

Rayleigh User Guide

Version 0.9.0

Nicholas Featherstone

January 23, 2018

Overview

Rayleigh solves the MHD equations, in a rotating frame, within spherical shells, using the anelastic or Boussinesq approximations. Derivatives in Rayleigh are calculated using a spectral transform scheme. Spherical harmonics are used as basis functions in the horizontal direction. Chebyshev polynomials are employed in radius. Time-stepping is accomplished using the semi-implicit Crank-Nicolson method for the linear terms, and the Adams-Bashforth method for the nonlinear terms. Both methods are second-order in time.

This document serves as a guide to installation and running Rayleigh. Rayleigh's diagnostics package is discussed in the companion document `Rayleigh/doc/Diagnostics_Plotting.html`, `pdf`

Rayleigh is written by Nicholas Featherstone, with National-Science-Foundation support through the Geodynamo Working Group of the Computational Infrastructure for Geodynamics (CIG; PI: Louise Kellogg).

The CIG Geodynamo Working Group Members are: Jonathon Aurnou, Benjamin Brown, Bruce Buffet, Nicholas Featherstone, Gary Glatzmaier, Eric Heien, Moritz Heimpel, Lorraine Hwang, Louise Kellogg, Hiroaki Matsui, Peter Olson, Krista Soderlund, Sabine Stanley, Rakesh Yadav

Rayleigh's implementation of the pseudo-spectral algorithm and its parallel design would not have been possible without earlier work by Gary Glatzmaier and Thomas Clune, described in:

1. Glatzmaier, G.A., 1984, *J. Comp. Phys.*, 55, 461
2. Clune, T.C., Elliott, J.R., Miesch, M.S. & Toomre, J., 1999, *Parallel Comp.*, **25**, 361

Contents

1	Compiling and Installing Rayleigh	3
1.1	Third-Party Dependencies	3
1.2	Compilation	3
1.3	Verifying Your Installation	3
2	Running the Code	4
2.1	Preparation	4
2.2	Code Execution and Load-Balancing	4
2.2.1	Load Balancing	5
2.2.2	Specifying Resolution & Domain Bounds	5
2.3	Controlling Run Length & Time Stepping	6
3	Running a Benchmark	7
4	Physics Controls	9
4.1	Anelastic Mode (dimensional)	9
4.2	Boussinesq Mode (nondimensional)	11
4.3	Anelastic Mode (nondimensional)	13
4.4	Boundary Conditions & Internal Heating	13
4.4.1	Internal Heating	14
4.5	General Physics Controls	14
4.6	Initializing a Model	14
5	Checkpointing	17
5.1	Standard Checkpoints	17
5.1.1	Generating Standard Checkpoints	17
5.1.2	Restarts From Standard Checkpoints	17
5.2	Quicksaves	18
5.2.1	1.2.1 Generating Quicksaves	18
5.2.2	1.2.2 Restarting from Quicksaves	19
5.3	1.3 Checkpoint Logs	19
6	Diagnostic Outputs	19
7	I/O Redirection	20
8	Ensemble Mode	21
9	Acknowledging Rayleigh	22

1 Compiling and Installing Rayleigh

A detailed explanation of the installation process may be found in the root directory of the code repository at: Rayleigh/INSTALL.

We provide an abbreviated version of those instructions here.

1.1 Third-Party Dependencies

In order to compile Rayleigh, you will need to have MPI (Message Passing Interface) installed along with a Fortran 2003-compliant compiler. Rayleigh has been successfully compiled with GNU, Intel, and IBM compilers (PGI has not been tested yet). Rayleigh's configure script provides native support for the Intel and GNU compilers. See Rayleigh\INSTALL for an example of configuration using the IBM compiler.

Rayleigh depends on the following third party libraries:

1. BLAS (Basic Linear Algebra Subprograms)
2. LAPACK (Linear Algebra PACKage)
3. FFTW 3.x (Fastest Fourier Transform in the West)

You will need to install these libraries before compiling Rayleigh. If you plan to run Rayleigh on Intel processors, we suggest installing Intel's Math Kernel Library (MKL) in lieu of installing these libraries individually. The Math Kernel Library provides optimized versions of BLAS, LAPACK, and FFTW. It has been tuned, by Intel, for optimal performance on Intel processors. At the time of this writing, MKL is provided free of charge. You may find it [here](#).

1.2 Compilation

Rayleigh is compiled using the standard Linux installation scheme of configure/make/make-install. From within the Rayleigh directory, run these commands:

1. **./configure** – See Rayleigh/INSTALL or run **./configure --help** to see relevant options.
2. **make** – This compiles the code
3. **make install** – This places the Rayleigh executables in Rayleigh/bin. If you would like to place them in (say) /home/my_rayleigh, run configure as: **./configure --prefix=/home/my_rayleigh**.

For most builds, two executables will be created: rayleigh.opt and rayleigh.dbg. Use them as follows:

1. When running production jobs, use **rayleigh.opt**.
2. If you encounter an unexpected crash and would like to report the error, rerun the job with **rayleigh.dbg**. This version of the code is compiled with debugging symbols. It will (usually) produce meaningful error messages in place of the gibberish that is output when rayleigh.opt crashes.

If *configure* detects the Intel compiler, you will be presented with a number of choices for the vectorization option. If you select *all*, rayleigh.opt will not be created. Instead, rayleigh.sse, rayleigh.avx, etc. will be placed in Rayleigh/bin. This is useful if running on a machine with heterogeneous node architectures (e.g., Pleiades). If you are not running on such a machine, pick the appropriate vectorization level, and rayleigh.opt will be compiled using that vectorization level.

1.3 Verifying Your Installation

Rayleigh comes with a benchmarking mode that helps you verify that the installation is performing correctly. If you are running Rayleigh for the first time, or running on a new machine, follow along with the example in §3, and verify that you receive an accurate benchmark report before running a custom model.

2 Running the Code

Whenever you run a new simulation, a similar series of steps must be performed. A summary of the typical Rayleigh work flow is:

1. Create a unique directory for storing simulation output
2. Create a `main_input` file
3. Copy or soft link the Rayleigh executable into the simulation directory
4. Modify `main_input` as desired
5. Run the code
6. Examine output and restart simulation as necessary

2.1 Preparation

Each simulation run using Rayleigh should have its own directory. The code is run from within that directory, and any output is stored in various subdirectories created by Rayleigh at run time. Wherever you create your simulation directory, ensure that you have sufficient space to store the output.

Do not run Rayleigh from within the source code directory.

Do not cross the beams: no running two models from within the same directory.

After you create your run directory, you will want to copy (`cp`) or soft link (`ln -s`) the executable from `Rayleigh/bin` to your run directory. Soft-linking is recommended; if you recompile the code, the executable remains up-to-date. If running on an IBM machine, copy the script named `Rayleigh/etc/make_dirs` to your run directory and execute the script. This will create the directory structure expected by Rayleigh for its outputs. This step is unnecessary when compiling with the Intel or GNU compilers.

Next, you must create a `main_input` file. This file contains the information that describes how your simulation is run. Rayleigh always looks for a file named `main_input` in the directory that it is launched from. Copy one of the sample input files from the `Rayleigh/etc/input_examples/` into your run directory, and rename it to `main_input`. The file named *benchmark_diagnostics_input* can be used to generate output for the diagnostics plotting tutorial (see §6).

Finally, Rayleigh has some OpenMP-related logic that is still in development. We do not support Rayleigh's OpenMP mode at this time, but on some systems, it can be important to explicitly disable OpenMP in order to avoid tripping any OpenMP flags used by external libraries, such as Intel's MKL. Please be sure and run the following command before executing Rayleigh. This command should be precede *each* call to Rayleigh.

```
export OMP_NUM_THREADS=1 (bash)
setenv OMP_NUM_THREADS 1 (c-shell)
```

2.2 Code Execution and Load-Balancing

Rayleigh is parallelized using MPI and a 2-D domain decomposition. The 2-D domain decomposition means that we envision the MPI Ranks as being distributed in rows and columns. The number of MPI ranks within a row is *nprow* and the number of MPI ranks within a column is *npcol*. When Rayleigh is run with *N* MPI ranks, the following constraint must be satisfied:

$$N = npcol \times nprow. \quad (1)$$

If this constraint is not satisfied, the code will print an error message and exit. The values of *nprow* and *npcol* can be specified in *main_input* or on the command line via the syntax:

```
mpiexec -np 8 ./rayleigh.opt -nprow 4 -npcol 2
```

2.2.1 Load Balancing

Rayleigh's performance is sensitive to the values of *nprow* and *npcol*, as well as the number of radial grid points N_r and latitudinal grid points N_θ . If you examine the `main_input` file, you will see that it is divided into Fortran namelists. The first namelist is the `problemsize_namelist`. Within this namelist, you will see a place to specify *nprow* and *npcol*. Edit `main_input` so that *nprow* and *npcol* agree with the N you intend to use (or use the command-line syntax mentioned above). The dominate effect on parallel scalability is the number of messages sent per iteration. For optimal message counts, *nprow* and *npcol* should be as close to one another in value as possible.

1. $N = \text{nprow} \times \text{npcol}$.
2. *nprow* and *npcol* should be equal or within a factor of two of one another.

The value of *nprow* determines how spherical harmonics are distributed across processors. Spherical harmonics are distributed in high- m /low- m pairs, where m is the azimuthal wavenumber. Each process is responsible for all ℓ -values associated with those m 's contained in memory.

The value of *npcol* determines how radial levels are distributed across processors. Radii are distributed uniformly across processes in contiguous chunks. Each process is responsible for a range of radii Δr .

The number of spherical harmonic degrees N_ℓ is defined by

$$N_\ell = \frac{2}{3}N_\theta \quad (2)$$

For optimal load-balancing, *nprow* should divide evenly into N_r and *npcol* should divide evenly into the number of high- m /low- m pairs (i.e., $N_\ell/2$). Both *nprow* and *npcol* must be at least 2.

In summary,

1. $\text{nprow} \geq 2$.
2. $\text{npcol} \geq 2$.
3. $n \times \text{npcol} = N_r$ (for integer n).
4. $k \times \text{npcol} = \frac{1}{3}N_\theta$ (for integer k).

2.2.2 Specifying Resolution & Domain Bounds

As discussed, the number of radial grid points is denoted by N_r , and the number of θ grid points by N_θ . The number of grid points in the ϕ direction is always $N_\phi = 2 \times N_\theta$. N_r and N_θ may each be defined in the `problemsize_namelist` of `main_input`:

```
&problemsize_namelist
  n_r = 48
  n_theta = 96
/
```

N_r and N_θ may also be specified at the command line (overriding the values in `main_input`) via:

```
mpiexec -np 8 ./rayleigh.opt -nr 48 -ntheta 96
```

If desired, the maximal spherical harmonic degree $\ell_{\max} \equiv N_\ell - 1$ can be specified in lieu of N_θ . The example above may equivalently be written:

```
&problemsize_namelist
  n_r = 48
  l_max = 63
/
```

The radial domain bounds are determined by the namelist variables *rmin* (the lower radial boundary) and *rmax* (the upper radial boundary):

```
&problemsize_namelist
  rmin = 1.0
  rmax = 2.0
/
```

Alternatively, the user may specify the shell depth ($rmax - rmin$) and aspect ratio ($rmin/rmax$) in lieu of *rmin* and *rmax*. The preceding example may then be written as:

```
&problemsize_namelist
  aspect_ratio = 0.5
  shell_depth = 1.0
/
```

Note that the interpretation of *rmin* and *rmax* depends on whether your simulation is dimensional or nondimensional. We discuss these alternative formulations in §4

2.3 Controlling Run Length & Time Stepping

A simulation's runtime and time-step size can be controlled using the **temporal_controls** namelist. The length of time for which a simulation runs before completing is controlled by the namelist variable **max_time_minutes**. The maximum number of time steps that a simulation will run for is determined by the value of the namelist **max_iterations**. The simulation will complete when it has run for *max_time_minutes* minutes or when it has run for *max_iterations* time steps – whichever occurs first.

Time-step size in Rayleigh is controlled by the Courant-Friedrichs-Lewy condition (CFL; as determined by the fluid velocity and Alfvén speed). A safety factor of **cflmax** is applied to the maximum time step determined by the CFL. Time-stepping is adaptive. An additional variable **cflmin** is used to determine if the time step should be increased.

The user may also specify the maximum allowed time-step size through the namelist variable **max_time_step**. The minimum allowable time-step size is controlled through the variable **min_time_step**. If the CFL condition is less than this value, the simulation will exit.

Let Δt be the current time-step size, and let t_{CFL} be the maximum time-step size as determined by the CFL limit. The following logic is employed by Rayleigh when calculating the time-step size:

- IF { $\Delta t \geq cflmax \times t_{CFL}$ } THEN { Δt is set to $cflmax \times t_{CFL}$ }.
- IF { $\Delta t \leq cflmin \times t_{CFL}$ } THEN { Δt is set to $cflmax \times t_{CFL}$ }.
- IF { $t_{CFL} \geq max_time_step$ } THEN { Δt is set to max_time_step }
- IF { $t_{CFL} \leq min_time_step$ } THEN { Rayleigh Exits }

The default values for these variables are:

```
&temporal_controls_namelist
max_iterations = 1000000
max_time_minutes = 1d8
cflmax = 0.6d0
cflmin = 0.4d0
max_time_step = 1.0d0
min_time_step = 1.0d-13
/
```

3 Running a Benchmark

Rayleigh has been programmed with internal testing suite so that its results may be compared against benchmarks described in the following two papers:

1. Christensen, U.R., et al. 2001, *A Numerical Dynamo Benchmark*, *PEPI*, 128, 25
2. Jones, C.A., et al., 2011, *Anelastic-Convective-Driven Dynamo Benchmarks*, *Icarus*, 216, 120

We recommend running a benchmark whenever running Rayleigh on a new machine for the first time, or after recompiling the code. The Christensen et al. (2001) reference describes two Boussinesq tests that Rayleigh’s results may be compared against. The Jones et al. (2011) reference describes anelastic tests. Rayleigh has been tested successfully against two benchmarks from each of these papers. Input files for these different tests are enumerated in Table 1 below. In addition to the input files listed in Table 1, input examples appropriate for use as a template for new runs are provided with the *_input* suffix (as opposed to the *minimal* suffix. These input files still have `benchmark_mode` active. Be sure to turn this flag off if not running a benchmark.

Important: If you are not running a benchmark, but only wish to modify an existing benchmark-input file, delete the line containing the text “*benchmark_mode=X*.” When benchmark mode is active, custom inputs, such as Rayleigh number, are overridden and reset to their benchmark-appropriate values.

We suggest using the `c2001_case0_minimal` input file for installation verification. Algorithmically, there is little difference between the MHD, non-MHD, Boussinesq, and anelastic modes of Rayleigh. As a result, when installing the code on a new machine, it is normally sufficient to run the cheapest benchmark, case 0 from Christensen 2001.

To run this benchmark, create a directory from within which to run your benchmark, and follow along with the commands below. Modify the directory structure a each step as appropriate:

1. `mkdir path_to_my_sim`
2. `cd path_to_my_sim`
3. `cp path_to_rayleigh/Rayleigh/etc/input_examples/c2001_case0_minimal main_input`
4. `cp path_to_rayleigh/Rayleigh/bin/rayleigh.opt rayleigh.opt` (or use `ln -s` in lieu of `cp`)
5. `mpiexec -np N ./rayleigh.opt -nprow X -npcol Y -nr R -ntheta T`

For the value **N**, select the number of cores you wish to run with. For this short test, 32 cores is more than sufficient. Even with only four cores, the lower-resolution test suggested below will only take around half an hour. The values **X** and **Y** are integers that describe the process grid. They should both be at least 2, and must satisfy the expression

$$N = X \times Y. \quad (3)$$

Some suggested combinations are $\{N,X,Y\} = \{32,4,8\}, \{16,4,4\}, \{8,2,4\}, \{4,2,2\}$. The values **R** and **T** denote the number of radial and latitudinal collocation points respectively. Select either $\{R,T\}=\{48,64\}$ or $\{R,T\}=\{64,96\}$. The lower-resolution case takes about 3 minutes to run on 32 Intel Haswell cores. The higher-resolution case takes about 12 minutes to run on 32 Intel Haswell cores.

Once your simulation has run, examine the file `path_to_my_sim/Benchmark_Reports/00025000`. You should see output similar to that presented in Tables 2 or 3. Your numbers may differ slightly, but all values should have a % difference of less than 1. If this condition is satisfied, your installation is working correctly.

Paper	Benchmark	Input File
Christensen et al. 2001	Case 0	Rayleigh/etc/input_examples/c2001_case0_minimal
Christensen et al. 2001	Case 1(MHD)	Rayleigh/etc/input_examples/c2001_case1_minimal
Jones et al. 2011	Steady Hydro	Rayleigh/etc/input_examples/j2011_steady_hydro_minimal
Jones et al. 2011	Steady MHD	Rayleigh/etc/input_examples/j2011_steady_MHD_minimal

Table 1: Benchmark-input examples useful for verifying Rayleigh’s installation. Those from Christensen et al. (2001) are Boussinesq. Those from Jones et al. (2011) are anelastic.

Observable	Measured	Suggested	% Difference	Std. Dev.
Kinetic Energy	58.347827	58.348000	-0.000297	0.000000
Temperature	0.427416	0.428120	-0.164525	0.000090
Vphi	-10.118053	-10.157100	-0.384434	0.012386
Drift Frequency	0.183272	0.182400	0.477962	0.007073

Table 2: Rayleigh benchmark report for Christensen et al. (2001) case 0 when run with nr=48 and ntheta=64. Run time was approximately 3 minutes when run on 32 Intel Haswell cores.

Run command: `mpiexec -np 32 ./rayleigh.opt -nprow 4 -npcol 8 -nr 48 -ntheta 64`

Observable	Measured	Suggested	% Difference	Std. Dev.
Kinetic Energy	58.347829	58.348000	-0.000294	0.000000
Temperature	0.427786	0.428120	-0.077927	0.000043
Vphi	-10.140183	-10.157100	-0.166551	0.005891
Drift Frequency	0.182276	0.182400	-0.067994	0.004877

Table 3: Rayleigh benchmark report for Christensen et al. (2001) case 0 when run with nr=64 and ntheta=96. Run time was approximately 12 minutes when run on 32 Intel Haswell cores.

Run command: `mpiexec -np 32 ./rayleigh.opt -nprow 4 -npcol 8 -nr 64 -ntheta 96`

4 Physics Controls

Rayleigh solves the MHD equations in spherical geometry under the Boussinesq and anelastic approximations. Both the equations that Rayleigh solves and its diagnostics can be formulated either dimensionally or nondimensionally. A nondimensional Boussinesq formulation, as well as dimensional and non-dimensional anelastic formulations (based on a polytropic reference state) are provided as part of Rayleigh.

In this section, we present the equation sets solved when running in each of these three modes, and discuss the relevant control parameters for each mode. We also discuss the boundary conditions available in Rayleigh and those namelist variables that can be used to modify the code's behavior in any of these three modes.

4.1 Anelastic Mode (dimensional)

Example Input: `Rayleigh/etc/input_examples/main_input_sun`

When run in dimensional, anelastic mode, **reference_type=2** must be specified in the `Reference_Namelist`. In that case, Rayleigh solves the following form of the MHD equations:

$$\begin{aligned}
 \hat{\rho}(r) \left[\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} + 2\Omega_0 \hat{\mathbf{z}} \times \mathbf{v} \right] &= \frac{\hat{\rho}(r)}{c_P} g(r) \Theta \hat{\mathbf{r}} + \hat{\rho}(r) \nabla \left(\frac{P}{\hat{\rho}(r)} \right) + \frac{1}{4\pi} (\nabla \times \mathbf{B}) \times \mathbf{B} + \nabla \cdot \mathcal{D} && \text{Momentum} \\
 \hat{\rho}(r) \hat{T}(r) \left[\frac{\partial \Theta}{\partial t} + \mathbf{v} \cdot \nabla \Theta \right] &= \nabla \cdot \left[\hat{\rho}(r) \hat{T}(r) \kappa(r) \nabla \Theta \right] + Q(r) + \Phi(r, \theta, \phi) + \frac{\eta(r)}{4\pi} [\nabla \times \mathbf{B}]^2 && \text{Thermal Energy} \\
 \frac{\partial \mathbf{B}}{\partial t} &= \nabla \times (\mathbf{v} \times \mathbf{B} - \eta(r) \nabla \times \mathbf{B}) && \text{Induction} \\
 \mathcal{D}_{ij} &= 2\hat{\rho}(r) \nu(r) \left[e_{ij} - \frac{1}{3} \nabla \cdot \mathbf{v} \right] && \text{Viscous Stress Tensor} \\
 \Phi(r, \theta, \phi) &= 2\hat{\rho}(r) \nu(r) \left[e_{ij} e_{ij} - \frac{1}{3} (\nabla \cdot \mathbf{v})^2 \right] && \text{Viscous Heating} \\
 \nabla \cdot (\hat{\rho}(r) \mathbf{v}) &= 0 && \text{Solenoidal Mass Flux} \\
 \nabla \cdot \mathbf{B} &= 0 && \text{Solenoidal Magnetic Field}
 \end{aligned}$$

Here, $\hat{\rho}$ and \hat{T} are the reference-state density and temperature respectively. g is the gravitational acceleration, c_P is the specific heat at constant pressure, and Ω_0 is the frame rotation rate. The velocity field vector is denoted by \mathbf{v} , the magnetic field vector by \mathbf{B} , and the pressure by P . The thermal anomaly is denoted by Θ and should be interpreted as entropy s in this formulation. The thermal variables satisfy the linearized equation of state

$$\frac{P}{\hat{P}} = \frac{T}{\hat{T}} + \frac{\rho}{\hat{\rho}} \quad (4)$$

The kinematic viscosity, thermal diffusivity, and magnetic diffusivity are given by ν , κ , and η respectively. Finally, $Q(r)$ is an internal heating function; it might represent radiative heating or heating due to nuclear fusion, for instance.

When running in anelastic mode, the **reference_type** variable in the `Reference_Namelist` must be set to 2.

Moreover, certain variables in the **Reference_Namelist** and the **Transport_Namelist** must be specified. The `Reference_Namelist` variables are described in Table 4 and the `Transport_Namelist` variables are described in Table 5. Default values indicated in brackets.

The polytropic reference state is the same as that used in the benchmarks and is described in detail in Jones et al. (2011).

See the example input file `main_input_sun` for an example of how to run a solar-like model using Rayleigh's dimensional, anelastic formulation.

Variable	Description
poly_n [0]	polytropic index ($P \propto \rho^n$)
poly_Nrho [0]	number of density scaleheights spanning the domain
poly_mass [0]	mass interior to $rmin$
poly_rho_i [0]	density at $rmin$, $\rho(r = rmin)$
pressure_specific_heat [0]	specific heat at constant pressure
angular_velocity [1.0]	cyclic frequency of the rotating frame

Table 4: Variables in the Reference_Namelist that must be specified when running in dimensional anelastic mode. In addition, reference_type=2 must also be specified.

Variable	Description
nu_top [1.0]	kinematic viscosity at $rmax$, $\nu(rmax)$
nu_type [1]	determines whether ν is constant with radius (1) or varies with density (2)
nu_power [0.0]	exponent in : $\nu(r) = \left(\frac{\rho(r)}{\rho(r=rmax)} \right)^{nu_power}$; use with nu_type=2
kappa_top [1.0]	thermal diffusivity at $rmax$, $\kappa(rmax)$
kappa_type [1]	determines whether κ is constant with radius (1) or varies with density (2)
kappa_power [0.0]	exponent in : $\kappa(r) = \left(\frac{\rho(r)}{\rho(r=rmax)} \right)^{kappa_power}$; use with kappa_type=2
eta_top [1.0]	magnetic diffusivity at $rmax$, $\eta(rmax)$
eta_type [1]	determines whether η is constant with radius (1) or varies with density (2)
eta_power [0.0]	exponent in : $\eta(r) = \left(\frac{\rho(r)}{\rho(r=rmax)} \right)^{eta_power}$; use with eta_type=2

Table 5: Variables in the Transport_Namelist that must be specified when running in dimensional anelastic mode. In addition, reference_type=2 must also be specified in the Reference_Namelist.

4.2 Boussinesq Mode (nondimensional)

Example Input: Rayleigh/etc/input_examples/c2001_case1_input

When run in nondimensional Boussinesq mode, **reference_type=1** must be specified in the Reference_Namelist. In that case, Rayleigh employs the nondimensionalization

$$\begin{aligned}
 \text{Length} &\rightarrow L && (\text{Shell Depth}) \\
 \text{Time} &\rightarrow \frac{L^2}{\nu} && (\text{Viscous Timescale}) \\
 \text{Temperature} &\rightarrow \Delta T && (\text{Temperature Contrast Across Shell}) \\
 \text{Magnetic Field} &\rightarrow \sqrt{\rho\mu\eta\Omega_0},
 \end{aligned}$$

where Ω_0 is the rotation rate of the frame, ρ is the (constant) density of the fluid, μ is the magnetic permeability, η is the magnetic diffusivity, and ν is the kinematic viscosity. After nondimensionalizing, the following nondimensional numbers appear in our equations

$$\begin{aligned}
 Pr &= \frac{\nu}{\kappa} && \text{Prandtl Number} \\
 Pm &= \frac{\nu}{\eta} && \text{Magnetic Prandtl Number} \\
 E &= \frac{\nu}{\Omega_0 L^2} && \text{Ekman Number} \\
 Ra &= \frac{\alpha g_0 \Delta T L^3}{\nu \kappa} && \text{Rayleigh Number,}
 \end{aligned}$$

where α is coefficient of thermal expansion, g_0 is the gravitational acceleration at the top of the domain, and κ is the thermal diffusivity.

In addition, ohmic and viscous heating, which do not appear in the Boussinesq formulation, are turned off when this nondimensionalization is specified at runtime. Rayleigh solves the following equations when running in nondimensional Boussinesq mode:

$$\begin{aligned}
 \left[\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} + \frac{2}{E} \hat{\mathbf{z}} \times \mathbf{v} \right] &= \frac{Ra}{Pr} \left(\frac{r}{r_o} \right)^n \Theta \hat{\mathbf{r}} - \frac{1}{E} \nabla P + \frac{1}{E Pm} (\nabla \times \mathbf{B}) \times \mathbf{B} + \nabla^2 \mathbf{v} && \text{Momentum} \\
 \left[\frac{\partial \Theta}{\partial t} + \mathbf{v} \cdot \nabla \Theta \right] &= \frac{1}{Pr} \nabla^2 \Theta && \text{Thermal Energy} \\
 \frac{\partial \mathbf{B}}{\partial t} &= \nabla \times (\mathbf{v} \times \mathbf{B}) + \frac{1}{Pm} \nabla^2 \mathbf{B} && \text{Induction} \\
 \nabla \cdot \mathbf{v} &= 0 && \text{Solenoidal Velocity Field} \\
 \nabla \cdot \mathbf{B} &= 0 && \text{Solenoidal Magnetic Field,}
 \end{aligned}$$

where $r_o \equiv r_{max}$. In this formulation, Θ should be interpreted as the temperature perturbation T . Those Reference_Namelist variables that must be set for this model are indicated in Table 6.

Note that our choice for the temperature scale assumes fixed-temperature boundary conditions. We might choose to specify fixed-flux boundary conditions and/or an internal heating, in which case the meaning of ΔT in our equation set changes, with $\Delta T \equiv L \frac{\partial T}{\partial r}$ instead, for some fiducial value of $\frac{\partial T}{\partial r}$. Which regard to the temperature scaling, it is up to the user to select boundary conditions appropriate for their desired values of ΔT . If ΔT denotes the temperature contrast across the domain, then their boundary condition variables should look like:

```

&boundary\_conditions\_namelist
T_Top    = 0.0d0
T_Bottom = 1.0d0
fix_tvar_top = .true.
fix_tvar_bottom = .true.
/

```

Variable	Description
Ekman_Number	The Ekman Number E
Rayleigh_Number	The Rayleigh Number Ra
Prandtl_Number	The Prandtl Number Pr
Magnetic_Prandtl_Number	The Magnetic Prandtl Number Pm
Gravity_Power	Buoyancy coefficient = $\frac{Ra}{Pr} \left(\frac{r}{r_{max}} \right)^{gravity_power}$

Table 6: Variables in the Reference_Namelist that must be specified when running in nondimensional Boussinesq mode. In addition, reference_type=1 must also be specified.

Alternatively, if the temperature scale is determined by a gradient at one boundary, the user should ensure that the amplitude of the temperature gradient at that boundary is 1. For example:

```
&boundary\_conditions\_namelist
dTdr_bottom = -1.0d0
fix_dtdr_bottom = .true.
/
```

Boundary conditions and internal heating are discussed in §4.4. Finally, in Boussinesq mode, the namelist variables **nu_type**, **kappa_type**, and **eta_type** should be set to 1. Their values will be determined by Pr and Pm, instead of nu_top, kappa_top, or eta_top.

4.3 Anelastic Mode (nondimensional)

Example Input: Rayleigh/etc/input_examples/main_input_jupiter

When running in nondimensional anelastic mode, you must set **reference.type=3** in the Reference_Namelist. When this parameter is set, the following nondimensionalization is used (following Heimpel et al., 2016, *Nat. Geo*, 9, 19):

$$\begin{aligned}
\text{Length} &\rightarrow L && (\text{Shell Depth}) \\
\text{Time} &\rightarrow \frac{1}{\Omega_0} && (\text{Rotational Timescale}) \\
\text{Temperature} &\rightarrow T_o \equiv \hat{T}(r_{\max}) && (\text{Reference} - \text{State Temperature at Upper Boundary}) \\
\text{Density} &\rightarrow \rho_o \equiv \hat{\rho}(r_{\max}) && (\text{Reference} - \text{State Density at Upper Boundary}) \\
\text{Entropy} &\rightarrow \Delta s && (\text{Entropy Constrast Across Shell}) \\
\text{Magnetic Field} &\rightarrow \sqrt{\rho_o \mu \eta \Omega_0}.
\end{aligned}$$

We assume a polytropic background state (similar to dimensional anelastic mode), with gravity varying as $\frac{1}{r^2}$. We further assume that the transport coefficients ν , κ , and η do not vary with radius. The results in the nondimensionalized equations (tildes used to indicated nondimensional reference-state values):

$$\begin{aligned}
\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} + 2\hat{\mathbf{z}} \times \mathbf{v} &= \text{Ra}^* \frac{r_{\max}^2}{r^2} \Theta \hat{\mathbf{r}} + \nabla \left(\frac{P}{\tilde{\rho}(r)} \right) + \frac{E}{\text{Pm} \tilde{\rho}} (\nabla \times \mathbf{B}) \times \mathbf{B} + \frac{E}{\rho(r)} \nabla \cdot \mathcal{D} && \text{Momentum} \\
\tilde{\rho}(r) \tilde{T}(r) \left[\frac{\partial \Theta}{\partial t} + \mathbf{v} \cdot \nabla \Theta \right] &= \frac{E}{\text{Pr}} \nabla \cdot [\tilde{\rho}(r) \tilde{T}(r) \nabla \Theta] + Q(r) + \frac{E \text{Di}}{\text{Ra}^*} \Phi(r, \theta, \phi) + \frac{\text{Di} E^2}{\text{Pm}^2 \text{R}^*} [\nabla \times \mathbf{B}]^2 && \text{Thermal Energy} \\
\frac{\partial \mathbf{B}}{\partial t} &= \nabla \times \left(\mathbf{v} \times \mathbf{B} - \frac{E}{\text{Pm}} \nabla \times \mathbf{B} \right) && \text{Induction} \\
\mathcal{D}_{ij} &= 2\tilde{\rho}(r) \left[e_{ij} - \frac{1}{3} \nabla \cdot \mathbf{v} \right] && \text{Viscous Stress Tensor} \\
\Phi(r, \theta, \phi) &= 2\tilde{\rho}(r) \left[e_{ij} e_{ij} - \frac{1}{3} (\nabla \cdot \mathbf{v})^2 \right] && \text{Viscous Heating} \\
\nabla \cdot (\tilde{\rho}(r) \mathbf{v}) &= 0 && \text{Solenoidal Mass Flux} \\
\nabla \cdot \mathbf{B} &= 0. && \text{Solenoidal Magnetic Field}
\end{aligned}$$

In the equations above, Di is the dissipation number, defined by

$$\text{Di} = \frac{g_o L}{c_p T_o}, \quad (5)$$

where g_o and T_o are the gravitational acceleration and temperature at the outer boundary respectively. Once more, the thermal anomaly Θ should be interpreted as entropy s . The symbol Ra^* is the modified Rayleigh number, given by

$$\text{Ra}^* = \frac{g_o}{c_p \Omega_0^2} \frac{\Delta s}{L} \quad (6)$$

Those Reference_Namelist variables that must be set for this model are indicated in Table 7. As with ΔT in the nondimensional Boussinesq mode, the user must choose boundary conditions suitable for their definition of Δs . As with the dimensional anelastic formulation, the background state is polytropic and is described through a polytropic index and number of density scale heights.

Note: As with the Boussinesq mode, please set the variables **nu_type**, **kappa_type**, **eta_type** in the Transport_Namelist.

4.4 Boundary Conditions & Internal Heating

Boundary conditions are controlled through the **Boundary_Conditions_Namelist**. All Rayleigh simulations are run with impenetrable boundaries. These boundaries may be either no-slip or stress-free (default). If you want to

Variable	Description
Ekman_Number	The Ekman Number E
Modified_Rayleigh_Number	The Modified Rayleigh Number Ra*
Prandtl_Number	The Prandtl Number Pr
Magnetic_Prandtl_Number	The Magnetic Prandtl Number Pm
poly_n [0]	polytropic index ($P \propto \rho^n$)
poly_Nrho [0]	number of density scaleheights spanning the domain

Table 7: Variables in the Reference_Namelist that must be specified when running in nondimensional anelastic mode. In addition, reference_type=3 must also be specified.

employ no-slip conditions at both boundaries, set **no_slip_boundaries = .true.**. If you wish to set no-slip conditions at only one boundary, set **no_slip_top=.true.** or **no_slip_bottom=.true.** in the Boundary_Conditions_Namelist.

Magnetic boundary conditions are set to match to a potential field at each boundary. There are no supported alternatives at this time.

By default, the thermal anomaly Θ is set to a fixed value at each boundary. The upper and lower boundary-values are specified by setting **T_top** and **T_bottom** respectively in the Boundary_Conditions_Namelist. Their defaults values are 1 and 0 respectively.

Alternatively, you may specify a constant value of $\partial\Theta/\partial r$ at each boundary. This is accomplished by setting the variables **fix_dTdr_top** and **fix_dTdr_bottom**. Values of the gradient may be enforced by setting the namelist variables **dTdr_top** and **dTdr_bottom**. Both default to a value of zero.

4.4.1 Internal Heating

The internal heating function $Q(r)$ is activated and described by two variables in the **Reference_Namelist**. These are **Luminosity** and **heating_type**. Note that these values are part of the **Reference_Namelist** and not the **Boundary_Conditions** namelist. Three heating types (0,1, and 4) are fully supported at this time. Heating type zero corresponds to no heating. This is the default.

Heating_type=1: This heating type is given by :

$$Q(r) = \gamma \hat{\rho}(r) \hat{T}(r) \quad (7)$$

where γ is a normalization constant defined such that

$$4\pi \int_{r=r_{\min}}^{r=r_{\max}} Q(r) r^2 dr = \text{Luminosity}. \quad (8)$$

This heating profile is particularly useful for emulating radiative heating in a stellar convection zone.

Heating_type=4: This heating type corresponds a heating that is variable in radius, but constant in *energy density*. Namely

$$\hat{\rho} \hat{T} \frac{\partial \Theta}{\partial t} = \gamma. \quad (9)$$

The constant γ in this case is also set by enforcing Equation 8.

4.5 General Physics Controls

A number of logical variables can be used to turn certain physics on (value = .true.) or off (value = .false.). These variables are described in Table 8, with default values indicated in brackets.

4.6 Initializing a Model

A Rayleigh simulation may be initialized with a random thermal and/or magnetic field, or it may be restarted from an existing checkpoint file (see §5 for a detailed discussion of checkpointing). This behavior is controlled through the **initial_conditions_namelist** and the **init_type** and **magnetic_init_type** variables. The init_type variable controls the behavior of the velocity and thermal fields at initialization time. Available options are:

Variable	Description
magnetism [.false.]	Turn magnetism on or off
rotation [.false.]	Turn rotation on or off (pressure is not scaled by E when off)
lorentz_forces [.true.]	Turn Lorentz forces on or off (magnetism must be .true.)
viscous_heating [.true.]	Turn viscous heating on or off (inactive in Boussinesq mode)
ohmic_heating [.true.]	Turn ohmic heating off or on (inactive in Boussinesq mode)

Table 8: Variables in the Physical_Controls_Namelist that may be specified to control run behavior (defaults indicated in brackets)

- `init_type=-1` ; read velocity and thermal fields from a checkpoint file
- `init_type=1` ; Christensen et al. (2001) case 0 benchmark init ($\{\ell = 4, m = 4\}$ temperature mode)
- `init_type=6` ; Jones et al. (2011) steady anelastic benchmark ($\{\ell = 19, m = 19\}$ entropy mode)
- `init_type=7` ; random temperature or entropy perturbation

When initializing a random thermal field, all spherical harmonic modes are independently initialized with a random amplitude whose maximum possible value is determined by the namelist variable **temp_amp**. The mathematical form of this random initialization is given by

$$T(r, \theta, \phi) = \sum_{\ell} \sum_m c_{\ell}^m f(r) g(\ell) Y_{\ell}^m(\theta, \phi), \quad (10)$$

where the c_{ℓ}^m 's are (complex) random amplitudes, distributed normally within the range $[-\text{temp_amp}, \text{temp_amp}]$. The radial amplitude $f(r)$ is designed to taper off to zero at the boundaries and is given by

$$f(r) = \frac{1}{2} \left[1 - \cos \left(2\pi \frac{r - r_{\min}}{r_{\max} - r_{\min}} \right) \right]. \quad (11)$$

The amplitude function $g(\ell)$ concentrates power in the central band of spherical harmonic modes used in the simulation. It is given by

$$g(\ell) = \exp \left[-9 \left(\frac{2\ell - \ell_{\max}}{\ell_{\max}} \right)^2 \right], \quad (12)$$

which is itself, admittedly, a bit random.

When initializing using a random thermal perturbation, it is important to consider whether it makes sense to separately initialize the spherically-symmetric component of the thermal field with a profile that is in conductive balance. This is almost certainly the case when running with fixed temperature conditions. The logical namelist variable **conductive_profile** can be used for this purpose. It's default value is `.false.` (off), and its value is ignored completely when restarting from a checkpoint. To initialize a simulation with a random temperature field superimposed on a spherically-symmetric, conductive background state, something similar to the following should appear in your `main_input` file:

```
&initial_conditions_namelist
init_type=7
temp_amp = 1.0d-4
conductive_profile=.true.
/
```

Magnetic-field initialization follows a similar pattern. Available values for `magnetic_input` type are:

- `magnetic_init_type = -1` ; read magnetic field from a checkpoint file
- `magnetic_init_type = 1` ; Christensen et al. (2001) case 0 benchmark init
- `magnetic_init_type = 7` ; randomized vector potential

For the randomized magnetic field, both the poloidal and toroidal vector-potential functions are given a random power distribution described by Equation 10. Each mode's random amplitude is then determined by namelist variable **mag_amp**. This variable should be interpreted as an approximate magnetic field strength (it's value is rescaled appropriately for the poloidal and toroidal vector potentials, which are differentiated to yield the magnetic field).

When initializing all fields from scratch, a main_input file should contain something similar to:

```
&initial_conditions_namelist
init_type=7
temp_amp = 1.0d-4
conductive_profile=.true. ! Not always necessary (problem dependent) ...
magnetic_init_type=7
mag_amp = 1.0d-1
/
```

Filename	Contents
00010000-W	Poloidal Stream function (at time step 10,000)
00010000-Z	Toroidal Stream function
00010000-P	Pressure
00010000-S	Entropy
00010000-C	Poloidal Vector Potential
00010000-A	Toroidal Vector Potential
00010000-WAB	Adams-Bashforth (A-B) terms for radial momentum (W) equation
00010000-ZAB	A-B terms for radial vorticity (Z) equation
00010000-PAB	A-B terms for horizontal divergence of momentum (dWdr) equation
00010000-SAB	A-B terms for Entropy equation
00010000-CAB	A-B terms for C-equation
00010000-AAB	A-B terms for A-equation
00010000-grid_etc	grid and time-stepping info

Table 9: Example checkpoint file collection for a time step 10,000. File contents are indicated.

5 Checkpointing

We refer to saved states in Rayleigh as **checkpoints**. A single checkpoint consists of 13 files when magnetism is activated, and 9 files when magnetism is turned off. A checkpoint written at time step X contains all information needed to advance the system to time step $X+1$. Checkpoint filenames end with a suffix indicating the contents of the file (see Table 9). Each checkpoint filename possess a prefix as well. Files belonging to the same checkpoint share the same prefix. A checkpoint file collection, written at time step 10,000 would look like that shown in Table 9.

These files contain all information needed to advance the system state from time step 10,000 to time step 10,001. Checkpoints come in two flavors, denoted by two different prefix conventions: **standard checkpoints** and **quick-saves**. This section discusses how to generate and restart from both types of checkpoints.

5.1 Standard Checkpoints

Standard checkpoints are intended to serve as regularly spaced restart points for a given run. These files begin with an 8-digit prefix indicating the time step at which the checkpoint was created.

5.1.1 Generating Standard Checkpoints

The frequency with which standard checkpoints are generated can be controlled by modifying the **checkpoint_interval** variable in the **temporal_controls_namelist**. For example, if you want to generate a checkpoint once every 50,000 time steps, you would modify your main_input file to read:

```
&temporal_controls_namelist
  checkpoint_interval = 50000 ! Checkpoint every 50,000 time steps
/
```

The default value of checkpoint_interval is 1,000,000, which is typically much larger than what you will use in practice.

5.1.2 Restarts From Standard Checkpoints

Restarting from a checkpoint is accomplished by first assigning a value of -1 to the variables **init_type** and/or **magnetic_init_type** in the **initial_conditions_namelist**. In addition, the time step from which you wish to restart from should be specified using the **restart_iter** variable in the initial_conditions_namelist. The example below will restart both the magnetic and hydrodynamic variables using the set of checkpoint files beginning with the prefix 00005000.

```
&initial_conditions_namelist
  init_type = -1           !Restart magnetic and hydro variables from time step 5,000
```

```
magnetic_init_type = -1
restart_iter = 5000
/
```

When both values are set to -1, hydrodynamic and magnetic variables are read from the relevant checkpoint files. Alternatively, magnetic and hydrodynamic variables may be initialized separately. This allows you to add magnetism to an already equilibrated hydrodynamic case, for instance. The example below will initialize the system with a random magnetic field, but it will read the hydrodynamic information (W,Z,S,P) from a checkpoint created at time step 50,000:

```
&initial_conditions_namelist
  init_type = -1           ! Restart hydro from time step 5,000
  magnetic_init_type = 7   ! Add a random magnetic field
  restart_iter = 5000
/
```

In addition to specifying the checkpoint time step manually, you can tell Rayleigh to simply restart using the last checkpoint written by assigning a value of zero to `init_type`:

```
&initial_conditions_namelist
  init_type = -1
  magnetic_init_type = 7
  restart_iter = 0        ! Restart using the most recent checkpoint
/
```

In this case, Rayleigh reads the **last_checkpoint** file (found within the Checkpoints directory) to determine which checkpoint it reads.

5.2 Quicksaves

In practice, Rayleigh checkpoints are used for two purposes: (1) guarding against unexpected crashes and (2) supplementing a run's record with a series of restart points. While standard checkpoints may serve both purposes, the frequency with which checkpoints are written for purpose (1) is often much higher than that needed for purpose (2). This means that a lot of data culling is performed at the end of a run or, if disk space is a particularly limiting factor, during the run itself. For this reason, Rayleigh has a **quicksave** checkpointing scheme in addition to the standard scheme. Quicksaves can be written with high-cadence, but require low storage due to the rotating reuse of quicksave files.

5.2.1 1.2.1 Generating Quicksaves

The cadence with which quicksaves are written can be specified by setting the **quicksave_interval** variable in the **temporal_controls_namelist**. Alternatively, the elapsed wall time (in minutes) that passes between quicksaves may be controlled by specifying the **quicksave_minutes** variable. If both `quicksave_interval` and `quicksave_minutes` are specified, `quicksave_minutes` takes precedence.

What distinguishes quicksaves from standard checkpoints is that only a specified number of quicksaves exist on the disk at any given time. That number is determined by the value of **num_quicksaves**. Quicksave files begin with the prefix *quicksave_XX*, where XX is a 2-digit code, ranging from 1 through `num_quicksaves` and indicates the quicksave number. Consider the following example:

```
&temporal_controls_namelist
  checkpoint_interval = 35000 ! Generate a standard checkpoint once every 35,000 time steps
  quicksave_interval = 10000 ! Generate a quicksave once every 10,000 time steps
  num_quicksaves = 2         ! Keep only two quicksaves on disk at a time
/
```

At time step 10,000, a set of checkpoint files beginning with prefix `quicksave_01` will be generated. At time step

20,000, a set of checkpoint files beginning with prefix `quicksave_02` will be generated. Following that, at time step 30,000, another checkpoint will be generated, *but it will overwrite the existing `quicksave_01` files*. At time step 40,000, the `quicksave_02` files will be overwritten, and so forth. Because the `num_quicksaves` was set to 2, filenames with prefix `quicksave_03` will never be generated.

Note that checkpoints beginning with an 8-digit prefix (e.g., 00035000) are still written to disk regularly and are not affected by the quicksave checkpointing. On time steps where a quicksave and a standard checkpoint would both be written, only the standard checkpoint is written. Thus, at time step 70,000 in the example above, a standard checkpoint would be written, and the files beginning with `quicksave_01` would remain unaltered.

5.2.2 1.2.2 Restarting from Quicksaves

Restarting from `quicksave_XX` may be accomplished by specifying the value of `restart_iter` to be `-XX` (i.e., the negative of the quicksave you wish to restart from). The following example shows how to restart the hydrodynamic variables from `quicksave_02`, while also initializing a random magnetic field. “

```
&initial_conditions_namelist
  init_type = -1      ! Restart hydro variables from a checkpoint
  magnetic_init_type = 7 ! Initialize a random magnetic field
  restart_iter = -2    ! Restart from quicksave number 2
/
```

Note that the file `last_checkpoint` contains the number of last checkpoint written. This might be a quicksave or a standard checkpoint. Specifying a value of zero for `restart_iter` thus works with quicksaves and standard checkpoints alike.

5.3 1.3 Checkpoint Logs

When checkpoints are written, the number of the most recent checkpoint is appended to a file named **checkpoint_log**, found in the Checkpoints directory. The checkpoint log can be used to identify the time step number of a quicksave file that otherwise has no identifying information. While this information is also contained in the `grid.etc` file, those are written in unformatted binary and cumbersome to access from the terminal command line.

An entry in the log of "00050000 02" means that a checkpoint was written at time step 50,000 to `quicksave_02`. An entry lacking a two-digit number indicates that a standard checkpoint was written at that time step. The most recent entry in the checkpoint log always comes at the end of the file.

6 Diagnostic Outputs

Rayleigh comes bundled with an in-situ diagnostics package that allows the user to sample a simulation in a variety of ways, and at user-specified intervals throughout a run. This package is comprised of roughly 17,000 lines of code (about half of the Rayleigh code base), and it is complex enough that we describe it in two other documents. We refer the user to :

1. The diagnostics plotting manual, provided in three formats:
 - `Rayleigh/etc/analysis/Diagnostics_Plotting.ipynb` (Jupyter Python notebook format; recommended for interactive use)
 - `Rayleigh/doc/Diagnostics_Plotting.html` (recommended for optimal viewing; generated from the `.ipynb` file)
 - `Rayleigh/doc/Diagnostics_Plotting.pdf` (same content as `.html` and `.ipynb`, but formatting quality is inferior)
2. `Rayleigh/doc/rayleigh_output_variables.pdf` – This companion document provides the output menu system referred to in the main diagnostics documentation.

A number of stand-alone Python plotting examples may also be found in the `Rayleigh/etc/analysis` directory.

7 I/O Redirection

Rayleigh writes all text output (e.g., error messages, iteration counter, etc.) to stdout by default. Different computing centers handle stdout in different ways, but typically one of two paths is taken. On some machines, a log file is created immediately and updated continuously as the simulation runs. On other machines, stdout is buffered on-node and written to disk only when the run has terminated.

There are situations where it can be advantageous to have a regularly updated log file whose update frequency may be controlled. This feature exists in Rayleigh and may be accessed by assigning values to **stdout_flush_interval** and **stdout_file** in the io controls namelist.

```
&io_controls_namelist
stdout_flush_interval = 1000
stdout_file = 'routput'
/
```

Set stdout_file to the name of a file that will contain Rayleigh's text output. In the example above, a file named *routput* will appear in the simulation directory and will be updated periodically throughout the run. The variable stdout_flush_interval determines how many lines of text are buffered before they are flushed to routput. Rayleigh prints time-step information during each time step, and so setting this variable to a relatively large number (e.g., 100+) prevents excessive disk access from occurring throughout the run. In the example above, a text buffer flush will occur once 1000 lines of text have been accumulated.

Changes in the time-step size and self-termination of the run will also force a text-buffer flush. Unexpected crashes and sudden termination by the system job scheduler do not force a buffer flush. Note that the default value of stdout_file is **'nofile'**. If this value is specified, output will be directed to normal stdout.

8 Ensemble Mode

Rayleigh can also be used to run multiple simulations under the umbrella of a single executable. This functionality is particularly useful for running parameter space studies, which often consist of multiple, similarly-sized simulations, in one shot. Moreover, as some queuing systems favor large jobs over small jobs, an ensemble mode is useful for advancing multiple small simulations through the queue in a reasonable timeframe.

Running Rayleigh in ensemble mode is relatively straightforward. To begin with, create a directory for each simulation as you normally would, and place an appropriately modified `main_input` into each directory. These directories should all reside within the same parent directory. Within that parent directory, you should place a copy of the Rayleigh executable (or a softlink). In addition, you should create a text file named **run_list** that contains the name of each simulation directory, one name per line. An ensemble job may then be executed by calling Rayleigh with **nruns** command line flag as:

```
user@machinename ~/runs/ $ mpiexec -np Y ./rayleigh.opt -nruns X
```

Here, Y is the total number of cores needed by all X simulations listed in `run_list`.

Example: Suppose you wish to run three simulations at once from within a parent directory named *ensemble* and that the simulation directories are named `run1`, `run2`, and `run3`. When performing an `ls` from within *ensemble*, you should see 5 items.

```
user@machinename ~/runs/ $ cd ensemble
user@machinename ~/runs/ensemble $ ls
rayleigh.opt      run1      run2      run3      run_list
```

In this example, the contents of `run_list` should be the *local* names of your ensemble run-directories, namely `run1`, `run2`, and `run3`.

```
user@machinename ~/runs/ensemble $ more run_list
run1
run2
run3
<-- place an empty line here
```

Note that some Fortran implementations will not read the last line in `run_list` unless it ends in a newline character. Avoid unexpected crashes by hitting "enter" following your final entry in `run_list`.

Before running Rayleigh, make sure you know how many cores each simulation needs by examining the `main_input` files:

```
user@machinename ~/runs/ensemble $ head run1/main_input
&problemlist
&problemsize_namelist
  n_r = 128
  n_theta = 192
  nprow = 16
  npcol = 16
/

user@machinename ~/runs/ensemble $ head run2/main_input
&problemlist
&problemsize_namelist
  n_r = 128
  n_theta = 384
  nprow = 32
  npcol = 16
/

user@machinename ~/runs/ensemble $ head run3/main_input
&problemlist
&problemsize_namelist
  n_r = 64
```

```
n_theta = 192
nprow = 16
npcol = 16
/
```

In this example, we need a total of 1024 cores (256+512+256) to execute three simulations, and so the relevant call to Rayleigh would be:

```
user@machinename ~/runs/ $ mpiexec -np 1024 ./rayleigh.opt -nruns 3
```

Closing Notes: When running in ensemble mode, it is *strongly recommended* that you redirect standard output for each simulation to a text file (see §7). Otherwise, all simulations write to the same default (machine-dependent) log file, making it difficult to read. Moreover, some machines such as NASA Pleiades will terminate a run if the log file becomes too long. This is easy to do when multiple simulations are writing to the same file.

Finally, The flags `-nprow` and `-npcol` **are ignored** when `-nruns` is specified. The row and column configuration for all simulations needs to be specified in their respective `main.input` files instead.

9 Acknowledging Rayleigh

We politely ask that you acknowledge the author, the National Science Foundation, and CIG in any work enabled by Rayleigh. This includes work deriving from results generated by Rayleigh. It also includes work that derives from the use of software that incorporates original or modified Rayleigh source code.

Sample acknowledgement text can always be found in Rayleigh/ACKNOWLEDGE.

You are reading the documentation for Rayleigh version 0.9.0. A formal paper describing the numerics and parallelization methods is forthcoming. In the interim, please cite the following two references in any academic publications that present results enabled by Rayleigh:

1. Featherstone, N.A. & Hindman, B.W., 2016, *Astrophys. J.*, **818**, 32, DOI: [10.3847/0004-637X/818/1/32](https://doi.org/10.3847/0004-637X/818/1/32)
2. Matsui et al., 2016, *Geochemistry, Geophysics, Geosystems*, **17**,1586, DOI: [10.1002/2015GC006159](https://doi.org/10.1002/2015GC006159)