustre_Stripe_Count = 64`

Specifies the approximate number of stripes to use when writing a CGNS solution file or a restart file.

NOTE: This is only used if `Output_Dir` is on a Lustre file system. The code will take this value and find the closest integer value satisfying: $\text{mod}(ncpu, \text{Lustre_Stripe_Count}) = 0$

`Lustre_Stripe_Size = "4m"`

Specifies the number of bytes to store on each stripe before moving to the next stripe.

NOTE: This is only used if `Output_Dir` is on a Lustre file system. The value of `Lustre_Stripe_Size` must be a multiple of 65,536 bytes (64 KB), and the suffixes 'k', 'm', or 'g' can be used to specify units of KB, MB, or GB, respectively.

EXAMPLE: "4m" $\Rightarrow$ 4 MB $\Rightarrow$ 4,194,304 bytes

11.12 nml_output_variables

This **namelist block** is used to specify which variables to output to the CGNS files.

```
&nml_output_variables
  Conserved_Variables      = F
  Momentum                 = F
  Vorticity                = F
  Cylindrical_Velocity     = F
  Cylindrical_Coordinates  = F
  Relative_Velocity        = F
  Relative_Conditions      = F
  Thermo_Variables        = F
  Temperature              = F
  Gradients                = F
  Transport_Variables      = F
  Stagnation_Variables     = F
  Stagnation_Pressure      = F
  Stagnation_Temperature   = F
  Marker_Variables        = F
  Qcriterion               = F
  Species_Mass_Fraction    = T
  Species_Density          = F
  Species_Mole_Fraction    = F
  Extra_Species_Variables  = F
  Debug_Species_Variables  = F
  Grdnt_Species_Variables  = F
  Exact_Density            = F
  Exact_Pressure           = F
/
```

Listing 17: Namelist nml_output_variables.

The following variables are automatically output to the CGNS files:

Density, **VelocityX**, **VelocityY**, **VelocityZ**, **Pressure**, and **Mach**.

NOTES:

- The grid coordinates are output in the **grid units** of the input grid file. For example, say your grid file has dimensions of inches. You will need to include the input **Grid_Scaling_Factor** = 0.0254 to convert the grid file to meters for it to be used correctly in the solver. However, this conversion will be undone when creating the CGNS solution files, which will contain coordinate dimensions of inches just like the grid file.
- All variables are output in SI units (except for the coordinate variables as discussed in the previous bullet).
- Due to the first two bullets, if using post-processing software to compute integrated quantities, care needs to be taken to ensure the coordinate variables and variable(s) being integrated are in compatible units.

- In the following descriptions, any vector quantities that are output for 1D and 2D cases will exclude vector components from higher spatial dimensions. For example, **VelocityY** and **VelocityZ** will not be included for a 1D flow, and for a gradient variable from a 2D simulation, the **[VariableName]GradientZ** variables would not be included.
- If multiple inputs add the same variable to the list of output variables, only one instance of the variable will be written, i.e., any duplicates in the list of output variables will be removed.
- The names of the variables written to the CGNS solution files attempt to follow the variable naming conventions of the CGNS standard, e.g., the x -coordinate variable will have the name **CoordinateX** instead of just “X”.

Conserved_Variables = F

Setting this to true adds the following variables to the CGNS solution files:

Density, **MomentumX**, **MomentumY**, **MomentumZ**, and **EnergyStagnationDensity** (i.e., ρe_t).

Momentum = F

Setting this to true adds the following variables to the CGNS solution files:

MomentumX, **MomentumY**, and **MomentumZ**.

Vorticity = F

Setting this to true adds the following variables to the CGNS solution files:

VorticityX, **VorticityY**, **VorticityZ**, **VorticityMagnitude**, **DivergenceOfVelocity**, **StrainRateMagnitude**, **Qcriterion**, and **Qcriterion_Positive**.

Cylindrical_Velocity = F

Setting this to true adds the following variables to the CGNS solution files:

VelocityR and **VelocityTheta**.

If either **Relative_Velocity** or **Relative_Conditions** is also true, then the following variables will also be added to the CGNS solution files:

RelativeVelocityR and **RelativeVelocityTheta**.

Cylindrical_Coordinates = F

Setting this to true adds the following variables to the CGNS *grid* file:

CoordinateAxial, **CoordinateR**, and **CoordinateTheta**.

This will also automatically set **Cylindrical_Velocity** to true.

NOTE: The axial coordinate direction will be set to the x -axis, i.e., **CoordinateAxial** = **CoordinateX**, unless:

- **AoA_Alpha** = $\pm 90^\circ$ and **AoA_Beta** = 0° : $\Rightarrow$ **CoordinateAxial** = **CoordinateY**
- **AoA_Alpha** = 0° and **AoA_Beta** = $\pm 90^\circ$: $\Rightarrow$ **CoordinateAxial** = **CoordinateZ**

The other two cylindrical coordinate variables are determined by the axial direction and the right-hand rule.

NOTE: These variables will only be written to the CGNS grid file. These variables will be added to each of the solution files by using CGNS links.

Relative_Velocity = F

Setting this to true adds the following variables to the CGNS solution files:

RelativeVelocityX, **RelativeVelocityY**, and **RelativeVelocityZ**.

Relative_Conditions = F

Setting this to true adds the following variables to the CGNS solution files:

RelativeVelocityX, **RelativeVelocityY**, **RelativeVelocityZ**,
RelativeMomentumX, **RelativeMomentumY**, **RelativeMomentumZ**, and **RelativeMach**.

Thermo_Variables = F

Setting this to true adds the following variables to the CGNS solution files:

Pressure, **Temperature**, **Enthalpy**, **SpecificHeatPressure** (i.e., c_p), and **Entropy**.

Temperature = F

Setting this to true adds the following variable to the CGNS solution files: **Temperature**.

Gradients = F

Setting this to true will also include in the CGNS solution file the directional gradient components for any of the following variables that are also set to be included in the CGNS solution file: **Density**, **Velocity**, **EnergyStagnationDensity**, **Temperature**, **Momentum**, **MassFraction**, and **SpeciesDensity**.

The name of the gradient variables will be the name of the non-gradient variable with **GradientX**, **GradientY**, and **GradientZ** appended on to the variable name, e.g., **MomentumYGradientZ**.

Transport_Variables = F

Setting this to true adds the following variables to the CGNS solution files: **ViscosityMolecular** and **ThermalConductivity**.

If using a multi-component gas mixture and **Extra.Species.Variables** is also true, then the variable **Prandtl** giving the mixture Prandtl number will also be included in the CGNS solution file.

Stagnation_Variables = F

Setting this to true adds the following variables to the CGNS solution files: **PressureStagnation**, **TemperatureStagnation**, and **EnthalpyStagnation**.

Stagnation_Pressure = F

Setting this to true adds the following variable to the CGNS solution files: **PressureStagnation**.

Stagnation_Temperature = F

Setting this to true adds the following variable to the CGNS solution files: **TemperatureStagnation**.

Marker_Variables = F

If using **Marker_Option**, the following variables will be added to the CGNS solution files:

Marker_Option	Output Variables	
1	Resolution_Marker	Scaled_Resolution_Marker
2	Jump_Marker	Scaled_Jump_Marker
3	MLP_Marker	Scaled_MLP_Marker

The variable specified by **Marker_Variable** will also be output.

Qcriterion = F

Setting this to true adds the following variables to the CGNS solution files:

Qcriterion and **Qcriterion_Positive**.

For the variable **Qcriterion**, the computed Q-criterion is normalized then rescaled such that the average Q-criterion is 0.0, and the values ± 1 respectively represent $\pm 1/100$ of a standard deviation from the average. The variable **Qcriterion_Positive** contains the true values of the Q-criterion, but only the positive parts, with the negative parts set to zero.

Species_Mass_Fraction = T

If using a multi-component gas mixture, setting this to true adds the mass fractions for each of the gas species to the CGNS solution files. The variable name will be **MassFraction** with the species' chemical formula appended to the end, e.g., **MassFractionH2O**.

Species_Density = F

If using a multi-component gas mixture, setting this to true adds the individual density for each of the gas species to the CGNS solution files. The variable name will be **SpeciesDensity** with the species' chemical formula appended to the end, e.g., **SpeciesDensityH2O**.

Species_Mole_Fraction = F

If using a multi-component gas mixture, setting this to true adds the mole fractions for each of the gas species to the CGNS solution files. The variable name will be **MoleFraction** with the species' chemical formula appended to the end, e.g., **MoleFractionH2O**.

Extra_Species_Variables = F

If using a multi-component gas mixture, setting this to true adds the following variables to the CGNS solution files:

SpeciesDensity, **Enthalpy**, **StagnationEnthalpy**, **SpecificHeatRatio** (i.e., mixture γ), **SpecificHeatPressure** (i.e., mixture c_p), **IdealGasConstant** (i.e., mixture R_{gas}), **MomentumX**, **MomentumY**, **MomentumZ**, **EnergyStagnationDensity**, **PressureStagnation**, and **TemperatureStagnation**.

Each gas component will have its own **SpeciesDensity** variable with its respective chemical formula appended to the end, e.g., **SpeciesDensityCO2**.

Debug_Species_Variables = F

If using a multi-component gas mixture, setting this to true will output the individual contribution of each species to various mixture thermodynamic properties, e.g., enthalpy, c_p , etc.

NOTE: These variables are intended to aid developers in debugging the thermodynamic properties of gas mixtures.

```
Grdnt_Species_Variables = F
```

If using a multi-component gas mixture, this is essentially the same as **Gradients**, but it also outputs the gradients of the species mass fractions and densities.

```
Exact_Density = F  
Exact_Pressure = F
```

For cases that have an analytical solution, the exact density and pressure can be computed at the output points and written to the solution file for direct comparison with the numerical solution. Cases that support this output include the following:

Initialization	Description
23	1D Sod shock tube
26	1D Jiang-Shu density pulse test

11.13 nml_sem

This **namelist block** is used to specify the global controls of the Synthetic Eddy Method. Additional controls for specifying the SEM on individual boundaries are contained in the **nml_bc** namelist block described in Section 11.7.

```
&nml_sem
  Use_Profile_For_Initialization = F
  Use_Random_Seed_From_Restart  = F
  Use_Spherical_Eddy_Volume     = F
  Include_Thermo_Fluctuations   = F
  Box_Num_Eddies_Multiplier     = 1.0
/
```

Listing 18: Namelist nml_sem.

Use_Profile_For_Initialization = F

Specifies whether the SEM profile will be used to initialize the *entire volume* of the flow domain based on the y coordinate value and the profile height variable. This is very experimental and only works for very simple geometries like a flat plate.

Use_Random_Seed_From_Restart = F

When using an SEM boundary condition, the SEM data will also be written to the restart files including the seed used for the random number generator.

If **Use_Random_Seed_From_Restart** is true, the seed for the random number generator will be read from the restart file and used to reset the random number generator to exactly where it left off when the restart file was written, as if the simulation never stopped.

If it is false, the current date and time will be used to create a new “random” seed for the random number generator. This might have a very small effect on the “randomness” of the generated strength and locations of the synthetic eddies in the SEM BC. However, the total number of time steps needed for a simulation will almost certainly prevent this from having any kind of measurable effect on the simulation results. **NOTE:** Very rarely (but it does happen), the SEM can randomly cause a simulation to die due to an NaN or other floating point exception. Setting this input to false is the only way to get past one of these occasions without modifying the solution with limiting and/or filtering.

Use_Spherical_Eddy_Volume = F

This input determines if the volume of each synthetic eddy is treated as a sphere (true) or as a box (false). It affects the number of eddies that will be used for an individual SEM boundary,

$$[\text{Number of eddies}] = [\text{Box_Num_Eddies_Multiplier}] \times \frac{[\text{Volume of SEM box}]}{[\text{Volume of synthetic eddy}]}$$

where the volumes are described in Section 5.4.1.1.

<code>Include_Thermo_Fluctuations = F</code>
--

Specifies whether SEM fluctuations will be applied to the temperature and density profiles based on the Strong Reynolds Analogy for temperature fluctuations. If this input is false, no fluctuations are added to the temperature and density profiles. See Section 5.4.1.6 for details.

<code>Box_Num_Eddies_Multiplier = 1.0</code>
--

This input is used as a multiplier to the total number of synthetic eddies used for the SEM boundary conditions, i.e.,

$$[\text{Number of eddies}] = [\text{Box_Num_Eddies_Multiplier}] \times \frac{[\text{Volume of SEM box}]}{[\text{Volume of synthetic eddy}]}$$

where the volumes are described in Section 5.4.1.1.

NOTE: The number of eddies for an individual SEM box are limited to be between a minimum of 5,000 and a maximum of 500,000. These limits were arbitrarily picked and hardwired into the SEM code, and either limit can be modified in the source code if needed. They were implemented to prevent (1) too few eddies from being present to adequately model the flow, and (2) too many eddies that add little to the modeling, but increase computational cost.

11.14 nml_chemistry

This **namelist block** is used to specify control of chemically reacting flows.

```
&nml_chemistry
  Disable_Chemical_Reactions      = F
  Semi_Implicit_Chemistry         = 0
  Ignition_Method                 = 0
  Ignition_Start_Timestep         = 0
  Ignition_Duration               = 1
  Ignition_Prod2Reac_L2_Ratio_Goal = -1.0
  Ignition_Max_Temperature        = 1600.0      ! [K]
  Ignition_Spark_Cap              = 1.0e+12     ! [N/m^2]
  Ignition_Source_Magnitude       = 20.0
  Ignition_Source_Radius          = 0.0         ! [m]
  Ignition_Source_Location        = [0.0,0.0,0.0] ! [m]
/
```

Listing 19: Namelist nml_chemistry.

Disable_Chemical_Reactions = F

If running a multi-component gas mixture, setting this to true will disable all chemical reactions, i.e., all reaction source terms are set to zero, resulting in a simulation of an inert gas mixture.

When reactions are disabled, all non-existent chemical species will be dropped from the simulation to reduce computational cost. For example, if running the 7-species (H_2 , O_2 , H_2O , OH , H , O , N_2)/7-reaction mechanism for a hydrogen/air mixture with reactions disabled and the flow initialized to a mixture of H_2 , O_2 , and N_2 , then the product species H_2O , OH , H , and O will be dropped completely from the simulation. If a simulation is run with reactions disabled, any dropped species will not be written to the restart files. However, these non-reacting restart files can still be used to restart a simulation with reactions enabled and the product species will be added to the simulation with an initial mass fraction of zero.

Semi_Implicit_Chemistry = 0

If running a multi-component gas mixture with reactions, this input controls the numerical treatment of the reaction source terms:

- 0 = Standard explicit treatment of the reaction source terms, which is likely to require the smallest time step.
- 1 = Point implicit treatment of the reaction source terms for the species transport equations, using an analytical evaluation of the Jacobian matrix.
- 2 = Point implicit treatment of the reaction source terms for the species transport equations, using a numerical approximation of the Jacobian matrix.
- 1 = Use special hard-coded source terms if using the 7-species /7-reaction hydrogen/air reaction mechanism. Otherwise, the same as option 0.

If the reaction terms require a smaller time step than what would be required without reactions, then using a point implicit treatment of the reaction source terms may (or may not) allow for using a time step closer to the non-reacting case.

Ignition_Method = 0

Specifies the ignition method to use for speeding up the startup of chemical reactions within the solution.

0 = Ignition process is disabled.

1 = A source term of $\rho c_p \Delta T$ is added to the energy equation each Runge-Kutta stage, where $\Delta T = \text{Ignition_Source_Magnitude} / [\# \text{ RK Stages}]$.

2 = A percentage of the reference total energy, $(\rho e_t)_\infty$, is added to the energy equation as a source term. The input **Ignition_Source_Magnitude** specifies the percentage to use, so the source term will be $\text{Ignition_Source_Magnitude} \times (\rho e_t)_\infty$.

3 = The temperature value specified with **Ignition_Source_Magnitude** is used instead of the actual local temperature for the solution point when computing the species reaction source terms.

Ignition_Start_Timestep = 0

Iteration number to begin using the ignition source terms for chemistry reactions.

Ignition_Duration = 1

Specifies the number of time steps for the ignition source term to remain active once either the temperature at any solution point reaches **Ignition_Max_Temperature** or the target goal for the overall ratio of L2(products)/L2(reactants), **Ignition_Prod2Reac_L2_Ratio_Goal**, within the global solution is achieved. The ignition source terms turn off after this number of time steps are completed.

Ignition_Prod2Reac_L2_Ratio_Goal = -1.0

Target goal for the ratio of L2(products)/L2(reactants) of the simulation to signify that reactions are occurring. Once this goal is met, the ignition source terms will continue to be active for the number of time steps specified by **Ignition_Duration**, after which, the ignition source terms will be disabled.

Ignition_Max_Temperature = 1600.0 ! [K]

Specifies a value for temperature (in Kelvin) that marks when to begin turning off the ignition source terms. Once this temperature has been reached, the ignition source terms will continue to be active for the number of time steps specified by **Ignition_Duration**, after which, the ignition source terms will be disabled. Any solution point that reaches this temperature while the ignition source terms are still active will have its ignition source term set to zero. When using **Ignition_Method** = 3, this means the actual local temperature at the solution point will be used to compute the reaction source terms instead of the reaction temperature specified by **Ignition_Source_Magnitude**.

Ignition_Spark_Cap = 1.0e+12 ! [N/m^2]

This specifies a maximum limit that can be used for the energy source term (in N/m²).

Ignition_Source_Magnitude = 20.0

Value to use for the ignition source term. The value specified depends on the ignition method specified by **Ignition_Method**.

`Ignition_Method = 1`

This input represents a temperature value (units of K) that is used to compute a source term that is added to the energy equation each Runge-Kutta stage. The energy source term is evaluated as $\rho c_p \Delta T$ where $\Delta T = \text{Ignition_Source_Magnitude}/[\# \text{ RK Stages}]$.

`Ignition_Method = 2`

This value represents a percentage value that is used to add a percentage of the reference total energy, $(\rho e_t)_\infty$, as a source term to the energy equation. The energy source term is evaluated as $\text{Ignition_Source_Magnitude} \times (\rho e_t)_\infty$.

`Ignition_Method = 3`

This value represents a reaction temperature (units of K) that is used to compute the species reaction source terms instead of using the actual local temperature for the solution point.

`Ignition_Source_Radius = 0.0 ! [m]`

Radius of influence (R) for the ignition source terms, specified in *grid scaled units* (i.e., meters) as described in Section 3.3. The behavior depends upon the value (R) specified:

- < 0 The ignition source term will immediately decay as $\exp(-36r^2/R^2)$ from the point specified by `Ignition_Source_Location`.
- = 0 The ignition source is disabled.
- > 0 Every point that is within the ignition source radius (R) from the `Ignition_Source_Location` will get the full value of the energy source term based on the input value of `Ignition_Source_Magnitude`. For all points outside this radius (R), the ignition source term will decay as $\exp(-36(r - R)^3/R^3)$. The higher exponent realizes a slower initial decay.

Fig. 20 compares the decay profiles.

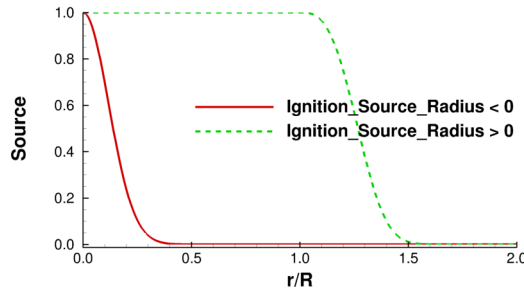


Figure 20: Ignition source decay.

`Ignition_Source_Location = [0.0,0.0,0.0] ! [m]`

The center of the ignition source region, specified in *grid scaled units* (i.e., meters) as described in Section 3.3.

11.15 nml_testcase_channel

This **namelist block** is used to specify inputs for the rectangular channel flow test case (**Initialization** = 2) or the annular-pipe channel flow test case (**Initialization** = 25). The inputs that start with “Grid_” are specific to running one of these cases using the internal grid generator.

```
&nml_testcase_channel
  Grid_Annulus_Angle = 5.0      ! [deg]
  Grid_Ave_Xplus     = -30.0
  Grid_Ave_Yplus     = -15.0
  Grid_Ave_Zplus     = -15.0
  Grid_Yplus_Wall    = 2.0
  Grid_Yplus_Of_One  = 8.42e-6 ! [m]
  Body_Force         = 0.0      ! [N]
  Nondim_Output_Coord = F
/
```

Listing 20: Namelist nml_testcase_channel.

```
Grid_Annulus_Angle = 5.0 ! [deg]
```

This input is used when running the annular-pipe channel flow case and specifies the angle (in degrees) of the circumferential extent of the grid domain. The input value will be limited to an angle between 0.1 and 90 degrees. The radial distance between the inner and outer walls of the annular pipe will be set to the channel height used for the rectangular channel flow case. The arc length of the annular pipe *outer wall* will be set to the span used for the regular channel flow case. From the arc length and circumferential angle, the outer radius of the annulus will be computed. This means the amount of curvature for the annular pipe will vary depending on the input **Grid_Annulus_Angle**, with a maximum value of 90 resulting in a highly curved annular pipe and a minimum value of 0.1 have little-to-no curvature.

```
Grid_Ave_Xplus = -30.0
```

When running either of the channel flow problems using the internal grid generator, this input is used to determine the average grid spacing in the axial (x) direction as follows:

- < 0 The abs(input value) specifies the average x^+ degree of freedom spacing.
This means the average x^+ cell spacing is given by $|\text{Grid_Ave_Xplus}| \times (\mathcal{P}_S + 1)$.
- > 0 The input value specifies the average x^+ cell spacing.

```
Grid_Ave_Yplus = -15.0
```

When running either of the channel flow problems using the internal grid generator, this input is used to determine the average grid spacing in the vertical (y) direction specifically at the center-height of the channel. Note, for the annular-pipe channel flow case, this is applied to the radial spacing.

- < 0 The abs(input value) specifies the average y^+ degree of freedom spacing at the center of the channel. This means the average y^+ cell spacing at the center of the channel is given by $|\text{Grid_Ave_Yplus}| \times (\mathcal{P}_S + 1)$.
- > 0 The input value specifies the average y^+ cell spacing at the center of the channel.

NOTE: This input will be used to determine the y^+ spacing for the cells at the middle of the channel height, and the input **Grid_Yplus_Wall** will be used to determine the y^+ spacing for the cells on channel walls. A tanh spacing will then be used to get the y^+ cell spacings for all the cells in between.

Grid_Ave_Zplus = -15.0

When running either of the channel flow problems using the internal grid generator, this input is used to determine the average grid spacing in the spanwise (z) direction. Note, for the annular-pipe channel flow case, this is applied to the circumferential spacing.

- < 0 The abs(input value) specifies the average z^+ degree of freedom spacing. This means the average z^+ cell spacing is given by $|\text{Grid_Ave_Zplus}| \times (\mathcal{P}_S + 1)$.
- > 0 The input value specifies the average z^+ cell spacing.

Grid_Yplus_Wall = 2.0

When running either of the channel flow problems using the internal grid generator, this input is used to determine the height of the first cell off the wall. Note, for the annular-pipe channel flow case, this is applied to the radial wall spacing.

- < 0 The location of $y^+=1$ is located at the `nint(|Grid_Yplus_Wall|)`-th degree of freedom off the wall.
- > 0 This specifies the height (in y^+) of the first cell off the wall.

Grid_Yplus_Of_One = 8.42e-6 ! [m]

This input specifies the dimensional length (in meters) corresponding to $y^+=1$ to use with the internal grid generator for either of the channel flow problems. Note, for the annular-pipe channel flow case, this is applied to the radial wall spacing.

Body_Force = 0.0 ! [N]

Body force (in Newtons) used to drive the channel flow problem.

Nondim_Output_Coord = F

If running the channel flow problem and this input is set to true, then all the grid coordinates written to the CGNS solution files will be nondimensionalized by the channel half height.

11.16 nml_testcase_pulse

This **namelist block** is used to specify inputs for the density pulse test case.

```
&nml_testcase_pulse
  Form      = 0
  Form_Dim  = 1
  Length    = 1.0           ! [m]
  Center    = [0.5,0.0,0.0] ! [m] (x,y,z)
  Width     = 0.2           ! [m]
  Gap       = 0.2           ! [m]
  Repeat    = 0
/
```

Listing 21: Namelist nml_testcase_pulse.

Form = 0

Specifies the form of the density pulse test shape.

0 = Combo (Gauss,Square,Triangle,Ellipse)

1 = Gauss

2 = Square

3 = Triangle

4 = Ellipse

Form_Dim = 1

Dimension of the density test shape: 1=1d, 2=2d.

Length = 1.0 ! [m]

1D domain length (in meters) for density pulse test internal grid.

Center = [0.5,0.0,0.0] ! [m] (x,y,z)

$[x, y, z]$ (in meters) at center of first density pulse.

Width = 0.2 ! [m]

Width (in meters) of density pulse.

Gap = 0.2 ! [m]

Gap (in meters) between density pulses.

Repeat = 0

Number of repeat density pulses.

11.17 nml_testcase_sod

This **namelist block** is used to specify inputs for the 1D Sod shock tube problem.

```
&nml_testcase_sod  
  Right_Vector = [1.0,0.0,0.0]  ! [m] (x,y,z)  
/  

```

Listing 22: Namelist nml_testcase_sod.

```
Right_Vector = [1.0,0.0,0.0]  ! [m] (x,y,z)
```

This defines the right side normal vector of the diaphragm for the Sod shock tube problem. It is intended for advanced 2D and 3D cases on an external grid where the flow problem is not aligned with the $+x$ -direction. This orientation vector is used to initialize the right side of the Sod shock tube, which is assumed to be centered at $[x, y, z] = [0, 0, 0]$.

See Also: [Exact_Density](#), [Exact_Pressure](#)

11.18 nml_testcase_tcf

This **namelist block** is used to specify inputs for the Taylor-Couette flow test case. This version of the Taylor-Couette flow case is based on that found in Ref 30, which uses the initial conditions from the Taylor-Green vortex problem as inspiration for the initial conditions for this case to speed up the time it takes for the flow to become fully turbulent. The default values in this namelist determining the domain extents are based on that paper. The default cell counts here are slightly coarser than their work which used 64, 20, and 80 grid cells in the respective x , r , and θ directions.

```
&nml_testcase_tcf
  Inner_Radius      = 0.5      ! [m]
  Inner_Wall_Spacing = 0.002    ! [m]
  Outer_Radius      = 1.0      ! [m]
  Outer_Wall_Spacing = 0.0025   ! [m]
  Aspect_Ratio      = 6.2832    ! 2*pi
  Axial_Cells       = 24
  Radial_Cells      = 20
  Azimuthal_Cells   = 48
  Theta_Shift       = 0.0      ! [deg]
  Periodic_Sectors  = 1
/
```

Listing 23: Namelist nml_testcase_tcf.

```
Inner_Radius = 0.5 ! [m]
```

This gives the radius of the inner wall for the Taylor-Couette flow problem.

```
Inner_Wall_Spacing = 0.002 ! [m]
```

This gives the wall spacing at the inner wall for the Taylor-Couette flow problem.

```
Outer_Radius = 1.0 ! [m]
```

This gives the radius of the outer wall for the Taylor-Couette flow problem.

```
Outer_Wall_Spacing = 0.0025 ! [m]
```

This gives the wall spacing at the outer wall for the Taylor-Couette flow problem.

```
Aspect_Ratio = 6.2832 ! 2*pi
```

This is used to specify the axial length, l_x , of the domain for the Taylor-Couette flow problem. The length is determined by the equation:

$$l_x = \text{Aspect_Ratio} \times |\text{Outer_Radius} - \text{Inner_Radius}|$$

```
Axial_Cells = 24
```

This specifies the number of axial cells to use for the Taylor-Couette flow problem.

```
Radial_Cells = 20
```

This specifies the number of radial cells to use for the Taylor-Couette flow problem.

```
Azimuthal_Cells = 48
```

This specifies the number of azimuthal cells to use for the Taylor-Couette flow problem.

```
Theta_Shift = 0.0 ! [deg]
```

This specifies an angle (in degrees) to shift the grid in theta for the Taylor-Couette flow problem. The purpose of this input is to rotate the grid so that a cell boundary is not located at the angle 0/360 degrees.

```
Periodic_Sectors = 1
```

This input can be used to run only a sector of the Taylor-Couette flow problem. This value specifies how many pie slices to divide the grid into, with the final grid domain consisting of only one pie slice.

11.19 nml_testcase_vortex

This **namelist block** is used to specify inputs for the vortex test cases. These cases include the different versions of the isentropic Euler vortex problem (**Initialization** = $[\pm 5, \pm 55, \pm 6]$), the density profile advection case (**Initialization** = 7), the entropy profile advection case (**Initialization** = 8), and the Taylor-Green vortex problem (**Initialization** = 9).

```
&nml_testcase_vortex
  Vortex_Bigr          = 1.0    ! [m]
  Vortex_Biggam        = 5.0    ! [unitless]
  Grid_Lref            = 10.0   ! [m]
  Grid_Random_Perturb  = F
  Grid_Perturb_Percent = 30
  Grid_Hex_To_Pyramids = F
  Grid_Is_3d           = F
/
```

Listing 24: Namelist nml_testcase_vortex.

Vortex_Bigr = 1.0 ! [m]

Constant used to specify the size or characteristic length (in meters) of the isentropic Euler vortex.

Vortex_Biggam = 5.0 ! [unitless]

Constant used to specify the strength of the isentropic Euler vortex.

Grid_Lref = 10.0 ! [m]

Reference length (in meters) for the grid domain, i.e., the grid is a square with bounds of $[-\text{Grid_Lref}, +\text{Grid_Lref}]$ in both the x - and y -directions.

Grid_Random_Perturb = F

Logical flag for applying a random perturbation to all the interior grid nodes for the isentropic Euler vortex and Taylor-Green vortex problems.

Grid_Perturb_Percent = 30

Percent of the uniform grid cell length to perturb the interior grid nodes if **Grid_Random_Perturb** is true.

Grid_Hex_To_Pyramids = F

When using the internal grid generator with one of the 3D vortex problems, the default is to use hex cells. Setting this to true will create a grid using only pyramid cells, by first creating the hex-only grid, and then subdividing each hex cell into 6 pyramids by adding a new node point in the center of the cell.

<code>Grid_Is_3d = F</code>

When running one of the traditionally 2D vortex cases (e.g., isentropic vortex), setting this to true will run a 3D version of the case by extruding the domain and initial solution in z to create a cube domain.

11.20 nml_interpolation

This **namelist block** is for use with the startup interpolation utility, which will interpolate a solution from 1) a set of Plot3d grid and solution files or 2) any restart file and respective grid file onto a new (and possibly unstructured) grid file.

```
&nml_interpolation
  Interpto_Grid_Scaling_Factor      = 1.0
  Interpto_Grid_Format               = -99    ! -99 = Unknown
  Interpto_Grid_File                 = ""
  Interpto_Plot3d_Cutfile            = ""
  Interpto_Plot3d_BCSfile            = ""
  Interpto_Plot3d_Stride_I           = -1
  Interpto_Plot3d_Stride_J           = -1
  Interpto_Plot3d_Stride_K           = -1
  Interpto_Plot3d_Stride_All         = -1
  Interpto_Plot3d_Agglomerate_Order = -1
  Interpto_Solution_Order            = 3
/
```

Listing 25: Namelist nml_interpolation.

Each input has the same functionality as the corresponding input (without the **Interpto_** prefix) from the **nml_grid** namelist block of Section 11.2 or the **nml_numerics** namelist block of Section 11.8. However, those inputs refer to the grid that the old solution is being interpolated *from*, whereas these inputs refer to the grid that is being interpolate *to*.

NOTE: The user must provide a valid **Interpto_Grid_Format**, since the default value of -99 will cause the code to terminate.

11.21 nml_utility

This **namelist block** is used to switch from solver mode to utility mode. **NOTE:** The inputs in this block perform post-processing tasks without running the flow solver.

```
&nml_utility
  Convert_Restart_To_CGNS           = F
  Delete_Converted_Restart_Files    = F
  Restart_List_File                 = ""
  Spatial_Ave_Restart_Channel        = F
  Spatial_Ave_Restart_X_And_Z_Planes = F
  Spatial_Ave_Restart_Z_Planes       = F
/
```

Listing 26: Namelist nml_utility.

Convert_Restart_To_CGNS = F

Flag used to read in a restart file and immediately write a CGNS solution file, bypassing the solver entirely. This first reads the instantaneous restart file from **Restart_File**, computes the requested output variables from **nml_output_variables** in Section 11.12, and writes them to a CGNS solution file. Next it reads the time-averaged restart file from **Input_Time_Ave_File**, and writes the accumulated data to **CGNS_files/solution.#.ta_data.cgns** and the derived statistics to **CGNS_files/solution.#.time_ave.cgns**. Lastly, it reads the list of instantaneous restart files from **Restart_List_File** and processes each one to compute the requested output variables from **nml_output_variables** in Section 11.12, and writes them to a CGNS solution file. Finally, it will write the file **CGNS_files/solution.time.cgns** to link all the instantaneous solution files together into a single time-history solution file.

The option **Delete_Converted_Restart_Files** controls whether the restart files are deleted after the CGNS output is complete.

Delete_Converted_Restart_Files = F

Specifies whether to delete each restart file from **Restart_List_File** after it has been processed by **Convert_Restart_To_CGNS**. This can be used to minimized disk space usage. The **Restart_File** and **Input_Time_Ave_File** will *not* be deleted after conversion to CGNS.

WARNING: This will permanently delete these files from the file system.

Restart_List_File = ""

File path to a simple ASCII file that contains a list of file paths to existing instantaneous restart files for use with **Convert_Restart_To_CGNS**. Each line of the list file should contain the name of a restart file. If the text on a line does not give a valid path to an existing file, it will ignore the line and continue with the next line. See also **Delete_Converted_Restart_Files**.

Spatial_Ave_Restart_Channel = F

This is essentially a specialized version of **Spatial_Ave_Restart_X_And_Z_Planes** for the channel flow problem where the 1D y profile is normalized by the channel half-height. It is used to post-process the time-averaged restart file (**Input_Time_Ave_File**).

```
Spatial_Ave_Restart_X_And_Z_Planes = F
```

Utility to read the time-averaged restart file (**Input.Time.Ave.File**) and spatially average in both the x and z coordinates. This option first produces the z -averaged output from **Spatial_Ave_Restart_Z_Planes**, then additionally averages in the x -direction and outputs a Tecplot ASCII data file containing a 1D line with the y grid coordinates and the x - z -averaged flow variables (e.g., a hex mesh formed from a y grid line by extruding in the x direction and then the z direction).

NOTE: This only works for simple cases like channel flows or flat plates where the y -spacing is the same for every x - z coordinate pair.

```
Spatial_Ave_Restart_Z_Planes = F
```

Utility to read the time-averaged restart file (**Input.Time.Ave.File**) and spatially average in the z coordinates. The output will be a Tecplot ASCII data file containing a 2D plane with the x - y grid coordinates and the z -averaged flow variables.

NOTE: This only works for simple cases like channel flows or flat plates where the 2D x - y grids are identical for all z -planes (e.g., a structured or unstructured 2D planar x - y grid that is extruded in the z direction).

11.22 nml_debug

This **namelist block** is used by developers for debugging.

```
&nml_debug
  Output_Limiter_Statistics      = F
  Output_Filter_Diagonals       = F
  Output_Plot3d_To_Tecplot      = F
  Output_Bface_Array            = F
  Dump_Memory_Usage             = F
  Check_Flux_Point_Coordinates  = F
  Force_Nd_Subroutines          = F
  Newton_Convergence_Goal       = 1.0e-12
  Chemistry_Sub_Iterations       = -1
  Traceback_Unconverged_Newton_Temps = F
  Ignore_Thermo_Newton_Warnings = F
/
```

Listing 27: Namelist nml_debug.

Output_Limiter_Statistics = F

This outputs the number of limited/projected cells each time step when **Local_Limiter_Option** is set to use projection.

Output_Filter_Diagonals = F

Specifies whether to output text files containing the filter diagonals and filter matrices for all geometry types after they are generated in the preprocessor. This is used to inspect values for possible errors in the filter definition.

Output_Plot3d_To_Tecplot = F

Specifies whether to output a Tecplot file of the input Plot3D grid file after reading it. This can be used to inspect whether the grid file was read correctly.

Output_Bface_Array = F

Specifies whether to have each MPI process output the contents of its own local bface array to the text file “bface.[MPI process #].dat”. The bface array contains the boundary conditions and connectivity information for each boundary face.

Dump_Memory_Usage = F

Specifies whether to profile and output the memory usage of GFR.

Check_Flux_Point_Coordinates = F

Flag used to output files to check that co-located face quantities are consistent in terms of interpolation between two cells sharing a face.

`Force_Nd_Subroutines = F`

GFR normally uses optimized 1D, 2D, and 3D versions of various subroutines. This input forces the use of the generalized “nD” versions of the various subroutines that loop over the number of equations instead of hardcoding in each equation.

`Newton_Convergence_Goal = 1.0e-12`

This specifies the convergence goal for all of the Newton iterations on temperature.

`Chemistry_Sub_Iterations = -1`

Number of chemistry sub-iterations to use for the KIVA scheme source terms. **NOTE:** The number of sub-iterations is set to 10 if `Chemistry_Sub_Iterations < 1`.

`Traceback_Unconverged_Newton_Temps = F`

If running a multi-component gas mixture, setting this input to true will write a traceback to stdout whenever the Newton solve for temperature fails to meet the convergence goal.

NOTE: This only works if using the Intel Fortran compiler and the namelist input variable `Ignore_Thermo_Newton_Warnings` is set to false.

`Ignore_Thermo_Newton_Warnings = F`

If running a multi-component gas mixture, setting this to true will prevent any warnings from being written to stdout whenever the Newton solve for temperature fails to converge.

11.23 Changes from Prior Version

Many new features have been added since the prior version, including:

- Mixed unstructured elements are now fully functional, Section 3.2.
- Rotating reference frame, Section 4.
- A number of new time stepping methods, Section 8.
- Unsteady turbulent inflow via SEM, Section 5.4.
- The Multidimensional Limiting Process (MLP).
- The `Thermofix%Method` to prevent negative density, temperature, or pressure.
- Test cases for:
 - 3D Taylor-Couette flow
 - 3D channel flow and annular pipe channel flow
 - 1D density pulse advection (Section 12.1)
 - 1D Sod shock tube (Section 12.3)
 - 1D Shu-Osher problem (Section 12.4)

One of the most noticeable changes is the renaming and reorganization of the user input. The code will attempt to convert an older input file to the current format. However, should you need to manually convert an older input file, the table below provides a mapping between the old and new inputs.

NOTE: An * means an equivalent input does not exist in the old namelist.

Old Namelist Old Variable Description	New Namelist New Variable
&input	&nml_version
*	<code>Namelist_Version</code>
&input	&nml_grid
Grid_Scaling_Factor	<code>Same</code>
Grid_Format	<code>Same</code>
Gridfile	<code>Grid_File</code>
Plot3d_Cutfile	<code>Same</code>
Plot3d_BCSfile	<code>Same</code>
Plot3d_I_Stride	<code>Plot3d_Stride_I</code>
Plot3d_J_Stride	<code>Plot3d_Stride_J</code>
Plot3d_K_Stride	<code>Plot3d_Stride_K</code>
Plot3d_All_Strides	<code>Plot3d_Stride_All</code>
Plot3d_Agglomerate_Order	<code>Same</code>
Correct_CGNS_Grid_Points	<code>Same</code>
Flame_Speed_Test_DoF	<code>Internal_Grid_DoF</code>
Pulse_Test_DoF	<code>Internal_Grid_DoF</code>

Old Namelist	New Namelist
Old Variable	New Variable
Description	
Shu_Osher_Test_DoF	Internal_Grid_DoF
Sod_Test_DoF	Internal_Grid_DoF
VortexGrid_DoF	Internal_Grid_DoF
Internal_Grid_Order	Same
Metis_Option	Same
Geom.Wts(1)%Tria	Same
Geom.Wts(1)%Quad	Same
Geom.Wts(1)%Tetr	Same
Geom.Wts(1)%Pyra	Same
Geom.Wts(1)%Pris	Same
Geom.Wts(1)%Hexa	Same
&input	&nml_reference
Machref	Mach
Reyref	Reynolds
Gam	Gamma
Rhoref	Density
Tref	Temperature
Pref	Pressure
Rgasref	Gas_Constant
Alpha_AoAref	AoA_Alpha
Beta_AoAref	AoA_Beta
Pr	Prandtl
Schmidt_Lam	Schmidt
Suth_Muref	Same
Suth_Tref	Same
Suth_Sref	Same
&input	&nml_physics
Governing_Equations	Same
Moving_Ref_Frame%Rotation(1:3)	Same
Moving_Ref_Frame%Translation(1:3)	Same
Moving_Ref_Frame%Spinup%Iter	Same
Moving_Ref_Frame%Spinup%Tstar	Same
&input	&nml_initialization
Itestcase	Initialization
Jet_Init_Func	Same
Load_Restart_File	Same
Restart_File	Same
Initial_Solution_File	Solution_File
Initial_Solution_Format	Solution_Format
&input	&nml_timestep
Runge_Kutta_Scheme	Same
Minimum_Timesteps	Num_Timesteps_Min
Num_Timesteps	Same
Final_Time	Same
Timestep_Type	Same

Old Namelist	New Namelist
Old Variable	New Variable
Description	
Constant_Dt	Timestep
CFL	Same
CFL_Beg	CFL_Ramping_Beg
CFL_End	CFL_Ramping_End
CFL_Cycles	CFL_Ramping_Iterations
CFL_Start_Iter	CFL_Ramping_Start_Iter
Convergence_Order_Abs_Res	Same
Convergence_Order_Max_Res	Same
&input	&nml_bc
Walls_Are_Exact	Same
Sub_Inflow_Method	Same
Use_BC_Conflicts	Same
BC_Input(1)	
%Name	Same
%Set_BC_Value	Same
%T_Static	%Temperature
%P_Static	%Pressure
%Rho_Static	%Density
%Mach	Same
%Vx	%VelocityX
%Vy	%VelocityY
%Vz	%VelocityZ
%T_Total	%Total_Temperature
%P_Total	%Total_Pressure
%Rho_Total	%Total_Density
%Alpha_AoA	%AoA_Alpha
%Beta_AoA	%AoA_Beta
%Wall_Temp	Same
%Rotation_Rate(1:3)	Same
%Is_Specified_In_MRF	Same
%Mass_Frac(1:7)	Same
%Species_Name(1:7)	Same
%Relax%Value	Same
%Relax%Time	Same
%Relax%Iter	Same
%SEM%Profile_File	Same
%SEM%Is_Cylindrical	Same
%SEM%Length_Scale	Same
%SEM%Convective_Velocity	Same
%SEM%Profile_Height	Same
%SEM%Sub_In_BC_Grp	Same
%SEM%Sub_In_BC_Idx	Same
%Periodic%Matching_BC_Name	Same
%Periodic%Matching_BC_Idx	Same
%Periodic%RotationCenter(1:3)	Same

Old Namelist	New Namelist
Old Variable	New Variable
Description	
%Periodic%RotationAngle(1:3)	Same
%Periodic%RotationAxis(1:4)	Same
%Periodic%Translation(1:3)	Same
&input	&nml_numerics
All_Orders	Same
Solution_Order	Same
Projection_Order	Same
Quadrature_Order	Same
Error_Order	Same
All_Points	Same
Loc_Solution_Pts	Same
Loc_Flux_Pts	Same
Loc_Quadrature_Pts	Same
Loc_Error_Pts	Same
Loc_Triangle_Pts	Same
Loc_Tetrahedra_Pts	Same
Invs_Flux_Method	Same
Visc_Flux_Method	Same
Visc_Variables_To_Ave	Same
Combine_Common_Flux_Projections	Same
Correction_Function	Same
&input	&nml_limiter
Filtering_Interval	Global_Filter_Interval
Filter_Option	Global_Filter_Option
Filter_Order	Global_Filter_Order
Filter_Alpha	Global_Filter_Alpha
Filter_Etac	Global_Filter_Etac
Limiting_Interval	Local_Limiter_Interval
Limiter_Option	
The old Limiter_Option input controlled both the marker and limiter. Local_Limiter_Option now controls only the limiter, and Marker_Option now controls the marker. Both are shown below.	
Limiter_Option	Local_Limiter_Option
-2 Jump Marker + Scaled Projection	0 None
-1 Res. Marker + Scaled Projection	1 Filtering
0 None	2 Scaled Filtering
1 Res. Marker + Limiter_Filter_Option	3 MLP
if (Limiter_Filter_Option == 0) then	9001 Recursive Projection
use Recursive Projection	9002 Scaled Projection
2 Jump Marker + Limiter_Filter_Option	
3 MLP Marker + MLP limiter	

Old Namelist Old Variable Description	New Namelist New Variable
Limiter_Option See description above.	Marker_Option 0 Default based on Local_Limiter_Option 1 Resolution marker 2 Jump marker 3 MLP marker
Limiter_Use_Dynamic_Filter Input value has a different meaning. Dynamic filtering now called scaled filtering.	Local_Limiter_Option
Limiter_Filter_Order	Local_Filter_Order
Limiter_Filter_Alpha	Local_Filter_Alpha
Limiter_Filter_Etac	Local_Filter_Etac
*	Sponge_Filter_Interval
Use_Sponge_Filter	Sponge_Filter_Option
F Off	0 Off
T On	1 On
Sponge_Filter_Order	Same
Sponge_Filter_Alpha	Same
Sponge_Filter_Etac	Same
Sponge_Beyond_X	Sponge_Filter_X_Start
Resolution_Marker_Variable	Marker_Variable
0 All conserved variables	-2 Ducros (velocity divergence)
1:5 Specific conserved variable	-1 Pressure 0 Default based on Marker_Option 1:5 Specific conserved variable 6 All conserved variables
Jump_Marker_Variable	Marker_Variable
0 Pressure	-2 Ducros (velocity divergence)
1:5 Specific conserved variable	-1 Pressure 0 Default based on Marker_Option 1:5 Specific conserved variable 6 All conserved variables
MLP_Marker_Variable	Marker_Variable
0 Density	-2 Ducros (velocity divergence)
1:5 Specific conserved variable	-1 Pressure 0 Default based on Marker_Option 1:5 Specific conserved variable 6 All conserved variables
Limiter_Marker_Min	Marker_Min
Limiter_Marker_Max	Marker_Max
Limiter_Marker_Dpsi	Marker_Dpsi
Limiting_Interval_Output	Marker_Stored_Interval
Thermofix%Method	Same
Thermofix%Iterations	Same
Thermofix%Check_Cells	Same
Thermofix%Check_Faces	Same

Old Namelist	New Namelist
Old Variable	New Variable
Description	
Thermofix%Check_Nodes	Same
Thermofix%Verbose	Same
Thermofix%Scale_Factor	Same
Thermofix%Min_Pressure	Same
Thermofix%Min_Temperature	Same
Thermofix%Min_Density	Same
&input	&nml_time_ave
Output_Time_Averaging	Compute_Time_Averaging
Num_Saved_Time_Ave_Restarts	Same
Time_Ave_File	Input_Time_Ave_File
Time_Ave_Vel_Is_Axisymm	Same
Start_Averaging_Timestep	Same
Start_Averaging_Tstar	Same
&input	&nml_output
Time_Scaling_Factor	Same
Restart_Interval	Same
Restart_Frequency	Same
Restart_Output_Uses_Host_Roots	Same
Completely_Disable_Restart_Output	Restart_Output_Disabled
Iter_Out_Interval	Display_Interval
Results_Interval	Residual_Interval
Output_Summary_File	Same
Output_Dir	Same
Output_Interval	Same
Output_Frequency	Same
Output_Order	Same
Dmat_Grad_Temp_Order	Same
Loc_Output_Pts	Same
CGNS_High_Order_Output	Same
CGNS_Mixed_Output	Same
Completely_Disable_CGNS	CGNS_Output_Disabled
Probes_Collection_Interval	same
Probes_Write_Interval	same
Point_Probes	same
Rake_Probes	same
Profile_IO_All	Same
Profile_IO_Restart	Same
Profile_IO_CGNS	Same
Profile_IO_CGNS_Subs	Same
Lustre_Stripe_Count	Same
Lustre_Stripe_Size	Same
&input	&nml_output_variables
Output_Conserved_Variables	Conserved_Variables
Output_Momentum	Momentum
Output_Vorticity	Vorticity

Old Namelist	New Namelist
Old Variable	New Variable
Description	
Output_Cylindrical_Velocity	Cylindrical_Velocity
Output_Cylindrical_Coordinates	Cylindrical_Coordinates
Output_Relative_Velocity	Relative_Velocity
Output_Relative_Conditions	Relative_Conditions
Output_Thermo_Variables	Thermo_Variables
Output_Temperature	Temperature
Output_Gradients	Gradients
Output_Transport_Variables	Transport_Variables
Output_Stagnation_Variables	Stagnation_Variables
Output_Stagnation_Pressure	Stagnation_Pressure
Output_Stagnation_Temperature	Stagnation_Temperature
Output_Marker_Variables	Marker_Variables
Output_Qcriterion	Qcriterion
Output_Species_Mass_Fractions	Species_Mass_Fraction
Output_Species_Density	Species_Density
Output_Species_Mole_Fractions	Species_Mole_Fraction
Output_Extra_Species_Variables	Extra_Species_Variables
Output_Debug_Species_Variables	Debug_Species_Variables
Output_Grnt_Species_Variables	Grnt_Species_Variables
&input	&nml_sem
Use_SEM_Profile_For_Initialization	Use_Profile_For_Initialization
SEM_Use_Random_Seed_From_Restart	Use_Random_Seed_From_Restart
SEM_Eddy_Volume_Is_Spherical	Use_Spherical_Eddy_Volume
SEM_Include_Thermo_Fluctuations	Include_Thermo_Fluctuations
SEM_Box_Num_Eddies_Multiplier	Box_Num_Eddies_Multiplier
&input	&nml_chemistry
Disable_Chemical_Reactions	Same
Semi_Implicit_Chemistry	Same
Ignition_Method	Same
Ignition_Start_Timestep	Same
Ignition_Duration	Same
Ignition_Prod2Reac_L2_Ratio_Goal	Same
Ignition_Max_Temperature	Same
Ignition_Spark_Cap	Same
Ignition_Source_Magnitude	Same
Ignition_Source_Radius	Same
Ignition_Source_Location	Same
&input	&nml_testcase_channel
Pipe_Channel_Angle	Grid_Annulus_Angle
Channel_Grid_Ave_Xplus	Grid_Ave_Xplus
Channel_Grid_Ave_Zplus	Grid_Ave_Zplus
Channel_Grid_Ave_Yplus	Grid_Ave_Yplus
Channel_Grid_Yplus_Wall	Grid_Yplus_Wall
Channel_Yplus_Of_One	Grid_Yplus_Of_One
Channel_Body_Force	Body_Force

Old Namelist Old Variable Description	New Namelist New Variable
Channel_Nondim_Output_Coord	Nondim_Output_Coord
&input	&nml_testcase_pulse
Pulse_Test_Form	Form
Pulse_Test_Form_Dim	Form_Dim
Pulse_Test_Length	Length
Pulse_Test_Center	Center
Pulse_Test_Width	Width
Pulse_Test_Gap	Gap
Pulse_Test_Repeat	Repeat
&input	&nml_testcase_sod
Sod_Test_Right_Vector	Right_Vector
&input	&nml_testcase_tcf
Tcf_Inner_Radius	Inner_Radius
Tcf_Inner_Wall_Spacing	Inner_Wall_Spacing
Tcf_Outer_Radius	Outer_Radius
Tcf_Outer_Wall_Spacing	Outer_Wall_Spacing
Tcf_Aspect_Ratio	Aspect_Ratio
Tcf_Axial_Cells	Axial_Cells
Tcf_Radial_Cells	Radial_Cells
Tcf_Azimuthal_Cells	Azimuthal_Cells
Tcf_Theta_Shift	Theta_Shift
Tcf_Periodic_Sectors	Periodic_Sectors
&input	&nml_testcase_vortex
Vortex_Bigr	Same
Vortex_Biggam	Same
VortexGrid_Lref	Grid_Lref
VortexGrid_Random_Perturb	Grid_Random_Perturb
VortexGrid_Perturb_Percent	Grid_Perturb_Percent
VortexGrid_Hex_To_Pyramids	Grid_Hex_To_Pyramids
VortexGrid_Is_3d	Grid_Is_3d
&input	&nml_interpolation
Interpto_Grid_Scaling_Factor	Same
Interpto_Grid_Format	Same
Interpto_Gridfile	Interpto_Grid_File
Interpto_Plot3d_Cutfile	Same
Interpto_Plot3d_BCSfile	Same
Interpto_Plot3d_I_Stride	Interpto_Plot3d_Stride_I
Interpto_Plot3d_J_Stride	Interpto_Plot3d_Stride_J
Interpto_Plot3d_K_Stride	Interpto_Plot3d_Stride_K
Interpto_Plot3d_All_Strides	Interpto_Plot3d_Stride_All
Interpto_Plot3d_Agglomerate_Order	Same
Interpto_Solution_Order	Same
&input	&nml_utility
Convert_Restart_To_CGNS	Same
Delete_Converted_Restart_Files	Same

Old Namelist	New Namelist
Old Variable	New Variable
Description	
Restart_List_File	Same
Channel_Spatial_Ave_Restart	Spatial_Ave_Restart_Channel
Spatial_Ave_Restart_X_And_Z_Planes	Same
Spatial_Ave_Restart_Z_Planes	Same
&input	&nml_debug
Output_Limiter_Statistics	Same
Output_Filter_Diagonals	Same
Output_Plot3d_To_Tecplot	Same
Output_Bface_Array	Same
Dump_Memory_Usage	Same
Check_Flux_Point_Coordinates	Same
Force_Nd_Subroutines	Same
Conv_Temp	Newton_Convergence_Goal
Chemistry_Sub_Iterations	Same
Traceback_Unconverged_Newton_Temps	Same
Ignore_Thermo_Newton_Warnings	Same
The variables below have been removed from the input namelist.	
&input	&nml_removed
Parametric_Agglomeration	Same
Interpto_All_Orders	Same
Interpto_Projection_Order	Same
Interpto_Quadrature_Order	Same
Interpto_Output_Order	Same
Interpto_Error_Order	Same
Prt	Prandtl_Turb
Schmidt_Turb	Same
Tinref	Turb_Intensity
Tvrref	Turb_Viscosity
Turbulence_Model	Same
Newton_Rlx	Newton_Relax
Use_Improved_SEM_Normalization	Use_Improved_Initialization
Average_SEM_Inflow_Plane	Average_Inflow_Plane
MMS_Opt	Option
MMS_Init	Initialization
MMS_Output_Solution	Output_Solution
Loc_Init_Pts	Same
Use_Old_Restart_Format	Same
CFL_Reduction_For_Nan_Restart	Same
Use_Unrolled_Dot_Products	Same
Flux_Divergence	Same
Pyramid_Filter_Option	Same
Ideal_Total_Temperature	Same
Limit_Total_Temperature	Same
Filter_Eps	Global_Filter_Eps

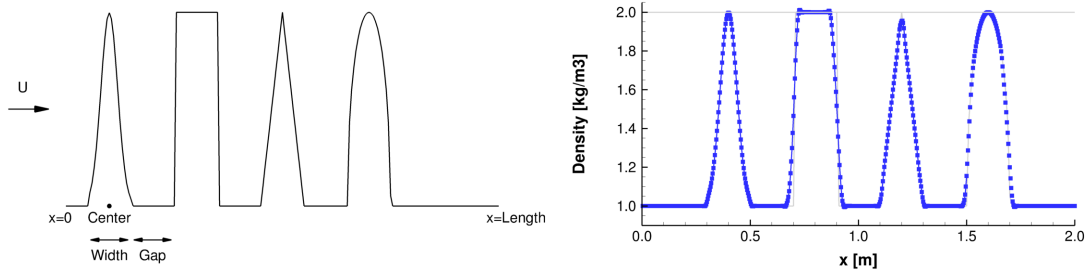
Old Namelist	New Namelist
Old Variable	New Variable
Description	
Limiter_Filter_Eps	Local_Filter_Eps
Limiter_Marker_C1	Marker_C1
Limiter_Marker_C2	Marker_C2
Output_Eperk_CFL	Same
Disable_Header_Rewrites	Same
CGNS_Use_Queue	Same
Use_Async_For_Error	Same
Output_Bnd_Profiles	Same
Channel_Init_File	Init_File
Time_Average_Restart_Files	Same
Postproc_Debug_Grid	Same
Dump_Fluxes	Same
Channel_Rake_Save_Interval	Rake_Save_Interval
Track_Limiter_Filter_Dissipation	Local_Filter_Dissipation
Continuous_Output	Same
Interpolate_Before_Output_Variable	Same
Num_Rk_Stages	Runge_Kutta_Stages
Geom_Wts(1)%Edge	Same
Geom_Wts(1)%Face_Tria	Same
Geom_Wts(1)%Face_Quad	Same

12 Test Cases

A number of test cases are provided with the source code to help users become familiar with how the code works. These range from simple one-dimensional problems to three-dimensional large-eddy simulations, and are discussed in the following sections:

- 1D Density Pulse Advection
- 1D Jiang-Shu Density Pulse Advection
- 1D Sod Shock Tube
- 1D Shu-Osher Problem
- 2D Inviscid Bump
- 2D Isentropic Vortex
- 2D Couette Flow

12.1 1D Density Pulse Advection



(a) Initial condition.

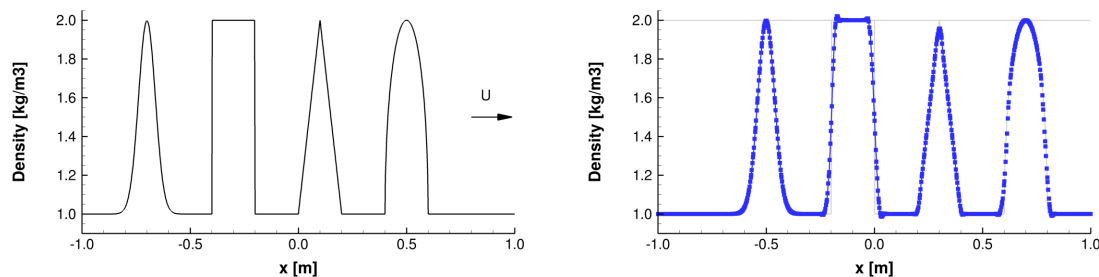
(b) $\mathcal{P}3$ result after advecting one pulse width.

Figure 21: Density pulse test case.

This test case features the inviscid advection of a density pulse, and can be used to assess the numerical dissipation of the scheme. Fig. 21a shows the starting density profile and the available user input parameters for adjusting the center, width, and gap of the profile as well as the domain length. Gauss, square, triangular, and elliptical pulse shapes with an amplitude of 1.0 kg/m³ can be superimposed onto the reference flow conditions. However, the combination pulse is recommended because it shows the effect on all the profiles shapes in one simulation. Periodic boundary conditions are applied. Fig. 21b shows a sample $\mathcal{P}3$ result using the resolution marker to identify troubled cells and applying a filter with `Local_Filter_Order` = 4 to damp oscillations.

See Also: `Initialization=24`, `Internal_Grid_DoF`, `nml_testcase_pulse`

12.2 1D Jiang-Shu Density Pulse Advection



(a) Initial condition.

(b) $\mathcal{P}3$ result after advecting one pulse width.

Figure 22: Jiang-Shu density pulse test case.

This test case features the inviscid advection of a density pulse, and can be used to assess the numerical dissipation of the scheme. Fig. 22a shows the starting density profile and the domain length. Both the reference conditions and the pulse shapes are preset within the code. Gauss, square, triangular, and elliptical pulse shapes with an amplitude of 1.0 kg/m^3 are superimposed onto the reference flow conditions. These pulse shapes differ slightly from those described in Section 12.1. Periodic boundary conditions are applied. Fig. 22b shows a sample $\mathcal{P}3$ result using the resolution marker to identify troubled cells and applying a filter with `Local_Filter_Order` = 4 to damp oscillations.

See Also: `Initialization=27`, `Internal_Grid_DoF`, `Exact_Density`, `Exact_Pressure`

12.3 1D Sod Shock Tube

The Sod shock tube is a simple case for evaluating shock capturing. When using the internally generated grid, the domain will be 1.0 meter long (centered at the origin) and the left/right boundary conditions will be set to extrapolation. For the initial condition at time $t=0$, imagine that there is an invisible diaphragm in the center of the tube as shown in Fig. 23a. The flow on the left side is set to 1 Pa and 1 kg/m^3 , and the flow on the right has $1/10^{\text{th}}$ the pressure and $1/8^{\text{th}}$ the density. Neither side has any initial velocity.

Once the simulation begins, the diaphragm is removed and the high-pressure fluid on the left moves to the right. This forms a shock wave moving to the right. Somewhat behind the shock wave, there is a contact discontinuity in density that also moves to the right. Meanwhile an expansion fan forms on the left side and moves to the left. Fig. 23b shows a sample $\mathcal{P}3$ result on a grid of 100 cells using the resolution marker to identify troubled cells and applying a filter with `Local_Filter_Order` = 4 to damp oscillations.

See Also: `Initialization=23`, `Internal_Grid_DoF`, `nml_testcase_sod`

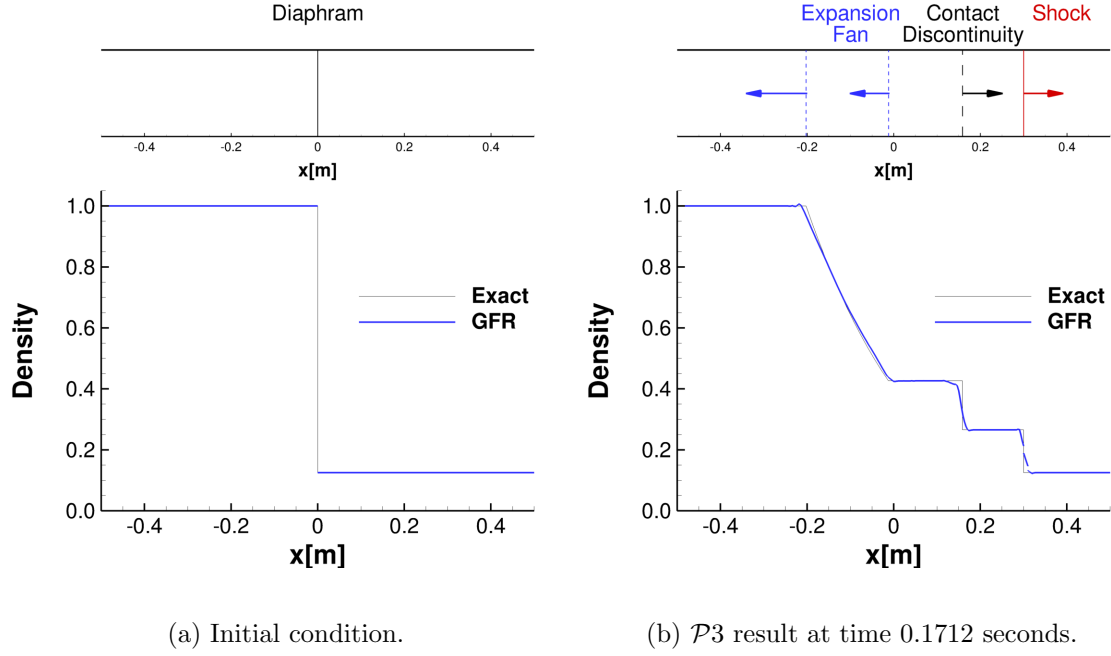


Figure 23: Sod shock tube test case.

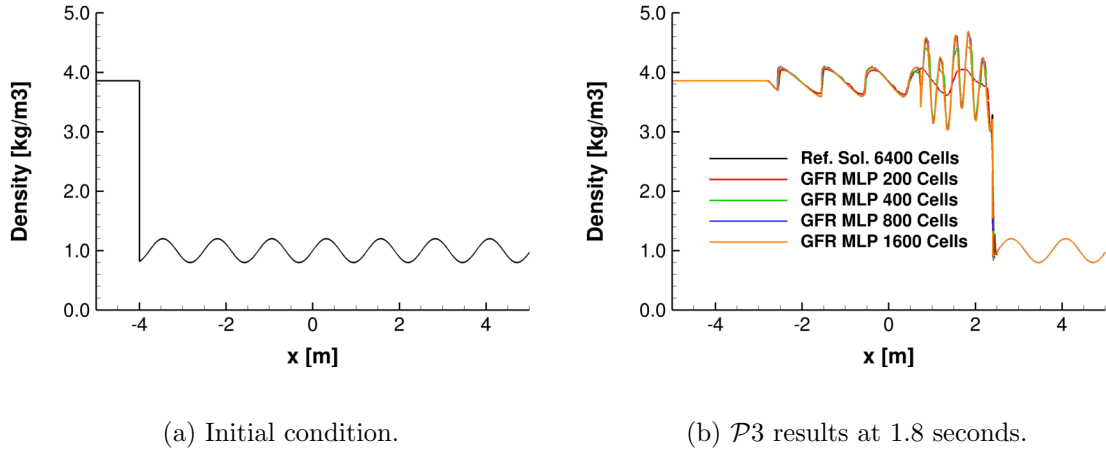


Figure 24: Shu-Osher test case.

12.4 1D Shu-Osher Problem

In this problem, a right-moving Mach 3 shock interacts with a field of small density (or correspondingly entropy) disturbances. The initial conditions (in SI units) are given as

$$\rho = 3.857143 \quad u = 2.629369 \quad p = 10.333333 \quad \text{when } x < -4 \quad (81)$$

$$\rho = 1 + \epsilon \sin(5x) \quad u = 0 \quad p = 1 \quad \text{when } x \geq -4 \quad (82)$$

with $\epsilon = 0.2$, and are shown in Fig. 24a. $\mathcal{P}3$ solutions on various grid levels using the multidimensional limiting process (MLP) are shown in Fig. 24b at time 1.8 seconds. There is no exact solution for this problem, so the results are compared with those obtained from the discontinuous Galerkin solver provided in Ref 39.

In order to prevent negative density and/or pressure, the following limiter options are used.

```
&nml_limiter
  Global_Filter_Interval    = 0      ! 0=Each Runge-Kutta stage
  Global_Filter_Option     = 0      ! 0=Off
  Local_Limiter_Interval   = 0      ! 0=Each Runge-Kutta stage
  Local_Limiter_Option     = 3      ! 3=MLP
  Marker_Stored_Interval   = 0      ! 0=Each Runge-Kutta stage
  Marker_Option            = 3      ! 3=MLP marker
  Marker_Variable          = 0      ! 0=Default=Density
  Thermofix%Method         = 2      ! 2=Scale higher modes
  Thermofix%Scale_Factor   = 0.5
  Thermofix%Min_Pressure   = 0.750 ! [Pa]
  Thermofix%Min_Temperature = 0.002 ! [K]
  Thermofix%Min_Density    = 0.500 ! [kg/m3]
/
```

See Also: [Initialization=26](#), [Internal_Grid_DoF](#)

12.5 2D Inviscid Bump

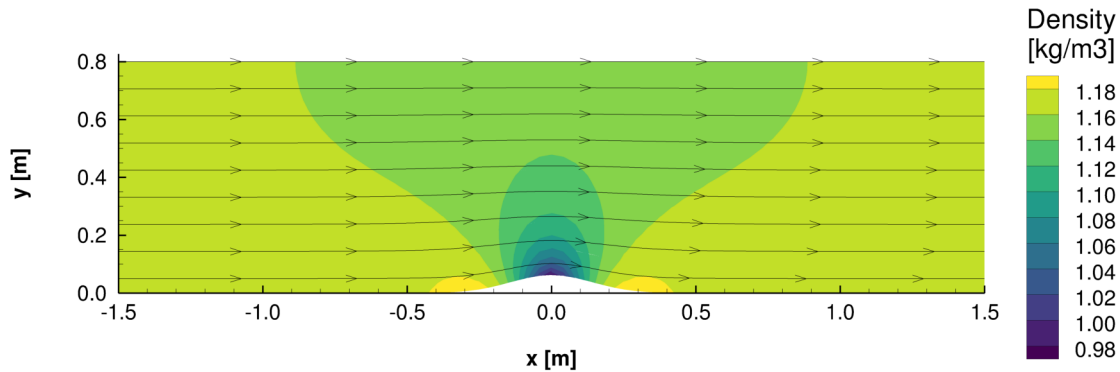


Figure 25: Inviscid Gaussian bump test case, $\mathcal{P}3$ solution.

This test case involves the subsonic flow over a smooth Gaussian bump in a channel, and is aimed at testing high-order methods for the computation of internal flow with a high-order curved boundary representation. In this subsonic flow problem, the geometry is smooth and symmetric in x , and so should the flow properties.

This test case can be used to experiment with how the grid order affects solution accuracy for various solution orders.

All of the provided grid files contain the same number of grid cells (256), and the names of the grid files refer to the order of the cells contained in the grid. The 'bump.P1.msh' grid contains linear ($\mathcal{P}1$) cells, the 'bump.P2.msh' grid contains quadratic ($\mathcal{P}2$) cells, the 'bump.P3.msh' grid contains cubic ($\mathcal{P}3$) cells, and the 'bump.P4.msh' grid contains quartic ($\mathcal{P}4$) cells. Fig. 25 shows the result on the $\mathcal{P}3$ grid.

See Also: [Initialization=3](#)

12.6 2D Isentropic Vortex

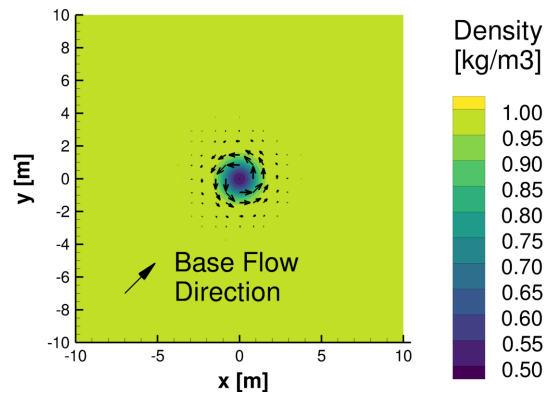


Figure 26: Isentropic Euler vortex test case.

This test case simulates the long term transport of a vortex by a uniform base flow over a 2D doubly periodic domain, as shown in Fig. 26. It is aimed at testing a high-order method's capability to preserve vorticity in an unsteady inviscid flow.

The flow solver will internally generate a uniform grid for this case if `Grid.Format=-1`. Other parameters for this case are specified through the `nml_testcase_vortex` namelist block. Grid resolution can be adjusted by changing the `Internal_Grid_DoF` input parameter which specifies the degree-of-freedom (DoF) to be applied to each direction. The number of grid cells in each direction is determined as:

$$[\# \text{ of cells}] = \text{nint}(\text{Internal_Grid_DoF}/(\text{Solution_Order} + 1)).$$

For example:

`Internal_Grid_DoF = 200`

`Solution_Order = 5`

`Grid = 33x33 Cells`

This case reaches its final simulation time in about 30 seconds using 52 MPI processes. A detailed validation of the GFR flow solver for the isentropic vortex test case can be found in Ref 11.

See Also: `Initialization=5`, `Internal_Grid_DoF`, `nml_testcase_vortex`

12.7 2D Couette Flow

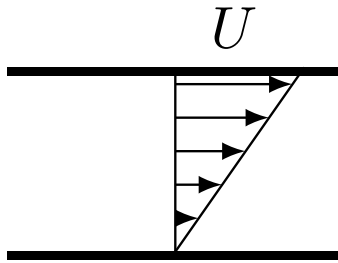


Figure 27: Schematic of Couette flow test case.

Couette flow is a classic problem consisting of viscous flow in a two-dimensional channel between two parallel walls, with one wall moving at a constant velocity and the other wall stationary, as shown in Fig. 27. For the specific case described here, both walls are treated as isothermal with a constant temperature of 300 Kelvin. The top wall has a velocity of 70 m/s.

13 Limitations

In this section, we summarize the limitations of the current implementation. The limitations in this section have been detailed in other parts of this document, but each was deemed significant enough to collect into this section for additional visibility and quick reference.

13.1 Geometry and Mesh

13.1.1 Collapsed Elements

GFR does not support collapsed elements. These are elements with one or more faces that have zero area. A common situation where this occurs is when a quadrilateral planar mesh is rotationally extruded about a singularity axis, often referred to as a pole boundary. In many cases these collapsed elements can be replaced with other supported polygon shapes with fewer faces. However, this is best done within the mesh generation software so that any curvature or high-order information is properly included in the revised elements.

13.2 Synthetic Eddy Method

13.2.1 Inflow Plane

Any SEM inflow boundary must be on a constant x plane. Users should orient their grids to meet this restriction.

13.2.2 Repeatability of Random Seed

Because GFR uses the date/time to generate the initial seed, it is not possible to reproduce the same sequence of random numbers used by the synthetic eddy method. This means that it is not possible to *exactly* reproduce the same results by rerunning a simulation from scratch. However, over a long period, the time-averaged results should be in very close agreement. This would be similar to any differences one might see comparing time-averaged results from one solution using two time-averaging windows of the same duration but shifted in the averaging start time.

13.3 Boundary Conditions

13.3.1 Periodic Boundaries

Opposing periodic boundaries must have identical face elements in order to have one-to-one connectivity between cell faces.

A Polynomials

The following sections summarize the key properties of the polynomial functions used in formulating the Flux Reconstruction method. Additional details can be found in Ref 1.

A.1 Legendre Polynomials

The Legendre polynomial of degree k on the interval $-1 \leq r \leq 1$ is given by the following recurrence relation:^{68,69}

$$P_{-1}(r)=0 \quad P_0(r)=1 \quad P_k(r)=\frac{2k-1}{k}rP_{k-1}(r)-\frac{k-1}{k}P_{k-2}(r) \quad \text{for } k \geq 1 \quad (83)$$

Its derivative can likewise be computed from:

$$P'_{-1}(r)=P'_0(r)=0 \quad P'_k(r)=\frac{2k-1}{k}P_{k-1}(r)+\frac{2k-1}{k}rP'_{k-1}(r)-\frac{k-1}{k}P'_{k-2}(r) \quad \text{for } k \geq 1 \quad (84)$$

Using these formulas, the first few Legendre polynomials and their derivatives can be written explicitly as:

$$P_0(r)=1 \quad P'_0(r)=0 \quad (85)$$

$$P_1(r)=r \quad P'_1(r)=1 \quad (86)$$

$$P_2(r)=(3r^2-1)/2 \quad P'_2(r)=3r \quad (87)$$

$$P_3(r)=(5r^3-3r)/2 \quad P'_3(r)=(15r^2-3)/2 \quad (88)$$

$$P_4(r)=(35r^4-30r^2+3)/8 \quad P'_4(r)=(35r^3-15r)/2 \quad (89)$$

These polynomials are plotted in Fig. 28. Note that each order of the Legendre polynomial alternates between being an even or odd function. That is to say,

$$P_k(-r)=(-1)^k P_k(r) \quad (90)$$

Values and derivatives on the left and right edges of the cell are given by¹ (for $k \geq 0$):

$$P_k(-1)=(-1)^k \quad P_k(1)=1 \quad (91)$$

$$P'_k(-1)=(-1)^{k-1}k(k+1)/2 \quad P'_k(1)=k(k+1)/2 \quad (92)$$

The zeros of the Legendre polynomial are referred to as the Gauss points. As seen in the Fig. 28, the number of such zeros is equal to the order of the polynomial. These points are symmetrically distributed across the cell, but biased toward the edges of the cell. For every odd polynomial order, one of the zeros is located in the center of the cell.

One can also show that the inner product of any two Legendre polynomials of orders (k, l) is given by:

$$(P_k, P_l) = \int_{-1}^1 P_k(r) P_l(r) dr = \begin{cases} 0 & \text{for } k \neq l \\ \frac{2}{2k+1} & \text{for } k = l \end{cases} \quad (93)$$

Therefore, Legendre polynomials of different orders are orthogonal. Also, the inner product of a Legendre polynomial with itself provides a measure of the magnitude of the polynomial, which is useful in DG schemes but not of particular importance for FR.

Using the above relation, one can further show that for $k \geq 1$:

$$(P_0, P_k) = \int_{-1}^1 P_0(r) P_k(r) dr = 0 \quad (94)$$

This indicates that the net area under any Legendre polynomial curve of degree ≥ 1 is zero.

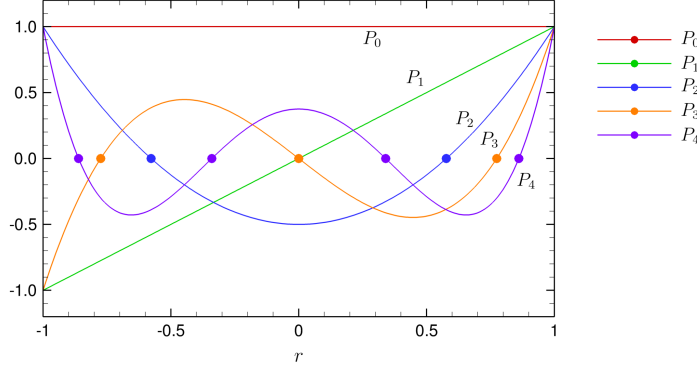


Figure 28: Legendre polynomials and Gauss points.

A.2 Radau Polynomials

The left and right Radau polynomials are defined in terms of Legendre polynomials.

$$\begin{aligned}
 R_{L,k}(r) &= \frac{1}{2} (P_k(r) + P_{k-1}(r)) & R_{L,k}(-1) &= 0 & R_{L,k}(1) &= 1 \\
 R_{R,k}(r) &= \frac{(-1)^k}{2} (P_k(r) - P_{k-1}(r)) & R_{R,k}(-1) &= 1 & R_{R,k}(1) &= 0
 \end{aligned} \tag{95}$$

and are shown in Fig. 2. From Eq. (90), Eq. (95), and Fig. 2, one can see they are mirror images of each other.

$$R_{R,k}(-r) = R_{L,k}(r) \tag{96}$$

We therefore focus on the left Radau polynomials, the first few of which and their derivatives can be written as:

$$R_{L,1}(r) = (r + 1) / 2 \qquad R'_{L,1}(r) = 1/2 \tag{97}$$

$$R_{L,2}(r) = (3r^2 + 2r - 1) / 4 \qquad R'_{L,2}(r) = (3r + 1) / 2 \tag{98}$$

$$R_{L,3}(r) = (5r^3 + 3r^2 - 3r - 1) / 4 \qquad R'_{L,3}(r) = (15r^2 + 6r - 3) / 4 \tag{99}$$

$$R_{L,4}(r) = (35r^4 + 20r^3 - 30r^2 - 12r + 3) / 16 \qquad R'_{L,4}(r) = (35r^3 + 15r^2 - 15r - 6) / 4 \tag{100}$$

References

1. Huynh, H. T., “A Flux Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin Methods,” AIAA Paper 2007–4079, Jun. 2007. (Cited on pages 7, 9, 14, 186).
2. Huynh, H. T., “A Reconstruction Approach to High-Order Schemes Including Discontinuous Galerkin for Diffusion,” AIAA Paper 2009–0403, Jan. 2009. (Cited on pages 7, 9).
3. Huynh, H. T., “High-Order Methods Including Discontinuous Galerkin by Reconstructions on Triangular Meshes,” AIAA Paper 2011–0044, Jan. 2011. (Cited on pages 7, 9).
4. Vincent, P., Castonguay, P., and Jameson, A., “A New Class of High-Order Energy Stable Flux Reconstruction Schemes,” *Journal of Scientific Computing*, Vol. 47, No. 1, 2011, pp. 50–72. (Cited on pages 7, 14).
5. Castonguay, P., Vincent, P., and Jameson, A., “A New Class of High-Order Energy Stable Flux Reconstruction Schemes for Triangular Elements,” *Journal of Scientific Computing*, Vol. 51, No. 1, 2012, pp. 224–256. (Cited on pages 7, 14).
6. Huynh, H. T., Wang, Z. J., and Vincent, P. E., “High-Order Methods for Computational Fluid Dynamics: A Brief Review of Compact Differential Formulations on Unstructured Grids,” *Computers & Fluids*, Vol. 98, 2014, pp. 209–220. (Cited on pages 7, 9, 77).
7. Vincent, P. E., Farrington, A. M., Witherden, F. D., and Jameson, A., “An Extended Range of Stable-Symmetric-Conservative Flux Reconstruction Correction Functions,” *Computer Methods in Applied Mechanics and Engineering*, Vol. 296, 2015, pp. 248–272. (Cited on pages 7, 14).
8. Spiegel, S. C., DeBonis, J. R., and Huynh, H. T., “Overview of the NASA Glenn Flux Reconstruction Based High-Order Unstructured Grid Code,” AIAA Paper 2016–1061, Jan. 2016. (Cited on pages 7, 9, 50).
9. Al Jahdali, R., Dalcin, L., Boukharfane, R., Nolasco, I. R., Keyes, D. E., and Parsani, M., “Optimized Explicit Runge–Kutta Schemes for High-Order Collocated Discontinuous Galerkin Methods for Compressible Fluid Dynamics,” *Computers & Mathematics with Applications*, Vol. 118, 2022, pp. 1–17. (Cited on pages 7, 72, 73, 114, 115).
10. Ranocha, H., Dalcin, L., Parsani, M., and Ketcheson, D. I., “Optimized Runge-Kutta Methods with Automatic Step Size Control for Compressible Computational Fluid Dynamics,” *Communications on Applied Mathematics and Computation*, Vol. 4, No. 4, 2022, pp. 1191–1228. (Cited on pages 7, 72, 73, 75, 76, 114, 115).
11. Spiegel, S. C., Huynh, H. T., and DeBonis, J. R., “A Survey of the Isentropic Euler Vortex Problem using High-Order Methods,” AIAA Paper 2015–2444, Jun. 2015. (Cited on pages 9, 66, 183).
12. Spiegel, S. C., Huynh, H. T., and DeBonis, J. R., “De-Aliasing through Over-Integration Applied to the Flux Reconstruction and Discontinuous Galerkin Methods,” AIAA Paper 2015–2744, Jun. 2015. (Cited on pages 9, 79).
13. Roe, P. L., “Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes,” *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372. (Cited on pages 13, 130).

14. Huynh, H. T., “Accurate Upwind Methods for the Euler Equations,” *SIAM Journal on Numerical Analysis*, Vol. 32, No. 5, Oct. 1995, pp. 1565–1619. (Cited on pages 13, 130).
15. Breuer, M., “Boundary Conditions for LES,” *Large-Eddy Simulation for Acoustics*, edited by C. Wagner, T. Hüttl, and P. Sagaut, chap. 5.2, Cambridge University Press, New York, 2007, pp. 201–237. (Cited on pages 21, 22).
16. Georgiadis, N. J., Rizzetta, D. P., and Fureby, C., “Large-Eddy Simulation: Current Capabilities, Recommended Practices, and Future Research,” NASA TM–2009-215616, July 2009, See also AIAA Paper 2009-948. (Cited on page 22).
17. Sagaut, P., “Theoretical Background: Large-Eddy Simulation,” *Large-Eddy Simulation for Acoustics*, edited by C. Wagner, T. Hüttl, and P. Sagaut, chap. 3, Cambridge University Press, New York, 2007, pp. 89–127. (Cited on page 22).
18. Sagaut, P. and Deck, S., “Large Eddy Simulation for Aerodynamics: Status and Perspectives,” *Philosophical Transactions of the Royal Society A*, Vol. 367, 2009, pp. 2849–2860. (Cited on page 22).
19. Schlatter, P., Li, Q., Brethouwer, G., Johansson, A. V., and Henningson, D. S., “Simulations of Spatially Evolving Turbulent Boundary Layers up to $Re_\theta=4300$,” *International Journal of Heat and Fluid Flow*, Vol. 31, No. 3, 2010, pp. 251–261, Sixth International Symposium on Turbulence and Shear Flow Phenomena. (Cited on page 22).
20. Debroeyer, R., Rasquin, M., Toulorge, T., Bartosiewicz, Y., and Winckelmans, G., “A Wall Model for Large-Eddy Simulation of Highly Compressible Flows Based on a New Scaling of the Law of the Wall,” *Journal of Fluid Mechanics*, Vol. 980, 2024, pp. A9. (Cited on page 22).
21. Ameri, A. A., “Requirements for Large Eddy Simulation Computations of Variable-Speed Power Turbine Flows,” NASA TM–2016-218962, Mar. 2016. (Cited on page 22).
22. Yoder, D. A., “Formulations of the Navier-Stokes Equations in Non-Inertial Reference Frames,” NASA TM–2026-xxxxxx, Oct. 2026. (Cited on page 25).
23. Jarrin, N., Benhamadouche, S., Laurence, D., and Prosser, R., “A Synthetic-Eddy-Method for Generating Inflow Conditions for Large-Eddy Simulations,” *International Journal of Heat and Fluid Flow*, Vol. 27, 2006, pp. 585–593. (Cited on page 28).
24. Toubert, E. and Sandham, N. D., “Oblique Shock Impinging on a Turbulent Boundary Layer: Low-Frequency Mechanisms,” AIAA Paper 2008-4170, Jun. 2008. (Cited on pages 33, 42).
25. Cooper, G. and Sirbaugh, J., “The PARC Distinction: A Practical Flow Simulator,” AIAA Paper 90-2002, July 1990. (Cited on page 55).
26. Carlson, J.-R., “Inflow/Outflow Boundary Conditions with Application to FUN3D,” NASA TM–217181, Oct. 2011. (Cited on pages 56, 120).
27. Spiegel, S. C., Yoder, D. A., DeBonis, J. R., Huynh, H. T., Heinlein, G. S., Borghi Jr., M. R., and Georgiadis, N. J., “New Capabilities and Improvements to the High-Order Glenn Flux Reconstruction Code,” AIAA Paper 2025-0061, Jan. 2025. (Cited on pages 69, 71, 76).

28. Giel, P. W., Shyam, V., Juangphanich, P., and Clark, J. P., “Effects of Trailing Edge Thickness and Blade Loading Distribution on the Aerodynamic Performance of Simulated CMC Turbine Blades,” *ASME Turbo Expo Turbomachinery Technical Conference & Exposition*, Vol. 84072, American Society of Mechanical Engineers, Sept. 2020, p. V02BT33A028. (Cited on page 70).
29. Miki, K. and Ameri, A. A., “Numerical Investigation of the Effect of Trailing Edge Thickness of Simulated CMC Blades on Loss Profiles,” *ASME Turbo Expo 2022: Turbomachinery Technical Conference and Exposition*, Vol. 86120, American Society of Mechanical Engineers, Jun. 2022, p. V10DT37A017. (Cited on page 70).
30. Wang, Z. J. and Jourdan, E., “Benchmark for Scale-Resolving Simulation with Curved Walls: The Taylor Couette Flow,” *Advances in Aerodynamics*, Vol. 3, No. 1, 2021, pp. 16. (Cited on pages 70, 160).
31. Spiteri, R. J. and Ruuth, S. J., “A New Class of Optimal High-Order Strong-Stability-Preserving Time Discretization Methods,” *SIAM Journal on Numerical Analysis*, Vol. 40, No. 2, 2002, pp. 469–491. (Cited on pages 72, 114).
32. Gottlieb, S. and Shu, C.-W., “Total Variation Diminishing Runge-Kutta Schemes,” *Mathematics of Computation*, Vol. 67, No. 221, 1998, pp. 73–85. (Cited on pages 72, 114).
33. Carpenter, M. H. and Kennedy, C. A., “Fourth-Order 2N-Storage Runge-Kutta Schemes,” NASA TM–109112, Jun. 1994. (Cited on pages 72, 114).
34. Dormand, J. R. and Prince, P. J., “A Family of Embedded Runge-Kutta Formulae,” *Journal of Computational and Applied Mathematics*, Vol. 6, No. 1, 1980, pp. 19–26. (Cited on pages 73, 115).
35. Hairer, E., Norsett, S., and Wanner, G., *Solving Ordinary Differential Equations I. Nonstiff Problems*, Springer Series in Computational Mathematics, Springer-Verlag, 2nd ed., 1993. (Cited on pages 73, 115).
36. Ketcheson, D. I., “Runge–Kutta Methods with Minimum Storage Implementations,” *Journal of Computational Physics*, Vol. 229, No. 5, 2010, pp. 1763–1773. (Cited on page 74).
37. Berzins, M., “Temporal Error Control for Convection-Dominated Equations in Two Space Dimensions,” *SIAM Journal on Scientific Computing*, Vol. 16, No. 3, 1995, pp. 558–580. (Cited on page 75).
38. Ware, J. and Berzins, M., “Adaptive Finite Volume Methods for Time-Dependent PDEs,” *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, Springer, 1995, pp. 417–430. (Cited on page 75).
39. Hesthaven, J. S. and Warburton, T., *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, Vol. 54 of *Texts in Applied Mathematics*, Springer New York, 2010. (Cited on pages 77, 79, 80, 129, 130, 181).
40. Kirby, R. M. and Karniadakis, G. E., “De-Aliasing on Non-Uniform Grids: Algorithms and Applications,” *Journal of Computational Physics*, Vol. 191, No. 1, 2003, pp. 249–264. (Cited on page 79).

41. Kirby, R. M. and Sherwin, S. J., “Aliasing Errors Due to Quadratic Nonlinearities on Triangular Spectral/hp Element Discretisations,” *Journal of Engineering Mathematics*, Vol. 56, No. 3, 2006, pp. 273–288. (Cited on page 79).
42. Kopriva, D. A., “Metric Identities and the Discontinuous Spectral Element Method on Curvilinear Meshes,” *Journal of Scientific Computing*, Vol. 26, No. 3, 2006, pp. 301–327. (Cited on page 79).
43. Castonguay, P., Vincent, P., and Jameson, A., “Application of High-Order Energy Stable Flux Reconstruction Schemes to the Euler Equations,” AIAA Paper 2011-686, Jan. 2011. (Cited on page 79).
44. Gassner, G. J. and Beck, A. D., “On the Accuracy of High-Order Discretizations for Underresolved Turbulence Simulations,” *Theoretical and Computational Fluid Dynamics*, Vol. 27, No. 3-4, 2013, pp. 221–237. (Cited on page 79).
45. Karniadakis, G. and Sherwin, S., *Spectral/hp Element Methods for Computational Fluid Dynamics*, Oxford University Press, 2nd ed., 2013. (Cited on page 79).
46. Williams, D. M. and Jameson, A., “Nodal Points and the Nonlinear Stability of High-Order Methods for Unsteady Flow Problems on Tetrahedral Meshes,” AIAA Paper 2013-2830, Jun. 2013. (Cited on pages 79, 129).
47. Witherden, F. D. and Vincent, P. E., “An Analysis of Solution Point Coordinates for Flux Reconstruction Schemes on Triangular Elements,” *Journal of Scientific Computing*, Vol. 61, No. 2, 2014, pp. 398–423. (Cited on page 79).
48. Persson, P.-O. and Peraire, J., “Sub-Cell Shock Capturing for Discontinuous Galerkin Methods,” AIAA Paper 2006-0112, Jan. 2006. (Cited on page 82).
49. Barter, G. E. and Darmofal, D. L., “Shock Capturing with Higher-Order, PDE-Based Artificial Viscosity,” AIAA Paper 2007-3823, Jun. 2007. (Cited on pages 82, 83).
50. Barter, G. E., *Shock Capturing with PDE-Based Artificial Viscosity for an Adaptive, Higher-Order Discontinuous Galerkin Finite Element Method*, Ph.D. thesis, Massachusetts Institute of Technology, 2008. (Cited on pages 82, 83).
51. Krivodonova, L., Xin, J., Remacle, J.-F., Chevaugnon, N., and Flaherty, J. E., “Shock Detection and Limiting with Discontinuous Galerkin Methods for Hyperbolic Conservation Laws,” *Applied Numerical Mathematics*, Vol. 48, No. 3-4, 2004, pp. 323–338, Workshop on Innovative Time Integrators for PDEs. (Cited on page 83).
52. Burgess, N. and Mavriplis, D., “An hp-Adaptive Discontinuous Galerkin Solver for Aerodynamic Flows on Mixed-Element Meshes,” AIAA Paper 2011-490, Jan. 2011. (Cited on page 83).
53. Burgess, N. K., *An Adaptive Discontinuous Galerkin Solver for Aerodynamic Flows*, Ph.D. thesis, University of Wyoming, 2011. (Cited on page 83).
54. Park, J. S., Chang, T. K., and Kim, C., “Higher-Order Multi-Dimensional Limiting Strategy for Correction Procedure via Reconstruction,” AIAA Paper 2014-0772, Jan. 2014. (Cited on page 83).

55. Park, J. S. and Kim, C., “Hierarchical Multi-Dimensional Limiting Strategy for Correction Procedure via Reconstruction,” *Journal of Computational Physics*, Vol. 308, 2016, pp. 57–80. (Cited on page 83).
56. Park, J. S., You, H., and Kim, C., “Higher-Order Multi-Dimensional Limiting Process for DG and FR/CPR Methods on Tetrahedral Meshes,” *Computers & Fluids*, Vol. 154, 2017, pp. 322–334, ICCFD8. (Cited on page 83).
57. Bridges, J. and Wernet, M. P., “The NASA Subsonic Jet Particle Image Velocimetry (PIV) Dataset,” NASA TM–2011-216807, Nov. 2011. (Cited on page 112).
58. Hartmann, R., “Higher-Order and Adaptive Discontinuous Galerkin Methods with Shock-Capturing Applied to Transonic Turbulent Delta Wing Flow,” *International Journal for Numerical Methods in Fluids*, Vol. 72, No. 8, 2013, pp. 883–894. (Cited on page 120).
59. Shunn, L. and Ham, F., “Symmetric Quadrature Rules for Tetrahedra Based on a Cubic Close-Packed Lattice Arrangement,” *Journal of Computational and Applied Mathematics*, Vol. 236, No. 17, 2012, pp. 4348–4364. (Cited on page 130).
60. Toro, E. F., Spruce, M., and Speares, W., “Restoration of the Contact Surface in the HLL-Riemann Solver,” *Shock waves*, Vol. 4, No. 1, 1994, pp. 25–34. (Cited on page 130).
61. Edwards, J. R., “A Low-Diffusion Flux-Splitting Scheme for Navier-Stokes Calculations,” *Computers & Fluids*, Vol. 26, No. 6, 1997, pp. 635–659. (Cited on page 130).
62. Nishikawa, H. and Kitamura, K., “Very Simple, Carbuncle-Free, Boundary-Layer-Resolving, Rotated-Hybrid Riemann Solvers,” *Journal of Computational Physics*, Vol. 227, No. 4, 2008, pp. 2560–2581. (Cited on page 130).
63. Liou, M.-S., “A Sequel to AUSM: AUSM+,” *Journal of Computational Physics*, Vol. 129, No. 2, 1996, pp. 364–382. (Cited on page 130).
64. Rusanov, V. V., “The Calculation of the Interaction of Non-Stationary Shock Waves with Barriers,” *Journal of Computational Mathematics and Mathematical Physics*, Vol. 1, No. 2, 1961, pp. 267–279. (Cited on page 130).
65. Bassi, F. and Rebay, S., “A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible Navier-Stokes Equations,” *Journal of Computational Physics*, Vol. 131, No. 2, 1997, pp. 267–279. (Cited on page 130).
66. Bassi, F., Crivellini, A., Rebay, S., and Savini, M., “Discontinuous Galerkin Solution of the Reynolds-Averaged Navier-Stokes and $k-\omega$ Turbulence Model Equations,” *Computers & Fluids*, Vol. 34, No. 4-5, 2005, pp. 507–540. (Cited on page 130).
67. Cockburn, B. and Shu, C.-W., “The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems,” *SIAM Journal on Numerical Analysis*, Vol. 35, No. 6, 1998, pp. 2440–2463. (Cited on page 130).
68. Szegő, G., “Orthogonal Polynomials,” *Colloquium Publications*, Vol. XXIII, American Mathematical Society, 1967. (Cited on page 186).
69. Hildebrand, F. B., *Introduction to Numerical Analysis*, International Series in Pure & Applied Mathematics, McGraw-Hill, New York, NY, 2nd ed., Mar. 1974. (Cited on page 186).

Index

- aliasing error, 79*, 128
- angle of attack, *see* flow angles
- angle of sideslip, *see* flow angles
- common flux value, 10, 11, 13*, 130
- continuous flux function, 11, 14, 15*, 131
 - derivative, 11, 15*
- correction function, 11, 14*, 14, 15, 131
- degree-of-freedom (DoF), 17*, 19
 - internally generated grid, 106*
- discontinuous flux function, 10, 11, 13*, 14, 15
- eddy volume, 29*, 31, 32
- filtering, 80*
 - global filter, 81*
 - local filter, 81*
 - sponge filter, 81*
- flow angles
 - angle of attack, 47*, 109, 122
 - angle of sideslip, 47*, 109, 122
- Flux Reconstruction Method, 9*
- grid scaled units, 20*, 36, 37, 65, 105
- grid units, 20*, 86, 105
- Initialization Sequence, 111*
- Input
 - All_Orders, 17, 127*, 127, 128, 172
 - All_Points, 17, 128*, 172
 - AoA_Alpha, 109*, 112, 134, 147, 170
 - AoA_Beta, 109*, 134, 147, 170
 - Aspect_Ratio, 160*, 176
 - Axial_Cells, 160*, 176
 - Azimuthal_Cells, 161*, 176
 - BC_Input(*i*), 34, 37, 44, 50, 66, 94, 119*, 171
 - %AoA_Alpha, 51, 122*, 171
 - %AoA_Beta, 51, 122*, 171
 - %Density, 51, 121*, 171
 - %Is_Specified_In_MRF, 63, 122*, 171
 - %Mach, 51, 121*, 171
 - %Mass_Frac, 122*, 171
 - %Name, 87, 94, 120*, 171
 - %Periodic%Matching_BC_Idx, 124, 125*, 171
 - %Periodic%Matching_BC_Name, 124*, 125, 171
 - %Periodic%RotationAngle, 68, 125*, 125, 172
 - %Periodic%RotationAxis, 68, 125*, 125, 172
 - %Periodic%RotationCenter, 63, 68, 125*, 125, 171
 - %Periodic%Translation, 67, 126*, 172
 - %Pressure, 51, 121*, 171
 - %Relax%Iter, 123*, 171
 - %Relax%Time, 123*, 171
 - %Relax%Value, 123*, 171
 - %Rotation_Rate, 63, 122*, 171
 - %SEM%Convective_Velocity, 29, 43, 123, 124*, 171
 - %SEM%Is_Cylindrical, 37, 42, 123*, 171
 - %SEM%Length_Scale, 29, 43, 123*, 124, 171
 - %SEM%Profile_File, 29, 42, 46, 123*, 124, 171
 - %SEM%Profile_Height, 43, 124*, 171
 - %SEM%Sub_In_BC_Grp, 43, 124*, 171
 - %SEM%Sub_In_BC_Idx, 43, 124*, 171
 - %Set_BC_Value, 46, 49, 52, 88, 121*, 171
 - %Species_Name, 122*, 171
 - %Temperature, 51, 121*, 171
 - %Total_Density, 51, 122*, 171
 - %Total_Pressure, 51, 122*, 171
 - %Total_Temperature, 51, 122*, 171
 - %VelocityX, 51, 63, 121*, 171
 - %VelocityY, 51, 63, 122*, 171
 - %VelocityZ, 51, 63, 122*, 171
 - %Wall_Temp, 64, 122*, 171
- Body_Force, 157*, 175
- Box_Num_Eddies_Multiplier, 32, 42, 152*, 175
- CFL_Ramping_Beg, 100, 116, 117*, 117, 171
- CFL_Ramping_End, 100, 116, 117*, 117, 171
- CFL_Ramping_Iterations, 100, 117*, 117, 171

CFL_Ramping_Start_Iter, 100, 117*, 171
 CFL, 74, 75, 116, 117*, 117, 171
 CGNS_High_Order_Output, 143*, 143, 174
 CGNS_Mixed_Output, 143*, 174
 CGNS_Output_Disabled, 142, 144*, 174
 Center, 158*, 176
 Check_Flux_Point_Coordinates, 167*, 177
 Chemistry_Sub_Iterations, 168*, 177
 Combine_Common_Flux_Projections, 130*, 172
 Compute_Time_Averaging, 96, 138*, 138, 174
 Conserved_Variables, 147*, 174
 Convergence_Order_Abs_Res, 118*, 171
 Convergence_Order_Max_Res, 118*, 171
 Convert_Restart_To_CGNS, 144, 165*, 165, 176
 Correct_CGNS_Grid_Points, 106*, 169
 Correction_Function, 131*, 172
 Cylindrical_Coordinates, 147*, 175
 Cylindrical_Velocity, 147*, 147, 175
 Debug_Species_Variables, 149*, 175
 Delete_Converted_Restart_Files, 165*, 165, 176
 Density, 109*, 170
 Disable_Chemical_Reactions, 153*, 175
 Display_Interval, 96, 141*, 174
 Dmat_Grad_Temp_Order, 143*, 174
 Dump_Memory_Usage, 167*, 177
 Error_Order, 17, 128*, 129, 172
 Exact_Density, 150*, 159, 180
 Exact_Pressure, 150*, 159, 180
 Extra_Species_Variables, 148, 149*, 175
 Final_Time, 115*, 115, 117, 141, 170
 Force_Nd_Subroutines, 168*, 177
 Form_Dim, 158*, 176
 Form, 158*, 176
 Gamma, 27, 109*, 170
 Gap, 158*, 176
 Gas_Constant, 27, 109*, 170
 Geom_Wts(:)%Hexa, 107*, 170
 Geom_Wts(:)%Pris, 107*, 170
 Geom_Wts(:)%Pyra, 107*, 170
 Geom_Wts(:)%Quad, 107*, 170
 Geom_Wts(:)%Tetr, 107*, 170
 Geom_Wts(:)%Tria, 107*, 170
 Geom_Wts, 107*, 107
 Global_Filter_Alpha, 81, 133*, 172
 Global_Filter_Etac, 81, 133*, 135, 172
 Global_Filter_Interval, 81, 132*, 172
 Global_Filter_Option, 81, 133*, 172
 Global_Filter_Order, 81, 133*, 172
 Governing_Equations, 27, 110*, 170
 Gradients, 148*, 150, 175
 Grdnt_Species_Variables, 150*, 175
 Grid_Annulus_Angle, 156*, 175
 Grid_Ave_Xplus, 156*, 175
 Grid_Ave_Yplus, 156*, 175
 Grid_Ave_Zplus, 157*, 175
 Grid_File, 86, 105*, 169
 Grid_Format, 86, 105*, 112, 169, 183
 Grid_Hex_To_Pyramids, 162*, 176
 Grid_Is_3d, 163*, 176
 Grid_Lref, 162*, 176
 Grid_Perturb_Percent, 162*, 176
 Grid_Random_Perturb, 162*, 162, 176
 Grid_Scaling_Factor, 20, 36, 37, 86, 105*, 146, 169
 Grid_Yplus_Of_One, 157*, 175
 Grid_Yplus_Wall, 157*, 157, 175
 Ignition_Duration, 154*, 154, 175
 Ignition_Max_Temperature, 154*, 154, 175
 Ignition_Method, 154*, 154, 175
 Ignition_Prod2Reac_L2_Ratio_Goal, 154*, 154, 175
 Ignition_Source_Location, 155*, 155, 175
 Ignition_Source_Magnitude, 154*, 154, 155, 175
 Ignition_Source_Radius, 155*, 175
 Ignition_Spark_Cap, 154*, 175
 Ignition_Start_Timestep, 154*, 175
 Ignore_Thermo_Newton_Warnings, 168*, 168, 177
 Include_Thermo_Fluctuations, 33, 42, 152*, 175
 Initial_Solution_File, 95, 111, 112, 113*, 113
 Initial_Solution_Format, 95, 111, 113*, 113
 Initialization, 86, 105, 106, 108, 111*, 111, 112, 113, 115, 150, 156, 162, 170, 179, 180, 182, 183

Inner_Radius, 160*, 176
 Inner_Wall_Spacing, 160*, 176
 Input_Time_Ave_File, 96, 138*, 165, 166, 174
 Internal_Grid_DoF, 106*, 112, 169, 170, 179, 180, 182, 183
 Internal_Grid_Order, 106*, 170
 Interpto_Grid_File, 164*, 176
 Interpto_Grid_Format, 164*, 176
 Interpto_Grid_Scaling_Factor, 164*, 176
 Interpto_Plot3d_Agglomerate_Order, 164*, 176
 Interpto_Plot3d_BCSfile, 164*, 176
 Interpto_Plot3d_Cutfile, 164*, 176
 Interpto_Plot3d_Stride_All, 164*, 176
 Interpto_Plot3d_Stride_I, 164*, 176
 Interpto_Plot3d_Stride_J, 164*, 176
 Interpto_Plot3d_Stride_K, 164*, 176
 Interpto_Solution_Order, 164*, 176
 Invs_Flux_Method, 50, 130*, 172
 Jet_Init_Func, 112*, 170
 Length, 158*, 176
 Load_Restart_File, 96, 111, 112, 113*, 113, 138, 170
 Loc_Error_Pts, 17, 128, 129*, 172
 Loc_Flux_Pts, 17, 128, 129*, 172
 Loc_Output_Pts, 17, 143*, 174
 Loc_Quadrature_Pts, 17, 128, 129*, 172
 Loc_Solution_Pts, 17, 86, 120, 128*, 128, 129, 172
 Loc_Tetrahedra_Pts, 17, 130*, 172
 Loc_Triangle_Pts, 17, 129*, 129, 172
 Local_Filter_Alpha, 81, 135*, 135, 173
 Local_Filter_Etac, 81, 135*, 135, 173
 Local_Filter_Order, 81, 135*, 135, 173, 179, 180
 Local_Limiter_Interval, 81, 134*, 135, 172
 Local_Limiter_Option, 134*, 135, 167, 172, 173
 Lustre_Stripe_Count, 145*, 174
 Lustre_Stripe_Size, 145*, 174
 Mach, 108*, 170
 Marker_Dpsi, 84, 136*, 173
 Marker_Max, 84, 136*, 173
 Marker_Min, 84, 136*, 173
 Marker_Option, 82, 83, 135*, 135, 136, 149, 172, 173
 Marker_Stored_Interval, 135*, 173
 Marker_Variables, 135, 149*, 175
 Marker_Variable, 82, 83, 135, 136*, 149, 173
 Metis_Option, 107*, 107, 170
 Momentum, 147*, 174
 Moving_Ref_Frame%Rotation, 25, 27, 110*, 170
 Moving_Ref_Frame%Spinup%Iter, 110*, 110, 170
 Moving_Ref_Frame%Spinup%Tstar, 110*, 110, 140, 170
 Moving_Ref_Frame%Translation, 25, 110*, 170
 Namelist_Version, 104*, 169
 Newton_Convergence_Goal, 168*, 177
 Nondim_Output_Coord, 157*, 176
 Num_Saved_Time_Ave_Restarts, 138*, 138, 174
 Num_Timesteps_Min, 115*, 170
 Num_Timesteps, 115*, 141, 142, 170
 Outer_Radius, 160*, 176
 Outer_Wall_Spacing, 160*, 176
 Output_Bface_Array, 167*, 177
 Output_Dir, 97, 138, 141, 142*, 145, 174
 Output_Filter_Diagonals, 167*, 177
 Output_Frequency, 96, 97, 141, 142*, 142, 174
 Output_Interval, 96, 97, 142*, 142, 174
 Output_Limiter_Statistics, 167*, 177
 Output_Order, 17, 142*, 174
 Output_Plot3d_To_Tecplot, 167*, 177
 Output_Summary_File, 143*, 174
 Periodic_Sectors, 161*, 176
 Plot3d_Agglomerate_Order, 93, 106*, 169
 Plot3d_BCSfile, 95, 106*, 169
 Plot3d_Cutfile, 93, 105*, 169
 Plot3d_Stride_All, 93, 106*, 169
 Plot3d_Stride_I, 93, 106*, 169
 Plot3d_Stride_J, 93, 106*, 169
 Plot3d_Stride_K, 93, 106*, 169
 Point_Probes, 98, 99, 144*, 174
 Prandtl, 27, 109*, 170
 Pressure, 109*, 170

Probes_Collection_Interval, 99, 144*, 174
 Probes_Write_Interval, 99, 144*, 144, 174
 Profile_IO_All, 145*, 174
 Profile_IO_CGNS_Sub, 145*, 174
 Profile_IO_CGNS, 145*, 145, 174
 Profile_IO_Restart, 145*, 145, 174
 Projection_Order, 17, 127*, 127, 172
 Qcriterion, 149*, 175
 Quadrature_Order, 17, 127, 128*, 129, 172
 Radial_Cells, 161*, 176
 Rake_Probes, 98, 99, 144*, 174
 Relative_Conditions, 147, 148*, 175
 Relative_Velocity, 147, 148*, 175
 Repeat, 158*, 176
 Residual_Interval, 96, 141*, 174
 Restart_File, 96, 111, 113*, 113, 138, 165, 170
 Restart_Frequency, 96, 138, 141*, 141, 174
 Restart_Interval, 96, 138, 141*, 174
 Restart_List_File, 165*, 165, 177
 Restart_Output_Disabled, 96, 142*, 174
 Restart_Output_Uses_Host_Roots, 141*, 174
 Reynolds, 27, 47, 108*, 109, 170
 Right_Vector, 159*, 176
 Runge_Kutta_Scheme, 71, 72, 73, 77, 114*, 170
 Schmidt, 109*, 170
 Semi_Implicit_Chemistry, 153*, 175
 Solution_File, 170
 Solution_Format, 170
 Solution_Order, 17, 86, 93, 106, 120, 127*, 127, 128, 133, 143, 172, 183
 Spatial_Ave_Restart_Channel, 165*, 177
 Spatial_Ave_Restart_X_And_Z_Planes, 165, 166*, 177
 Spatial_Ave_Restart_Z_Planes, 166*, 166, 177
 Species_Density, 149*, 175
 Species_Mass_Fraction, 149*, 175
 Species_Mole_Fraction, 149*, 175
 Sponge_Filter_Alpha, 81, 134*, 173
 Sponge_Filter_Etac, 81, 134*, 173
 Sponge_Filter_Interval, 81, 134*, 173
 Sponge_Filter_Option, 81, 134*, 173
 Sponge_Filter_Order, 81, 134*, 173
 Sponge_Filter_X_Start, 81, 134*, 173
 Stagnation_Pressure, 148*, 175
 Stagnation_Temperature, 148*, 175
 Stagnation_Variables, 148*, 175
 Start_Averaging_Timestep, 139*, 139, 174
 Start_Averaging_Tstar, 138, 139*, 141, 174
 Sub_Inflow_Method, 55, 120*, 171
 Suth_Muref, 27, 108, 109*, 170
 Suth_Sref, 27, 109*, 170
 Suth_Tref, 27, 109*, 170
 Temperature, 109*, 148*, 170, 175
 Thermo_Variables, 148*, 175
 Thermofix%Check_Cells, 137*, 173
 Thermofix%Check_Faces, 137*, 173
 Thermofix%Check_Nodes, 137*, 174
 Thermofix%Iterations, 137*, 173
 Thermofix%Method, 137*, 169, 173
 Thermofix%Min_Pressure, 137*, 174
 Thermofix%Min_Density, 137*, 174
 Thermofix%Min_Temperature, 137*, 174
 Thermofix%Scale_Factor, 137*, 174
 Thermofix%Verbose, 137*, 174
 Theta_Shift, 161*, 176
 Time_Ave_Vel_Is_Axisymm, 97, 98, 139*, 174
 Time_Scaling_Factor, 47, 115, 117, 140*, 174
 Timestep_Type, 74, 75, 116*, 117, 170
 Timestep, 47, 75, 116, 117*, 117, 171
 Traceback_Unconverged_Newton_Temps, 168*, 177
 Transport_Variables, 148*, 175
 Use_BC_Conflicts, 120*, 171
 Use_Profile_For_Initialization, 41, 151*, 175
 Use_Random_Seed_From_Restart, 35, 41, 151*, 175
 Use_Spherical_Eddy_Volume, 29, 32, 41, 45, 151*, 175
 Visc_Flux_Method, 50, 130*, 172
 Visc_Variables_To_Ave, 50, 130*, 172
 Vortex_Biggam, 162*, 176
 Vortex_Bigr, 162*, 176

Vorticity, 147*, 174
 Walls_Are_Exact, 120*, 171
 Width, 158*, 176
 nml_bc, 28, 42, 46, 119*
 nml_chemistry, 153*
 nml_debug, 167*
 nml_grid, 105*
 nml_initialization, 111*
 nml_interpolation, 95, 164*
 nml_limiter, 132*
 nml_numerics, 127*
 nml_output_variables, 146*
 nml_output, 140*
 nml_physics, 25, 27, 110*
 nml_reference, 27, 47, 95, 108*, 112, 123
 nml_sem, 41, 151*
 nml_testcase_channel, 106, 156*
 nml_testcase_pulse, 158*, 179
 nml_testcase_sod, 159*, 180
 nml_testcase_tcf, 106, 160*
 nml_testcase_vortex, 162*, 183
 nml_time_ave, 98, 138*
 nml_timestep, 114*
 nml_utility, 165*
 nml_version, 104*
 internally generated grid, 106, 112*
 degree-of-freedom (DoF), 106*
 jump marker, *see* marker
 large-eddy simulation (LES), 20*
 implicit LES (ILES), 21*
 mesh requirements, 20*

marker, 81, 82*
 jump marker, 83*
 MLP marker, 83*
 resolution marker, 82*
 scaled marker, 84*
 METIS, 107
 MLP marker, *see* marker
 points
 equidistant, 16*, 16, 17, 129, 143
 Legendre-Gauss (Gauss), 16*, 16, 17, 129, 143
 Legendre-Gauss-Lobatto (Lobatto), 16*, 16, 129, 143
 output, 17*, 143
 quadrature, 17*, 80, 129
 solution, 16*, 128
 polynomial
 Lagrange, 12*
 Legendre, 186*, 187
 modal form, 12*, 13, 82
 nodal form, 12*, 13
 Radau, 4, 14, 15, 187*
 probes, 98*, 144
 resolution marker, *see* marker
 Sutherland's viscosity law, 27*, 108, 109
 Synthetic Eddy Method (SEM), 28*
 volume, 29*, 32, 34
 t^* , 110, 115, 138, 139, 140*, 141, 142, 143
 time averaging, 97*
 troubled cell marker, *see* marker

