COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

RAYLEIGH

User Manual

Version 1.1.0

(generated May 7, 2024)

Nicholas Featherstone

with contributions by:
Kyle Augustson, Wolfgang Bangerth, Rene Gassmöller, Sebastian Glane, Brad Hindman, Lorraine Hwang, Hiro Matsui, Ryan Orvedahl, Krista Soderlund, Cian Wilson, Maria Weber, Rakesh Yadav

geodynamics.org

CONTENTS

Rayleigh solves the magnetohydrodynamic (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 documentation is structured into the following sections:

CHAPTER

ONE

USER GUIDE

1.1 Getting Started

1.1.1 Accessing Rayleigh

You can download the Rayleigh source code from Rayleigh's GitHub respository .

1.1.2 Setting up a Rayleigh Development Environment

When running Rayleigh on HPC resources, always compile the software with the recommended compiler and link against libraries optimized for the architecture you are running on. We provide example instructions for some common systems at **`Installation Instructions for HPC systems`_**.

When developing Rayleigh or editing its documentation, however, such optimizations are rarely necessary. Instead, it is sufficient for the code and documentation to compile. For this purpose, we recommend setting up a *conda environment* or using our *Docker container*. Instructions for setting up an environment on Linux and Mac OS are provided below.

Conda Environment

First, if you don't have Conda, you should download and install the version appropriate for your architecture here.

Once you have Conda installed, create a Conda environment using the environment files we provide in Rayleigh's main directory.

```
conda config --set channel_priority strict
conda env create -f environment.yml
conda activate radev
```

Because Rayleigh installs a number of different packages the first of these lines simplifies the search for a compatible selection of package versions. The second line creates a new environment named "radev" and installs all necessary packages. The third line activates the environment. This command will likely take a while (a few minutes).

If you want to undo the change to the channel priority setting afterwards, you can run .. code-block:: bash

    conda config –set channel_priority flexible

after installing the environment (only necessary if you depend on the more flexible conda solver).

MKL Setup: Linux and Mac

Once your packages are installed, you will most likely want to have the `MKLROOT` environment variable set whenever you activate your Conda environment. To do this we set `MKLROOT` to the location of the currently activated conda environment from the enviroment variable `CONDA_PREFIX`.

```
export MKLROOT="$CONDA_PREFIX"
```

Note that this is Bash syntax (use setenv if running c-shell). Note that there should be no spaces on either side of the "=" sign. If you stop here, you will have to do this every time you activate your development en-

vironment. To have this happen automatically, you only need to add two small scripts to radev/etc/conda/activate.d and radev/etc/conda/deactivate.d directories. Scripts in these directories are automatically executed when your conda environment is activated and deactivated, respectively.

Change to your activate.d directory (for me, this was /custom/software/miniconda3/envs/radev/etc/conda/activate.d) and create a file named activate_mkl.sh with the following three lines:

```
#!/bin/bash
export MKLSAVE="$MKLROOT"
export MKLROOT="$CONDA_PREFIX"
```

In the deactivate.d directory, create a file named deactivate_mkl.sh with the following two lines:

```
#!/bin/bash
export MKLROOT="$MKLSAVE"
```

Now, try it out.

```
conda deactivate
echo $MKLROOT
conda activate radev
echo $MKLROOT
```

The MKLSAVE variable is used so that a separate MKL installation on your machine, if one exists, is properly reset in your environment following deactivation.

Configuration and Compilation

Building the documentation is the same on Linux and Mac.

```
conda activate radev
cd /path/to/Rayleigh
make doc
```

Once the documetation builds, you can access it by opening Rayleigh/doc/build/html/index.html in your web browser.

Building the code is again the same on Linux and Mac. Execute the following:

```
conda activate radev
cd /path/to/Rayleigh
./configure -conda-mkl --FC=mpifort
make
```

At this point, you can run “make install,” and run the code using mpirun as you normally would (keep the radev environment active when doing this).

Docker Container

Docker provides a standardized way to build, distribute and run containerized environments on Linux, macOS, and Windows. To get started you should install Docker on your system following the instructions from here. On Linux you can likely also install it from a distribution package (e.g., `docker-io` on Debian/Ubuntu). Podman is an alternative runtime that can run Docker containers and can be used as a drop-in replacement for Docker. Singularity/Apptainer are other available alternatives that are more commonly used on HPC systems.

Launching the container

You can launch our pre-built container that is hosted on Docker Hub from a terminal. This container is set up to get used to Rayleigh not to run productive models with it.

This command will create a terminal inside the container and drop you in a directory that contains a pre-compiled version of Rayleigh. You can run input examples or tests by executing *rayleigh.opt* or *rayleigh.dbg* and look at the output files, but all files will be deleted when you *exit* the container.

Note: If you use Apptainer/Singularity instead of docker you can keep the model output files, because Apptainer by default mounts the current directory into the container. The command to run Rayleigh inside the container is `mpirun -np X apptainer exec geodynamics/rayleigh:latest rayleigh.opt`` (assuming you have a Rayleigh input file in the current directory).

We also provide a container with a development environment for Rayleigh that allows you to change the code, build the documentation and the code, and to keep model outputs. The following command is for GNU/Linux and macOS users.

```
./docker-devel
# This runs the following command:
# docker run -it --rm -v $HOME:/work -e HOSTUID=$UID -e HOSTGID=$GROUPS -e␣
↪HOSTUSER=$USER geodynamics/rayleigh-devel-jammy:latest
```

This will give you a shell inside the container and mount your home directory at `/work`. You can clone, configure, build, and run the code and analyze the outputs using Python inside the container. Any changes below `/work` will be reflected in your home directory. Any other changes to the container will be deleted once you exit the shell.

Note: Your user has `sudo` rights within the container. This allows to install packages using the `apt` command or modify the system in any other way.

Windows users should run the script `docker-devel.bat` instead.

Configuration and Compilation

Note: All these commands are run inside the Docker container and assume you have a copy of Rayleigh at `$HOME/path/to/Rayleigh` (which corresponds to `/work/path/to/Rayleigh` inside the container).

Building the documentation

```
cd /work/path/to/Rayleigh
make doc
```

Building the code

```
cd /work/path/to/Rayleigh
./configure --with-fftw=/usr
make
```

Updating the container

On the first launch of the container, your local Docker engine will automatically download our pre-built container from Docker Hub. Subsequent launches will just use this container and will not check for updates. You can download a newer version of the container using the following command.

```
docker pull geodynamics/rayleigh-devel-jammy:latest
```

Building the container

Note: This step purely optional. You only need to do this if you cannot pull the container from Docker Hub or you want to modify the Dockerfile.

To build the container you have to run this command from your host system (i.e., not from inside the container).

```
cd docker
docker build -t geodynamics/rayleigh-devel-jammy:latest rayleigh-devel-jammy
```

You can check the newly built container is there using this command.

```
docker images
```

Spack Environment

Spack can be used to create a development environment to build the code in a local directory. First set up Spack using the instructions in *Alternative: Installation using Spack*

Afterwards create a new environment, activate it and set the status of the Rayleigh package to development. We select `$PWD` as the path, so run this command from the base directory of your git clone.

```
spack env create rayleigh
spack env activate rayleigh
spack add rayleigh@master
spack develop -p "$PWD" rayleigh@master
```

A subsequent `spack install` will install necessary dependencies and build Rayleigh in the selected directory.

1.1.3 Installing Rayleigh

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

https://github.com/geodynamics/Rayleigh/blob/main/INSTALL.

We provide an abbreviated version of those instructions here.

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, IBM, AOCC, and Cray compilers (PGI has not been tested yet). Rayleigh's configure script provides native support for the Intel, GNU, AOCC, and Cray 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.

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 produces the code. You can run **make -j** to build several files in parallel and speed up the build this way.
3. **make install** – This places the Rayleigh executables in Rayleigh/bin. If you would like to place them in (say) /home/my_rayleigh/bin, run configure as: **./configure –prefix=/home/my_rayleigh**, i.e., the executables will be placed in the **$(prefix)/bin** directory.

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.

The default behavior of the **make** command is to build both the optimized, **rayleigh.opt**, and the debug versions, **rayleigh.dbg**. As described above, if Intel is used and *all* is selected, every version will be compiled. To build only a single version, the **target=<target>** option may be used at the **make** stage, for example:

1. **make target=opt** - build only the optimized version, **rayleigh.opt**
2. **make target=dbg** - build only the debug version, **rayleigh.dbg**
3. **make target=avx** - build only the AVX version, **rayleigh.avx**

When building a single target, the final name of the executable can be changed with the **output=<output>** option during the **make install** command. For example, to build the optimized version and name the executable **a.out**:

1. **make target=opt** - only build the optimized version
2. **make target=opt output=a.out install** - install the optimized version as **a.out**

Inspection of the **$(prefix)/bin** directory (specified at configure time with the -prefix option) will show a new file named **a.out**.

If both the optimized version and the debug version have already been built, they can be renamed at install time as:

1. **make** - build both optimized and debug version (or all versions)
2. **make target=opt output=a.out.opt install** - install and rename the optimized version
3. **make target=dbg output=a.out.dbg install** - install and rename the debug version

The **output** option is only respected when a particular **target** is specified. Running **make output=a.out install** will install all **rayleigh.*** executables, they will not be renamed.

Alternative: Installation using Spack

Spack is a package management tool designed to support multiple versions and configurations of software on a wide variety of platforms and environments. It can be used to build Rayleigh with different compilers and a custom set of libraries for MPI, LAPACK, and FFTW. It can automatically build dependencies itself or use those provided by the HPC environment.

To set up Spack in your environment follow the instructions in the documentation. Add local compilers and packages as desired.

The next step has only to be performed once to add the Rayleigh package repository. Run this from the base directory of the Rayleigh repository.

```
spack repo add spack-repo
```

Afterwards you can just install Rayleigh and its dependencies using:

```
spack install rayleigh
```

Once the build succeeded the package can be loaded using the following command, which will make the `rayleigh.opt` and `rayleigh.dbg` executables available in the `PATH` and can be run to start simulations as usual.

```
spack load rayleigh
```

There are many ways in which to modify the compiler and dependencies being used. They can be found in the Spack documentation.

As an example you can install Rayleigh using MKL for LAPACK and FFTW using:

```
spack install rayleigh ^intel-mkl
```

To see the dependencies being installed you can use:

```
spack spec rayleigh ^intel-mkl
```

1.1.4 Installation on HPC systems

Given the amount of computational resources required to simulate convection in highly turbulent parameter regimes, many users will want to run Rayleigh in a HPC environment. Here we provide instructions for compilation on two widely-used, national-scale supercomputing systems: TACC Stampede2 and NASA Pleiades.

Example jobscripts containing the necessary commands to compile and run Rayleigh on various systems may be found in *Rayleigh/job_scripts/*.

TACC Stampede2

Installing Rayleigh on NSF's Stampede 2 system is straightforward. At the time this documentation is written (Sep 2022) the loaded default modules work out of the box for Rayleigh. In case the modules change in the future here is a listed for reference:

```
1) intel/18.0.2      3) impi/18.0.2   5) autotools/1.1    7) cmake/3.20.2   9) ␣
↪TACC
2) libfabric/1.7.0   4) git/2.24.1    6) python2/2.7.15   8) xalt/2.10.37
```

After cloning a Rayleigh repository, rayleigh can be configured and compiled as:

```
FC=mpifc CC=mpicc ./configure  # select 'AVX512'
make -j
make install
```

We suggest choosing 'AVX512' at the configure menu. This vectorization is supported by both the Skylake and Ice Lake nodes available on Stampede2. An example jobscript for Stampede2 may be found in *Rayleigh/job_scripts/TACC_Stampede2*.

NASA Pleiades

Installation on NASA's Pleiades cluster is similarly straightforward. After cloning the repository, Rayleigh can be configured and compiled via the following commands:

```
module purge
module load comp-intel
module load mpi-hpe
./configure --FC=mpif90 --CC=icc  # select 'ALL'
make -j
make install
```

We suggest using the default Intel and MPI compilers provided by Pleiades as in the example above. As of December, 2022, this corresponded to the following version combination:

```
1) comp-intel/2020.4.304   2) mpi-hpe/mpt.2.25
```

Note that Pleiades is a heterogeneous cluster, composed of many (primarily Intel) processor types. We suggest selecting the 'ALL' option when configuring Rayleigh to ensure that a unique executable is created for each of the possible vectorization options. An example jobscript for Pleiades may be found in *Rayleigh/job_scripts/NASA_Pleiades*.

1.1.5 Verifying Your Installation

Rayleigh has been programmed with internal testing suite so that its results may be compared against benchmarks described in Christensen et al. (2001) [CAC+01] and Jones et al. (2011) [JBB+11]

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) [CAC+01] reference describes two Boussinesq tests that Rayleigh's results may be compared against. The Jones et al. (2011) [JBB+11] 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 *table_benchmark* below. In addition to the input files listed in Table *table_benchmark*, 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. For example, setting `benchmark_mode = 1` defines the appropriate Case 0 Christensen et al. (2001) [CAC+01] initial conditions. A benchmark report is written every 5000 time steps by setting `benchmark_report_interval = 5000`. The benchmark reports are text files found within directory **path_to_my_sim/Benchmark_Reports/** and numbered according to the appropriate time step. The | `benchmark_integration_interval` variable sets the interval at which measurements are taken to calculate the values reported in the benchmark reports.

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 [CAC+01].

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/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

$$N = X \times Y.$$

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 *table_benchmark_low* or *table_benchmark_high* . 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.

Benchmark Low

Rayleigh benchmark report for Christensen et al. (2001) [CAC+01] 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	58.347827	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

Benchmark High

Rayleigh benchmark report for Christensen et al. (2001) [CAC+01] 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
```

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

1.1.6 Available Benchmarks

Benchmark

Benchmark-input examples useful for verifying Rayleigh's installation. Those from Christensen et al. (2001) [CAC+01] are Boussinesq. Those from Jones et al. (2011) [JBB+11] are anelastic. Examples are found in the directory: Rayleigh/input_examples/

Paper	Benchmark	Input File	Specify in the main_input file
Christensen et al.	Case 0	c2001_case0_minimal	benchmark_mode = 1
Christensen et al.	Case 1(MHD)	c2001_case1_minimal	benchmark_mode = 2
Jones et al. 2011	Steady Hydro	j2011_steady_hydro_minimal	benchmark_mode = 3
Jones et al. 2011	Steady MHD	j2011_steady_MHD_minimal	benchmark_mode = 4
Breuer et al. 2010	Case 0	b2010_case0_*T_input	

Standard benchmarks that generate minimal output files are discussed in the next four benchmarks:

- *Boussinesq non-MHD Benchmark: c2001_case0_minimal*
- *Boussinesq MHD Benchmark: c2001_case1_minimal*
- *Steady Anelastic non-MHD Benchmark: j2011_steady_hydro_minimal*
- *Steady Anelastic MHD Benchmark: j2011_steady_mhd_minimal*
- *Steady Thermal-Chemical Boussinesq Convection Benchmark: b2010_case0_*T_input*

Boussinesq non-MHD Benchmark: c2001_case0_minimal

This is the standard benchmark test when running Rayleigh on a new machine. Christensen et al. (2001) [CAC+01] describes two Boussinesq tests that Rayleigh's results may be compared against. Case 0 in Christensen et al. (2001) [CAC+01] solves for Boussinesq (non-dimensional) non-magnetic convection, and we will discuss the input parameters necessary to set up this benchmark in Rayleigh below. Rayleigh's input parameters are grouped in so-called namelists, which are subcategories of related input parameters that will be read upon program start and assigned to Fortran variables with identical names. Below are the first four Fortran namelists in the input file **c2001_case0_minimal**.

```
&problemsize_namelist
 n_r = 64
 n_theta = 96
 nprow = 16
 npcol = 32
/
&numerical_controls_namelist
/
&physical_controls_namelist
 benchmark_mode = 1
 benchmark_integration_interval = 100
 benchmark_report_interval = 5000
/
&temporal_controls_namelist
 max_iterations = 25000
 checkpoint_interval = 100000
 quicksave_interval = 10000
 num_quicksaves = 2
/
```

Boussinesq MHD Benchmark: c2001_case1_minimal

The MHD Boussinesq benchmark with an insulating inner core of Christensen et al. (2001) [CAC+01] is denoted as Case 1 and is specified with input file **c2001_case1_minimal**. Only the namelists modified compared to Case 0 (*Boussinesq non-MHD Benchmark: c2001_case0_minimal* above) are shown below.

```
&physical_controls_namelist
 benchmark_mode = 2
 benchmark_integration_interval = 100
```

(continues on next page)

(continued from previous page)

```
 benchmark_report_interval = 10000
/
&temporal_controls_namelist
 max_iterations = 150000
 checkpoint_interval = 100000
 quicksave_interval = 10000
 num_quicksaves = 2
/
```

Steady Anelastic non-MHD Benchmark: j2011_steady_hydro_minimal

Jones et al. (2011) describes a benchmark for an anelastic hydrodynamic solution that is steady in a drifting frame. This benchmark is specified for Rayleigh with input file **j2011_steady_hydro_minimal**. Below are the relevant Fortran namelists.

```
&problemsize_namelist
 n_r = 128
 n_theta = 192
 nprow = 32
 npcol = 16
/
&numerical_controls_namelist
/
&physical_controls_namelist
 benchmark_mode = 3
 benchmark_integration_interval = 100
 benchmark_report_interval = 10000
/
&temporal_controls_namelist
 max_iterations = 200000
 checkpoint_interval = 100000
 quicksave_interval = 10000
 num_quicksaves = 2
/
```

Steady Anelastic MHD Benchmark: j2011_steady_mhd_minimal

The anelastic MHD benchmark described in Jones et al. (2011) can be run with main input file **j2011_steady_mhd_minimal**. The Fortran namelists differing from the Jones et al. (2011) anelastic hydro benchmark (§:ref:*cookbookHydroAnelastic* above) are shown here.

```
&physical_controls_namelist
 benchmark_mode = 4
 benchmark_integration_interval = 100
```

(continues on next page)

(continued from previous page)

```
 benchmark_report_interval = 10000
/
&temporal_controls_namelist
 max_iterations = 5000000
 checkpoint_interval = 100000
 quicksave_interval  = 25000
 num_quicksaves = 2
/
```

Steady Thermal-Chemical Boussinesq Convection Benchmark: b2010_case0_*T_input

This is a Boussinesq convection benchmark described in Breuer et al. (2010) [BMW+10] in a dual buoyancy system that allows both thermal and chemical buoyancy sources. The case 0 contains three input lists that describes varying contributions of thermal vs chemical Rayleigh numbers whereas the total Rayleigh number stays the same. This benchmark is specified for Rayleigh with input file b2010_case0_*T_input. Below is an example for 80% thermal and 20% chemical convection scene for the relevant Fortran namelists:

```
&problemsize_namelist
 n_r = 128
 n_theta = 192
 nprow = 32
 npcol = 16
&Reference_Namelist
 Ekman_Number = 1.0d-3
 Rayleigh_Number = 4.8d4
 Prandtl_Number = 3.0d-1
 chi_a_Rayleigh_Number = -1.2d5
 chi_a_Prandtl_Number = 3.0d0
```

1.2 Underlying Physics

Rayleigh solves the MHD equations in spherical geometry under the Boussinesq and anelastic approximations. This section will provide a basic overview of those equations as well as the mathematical approach Rayleigh uses to solve them.

1.2.1 Notation and Conventions

Vector and Tensor Notation

All vector quantities are represented in bold italics. Components of a vector are indicated in non-bold italics, along with a subscript indicating the direction associated with that component. Unit vectors are written in lower-case, bold math font and are indicated by the use of a *hat* character. For example, a vector quantity $\boldsymbol{a}$ would represented as

$$\boldsymbol{a} = a_r\hat{\boldsymbol{a}} + a_\theta\hat{\boldsymbol{\theta}} + a_\phi\hat{\boldsymbol{\phi}}. \tag{1.1}$$

The symbols ($\hat{\boldsymbol{r}}$, $\hat{\boldsymbol{\theta}}$, $\hat{\boldsymbol{\phi}}$) indicate the unit vectors in the (r,θ,ϕ) directions, and (a_r, a_θ, a_ϕ) indicate the components of $\boldsymbol{a}$ along those directions respectively.

Vectors may be written in lower case, as with the velocity field $\boldsymbol{v}$, or in upper case as with the magnetic field $\boldsymbol{B}$. Tensors are indicated by bold, upper-case, script font, as with the viscous stress tensor $\boldsymbol{\mathcal{D}}$. Tensor components are indicated in non-bold, and with directional subscripts (i.e., $\mathcal{D}_{r\theta}$).

Reference-State Values

The *hat* notation is also used to indicate reference-state quantities. These quantities are scalar, and they are not written in bold font. They vary only in radius and have no θ-dependence or ϕ-dependence. The reference-state density is indicated by $\hat{\rho}$ and the reference-state temperature by $\hat{T}$, for instance.

Averaged and Fluctuating Values

Most of the output variables have been decomposed into a zonally-averaged value, and a fluctuation about that average. The average is indicated by an overbar, such that

$$\overline{a} \equiv \frac{1}{2\pi}\int_0^{2\pi} a(r,\theta,\phi)\,\mathrm{d}\phi. \tag{1.2}$$

Fluctations about that average are indicated by a *prime* superscript, such that

$$a'(r,\theta,\phi) \equiv a(r,\theta,\phi) - \overline{a}(r,\theta) \tag{1.3}$$

Finally, some quantities are averaged over the full sphere. These are indicated by a double-zero subscript (i.e. $\ell = 0$, $m = 0$), such that

$$a_{00} \equiv \frac{1}{4\pi}\int_0^{2\pi}\int_0^{\pi} a(r,\theta,\phi)\, r\sin\theta \mathrm{d}\theta\mathrm{d}\phi. \tag{1.4}$$

1.2.2 The System of Equations Solved in Rayleigh

Rayleigh solves the Boussinesq or anelastic MHD equations in spherical geometry. 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 nondimensional anelastic formulations (based on a polytropic reference state) are provided as part of Rayleigh. The user may employ alternative formulations via the custom Reference-state interface. To do so, they must specify the functions f_i and the constants c_i in Equations (1.5)-(1.11) at input time (*in development*).

The general form of the momentum equation solved by Rayleigh is given by

$$\begin{aligned}\mathrm{f}_1(r)\left[\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v} + c_1\hat{\boldsymbol{z}}\times\boldsymbol{v}\right] = c_2\,\mathrm{f}_2(r)\Theta\,\hat{\boldsymbol{r}} - c_3\,\mathrm{f}_1(r)\boldsymbol{\nabla}\left(\frac{P}{\mathrm{f}_1(r)}\right) \\ + c_4\,(\boldsymbol{\nabla}\times\boldsymbol{B})\times\boldsymbol{B} + c_5\boldsymbol{\nabla}\cdot\boldsymbol{\mathcal{D}},\end{aligned} \tag{1.5}$$

where the stress tensor $\mathcal{D}$ is given by

$$\mathcal{D}_{ij} = 2\mathrm{f}_1(r)\,\mathrm{f}_3(r)\left[e_{ij} - \frac{1}{3}\left(\boldsymbol{\nabla}\cdot\boldsymbol{v}\right)\delta_{ij}\right]. \tag{1.6}$$

Here e_{ij} and δ_{ij} refer to the standard rate-of-strain tensor and Kronecker delta, respectively.

The velocity field is denoted by $\boldsymbol{v}$, the thermal anomoly by Θ, the pressure by P, and the magnetic field by $\boldsymbol{B}$. All four of these quantities (eight, if you count the three components each for $\boldsymbol{v}$ and $\boldsymbol{B}$) are 3-dimensional functions of position, in contrast to the 1-dimensional functions of radius $\mathrm{f}_i(r)$. The velocity and magnetic fields are subject to the constraints

$$\boldsymbol{\nabla}\cdot\left[\mathrm{f}_1(r)\,\boldsymbol{v}\right] = 0 \tag{1.7}$$

and

$$\boldsymbol{\nabla}\cdot\boldsymbol{B} = 0, \tag{1.8}$$

respectively. The evolution of Θ is described by

$$\begin{aligned}\mathrm{f}_1(r)\,\mathrm{f}_4(r)\left[\frac{\partial\Theta}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\Theta + c_{11}\,\mathrm{f}_{14}(r)v_r\right] = \; & c_6\,\boldsymbol{\nabla}\cdot\left[\mathrm{f}_1(r)\,\mathrm{f}_4(r)\,\mathrm{f}_5(r)\,\boldsymbol{\nabla}\Theta\right] \\ & + c_{10}\,\mathrm{f}_6(r) + c_8\,\Phi(r,\theta,\phi) + c_9\,\mathrm{f}_7(r)|\boldsymbol{\nabla}\times\boldsymbol{B}|^2,\end{aligned} \tag{1.9}$$

where the viscous heating Φ is given by

$$\begin{aligned}\Phi(r,\theta,\phi) = c_5\,\mathcal{D}_{ij}e_{ij} &= 2\,c_5\,\mathrm{f}_1(r)\mathrm{f}_3(r)\left[e_{ij}e_{ij} - \frac{1}{3}\left(\boldsymbol{\nabla}\cdot\boldsymbol{v}\right)^2\right] \\ &= 2\,c_5\,\mathrm{f}_1(r)\mathrm{f}_3(r)\left[e_{ij} - \frac{1}{3}\left(\boldsymbol{\nabla}\cdot\boldsymbol{v}\right)\delta_{ij}\right]^2.\end{aligned} \tag{1.10}$$

Finally, the evolution of $\boldsymbol{B}$ is described by the induction equation

$$\frac{\partial\boldsymbol{B}}{\partial t} = \boldsymbol{\nabla}\times\left[\boldsymbol{v}\times\boldsymbol{B} - c_7\,\mathrm{f}_7(r)\boldsymbol{\nabla}\times\boldsymbol{B}\right]. \tag{1.11}$$

Note that when Rayleigh actually solves the equations, the following additional derivative functions are used:

$$\begin{aligned}
\mathrm{f}_8(r) &= \frac{d \ln \mathrm{f}_1}{dr} \\
\mathrm{f}_9(r) &= \frac{d^2 \ln \mathrm{f}_1}{dr^2} \\
\mathrm{f}_{10}(r) &= \frac{d \ln \mathrm{f}_4}{dr} \\
\mathrm{f}_{11}(r) &= \frac{d \ln \mathrm{f}_3}{dr} \\
\mathrm{f}_{12}(r) &= \frac{d \ln \mathrm{f}_5}{dr} \\
\mathrm{f}_{13}(r) &= \frac{d \ln \mathrm{f}_7}{dr}.
\end{aligned}$$

When supplying a custom reference state, the user may specify the six derivative functions "by hand." If the user fails to do so, Rayleigh will compute the required derivatives (only if the user supplies the function whose derivative is to be taken) from the function's Chebyshev coefficients.

Note that equations (1.5)-(1.11) could have been formulated in other ways. For instance, we could combine f_1 and f_3 into a single function in Equation (1.10). The form of the equations presented here has been chosen to reflect that actually used in the code, which was originally written dimensionally.

We now describe the nondimensional Boussinesq, and dimensional/nondimensional anelastic formulations used in the code.

Nondimensional Boussinesq Formulation of the MHD Equations

Rayleigh can be run using a nondimensional, Boussinesq formulation of the MHD equations (**reference_type=1**). The nondimensionalization employed is as follows:

$$\begin{aligned}
\text{Length} &\rightarrow L && \text{(Shell Depth)} \\
\text{Time} &\rightarrow \frac{L^2}{\nu_o} && \text{(Viscous Timescale)} \\
\text{Temperature} &\rightarrow \Delta T && \text{(Temperature Contrast Across Shell)} \\
\text{Magnetic Field} &\rightarrow \sqrt{\hat{\rho}\mu\eta_o\Omega_0} && \\
\text{Reduced Pressure} &\rightarrow \nu_o\Omega_0 && \text{([Thermodynamic Pressure]}/\hat{\rho}),
\end{aligned}$$

where Ω_0 is the rotation rate of the frame, $\hat{\rho}$ is the (constant) density of the fluid, η_o is the magnetic diffusivity at the top of the domain (i.e., at $r = r_o$), ν_o is the kinematic viscosity at the top of the domain, and μ is the magnetic permeability. Note that in Gaussian units for vacuum, $\mu = 4\pi$. After nondimensionalizing, the following nondimensional numbers appear in our equations:

$$\begin{aligned}
Pr &= \frac{\nu_o}{\kappa_o} && \text{Prandtl Number} \\
Pm &= \frac{\nu_o}{\eta_o} && \text{Magnetic Prandtl Number} \\
E &= \frac{\nu_o}{\Omega_0 L^2} && \text{Ekman Number} \\
Ra &= \frac{\alpha g_o \Delta T L^3}{\nu_o \kappa_o} && \text{Rayleigh Number,}
\end{aligned}$$

where α is coefficient of thermal expansion, g_o is the gravitational acceleration at the top of the domain, and κ is the thermal diffusivity. Adopting this nondimensionalization is equivalent to assigning the following to the functions $\mathrm{f}_i(r)$ and the constants c_i:

$$
\begin{aligned}
\mathrm{f}_1(r) &\to 1 & c_1 &\to \frac{2}{E} \\
\mathrm{f}_2(r) &\to \left(\frac{r}{r_o}\right)^n & c_2 &\to \frac{Ra}{Pr} \\
\mathrm{f}_3(r) &\to \tilde{\nu}(r) & c_3 &\to \frac{1}{E} \\
\mathrm{f}_4(r) &\to 1 & c_4 &\to \frac{1}{E\,Pm} \\
\mathrm{f}_5(r) &\to \tilde{\kappa}(r) & c_5 &\to 1 \\
\mathrm{f}_6(r) &\to 0 & c_6 &\to \frac{1}{Pr} \\
\mathrm{f}_7(r) &\to \tilde{\eta}(r) & c_7 &\to \frac{1}{Pm} \\
&\vdots & c_8 &\to 0 \\
&\vdots & c_9 &\to 0 \\
&\vdots & c_{10} &\to 0 \\
\mathrm{f}_{14}(r) &\to 0 & c_{11} &\to 0.
\end{aligned}
$$

Here the tildes denote nondimensional radial profiles, e.g., $\tilde{\nu}(r) = \nu(r)/\nu_o$.

Our choice of $\mathrm{f}_{14}(r) \to 0$ sets the default atmosphere in non-dimensional Boussinesq to be neutrally stable. For other choices (i.e., convectively stable or unstable), one must use the custom-reference-state framework.

Our choice of $\mathrm{f}_2(r)$ allows gravity to vary with radius based on the value of the exponent n, which has a default value of 0 in the code. Note also that our definition of Ra assumes fixed-temperature boundary conditions. We might specify fixed-flux boundary conditions and/or an internal heating through a suitable choice $c_{10}\mathrm{f}_6(r)$, in which case the meaning of Ra in our equation set changes, with Ra denoting a flux Rayleigh number instead. In addition, ohmic and viscous heating, which do not appear in the Boussinesq formulation, are turned off when this nondimensionalization is specified at runtime. When these substitutions are made, Equations (1.5)-(1.11) transform as follows.

$$
\begin{aligned}
\left[\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v} + \frac{2}{E}\hat{\boldsymbol{z}}\times\boldsymbol{v}\right] &= \frac{Ra}{Pr}\left(\frac{r}{r_o}\right)^n \Theta\,\hat{\boldsymbol{r}} - \frac{1}{E}\boldsymbol{\nabla}P + \frac{1}{E\,Pm}\left(\boldsymbol{\nabla}\times\boldsymbol{B}\right)\times\boldsymbol{B} + \boldsymbol{\nabla}\cdot\boldsymbol{\mathcal{D}} \\
\left[\frac{\partial \Theta}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla}\Theta\right] &= \frac{1}{Pr}\boldsymbol{\nabla}\cdot\left[\tilde{\kappa}(r)\boldsymbol{\nabla}\Theta\right] \\
\frac{\partial \boldsymbol{B}}{\partial t} &= \boldsymbol{\nabla}\times\left[\boldsymbol{v}\times\boldsymbol{B} - \frac{1}{Pm}\tilde{\eta}(r)\boldsymbol{\nabla}\times\boldsymbol{B}\right] \\
\mathcal{D}_{ij} &= 2\tilde{\nu}(r)e_{ij} \\
\boldsymbol{\nabla}\cdot\boldsymbol{v} &= 0 \\
\boldsymbol{\nabla}\cdot\boldsymbol{B} &= 0
\end{aligned}
$$

Here Θ refers to the temperature (perturbation from the background) and P to the reduced pressure (ratio of the thermodynamic pressure to the constant density).

Dimensional Anelastic Formulation of the MHD Equations

When run in dimensional, anelastic mode (cgs units; **reference_type=2**), the following values are assigned to the functions f_i and the constants c_i:

$$\begin{aligned}
\mathrm{f}_1(r) &\rightarrow \hat{\rho}(r) & c_1 &\rightarrow 2\Omega_0 \\
\mathrm{f}_2(r) &\rightarrow \frac{\hat{\rho}(r)}{c_P} g(r) & c_2 &\rightarrow 1 \\
\mathrm{f}_3(r) &\rightarrow \nu(r) & c_3 &\rightarrow 1 \\
\mathrm{f}_4(r) &\rightarrow \hat{T}(r) & c_4 &\rightarrow \frac{1}{4\pi} \\
\mathrm{f}_5(r) &\rightarrow \kappa(r) & c_5 &\rightarrow 1 \\
\mathrm{f}_6(r) &\rightarrow \frac{Q(r)}{L_*} & c_6 &\rightarrow 1 \\
\mathrm{f}_7(r) &\rightarrow \eta(r) & c_7 &\rightarrow 1 \\
&\vdots & c_8 &\rightarrow 1 \\
&\vdots & c_9 &\rightarrow \frac{1}{4\pi} \\
&\vdots & c_{10} &\rightarrow L_* \\
\mathrm{f}_{14}(r) &\rightarrow \frac{d\hat{S}}{dr} & c_{11} &\rightarrow 1.
\end{aligned}$$

Here $\hat{\rho}(r)$, $\hat{T}(r)$, and $d\hat{S}/dr$ are the spherically symmetric, time-independent reference-state density, temperature, and entropy gradient, respectively. The thermal variables satisfy the linearized equation of state

$$\frac{P}{\hat{P}} = \frac{T}{\hat{T}} + \frac{\rho}{\hat{\rho}}$$

$g(r)$ is the gravitational acceleration, c_P is the specific heat at constant pressure, and Ω_0 is the frame rotation rate. The viscous, thermal, and magnetic diffusivities (also assumed to be spherically symmetric and time-independent) are given by $\nu(r)$, $\kappa(r)$, and $\eta(r)$, respectively. Note that the entropy gradient term $f_{14}(r)v_r$ is only used in Equation (1.9) if **advect_reference_state=.true.**. Finally, $Q(r)$ is an internal heating function; it might represent radiative heating or heating due to nuclear fusion, for instance. In our convention, the volume integral of $\mathrm{f}_6(r)$ equals unity, and c_{10} equals the **luminosity** or **heating_integral** L_* specified in the main_input file. When using a custom reference state, this allows easy adjustment of the luminosity using the **override_constants** formalism, e.g.,

override_constants(10) = T

ra_constants(10) = 3.846d33

specified in the in the **reference_namelist**.

Note that in the anelastic formulation, the thermal variable Θ is interpreted as the entropy perturbation, rather than the temperature perturbation. When these substitutions are made, Equations (1.5)-(1.11) transform as

follows.

$$\begin{aligned}
\hat{\rho}(r)\left[\frac{\partial \boldsymbol{v}}{\partial t}+\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}+2\Omega_0\hat{\boldsymbol{z}}\times\boldsymbol{v}\right] &= \frac{\hat{\rho}(r)}{c_P}g(r)\Theta\,\hat{\boldsymbol{r}}+\hat{\rho}(r)\boldsymbol{\nabla}\left(\frac{P}{\hat{\rho}(r)}\right)\\
&\quad+\frac{1}{4\pi}\left(\boldsymbol{\nabla}\times\boldsymbol{B}\right)\times\boldsymbol{B}+\boldsymbol{\nabla}\cdot\boldsymbol{\mathcal{D}}\\
\hat{\rho}(r)\,\hat{T}(r)\left[\frac{\partial\Theta}{\partial t}+\boldsymbol{v}\cdot\boldsymbol{\nabla}\Theta+v_r\frac{d\hat{S}}{dr}\right] &= \boldsymbol{\nabla}\cdot\left[\hat{\rho}(r)\,\hat{T}(r)\,\kappa(r)\,\boldsymbol{\nabla}\Theta\right]+Q(r)\\
&\quad+\Phi(r,\theta,\phi)+\frac{\eta(r)}{4\pi}\left[\boldsymbol{\nabla}\times\boldsymbol{B}\right]^2\\
\frac{\partial\boldsymbol{B}}{\partial t} &= \boldsymbol{\nabla}\times\left[\,\boldsymbol{v}\times\boldsymbol{B}-\eta(r)\boldsymbol{\nabla}\times\boldsymbol{B}\,\right]\\
\mathcal{D}_{ij} &= 2\hat{\rho}(r)\,\nu(r)\left[e_{ij}-\frac{1}{3}\left(\boldsymbol{\nabla}\cdot\boldsymbol{v}\right)\delta_{ij}\right]\\
\Phi(r,\theta,\phi) &= 2\,\hat{\rho}(r)\nu(r)\left[e_{ij}e_{ij}-\frac{1}{3}\left(\boldsymbol{\nabla}\cdot\boldsymbol{v}\right)^2\right]\\
\boldsymbol{\nabla}\cdot\left[\hat{\rho}(r)\,\boldsymbol{v}\right] &= 0\\
\boldsymbol{\nabla}\cdot\boldsymbol{B} &= 0.
\end{aligned}$$

Nondimensional Anelastic MHD Equations

To run in nondimensional anelastic mode, you must set **reference_type=3** in the Reference_Namelist. The reference state is assumed to be polytropic with a $\frac{1}{r^2}$ profile for gravity. When this mode is active, the following nondimensionalization is used (following Heimpel et al., 2016, Nat. Geo., 9, 19):

$$\begin{aligned}
\text{Length} &\rightarrow L\equiv r_o-r_i && \text{(Shell Depth)}\\
\text{Time} &\rightarrow \frac{1}{\Omega_0} && \text{(Rotational Timescale)}\\
\text{Temperature} &\rightarrow T_o\equiv\hat{T}(r_o) && \text{(Reference Temperature at Upper Boundary)}\\
\text{Density} &\rightarrow \rho_o\equiv\hat{\rho}(r_o) && \text{(Reference Density at Upper Boundary)}\\
\text{Entropy} &\rightarrow \Delta s && \text{(Entropy Constrast Across Shell)}\\
\text{Magnetic Field} &\rightarrow \sqrt{\hat{\rho}_o\mu\eta_o\Omega_0}\\
\text{Pressure} &\rightarrow \rho_o L^2\Omega_0^2.
\end{aligned}$$

When run in this mode, Rayleigh employs a polytropic background state, with an assumed $\frac{1}{r^2}$ variation in gravity. These choices result in the functions f_i and the constants c_i (tildes indicate nondimensional reference-

state variables):

$$
\begin{aligned}
&\mathrm{f}_1(r) \rightarrow \tilde{\rho}(r) && c_1 \rightarrow 2 \\
&\mathrm{f}_2(r) \rightarrow \tilde{\rho}(r)\frac{r_{\max}^2}{r^2} && c_2 \rightarrow \mathrm{Ra}^* \\
&\mathrm{f}_3(r) \rightarrow \tilde{\nu}(r) && c_3 \rightarrow 1 \\
&\mathrm{f}_4(r) \rightarrow \tilde{T}(r) && c_4 \rightarrow \frac{\mathrm{E}}{\mathrm{Pm}} \\
&\mathrm{f}_5(r) \rightarrow \tilde{\kappa}(r) && c_5 \rightarrow \mathrm{E} \\
&\mathrm{f}_6(r) \rightarrow \frac{\tilde{Q}(r)}{L_*}; && c_6 \rightarrow \frac{\mathrm{E}}{\mathrm{Pr}} \\
&\mathrm{f}_7(r) \rightarrow \tilde{\eta}(r) && c_7 \rightarrow \frac{\mathrm{E}}{\mathrm{Pm}} \\
&\quad\vdots && c_8 \rightarrow \frac{\mathrm{E\,Di}}{\mathrm{Ra}^*} \\
&\quad\vdots && c_9 \rightarrow \frac{\mathrm{E}^2\,\mathrm{Di}}{\mathrm{Pm}^2\mathrm{Ra}^*} \\
&\quad\vdots && c_{10} \rightarrow L_* \\
&\mathrm{f}_{14}(r) \rightarrow 0 && c_{11} \rightarrow 0.
\end{aligned}
$$

As in the Boussinesq case, the nondimensional diffusivities are defined according to, e.g., $\tilde{\nu}(r) \equiv \nu(r)/\nu_o$. The nondimensional heating $\tilde{Q}(r)$ is defined such that its volume integral equals the nondimensional **luminosity** or **heating_integral** set in the *main_input* file. As in the dimensional anelastic case, the volume integral of $\mathrm{f}_6(r)$ equals unity, and $\mathrm{c}_{10} = L_*$. The unit for luminosity in this nondimensionalization (to get a dimensional luminosity from the nondimensional L_*) is $\rho_o L^3 T_o \Delta s \Omega_0$.

Two new nondimensional numbers appear in our equations, in addition to those defined for the Boussinesq case. Di, the dissipation number, is defined by

$$\mathrm{Di} = \frac{g_o\,\mathrm{L}}{c_\mathrm{P}\,T_o}, \tag{1.12}$$

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 (nondimensional) entropy. The symbol Ra^* is the modified Rayleigh number, given by

$$\mathrm{Ra}^* = \frac{g_o}{c_\mathrm{P}\Omega_0^2}\frac{\Delta s}{L} \tag{1.13}$$

We thus arrive at the following nondimensionalized equations:

$$\begin{aligned}
\tilde{\rho}(r)\left[\frac{\partial \boldsymbol{v}}{\partial t}+\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{v}+2\hat{\boldsymbol{z}}\times\boldsymbol{v}\right] &= \mathrm{Ra}^{*}\tilde{\rho}(r)\left(\frac{r_{\mathrm{max}}^2}{r^2}\right)\Theta\,\hat{\boldsymbol{r}}+\tilde{\rho}(r)\boldsymbol{\nabla}\left(\frac{P}{\tilde{\rho}(r)}\right)\\
&\quad+\frac{\mathrm{E}}{\mathrm{Pm}}\left(\boldsymbol{\nabla}\times\boldsymbol{B}\right)\times\boldsymbol{B}+\mathrm{E}\boldsymbol{\nabla}\cdot\boldsymbol{\mathcal{D}}\\
\tilde{\rho}(r)\,\tilde{T}(r)\left[\frac{\partial\Theta}{\partial t}+\boldsymbol{v}\cdot\boldsymbol{\nabla}\Theta\right] &= \frac{\mathrm{E}}{\mathrm{Pr}}\boldsymbol{\nabla}\cdot\left[\tilde{\kappa}(r)\tilde{\rho}(r)\,\tilde{T}(r)\,\boldsymbol{\nabla}\Theta\right]+\tilde{Q}(r)\\
&\quad+\frac{\mathrm{E}\,\mathrm{Di}}{\mathrm{Ra}^{*}}\Phi(r,\theta,\phi)+\frac{\mathrm{Di}\,\mathrm{E}^2}{\mathrm{Pm}^2\mathrm{Ra}^{*}}\tilde{\eta}(r)|\boldsymbol{\nabla}\times\boldsymbol{B}|^2\\
\frac{\partial\boldsymbol{B}}{\partial t} &= \boldsymbol{\nabla}\times\left[\boldsymbol{v}\times\boldsymbol{B}-\frac{\mathrm{E}}{\mathrm{Pm}}\tilde{\eta}(r)\boldsymbol{\nabla}\times\boldsymbol{B}\right]\\
\mathcal{D}_{ij} &= 2\tilde{\rho}(r)\tilde{\nu}(r)\left[e_{ij}-\frac{1}{3}\boldsymbol{\nabla}\cdot\boldsymbol{v}\right]\\
\Phi(r,\theta,\phi) &= 2\tilde{\rho}(r)\tilde{\nu}(r)\left[e_{ij}e_{ij}-\frac{1}{3}\left(\boldsymbol{\nabla}\cdot\boldsymbol{v}\right)^2\right]\\
\boldsymbol{\nabla}\cdot\left[\tilde{\rho}(r)\boldsymbol{v}\right] &= 0\\
\boldsymbol{\nabla}\cdot\boldsymbol{B} &= 0.
\end{aligned}$$

1.2.3 The Streamfunction Formulation

The velocity field in Rayleigh is evolved subject to the solenoidal constraint

$$\boldsymbol{\nabla}\cdot\left[\mathrm{f}_1(r)\,\boldsymbol{v}\right]=0. \tag{1.14}$$

This is accomplished by casting $\mathrm{f}_1\boldsymbol{v}$ in terms of streamfunctions such that

$$\mathrm{f}_1\,\boldsymbol{v}=\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\left(W\,\hat{\boldsymbol{r}}\right)+\boldsymbol{\nabla}\times\left(Z\,\hat{\boldsymbol{r}}\right), \tag{1.15}$$

where W and Z are referred to as the poloidal and toroidal stream functions respectively. Rather than evolving the three components of $\boldsymbol{v}$ directly, the momentum equations are cast in terms of these variables before advancing the timestep. The velocity components are related to the streamfunctions via the relations:

$$\mathrm{f}_1 v_r=-\frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial W}{\partial\theta}\right)-\frac{1}{r^2\sin^2\theta}\frac{\partial^2 W}{\partial\phi^2}, \tag{1.16}$$

$$\mathrm{f}_1\,v_\theta=\frac{1}{r}\frac{\partial^2 W}{\partial r\partial\theta}+\frac{1}{r\sin\theta}\frac{\partial Z}{\partial\phi}, \tag{1.17}$$

and

$$\mathrm{f}_1 v_\phi=\frac{1}{r\sin\theta}\frac{\partial^2 W}{\partial r\partial\phi}-\frac{1}{r}\frac{\partial Z}{\partial\theta}. \tag{1.18}$$

When the velocity field and streamfunctions are projected onto a spherical harmonic basis, two additional useful relations are given by

$$\left[\mathrm{f}_1\,v_r\right]_\ell^m=\frac{\ell(\ell+1)}{r^2}W_\ell^m \tag{1.19}$$

and

$$[\{\boldsymbol{\nabla} \times (\mathrm{f}_1 \boldsymbol{v})\}_r]_\ell^m = \frac{\ell(\ell+1)}{r^2} Z_\ell^m. \tag{1.20}$$

A similar decomposition is performed on the magnetic field to ensure it remains divergence free. In that case, the magnetic field is projected onto flux functions such that

$$\boldsymbol{B} = \boldsymbol{\nabla} \times \boldsymbol{\nabla} \times (C\,\hat{\boldsymbol{r}}) + \boldsymbol{\nabla} \times (A\,\hat{\boldsymbol{r}})\,, \tag{1.21}$$

where C and A are the poloidal and toroidal flux functions respectively. Similar to the velocity field, the components of $\boldsymbol{B}$ satisfy

$$B_r = -\frac{1}{r^2 \sin\theta} \frac{\partial}{\partial \theta} \left(\sin\theta \frac{\partial C}{\partial \theta} \right) - \frac{1}{r^2 \sin^2\theta} \frac{\partial^2 C}{\partial \phi^2}, \tag{1.22}$$

$$B_\theta = \frac{1}{r} \frac{\partial^2 C}{\partial r \partial \theta} + \frac{1}{r \sin\theta} \frac{\partial A}{\partial \phi}, \tag{1.23}$$

$$B_\phi = \frac{1}{r \sin\theta} \frac{\partial^2 C}{\partial r \partial \phi} - \frac{1}{r} \frac{\partial A}{\partial \theta}, \tag{1.24}$$

$$[B_r]_\ell^m = \frac{\ell(\ell+1)}{r^2} C_\ell^m, \tag{1.25}$$

and

$$[\{\boldsymbol{\nabla} \times \boldsymbol{B}\}_r]_\ell^m = \frac{\ell(\ell+1)}{r^2} A_\ell^m. \tag{1.26}$$

1.2.4 The Pseudospectral Approach

Section needed.

1.2.5 Parallelization and Performance

Section needed.

1.3 Setting Up a Model

This section details the basics of running a custom model in Rayleigh. For a commplete list of all Rayleigh input parameters, see *Input parameters*.

1.3.1 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, GNU, AOCC, or Cray 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/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 §diagnostics).

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)
```

1.3.2 Grid Setup

By default, Rayleigh employs a single-domain Chebyshev decomposition in radius and a spherical-harmonic decomposition in the $\theta - \phi$ directions. Additionally, multiple Chebyshev domains or a finite-difference scheme may be alternatively employed in radius. We focus on the default mode first.

Standard Grid Specification

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 number of spherical harmonic degrees N_ℓ or the maximal spherical harmonic degree $\ell_{\rm max} \equiv N_\ell - 1$ may be specified in lieu of N_θ. The example above may equivalently be written as

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

or

```
&problemsize_namelist
 n_r = 48
 n_l = 64
/
```

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 §*Underlying Physics*

Using Multiple Chebyshev Domains in Radius

It is possible to run Rayleigh with multiple, stacked domains in the radial direction. Each of these is discretized using their own set of Chebyshev polynomials. The boundaries and number of polynomials can be set for each domain indiviadually, which makes it possible to control the radial resolution at different radii.

To use this feature the problem size has to be specified using `domain_bounds` and `ncheby` instead of `rmin`, `rmax`, and `n_r`. `ncheby` takes a comma-separated list of the number of radial points to use in each domain. `domain_bounds` takes a comma-separated list of the radii of the domain boundaries, starting with the smallest radius. It has one element more than the number of domains. This is an example of two radial domains, one covering the radii 1 to 2 with 16 radial points, the other the radii 2 to 4 with 64 radial points.

```
&problemsize_namelist
 domain_bounds = 1.0, 2.0, 4.0
 ncheby = 16, 64
/
```

Radial values in the diagnostic output will be repeated at the inner domain boundaries. Most quantities are forced to be continuous at these points.

Employing a Finite-Difference Approach in Radius

Rayleigh's default behavior is to employ a Chebyshev collocation scheme in radius. If desired, a finite-difference method can be applied instead. This mode is activated by setting the value of `chebyshev` to .false. in the `numerical_controls_namelist`. At present, Rayleigh's finite-difference scheme employs a five-point stencil with 4th-order accuracy in the interior points. Boundary derivatives are taken with second-order accuracy. By default, a uniform radial grid is assumed. Consider the following example:

```
&problemsize_namelist
 rmin = 1.0
 rmax = 2.0
 n_r = 4
 dr_weights = 0.1,0.3,0.2
 nr_count = 2,4,2
/
&numerical_controls_namelist
 chebyshev=.false.
/
```

This results in the uniform grid:

```
radius = 1.000 , 1.333 , 1.667 , 2.000
    dr = 0.333 , 0.333 , 0.333
```

An example input file using a uniform radial grid and a finite-difference scheme is provided in `input_examples\main_input_mhd_jones_FD`. If desired, a nonuniform grid can also be generated. There are two ways to do this: via *main_input* and via a grid-description file.

Using Main_Input to Specify a Nonuniform Grid

The first method of specifying a nonuniform grid is to join together a series of uniformly-gridded subdomains with different grid spacings. This is accomplished using the `dr_weight` and `nr_count` parameters. `nr_count` indicates the number of gridpoints within each subregion, and `dr_weight` indicates the relative size of the grid spacing within each region. Consider the following example:

```
&problemsize_namelist
 rmin = 1.0
 rmax = 2.0
 n_r = 4
 dr_weights = 0.1,0.3,0.2
 nr_count = 2,4,2
/
```

This example defines a nonuniform grid ranging from 1.0 to 2.0 with 8 gridpoints (Rayleigh will reset the value of `n_r` to be the total of nr_count). The grid spacing within the first 2-point region will be 1/3 of that in the second, 4-point region. Similarly, the grid-spacing in the third, 2-point region will be 2/3 that of the second region and twice that of the first region. The resulting radial grid and spacing is:

```
radius = 1.000 , 1.059 , 1.235 , 1.412 , 1.588 , 1.765 , 1.882 , 2.
    dr = 0.059 , 0.176 , 0.176 , 0.176 , 0.176 , 0.118 , 0.118
```

Note that for n_r points, there are n_r-1 spaces between gridpoints. Rayleigh's convention is to apply dr_weights(1) nr_count(1)-1 times. As a result, specifying a symmetric `nr_count` will lead to assymetry in the grid spacing. We can adjust this by adding one to nr_count(1) and subtracting one from nr_count(2) so that we have:

```
&problemsize_namelist
 rmin = 1.0
 rmax = 2.0
 n_r = 4
 dr_weights = 0.1,0.3,0.2
 nr_count = 3,3,2
/
```

This results in the symmetric grid:

```
radius = 1.000 , 1.067 , 1.133 , 1.333 , 1.533 , 1.733 , 1.867 , 2.000
    dr = 0.067 , 0.067 , 0.200 , 0.200 , 0.200 , 0.133 , 0.133
```

Be sure to leave the `nr_count` and `dr_weights` parameters unset in `main_input` if you wish to use a uniform grid in radius.

Using a Grid-Description File to Specify a Nonuniform Grid

NOTE: The functionality described below is currently incompatible with Rayleigh's ensemble mode.

An arbitrary radial grid may also be generated using Python and then stored to a file that is read when Rayleigh initializes. To do so, import the *reference_tools* module and define a custom grid as illustrated by the code snippet below.

```
import numpy  # Import necessary modules
import reference_tools as rt

ri = 0.5   # Inner radius
ro = 1.5   # Outer radius
nr = 128   # Number of radial points
radius = numpy.linspace(ri,ro,nr)  # The radial grid

my_grid = rt.radial_grid(radius)   # Instantiate the grid object

my_grid.write('grid_layout_128.dat')   # Store contents to file
```

Note that we could have generated the grid in either ascending or descending order. The *write* method accounts for the grid-ordering before storing its contents to the file. Now that we have created a grid-description file ('grid_layout_128.dat' in this example), we indicate the relevant filename in `main_input` using the `radial_grid_file` parameter:

```
&problemsize_namelist
 n_theta = 32
 radial_grid_file = 'grid_layout_128.dat'
/
&numerical_controls_namelist
 chebyshev=.false.
/
```

There are two important points to be aware of:

1. When `radial_grid_file` is specified, all information concerning the grid structure is derived from that file. The values of `rmin`, `rmax`, `N_R` etc. are completely ignored. For this reason, we strongly suggest indicating `N_R` in the grid file's name.
2. In the event that `radial_grid_file`, `nr_count` and `dr_weights` are simultaneously specified, the grid-description file takes precedence.

There is one exception to point 1 above because there may be instances where the same grid structure is useful for problems with different values of *rmin* and *rmax*. If desired, the grid stored in *radial_grid_file* can be rescaled to a new *rmin* and *rmax* by setting the *rescale_radial_grid* keyword to true:

```
&problemsize_namelist
 n_theta = 32
 rmin = 1.0
 rmax = 2.0
```

(continues on next page)

(continued from previous page)

```
 rescale_radial_grid = .true.
 radial_grid_file = 'grid_layout_128.dat'
/
&numerical_controls_namelist
 chebyshev=.false.
/
```

The example above will generate a grid identical to that stored in the grid file, but rescaled to run from r=1 to r=2, rather than r=0.5 to r=1.5 as specified in the original Python code.

1.3.3 Numerical Controls

The Numerical_Controls namelist was added to facilitate fine-control over some aspects of Rayleigh's parallelization and is documented in *Input parameters*. Two numerical_controls parameters worth mentioning that are particularly important for setting up a new model are the `chebyshev` and `bandsolve` keywords.

The value of `chebyshev` is set to `.true.` by default. When set to `.false.`, a finite-difference scheme will be employed in radius rather than a Chebyshev collocation scheme.

The value of the `bandsolve` keyword is also set to `.false.` by default. When set to `.true.`, the otherwise dense matrices used in the implicit timestepping scheme will be recast in banded or block-banded form for the finite-difference and Chebyshev schemes respectively. This can save memory and may offer performance gains. Note that this mode has no effect for models run in Chebyshev mode with only 1 or 2 Chebyshev domains in radius. A minimum of three Chebyshev domains is required before any memory savings is possible.

1.3.4 Physics Controls

Many physical effects can be turned on or off in Rayleigh. The details of what physics you want to include will depend on the type of model you want to run. Be careful, however, that if you are adapting an input file from the benchmark described in *Installation on HPC systems* that you set `benchmark_mode` to 0 or omit it entirely, as this will override other input flags in favor of running the specified benchmark.

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 *table_logicals*, with default values indicated in brackets.

Table. Logicals.

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

Variable [Default value]	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)

1.3.5 Initial Conditions

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 §*Checkpointing* 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:

- 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
- init_type=8 ; user generated temperature or entropy perturbation (see Generic Initial Conditions below)

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 of this random initialization is given by

$$T(r,\theta,\phi) = \sum_{\ell}\sum_{m} c_\ell^m f(r) g(\ell) \mathrm{Y}_\ell^m(\theta,\phi),$$

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

$$f(r) = \frac{1}{2}\left[1 - \cos\left(2\pi \frac{r - rmin}{rmax - rmin}\right)\right].$$

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

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

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.
/
```

Alternatively, you may wish to specify an ell=0 initial thermal profile that is neither random nor conductive. To create your own profile, follow the example found in Rayleigh/examples/custom_thermal_profile/custom_thermal_profile.ipynb. Then, use the following combination of input parameters in main_input:

```
&initial_conditions_namelist
init_type=7
temp_amp = 1.0d-4
custom_thermal_file = 'my_custom_profile.dat'
/
```

This will use the radial profile stored in my_custom_profile.dat for the ell=0 component of entropy/temperature Random values will be used to initialize all other modes.

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
- magnetic_init_type=8 ; user generated magnetic potential fields (see Generic Initial Conditions below)

For the randomized magnetic field, both the poloidal and toroidal vector-potential functions are given a random power distribution described by Equation *eq_init*. 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
/
```

Generic Initial Conditions

The user can input any initial conditions from data files generated by a python routine "rayleigh_spectral_input.py", which can be called as a script or imported as a python class.

The available generic initial conditions options are

```
&initial_conditions_namelist
init_type=8
T_init_file = '<filename>'  !! Temperature
W_init_file = '<filename>'  !! Poloidal velocity potential
Z_init_file = '<filename>'  !! Toroidal velocity potential
```

(continues on next page)

(continued from previous page)

```
P_init_file = '<filename>'  !! `Pressure` potential
magneic_init_type=8
C_init_file = '<filename>'  !! Poloidal magnetic potential
A_init_file = '<filename>'  !! Toroidal magnetic potential

/
```

where *T_init_file* is a user generated initial temperature field and <filename> is the name of the file generated by the python script. If *T_init_file* is not specified the initial field will be zero by default. The same for the other fields. Fields T, W, Z, and P are only initialized from the file if *init_type=8*. Fields C and A are only initialized from file if *magnetic_init_type=8*.

To generate a generic initial condition input file, for example, if a user wanted to specify a single mode in that input file then they could just run the script:

```
rayleigh_spectral_input.py -m 0 0 0 1.+0.j -o example
```

to specify (n,l,m) = (0,0,0) to have a coefficient 1.+0.j and output it to the file example.

This could also be done using the python as a module. In a python shell this would look like:

```
from rayleigh_spectral_input import *
si = SpectralInput()
si.add_mode(1., n=0, l=0, m=0)
si.write('example')
```

For a more complicated example, e.g. the hydrodynamic benchmark from Christensen et al. 2001, the user can specify functions of theta, phi and radius that the python will convert to spectral:

```
rayleigh_spectral_input.py -ar 0.35 -sd 1.0 -nt 96 -nr 64 -o example \
 -e 'import numpy as np; x = 2*radius - rmin - rmax;
 rmax*rmin/radius - rmin + 210*0.1*(1 - 3*x*x + 3*(x**4) -
 x**6)*(np.sin(theta)**4)*np.cos(4*phi)/np.sqrt(17920*np.pi)'
```

in "script" mode.

Alternatively, in "module" mode in a python shell:

```
from rayleigh_spectral_input import *
si = SpectralInput(n_theta=96, n_r=64)
rmin, rmax = radial_extents(aspect_ratio=0.35, shell_depth=1.0)
def func(theta, phi, radius):
   x = 2*radius - rmin - rmax
   return rmax*rmin/radius - rmin + 210*0.1*(1 - 3*x*x + 3*(x**4) - x**6)*(np.
↪sin(theta)**4)*np.cos(4*phi)/np.sqrt(17920*np.pi)
si.transform_from_rtp_function(func, aspect_ratio=0.35, shell_depth=1.0)
si.write('example')
```

The above commands will generate a file called *example* which can be called by

```
&initial_conditions_namelist
init_type=8
T_init_file = 'example'
```

Note that these two examples will have produced different data formats - the first one sparse (listing only the mode specified) and the second one dense (listing all modes).

For more examples including magnetic potentials see *tests/generic_input*.

1.3.6 Custom Reference States

If desired, the constant and nonconstant equation coefficients enumerated here may be completely or partially specified by the user. This allows the user to specify diffusivity profiles, background states, or nondimensionalizations that are not supplied by Rayleigh. Two use cases are supported:

1. One of Rayleigh's predefined reference states may be used, but with some coefficients supplied instead through an auxilliary coefficients file. For each predefined reference type, the coefficients file may be used to override volumetric heating ($c_{10} + f_6$) and the transport coefficients ν, κ, η ($c_5 + f_3,\ c_6 + f_5,\ c_7 + f_7$). For Boussinesq runs, the buoyancy term may also be modified ($c_2 + f_2$).
2. Nonconstant coefficents may be completely specified through the coefficients file. In this mode, activated by setting reference_type=4, the user must fully specify nonconstant coefficients $f_1 - f_7$.

In either case, constant coefficients may be defined within the coefficients file or, read in from main_input, or some combination of the two. Moreover, the radial variation of transport coefficients, as specified by nu_type, kappa_type, and eta_type flags is respected. We elaborate on this behavior below.

Creating a Coefficients File

The first step in modifying Rayleigh's equation coefficients is to generate an equation coefficients file. This file will be used alongside options defined in main_input to determine which combination of coefficients are overridden. In order create your coefficients file, you will need to create an instance of the equation_coefficients class, provided in post_processing/reference_tools.py. Constant and nonconstant coefficients may then be set through set_constant and set_function methods respectively.

The equation_coefficient class is instantiated by passing a radial grid to its init method. This grid can be cast in ascending or descending order, but it should generally possess a much finer mesh than what you plan to use in Rayleigh. Nonconstant coefficients specified in the coefficients file will be interpolated onto the Rayleigh grid at input time.

The file structure created through the class's write method contains a record of those functions and contants that have been set. Rayleigh uses this information at runtime along with main_input to to perform consistency checks and to determine the values ultimately assigned to each constant coefficient.

The sample code below defines a file with sufficient information to alter the viscous, heating, and buoyancy functions of a Rayleigh-provided reference state. This information would be insufficient for use with reference_type=4, but several example notebooks handling that scenario are provided below.

```
import numpy
from reference_tools import equation_coefficients
```

(continues on next page)

(continued from previous page)

```
#Define a name for your equation coefficients file
ofile = 'my_coeffs.dat'

# Define the radial grid.  We suggest using a uniform,
# but finer radial mesh than what you plan for Rayleigh.
# Rayleigh's radial domain bounds should match or fall
# within the domain bounds used for this radial grid.
nr = 2048                     # number of radial points
ri = 0.5                      # Inner radius
ro = 1.0                      # Outer radius [aspect ratio = 0.5]
radius=numpy.linspace(ri,ro,nr, dtype='float64')

#Instantiate an equation_coefficients object
eqc = equation_coefficients(radius)

# Set the buoyancy, heating, and viscosity functions
# These particular choices may be questionable!
buoy = radius
nu   = radius**2
heat = radius**3
eqc.set_function(buoy , 2)  # set function 2
eqc.set_function(nu   , 3)  # set function 3
eqc.set_function(heat , 6)  # set function 6

# Set the corresponding constants
cbuoy = 10.0
cnu   = 20.0
cheat = 30.0
eqc.set_constant(cbuoy , 2)   # set constant 2
eqc.set_constant(cnu   , 5)   # set constant 5
eqc.set_constant(cheat , 10)  # set constant 10

#Generate the coefficients file
eqc.write(ofile)
```

Constant Coefficients: Runtime Control

While constant coefficients may be specified via the coefficients file, many of these coefficients represent simulation "control knobs" that the user may wish to modify at run-time. For instance, the user may want to frequently use a particular profile for viscous diffusion (f_3), but would like to vary its amplitude (c_5) between simulations without generating a new coefficients file. Rayleigh provides the opportunity to override all constant coefficients, or a subset of them, through the main_input file.

Consider the example below.

```
&Reference_Namelist
 ...
 custom_reference_file='mycoeffs.dat'
 override_constants=T
 ra_constants( 2) = 1.0
 ra_constants( 5) = 10.0
 ra_constants(10) = 14.0
 ...
/
```

In this example, values of constant coefficients c_2, c_5, c_{10} will be determined entirely via the main_input file and assigned the values of 1.0, 10.0, and 14.0 respectively. Values specified in mycoeffs.dat will be ignored completely.

This behavior is dictated by the override_constants flag, which instructs Rayleigh to ignore ALL constant coefficients specified in the coefficients files. If a coefficient is not specified in main_input, its value will be set to Rayleigh's internal default value of 0. Consider the following example

```
&Reference_Namelist
 ...
 custom_reference_file='mycoeffs.dat'
 override_constants=T
 ra_constants( 2) = 1.0
 ra_constants(10) = 14.0
 ...
/
```

The resulting values of c_2, c_5, c_{10} will be 1.0, 0.0, and 14.0 respectively. The constant c_5 will not be set to 20.0 (the value specified in the coefficients file).

To specify a subset of constants, use the override_constant flag for each constant you wish to override, as shown below.

```
&Reference_Namelist
 ...
 custom_reference_file='mycoeffs.dat'
 override_constant( 2) = T
 override_constant(10) = T
 ra_constants( 2) = 1.0
 ra_constants(10) = 14.0
 ...
/
```

In this case, the values of constants c_2 and c_{10} will be taken the main_input file. The value of c_5 will be taken from the coefficients file. If a constant's override flag is set, but its value is not specified in main_input, the default value of zero will be used.

Augmenting a Rayleigh-Provided Reference State

When augumenting one of Rayleigh's internal reference-state types, set the with_custom_reference flag (Reference_Namelist) to true in main_input. In addition, assign a list of values to with_custom_constants and with_custom_functions. As an example, to modify the heating and buoyancy profiles using entirely information provided through the equation coefficients file, main_input would contain the following

```
&Reference_Namelist
 ...
 reference_type=1
 custom_reference_file='mycoeffs.dat'
 with_custom_reference=T
 with_custom_constants=2,10
 with_custom_functions=2,6
 ...
/
```

These flags can be used in tandem with the override flags to specify values via main_input. For example, the following input combination would set a value of c_2 of 13.0

```
&Reference_Namelist
 ...
 reference_type=1
 custom_reference_file='mycoeffs.dat'
 with_custom_reference=T
 with_custom_constants=2,10
 with_custom_functions=2,6
 override_constant(2)=T
 ra_constants(2) = 13.0
 ...
/
```

Specifing an Entire Custom Reference State

To specify a full set of custom equation coefficients, set reference_type to 4. Constant coefficients may be overridden, if desired, and as described above. Note that you must fully specify nonconstant coefficients $f_1 - f_7$. If desired, you may also specify their logarithmic derivatives on the fine mesh (see the anelastic notebooks below). This is optional, however, as Rayleigh will compute those funtions if not provided.

```
&Reference_Namelist
 ...
 reference_type = 4
 custom_reference_file='mycoeffs.dat'
 override_constant( 2) = T
 override_constant(10) = T
 ra_constants( 2) = 1.0
 ra_constants(10) = 14.0
```

(continues on next page)

(continued from previous page)

```
 ...
/
```

Transport coefficients may also be specified as desired, but nu_type, kappa_type, and eta_type still behave as described *below*. If you wish to specify a custom diffusivity profile, set the corresponding type to 3. In that case, the corresponding nonconstant coefficient MUST be set in the equation coefficients file. Moreover, if reference_type=4, these corresponding constant must be set in either the coefficients file or in main_input (regardless of the diffusion type specified).

For diffusion types 2 and 3, if the reference_type is not 4, the value of {nu,kappa,eta}_top normally used by that reference_type will be invoked if the corresponding constant coefficient is not set.

Finally, if specifying a custom form for the volumetric heating, please ensure that heating_type is set to a positive, nonzero value in the reference_namelist. Otherwise, reference heating will be deactivated. Any Rayleigh-initialization of the heating function that takes place initially will be overridden by the with_custom_reference or reference_type=4 flags.

Custom Reference State Examples

The notebooks below provide several examples of how to generate a custom-equation-coefficient file. These notebooks are located in the examples/custom_reference_states subdirectory of the main Rayleigh directory. Each notebook has an accompanying main_input file, also located in this directory.

Custom reference state for a non-dimensional Boussinesq MHD setup in a convective spherical shell

Custom reference state for a Boussinesq convective setup. We non-dimensionalize the MHD Boussinesq equations using the rotation period such that $[t] = 1/\Omega_o$ is the timescale, the shell depth $[r] = r_o - r_i = L$ is the lengthscale (where r_o is the outer radius and r_i is the inner radius), $[u] = L\Omega_o$ is the velocity scale, and $[T] = \Delta T$ is the temperature scale*. Assuming that Θ are the temperature perturbations, the non-dimensional Boussinesq equations can be written as:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u} + 2\hat{z}\times\vec{u} = -\frac{\nabla P}{\rho_m} + \mathrm{Ra}^*\left(\frac{r_o}{r}\right)^n \Theta\hat{e}_r + \frac{1}{4\pi}\frac{E}{\mathrm{P_m}}(\nabla\times\vec{B})\times\vec{B} + E\nabla^2\vec{u} \tag{1.27}$$

$$\nabla\cdot\vec{u} = 0 \tag{1.28}$$

$$\frac{\partial \Theta}{\partial t} + \vec{u}\cdot\nabla\Theta = \frac{E}{\mathrm{Pr}}\nabla^2\Theta \tag{1.29}$$

and

$$\frac{\partial \vec{B}}{\partial t} - \nabla\times(\vec{u}\times\vec{B}) = \frac{E}{\mathrm{P_m}}\nabla^2\vec{B}. \tag{1.30}$$

We have assumed that ν, κ, and η are constants, ρ_m is the mean denisty and n is the gravity power (hence e.g. $n = 0$ for constant gravity). We also have the modified Rayleigh number Ra^* given by

$$\mathrm{Ra}^* = \frac{\alpha g_o \Delta T}{L\Omega_o^2} = \frac{\mathrm{Ra}}{\mathrm{Pr}} E^2, \tag{1.31}$$

where $\vec{g}(r) = g_o(r_o/r)^n$, Pr=ν/κ is the Prandtl number, $E = \dfrac{\nu}{L^2\Omega_o}$ is the Ekman number and Ra= $\alpha g_o \Delta T L^3/(\kappa\nu)$. Finally, the magnetic Prandtl number is $\mathrm{P}_m = \nu/\eta$. Then the corresponding fuctions f used here are:

$f_1(r) \to 1$,

$f_2(r) \to (r_o/r)^n$,

$f_3(r) \to 1$,

$f_4(r) \to 1$,

$f_5(r) \to 1$,

$f_6(r) \to 0$,

$f_7(r) \to 1$,

$f_8(r) \to 0$,

$f_9(r) \to 0$,

$f_{10}(r) \to 0$,

$f_{14}(r) \to 0$,

$f_{15}(r) \to 0$,

$f_{16}(r) \to 1$

and the constants c are:

$c_1 \to 2$,

$c_2 \to RaE^2/Pr$,

$c_3 \to 1$,

$c_4 \to E/(4\pi \mathrm{P_m})$,

$c_5 \to E$,

$c_6 \to E/\mathrm{Pr}$,

$c_7 \to E/\mathrm{P_m}$,

$c_8 \to 0$,

$c_9 \to 0$,

$c_{10} \to 0$.

*This is the relevant temperature scale for isothermal BCs – for fixed flux, fixed temperature BCs, the temperature scale should be something like $[T] = LdT_o/dr$.

```
[ ]: ########################
     import numpy
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot, draw, show

import os, sys
sys.path.insert(0, os.path.abspath('../../'))

import post_processing.reference_tools as rt # You will need the refernce_tools.
↪py to run this notebook
```

[ ]:
```
# Grid Parameters
nr    = 500     # Number of radial points
ri    = 0.7e0   # Inner boundary of radial domain
ro    = 1.0e0   # Outer boundary of radial domain

# radial grid
r=numpy.linspace(ri,ro,nr)

#aspect ratio
beta=ri/ro

# shell depth depending on the non-dimensionalization
d=1.e0

#non-dimensional r_i
ri_nd=beta*d/(1-beta)

#non-dimensional r_o
ro_nd=d/(1-beta)

#non-dimensional radial grid
radius1=numpy.linspace(ri_nd,ro_nd,nr)
print(ri_nd,ro_nd)
#print(radius1[0],radius1[nr-1])
```

[ ]:
```
ones = numpy.ones(nr,dtype='float64')
zeros = numpy.zeros(nr,dtype='float64')

# Here we define the reference state i.e. density, temperature,etc.
# For a classic RBC setup, this is the reference state to be used.

density = ones  # density rho
```

(continues on next page)

(continued from previous page)

```
dlnrho= zeros   # dlnrho/dr
d2lnrho= zeros  # d^2lnrho/dr^2
temperature=ones # temperature T
dlnt=zeros       # dlnT/dr
pressure=ones    # pressure P
entropy = zeros  # entropy S, not used in Boussinesq -- set it = 0
gravity=zeros   # gravity -- it is part of the buoyancy term in the non-
↪dimensional momentum equation (see notes above)
hprofile=zeros  # heating function (if we want one)
dsdr=zeros      # dS/dr --  not useful in Boussinesq -- set it = 0
```

[ ]:
```
my_ref = rt.equation_coefficients(radius1)
```

[ ]:
```
## Here we define all the functions and constants that will be written in our␣
↪data file and
## read by Rayleigh if we choose the custom reference state (=4). For more info,␣
↪check notes above!
## Also, chech main_input_Boussinesq to see how to run a simulation with␣
↪Rayleigh and this custom
## reference state ( "Boussinesq.dat" input file generated below!)

unity = numpy.ones(nr,dtype='float64')
gravity_power=0.0
buoy = (radius1[nr-1]/radius1)**gravity_power # buoyancy term calculation
my_ref.set_function(density,1) # denisty rho
my_ref.set_function(buoy,2)    # buoyancy term
my_ref.set_function(unity,3)  # nu(r) -- can be overwritten via nu_type in␣
↪Rayleigh
my_ref.set_function(temperature,4) # temperature T
my_ref.set_function(unity,5)  # kappa(r) -- works like nu
my_ref.set_function(hprofile,6)  # heating function
my_ref.set_function(dlnrho,8)   # dlnrho/dr
my_ref.set_function(d2lnrho,9)  # d^2lnrho/dr^2
my_ref.set_function(dlnt,10)    # dlnT/dr
my_ref.set_function(unity,7)  # eta -- works like nu and kappa
my_ref.set_function(dsdr,14)  # This is not used in Boussinesq -- set it = 0


# The constants can all be set/overridden in the input file
# NOTE that they default to ZERO, but we want
# most of them to be UNITY.  These constants will explicitly depend on the non-
↪dimensionalization chosen.

# The comments corresponding to each one of the constants are generic but  here,␣
↪we also specify
```

(continues on next page)

(continued from previous page)

```
# what they are exactly in our example which is based on the non-
↪dimensionalization used in this notebook!

## This is a  non-magnetic example with Pr=1, E=0.001, and Ra=10^6, such that␣
↪Ra*=1 !
my_ref.set_constant(2.0,1)  # multiplies the Coriolis term, here it is: 2
my_ref.set_constant(1.0,2)  # multiplies the buoyancy, here it is Ra*=Ra.E^2/Pr␣
↪as defined above
my_ref.set_constant(1.0,3)  # multiplies the pressure gradient
my_ref.set_constant(0.0 , 4)  # multiplies the lorentz force, here it is: E/
↪(4*pi*Pm)
my_ref.set_constant(0.001e0,5)  # multiplies the viscosity, here it is: E
my_ref.set_constant(0.001e0,6)  # multiplies the entropy diffusion (kappa), here␣
↪it is: E/Pr
my_ref.set_constant(0.0,7)  # multiplies eta in induction equation, here it is E/
↪Pm
my_ref.set_constant(0.0,8)  # multiplies viscous heating, here it is always 0,␣
↪since we assume the Boussinesq approximation
my_ref.set_constant(1.0,9)  # multiplies ohmic heating, here it is always 0,␣
↪since we assume the Boussinesq approximation
my_ref.set_constant(1.0,10) # multiplies the heating, here it is 0, since we␣
↪have assumed that there is no heating function!
my_ref.write('Boussinesq.dat') # Here we write our data file to be used to run␣
↪our simulation with Rayleigh!
print(my_ref.fset)
print(my_ref.cset)
```

[ ]:

[ ]:

Boussinesq dynamo: nondimensionalized using viscous timescale

[ ]:
```
'''

This code provides an example for using a custom non-dimensionalization
of Rayleigh in Boussinesq dynamo mode.

The non-dimensionalization used here is based on the Rayleigh default
viscous diffusion scaling for convection, thus providing a check
on the custom reference state.

The parameters correspond to a case listed in Table 2 of
```

(continues on next page)

(continued from previous page)

```
Soderlund et al.: "The influence of magnetic fields in planetary
dynamo models", Earth Planet. Sci. Lett., v.333-334, p.9-20 (2012)

The numbers referenced below for the various functions and constants
refer to equation (5) in "Rayleigh_Output_Variables.pdf". Please refer to
that document for further details.

Requirements: (1) "rayleigh_diagnostics.py" ; and (2) "reference_tools.py"

'''

import numpy as np


import os, sys
sys.path.insert(0, os.path.abspath('../../'))

import post_processing.reference_tools as rt

# name of output file containing custom reference data
filename = 'custom_ref_viscous.dat'
```

[ ]:
```
# Non-dimensional input parameters

Ra = 1.12e5             # Rayleigh number
Pr = 1.0                # Prandtl number
Ek = 2.0e-3             # Ekman number
Pm = 5.0                # Magnetic Prandtl number
beta = 0.4              # Aspect ratio = r_inner/r_outer
gravity_power = 1.0     # power law variation of gravitational acceleration
```

[ ]:
```
# Create radial grid

# Numer of radial grid points for radial functions (f_1, f_2, etc.)
# Make large enough for accurate interpolation onto Chebyshev grid
nr = 2000

# non-dimensional r_inner
ri = beta/(1-beta)

# non-dimensional r_outer
ro=1.0/(1-beta)

# non-dimensional radial grid
```

(continues on next page)

(continued from previous page)

```
radius=np.linspace(ri,ro,nr)
```

[ ]:
```
# Define the reference state functions and constants

ones = np.ones(nr,dtype='float64')
zeros = np.zeros(nr,dtype='float64')

# the function list below is default for Boussinesq
f_1 = ones
f_2 = (radius/radius[nr-1])**gravity_power
f_3 = ones
f_4 = ones
f_5 = ones
f_6 = zeros
f_7 = ones
f_8 = zeros
f_9 = zeros
f_10 = zeros
f_11 = zeros
f_12 = zeros
f_13 = zeros

c_1 = 2.0/Ek                # Coriolis force
c_2 = Ra/Pr                 # Buoyancy force
c_3 = 1.0/Ek                # Pressure gradient
c_4 = 1.0/(Ek*Pm)           # Lorentz force
c_5 = 1.0                   # Viscous force
c_6 = 1.0/Pr                # Thermal diffusion
c_7 = 1.0/Pm                # Ohmic diffusion
c_8 = 0.0                   # Viscous heating
c_9 = 0.0                   # Ohmic heating
c_10 = 0.0                  # Internal heating
```

[ ]:
```
# Set all of the functions and constants

my_ref = rt.equation_coefficients(radius)

# Set functions here
my_ref.set_function(f_1, 1)
my_ref.set_function(f_2, 2)
my_ref.set_function(f_3, 3)
my_ref.set_function(f_4, 4)
my_ref.set_function(f_5, 5)
```

(continues on next page)

(continued from previous page)

```
my_ref.set_function(f_6, 6)
my_ref.set_function(f_7, 7)
my_ref.set_function(f_8, 8)
my_ref.set_function(f_9, 9)
my_ref.set_function(f_10, 10)
my_ref.set_function(f_11, 11)
my_ref.set_function(f_12, 12)
my_ref.set_function(f_13, 13)

# Set constants here
my_ref.set_constant(c_1, 1)
my_ref.set_constant(c_2, 2)
my_ref.set_constant(c_3, 3)
my_ref.set_constant(c_4, 4)
my_ref.set_constant(c_5, 5)
my_ref.set_constant(c_6, 6)
my_ref.set_constant(c_7, 7)
my_ref.set_constant(c_8, 8)
my_ref.set_constant(c_9, 9)
my_ref.set_constant(c_10, 10)

my_ref.write(filename)

print('Custom reference file', filename, 'was written successfully.')
```

[ ]:

Custom reference state for the dimensional anelastic MHD formulation in a convective spherical shell based on polytropes

Problem Setup

We define the reference state of the convective region based on a polytrope that has a form given by

$\rho = \rho_0 z^n$,

$P = P_0 z^{n+1}$,

and

$T = T_0 z$,

where ρ, P, and T are the density, pressure, and temperature respectively, and where the polytropic variable z is an as-of-yet undetermined function of radius. We further assume that these quantities are related by the ideal gas law, with

$P = R\rho T$,

where R is the gas constant. It is related to the specific heats at constant pressure (c_p) and contant volume (c_v) through the relation

$R = c_p - c_v = c_p(1 - \frac{1}{\gamma})$,

with

$\gamma \equiv \frac{c_p}{c_v}$.

For a monotomic ideal gas, we have that

$\gamma \equiv \frac{5}{3}$.

Finally, we require that the polytrope satisfies hydrostatic balance. Namely,

$\rho \frac{GM}{r^2} = -\frac{\partial P}{\partial r}$,

where G is the gravitational constant, and M is the mass of the star.

Polytropic Solution

Substituting our relations for P, ρ, and T into the equation of hydrostatic balance, we arive at

$\frac{\partial z}{\partial r} = -\frac{GM}{(n+1)RT_0 r^2} = -\frac{2GM}{5(n+1)c_p T_0 r^2}$.

This motivates us to seek a form for z of

$z = a + \frac{b}{r}$,

and immediately, we see that b must be given by

$b = \frac{2GM}{5(n+1)c_p T_0}$.

Note that while T_0 remains undetermined, we can now compute $\partial T/\partial r$. In order to determine a, we need one more constraint. In our case, we will specify the number of density scaleheights, N_ρ, across the convection zone. We denote the top of the convection zone by a subscript t and the base of the convection zone by a subscript b. We then have $N_\rho = \frac{\rho_b}{\rho_t} = \frac{z_b^n}{z_t^n}$,

or equivalently, using C to denote the exponential factor, and $\beta \equiv \frac{r_b}{r_t}$, we have

$C \equiv e^{\frac{N_\rho}{n}} = \frac{a+b/r_b}{a+b/r_t}$.

Rearranging, we find our expression for a

$a = \frac{fb}{r_b}$,

where

$f \equiv \frac{\beta C - 1}{1 - C}$.

This yields our expression for z in terms of T_0

$z = b(\frac{1}{r} + \frac{f}{r_b}) = \frac{2GM}{5(n+1)c_p T_0}\left(\frac{1}{r} + \frac{f}{r_b}\right)$.

Factors of T_0 cancel out, when calculating the temperature, leaving us with a complete description of its functional form. We are free to choose any value of T_0 as a result; we use T_b, the temperature at the base of the convection zone. We have that

$T_0 = T_b = \frac{2GM}{5(n+1)c_p}\left(\frac{1+f}{r_b}\right)$,

completing our description of z. Values for ρ_0 and P_0 can now similarly be computed by enforcing the value

of ρ and P as a particular point. As with T, we choose the base of the convection zone in the code that follows.

[ ]:
```
######################
import numpy
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot, draw, show

import os, sys
sys.path.insert(0, os.path.abspath('../../'))

import post_processing.reference_tools as rt # You will need the refernce_tools.
↪py to run this notebook
import post_processing.rayleigh_diagnostics as rdiag
```

[ ]:
```
# Grid Parameters
# Here, we use solar values as an example

nr    = 512         # Number of radial points
ri    = 5e10        # Inner boundary of radial domain
ro    = 6.586e10    # Outer boundary of radial domain
rcz   = 5e10        # base of the CZ

#apsect ratio
beta=ri/ro

#Polytrope Parameters
ncz   = 1.5         # polytropic index of convection zone
nrho  = 3.          # number of density scaleheights across convection zone (not␣
↪full domain)
mass  = 1.98891e33  # Mass of the star
G     = 6.67e-8     # gravitational constant
rhoi  = 0.18053428  # density at the base of convection zone
cp    = 3.5e8        # specific heat at constant pressure
gamma = 5.0/3.0     # Ratio of specific heats for ideal gas

lsun = 3.846e33     # solar luminosity for scaling the volumetric heating
```

[ ]:
```
radius = numpy.linspace(ri,ro,nr)           #Radial domain of the CZ
```

[ ]:
```
#Compute a  CZ polytrope (see reference_tools)

poly1 = rt.gen_poly(radius,ncz,nrho,mass,rhoi,G,cp,rcz)
gas_constant = cp*(1.0-1.0/gamma)  # R
```

(continues on next page)

(continued from previous page)

```
temperature= poly1.temperature # temperature T
density = poly1.density  # density rho
pressure = poly1.pressure # pressure P, this won't matter -- set it equal to␣
↪something
dsdr = poly1.entropy_gradient # dS/dr
entropy= poly1.entropy # entropy S, this won't matter -- set it equal to something
dpdr = poly1.pressure_gradient # dP/dr
gravity= mass*G/radius**2  # gravity g
```

[ ]:
```
# Calculation of the first and second derivative of lnrho and the first␣
↪derivative of lnT.
# If we do not calculate those here, then they are calculated within Rayleigh!

d_density_dr = numpy.gradient(density,radius, edge_order=2)
dlnrho = d_density_dr/density
d2lnrho = numpy.gradient(dlnrho,radius, edge_order=2)
dtdr = numpy.gradient(temperature,radius, edge_order =2)
dlnt = dtdr/temperature
```

[ ]:
```
# Plots of  the dimensional reference state profiles, i.e. the density profile,␣
↪the temperature profile, etc.

fig, ax = plt.subplots(nrows =3,ncols = 3, figsize=(15,10) )
ax[0][0].plot(radius,density,'r')
ax[0][0].set_xlabel('Radius')
ax[0][0].set_ylabel('Density')

ax[0][1].plot(radius,entropy)
ax[0][1].set_xlabel('Radius')
ax[0][1].set_ylabel('Entropy')

ax[0][2].plot(radius,temperature)
ax[0][2].set_xlabel('Radius')
ax[0][2].set_ylabel('Temperature')

ax[1][0].plot(radius,dsdr)
ax[1][0].set_xlabel('Radius')
ax[1][0].set_ylabel('Entropy Gradient')

ax[1][1].plot(radius,pressure)
ax[1][1].set_xlabel('Radius')
ax[1][1].set_ylabel('Pressure')

ax[1][2].plot(radius,gravity)
ax[1][2].set_xlabel('Radius')
```

(continues on next page)

(continued from previous page)

```
ax[1][2].set_ylabel('Gravity')

ax[2][0].plot(radius,dlnrho)
ax[2][0].set_xlabel('Radius')
ax[2][0].set_ylabel('dlnrho')

ax[2][1].plot(radius,d2lnrho)
ax[2][1].set_xlabel('Radius')
ax[2][1].set_ylabel('d2lnrho')

ax[2][2].plot(radius,dlnt)
ax[2][2].set_xlabel('Radius')
ax[2][2].set_ylabel('dlnt')

plt.tight_layout()

plt.show()
```

[ ]:
```
# Calculation of the heating profile based on the pressure, if we want to have a␣
↪heating function in our setup.

hprofile = numpy.zeros(nr,dtype='float64')
hprofile[:] = pressure[:]

#####################################################################
# We normalize the heating function so that it integrates to 1.

integrand= numpy.pi*4*radius*radius*hprofile
hint = numpy.trapz(integrand,x=radius)
hprofile = hprofile/hint
#plt.plot(radius,hprofile)
plt.plot(radius, hprofile*lsun)
```

[ ]:
```
my_ref = rt.equation_coefficients(radius)
```

[ ]:
```
# Here we define all the functions and constants that will be written in our␣
↪data file and
# read by Rayleigh if we choose the custom reference state (=4)

unity = numpy.ones(nr,dtype='float64')
buoy =density*gravity/cp # calculation of the buoyancy term used in the momentum␣
↪equation

my_ref.set_function(density,1) # density rho
```

(continues on next page)

(continued from previous page)

```
my_ref.set_function(buoy,2)    # buoyancy term
my_ref.set_function(unity,3)   # nu(r) -- can be overwritten via nu_type in␣
↪Rayleigh
my_ref.set_function(temperature,4) # temperature T
my_ref.set_function(unity,5)  # kappa(r) -- works like nu
my_ref.set_function(hprofile,6)  # heating -- this is normalized to one

my_ref.set_function(dlnrho,8)  # dlnrho/dr
my_ref.set_function(d2lnrho,9) # d^2lnrho/dr^2
my_ref.set_function(dlnt,10)   # dlnT/dr
my_ref.set_function(unity,7)  # eta -- works like nu and kappa --  magnetic␣
↪diffusivity
my_ref.set_function(dsdr,14)   # dS/dr

# The constants can all be set/overridden in the input file
# NOTE that they default to ZERO, but we want
# most of them to be UNITY.
# These aren't very useful in the dimensional anelastic formulation but are VERY␣
↪useful
# for the non-dimensional version of the anelastic custom reference state!

my_ref.set_constant(1.0,1)  # multiplies the Coriolis force--Should be 2 Omega␣
↪(complication here)
my_ref.set_constant(1.0,2)  # multiplies buoyancy
my_ref.set_constant(1.0,3)  # multiplies pressure gradient
my_ref.set_constant(0.0,4)  # multiplies lorentz force
my_ref.set_constant(1.0,5)  # multiplies viscosity
my_ref.set_constant(1.0,6)  # multiplies entropy diffusion (kappa)
my_ref.set_constant(0.0,7)  # multiplies eta in induction equation
my_ref.set_constant(1.0,8)  # multiplies viscous heating
my_ref.set_constant(1.0,9)  # multiplies ohmic heating
my_ref.set_constant(lsun,10) # multiplies the heating (if normalized to 1, this␣
↪should be the luminosity)
my_ref.write('dimensional.dat')  # Here we write our data file to be used to run␣
↪our simulation with Rayleigh!
print(my_ref.fset)
print(my_ref.cset)
```

[ ]:
```
# Once you've run for one time step, set have_run = True !!

# Here we check if the output reference state is the same as the one we used as␣
↪an input (sanity check)

# NOTE: We need the output file "equation_coefficients" to run this, as well as␣
↪the PDE_Coefficients
```

(continues on next page)

(continued from previous page)

```
# from rayleigh_diagnostics.py

try:
    cref = rdiag.PDE_Coefficients() # This will give us the output reference␣
↪state
    have_run = True
except:
    have_run = False

if (have_run):

    fig, ax = plt.subplots(ncols=3,nrows=4, figsize=(16,4*4))
    # density, derivatives of lnrho
    ax[0][0].plot(cref.radius,cref.density,'yo')
    ax[0][0].plot(radius,density)
    ax[0][0].set_xlabel('Radius (cm)')
    ax[0][0].set_title('Density')

    ax[0][1].plot(cref.radius, cref.dlnrho,'yo')
    ax[0][1].plot(radius, dlnrho)
    ax[0][1].set_xlabel('Radius (cm)')
    ax[0][1].set_title('Log density gradient')

    ax[0][2].plot(cref.radius,cref.d2lnrho,'yo')
    ax[0][2].plot(radius,d2lnrho)
    ax[0][2].set_xlabel('Radius (cm)')
    ax[0][2].set_title('d_dr{Log density gradient}')

    # temperature and derivative of lnT
    ax[1][0].plot(cref.radius,cref.temperature,'yo')
    ax[1][0].plot(radius,temperature)
    ax[1][0].set_xlabel('Radius (cm)')
    ax[1][0].set_title('Temperature')

    ax[1][1].plot(cref.radius, cref.dlnT,'yo')
    ax[1][1].plot(radius, dlnt)
    ax[1][1].set_xlabel('Radius (cm)')
    ax[1][1].set_title('Log temperature gradient')

    # entropy gradient
    ax[2][1].plot(cref.radius, cref.dsdr,'yo')
    ax[2][1].plot(radius, dsdr,'b')
    ax[2][1].set_xlabel('Radius (cm)')
    ax[2][1].set_title('Entropy gradient')

    # Note that you must build the buoyancy term from the functions/constants
```

(continues on next page)

(continued from previous page)

```
    ax[3][1].plot(cref.radius, cref.functions[:,1]*cref.constants[1],'yo')
    ax[3][1].plot(radius, gravity*density/cp)
    ax[3][1].set_xlabel('Radius (cm)')
    ax[3][1].set_title('Buoyancy')

    # Note that the output heating (cref.heating) is hprofile/density/
↪temperature*luminosity
    ax[3][2].plot(cref.radius, cref.heating*cref.rho*cref.T,'yo')
    ax[3][2].plot(radius, hprofile*lsun)
    ax[3][2].set_xlabel('Radius (cm)')
    ax[3][2].set_title('Heating')

    plt.tight_layout()
    plt.show()
```

[ ]:

Custom reference state for non-dimensional anelastic MHD formulation in a convective spherical shell

We non-dimensionalize the dimensional anelastic MHD equations (see Rayleigh documentation for the full set of equations, and the definitions of the variables and parameters used here) using the width of the convection zone $[l] = L = r_o - r_{cz}$ as the lengthscale (where r_o is the outer radius and r_{cz} is the radius at the bottom of the convection zone (CZ)), the viscous timescale $[t] = L^2/\nu$, and the velocity scale $[u] = \nu/L$. For the density and temperature scales, we choose $[\rho] = \tilde{\rho}$, and $[T] = \tilde{T}$, respectively, where the tildes correspond to the volume average of the respective quantity such that $\tilde{q} = \frac{1}{V}\int_V q dV$, where V is the volume of the convective region of the spherical shell (and $q \rightarrow \rho, T$). For the magnetic field, we use $[B] = \sqrt{\tilde{\rho}\mu\eta\Omega_o}$ and finally for the entropy S we use a scaling related to the thermal energy flux $\tilde{F}$ such that $[S] = \frac{L\tilde{F}}{\tilde{\rho}\tilde{T}\kappa}$. Then, we can write the non-dimensional anelastic MHD equations as:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u} + \frac{1}{\mathrm{E}}2\hat{z}\times\vec{u} = \frac{\mathrm{Ra}}{\mathrm{Pr}}\frac{g(r)}{\tilde{g}}S\hat{r} - \nabla(p/\bar{\rho}) + \frac{1}{\bar{\rho}}\nabla\cdot\vec{D} + \frac{1}{4\pi\bar{\rho}}\frac{1}{\mathrm{P_m E}}(\nabla\times\vec{B})\times\vec{B}. \tag{1.32}$$

$$\nabla\cdot(\bar{\rho}\vec{u}) = 0, \tag{1.33}$$

$$\nabla\cdot\vec{B} = 0. \tag{1.34}$$

$$\bar{\rho}\bar{T}\left(\frac{dS}{dt} + \vec{u}\cdot\nabla S\right) = \frac{1}{\mathrm{Pr}}\nabla\cdot(\bar{\rho}\bar{T}\nabla S) + \frac{1}{\mathrm{Pr}}Q_{nd} + \frac{\mathrm{DiPr}}{\mathrm{Ra}}\Phi + \frac{1}{4\pi}\frac{\mathrm{DiPr}}{\mathrm{EP_m^2 Ra}}|\nabla\times\vec{B}|^2, \tag{1.35}$$

where $Q_{nd} = \frac{L}{\tilde{F}}Q$, and

$$\frac{\partial \vec{B}}{\partial t} - \nabla \times (\vec{u} \times \vec{B}) = -\nabla \times \left(\frac{1}{\mathrm{P_m}} \nabla \times \vec{B} \right) \left\{ = \frac{1}{\mathrm{P_m}} \nabla^2 \vec{B} \right\}, \tag{1.36}$$

where we have assumed that κ, ν and η are constants. We now have five non-dimensional numbers: the flux Rayleigh number Ra, the Prandtl number Pr, the magnetic Prandtl number $\mathrm{P_m}$, the Ekman number E and the dissipation number Di, which are defined respectively as: $\mathrm{Ra} = \frac{\tilde{g}\tilde{F}L^4}{c_p \tilde{\rho} \tilde{T} \kappa^2 \nu}$, $\mathrm{Pr} = \frac{\nu}{\kappa}$, $\mathrm{P_m} = \frac{\nu}{\eta}$, $\mathrm{E} = \frac{\nu}{\Omega_o L^2}$, and $\mathrm{Di} = \frac{\tilde{g}L}{c_p \tilde{T}}$. Then the functions f are:

$f_1(r) \to \bar{\rho}$,

$f_2(r) \to \bar{\rho} g(r)/\tilde{g}$,

$f_3(r) \to 1$,

$f_4(r) \to \bar{T}$,

$f_5(r) \to 1$,

$f_6(r) \to Q_{nd}(r)$,

$f_7(r) \to 1$,

and the constants are

$c_1 \to 2/\mathrm{E}$,

$c_2 \to \mathrm{Ra}/\mathrm{Pr}$,

$c_3 \to 1$,

$c_4 \to \frac{1}{4\pi} \frac{1}{\mathrm{P_m E}}$,

$c_5 \to 1$,

$c_6 \to 1/\mathrm{Pr}$,

$c_7 \to 1/\mathrm{P_m}$,

$c_8 \to \mathrm{DiPr}/\mathrm{Ra}$,

$c_9 \to \frac{1}{4\pi} \frac{\mathrm{DiPr}}{\mathrm{E P_m^2 Ra}}$, and

$c_{10} \to \frac{1}{\mathrm{Pr}}$.

The non-dimensional reference density and temperature profiles are equal to their dimensional profile divided by its volume average in the CZ, i.e. $\bar{\rho} = \bar{\rho}_{dim}/\tilde{\rho}$, and $\bar{T} = \bar{T}_{dim}/\tilde{T}$.

Note that the reference state is based on a polytrope, similarly to the dimensional case. For more info on that, check the dimensional anelastic notebook version!

[ ]:
```
########################
import numpy
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot, draw, show
```

(continues on next page)

(continued from previous page)

```
import os, sys
sys.path.insert(0, os.path.abspath('../../'))

import post_processing.reference_tools as rt # You will need the refernce_tools.
↪py to run this notebook
import post_processing.rayleigh_diagnostics as rdiag
```

[ ]:
```
# Grid Parameters
nr    = 512         # Number of radial points
ri    = 5e10        # Inner boundary of radial domain
ro    = 6.586e10    # Outer boundary of radial domain
rcz   = 5e10        # Base of the CZ

# apsect ratio
beta=ri/ro

# shell depth according to non-dimensionalization
d=1.e0

## In the main_input file, we should use the following radial boundaries or else␣
↪set the right aspect ratio and shell depth

ri_nd=beta*d/(1-beta) # non-dimensional inner boundary

ro_nd=d/(1-beta) # non-dimensional outer boundary

print(ri_nd,ro_nd)

# Polytrope Parameters
# In this example, we use solar values

ncz   = 1.5         # polytropic index of convection zone
nrho  = 3.          # number of density scaleheights across convection zone (not␣
↪full domain)
mass  = 1.98891e33  # Mass of the star
G     = 6.67e-8     # gravitational constant
rhoi  = 0.18053428  # density at the base of convection zone
cp    = 3.5e8       # specific heat at constant pressure
gamma = 5.0/3.0     # Ratio of specific heats for ideal gas
```

[ ]:
```
# Define the Radial Grid
```

(continues on next page)

(continued from previous page)

```
radius = numpy.linspace(ri,ro,nr)           # Full domain radial grid for␣
↪reference state
```

```
[ ]: #Compute  CZ polytrope -- see reference_tools.py script

poly1 = rt.gen_poly(radius,ncz,nrho,mass,rhoi,G,cp,rcz)
gas_constant = cp*(1.0-1.0/gamma)  # R

#Dimensional profiles for reference state functions
temperature1 = poly1.temperature # T
density1 = poly1.density         # rho
pressure1 = poly1.pressure       # P
dsdr1 = poly1.entropy_gradient   # dS/dr
s1= poly1.entropy                # S
dpdr1 = poly1.pressure_gradient  # dP/dr
gravity1= mass*G/radius**2       # g

# We use the volume average of rho,T and g in the CZ to non-dimensionalize our␣
↪reference state functions
def vol_av(f,radius):
    int1=radius**2.
    int2=f*radius**2.
    vol = numpy.trapz(int1,x=radius)
    f_av1 = numpy.trapz(int2,x=radius)
    f_av=f_av1/vol
    return f_av

temperature_av=vol_av(temperature1,radius) # CZ volume average of T
density_av=vol_av(density1,radius)         # CZ volume average of rho
gravity_av=vol_av(gravity1,radius)         # CZ volume average of g


print(temperature_av,density_av,gravity_av)

# Here we define the non-dimensional T, rho, g, S, r, dS/dr and P

# The non-dimensional temperature: T=T_dimensional/T_av
temperature=temperature1/temperature_av

# The non-dimensional density: rho=rho_dimensional/rho_av
density=density1/density_av

# The non-dimensional gravity: g=g_dimensional/g_av
```

(continues on next page)

(continued from previous page)

```
gravity=gravity1/gravity_av


# Here we define our non-dimensional radius
radius1=radius/(ro-ri) # OR: radius1=numpy.linspace(ri_nd,ro_nd,nr)

#The pressure profile won't matter, we set it equal to rho*T as a reference
pressure=temperature*density

# For a purely convective region, we use dS/dr=0
dsdr=dsdr1

# Entropy won't matter, set it to something-- here I use the non-dimensional S i.
↪e. S_nondim=[S_dim*(L^3*g_av)]/Ra*cp*kappa*nu]
entropy=numpy.ones(len(radius))*s1*(1.586e10**3.*gravity_av)/(13303.
↪43109666924*cp*(8e12)**2)
```

[ ]:
```
# Here we calculate the derivatives of lnrho and lnT based on the non-
↪dimensional radius1, since
# we want their non-dimensional profiles

d_density_dr = numpy.gradient(density,radius1, edge_order=2)
dlnrho = d_density_dr/density
d2lnrho = numpy.gradient(dlnrho,radius1, edge_order=2)
dtdr = numpy.gradient(temperature,radius1, edge_order =2)
dlnt = dtdr/temperature


### Here, we plot our non-dimensional profiles for our reference state functions,
↪ i.e. for rho, T, etc.

fig, ax = plt.subplots(nrows =3,ncols = 3, figsize=(15,10) )

ax[0][0].plot(radius1,density,'r')
ax[0][0].set_xlabel('Radius')
ax[0][0].set_ylabel('Density')

ax[0][1].plot(radius1,entropy)
ax[0][1].set_xlabel('Radius')
ax[0][1].set_ylabel('Entropy')


ax[0][2].plot(radius1,temperature)
ax[0][2].set_xlabel('Radius')
```

(continues on next page)

(continued from previous page)

```
ax[0][2].set_ylabel('Temperature')

ax[1][0].plot(radius1,dsdr)
ax[1][0].set_xlabel('Radius')
ax[1][0].set_ylabel('Entropy Gradient')

ax[1][1].plot(radius1,pressure)
ax[1][1].set_xlabel('Radius')
ax[1][1].set_ylabel('Pressure')

ax[1][2].plot(radius1,gravity)
ax[1][2].set_xlabel('Radius')
ax[1][2].set_ylabel('Gravity')

ax[2][0].plot(radius1,dlnrho)
ax[2][0].set_xlabel('Radius')
ax[2][0].set_ylabel('dlnrho')

ax[2][1].plot(radius1,d2lnrho)
ax[2][1].set_xlabel('Radius')
ax[2][1].set_ylabel('d2lnrho')

ax[2][2].plot(radius1,dlnt)
ax[2][2].set_xlabel('Radius')
ax[2][2].set_ylabel('dlnt')

plt.tight_layout()

plt.show()
print(density[0],temperature[0])
```

[ ]:
```
## This is where we define a heating function, if we want one in our model.

# Units are energy / volume / time such that {rho_hat T_hat dS/dt} = hprofile(r)


hprofile = numpy.zeros(nr,dtype='float64')
hprofile[:] = pressure1[:]

# Next, we need to integrate the heating profile
# We normalize it such that its integral over the volume is 1
# This way, we can set the luminosity via a constant in the input file for the␣
↪dimensional case!

integrand= numpy.pi*4*radius*radius*hprofile
```

(continues on next page)

(continued from previous page)

```
hint = numpy.trapz(integrand,x=radius)
hprofile = hprofile/hint




############################################################
# Plot the integrated luminosity as a function of radius
# (should integrate to 1 at r = r_top)
############################################################


# We then need to calculate the non-dimensional heating function

lq1 = numpy.zeros(nr)
lq1[0]=0
lq2 = numpy.zeros(nr)
lq2[0]=0
lq3 = numpy.zeros(nr)
lq3[0]=0

lsun= 3.846e33 # solar luminosity
integrand1= lsun*radius*radius*hprofile # Luminosity*r^2*heating
for i in range(1,nr):
    lq1[i] =(1/(radius[i]**2.))*numpy.trapz(integrand1[0:i+1],x=radius[0:i+1])

integrand2=4*numpy.pi*radius**2.
lq2= numpy.trapz(integrand2,x=radius)
integrand3=lq1*4*numpy.pi*radius**2.
lq3=numpy.trapz(integrand3,x=radius)/lq2 # That is the volume Flux F_tilde


# The  non-dimensional heating profile is hprofile_nd = lsun*hprofile*L/F_
↪tilde=Q_dim*(r_o-r_cz)/F_tilde
hprofile_nd=lsun*hprofile*(ro-ri)/lq3
#print((ro-ri)/lq3) # This comes out of the non-dimensionalization used (L/F_
↪tilde)

nu=8.e12
kappa=8.e12

Ra=gravity_av*lq3*(ro-ri)**4./(cp*density_av*temperature_av*nu*kappa**2.) # Ra␣
↪is defined earlier in the notes
Di=gravity_av*(ro-ri)/(cp*temperature_av) # Dissipation number is defined in the␣
↪notes above
print(Ra,Di/Ra)
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(ncols=2,figsize=(12,4))

ax[0].plot(radius1,hprofile_nd,'ob')
ax[0].set_title('Non-Dimensional Heating Profile')
ax[0].set_xlabel('Radius')

ax[1].plot(radius,hprofile,'.b')
ax[1].set_xlabel('Radius (cm)')
ax[1].set_title('Dimensional Heating Profile')
plt.show()
```

[ ]:
```
# Have everything in terms of the non-dimensional radius
my_ref = rt.equation_coefficients(radius1)
```

[ ]:
```
# Here we define all the functions and constants that will be written in our␣
↪data file and
# read by Rayleigh if we choose the custom reference state (=4)

unity = numpy.ones(nr,dtype='float64')
buoy =density*gravity # buoyancy term calculation

my_ref.set_function(density,1) # density rho
my_ref.set_function(buoy,2)    # buoyancy term
my_ref.set_function(unity,3)  # nu(r) -- can be overwritten via nu_type in␣
↪Rayleigh
my_ref.set_function(temperature,4) # temperature T
my_ref.set_function(unity,5)  # kappa(r) -- works like nu
my_ref.set_function(hprofile_nd,6)  # heating profile

my_ref.set_function(dlnrho,8) # dlnrho/dr
my_ref.set_function(d2lnrho,9) # d^2lnrho/dr^2
my_ref.set_function(dlnt,10)   # dlnT/dr
my_ref.set_function(unity,7)  # eta -- works like nu and kappa
my_ref.set_function(dsdr,14) # dS/dr


# The constants can all be set/overridden in the input file
# NOTE that they default to ZERO, but we want
# most of them to be UNITY.  These constants will explicitly depend on the non-
↪dimensionalization chosen.

# The comments corresponding to each one of the constants are generic but  we␣
↪also specify
# what they are exactly in our example here, assuming the non-dimensionalization␣
↪we used.
```

(continues on next page)

(continued from previous page)

```
# This is a non-magnetic, non-rotating example that reproduces the results from␣
↪Featherstone & Hindman (2016)
# for the case with Nrho=3,and  kappa=nu=8e12

my_ref.set_constant(1.0,1)  # multiplies the Coriolis force, here it is: 2/E
my_ref.set_constant(13303.43109666924,2)  # multiplies buoyancy -- Ra/Pr here
my_ref.set_constant(1.0,3)  # multiplies the pressure gradient
my_ref.set_constant(0.0 , 4)  # multiplies the lorentz force, here it is: (1/
↪(4*pi)*1/(Pm*E))
my_ref.set_constant(1.0,5)  # multiplies viscosity
my_ref.set_constant(1.0,6)  # multiplies entropy diffusion , here it is 1/Pr
my_ref.set_constant(0.0,7)  # multiplies magnetic diffusion in induction␣
↪equation, here: 1/Pm
my_ref.set_constant(0.00012954929449971041,8)  # multiplies viscous heating,␣
↪here it is: Di*Pr/Ra
my_ref.set_constant(1.0,9)  # multiplies ohmic heating, here it is: (1/
↪(4*pi)*Di*Pr/(E*Pm^2*Ra))
my_ref.set_constant(1.0,10) # multiplies the heating function -- here it is 1/Pr␣
↪(if normalized to 1, this is the luminosity)
my_ref.write('CZtest.dat') # Here we write our data file to be used to run our␣
↪simulation with Rayleigh!
print(my_ref.fset)
print(my_ref.cset)
```

[ ]:
```
# Once you've run for one time step, set have_run = True

# Here we check if the output reference state is the same as the one we used as␣
↪an input (sanity check)!

# NOTE: We need the output file "equation_coefficients" to run this, as well as␣
↪the PDE_Coefficients
# from rayleigh_diagnostics.py

try:
    cref = rdiag.PDE_Coefficients()
    have_run = True
except:
    have_run = False

if (have_run):

    fig, ax = plt.subplots(ncols=3,nrows=4, figsize=(16,4*4))

    # Density and derivatives of lnrho
```

(continues on next page)

(continued from previous page)

```
    ax[0][0].plot(cref.radius,cref.density,'yo')
    ax[0][0].plot(radius1,density)
    ax[0][0].set_xlabel('Radius')
    ax[0][0].set_title('Density')

    ax[0][1].plot(cref.radius, cref.dlnrho,'yo')
    ax[0][1].plot(radius1, dlnrho)
    ax[0][1].set_xlabel('Radius')
    ax[0][1].set_title('Log density gradient')

    ax[0][2].plot(cref.radius,cref.d2lnrho,'yo')
    ax[0][2].plot(radius1,d2lnrho)
    ax[0][2].set_xlabel('Radius')
    ax[0][2].set_title('d_dr{Log density gradient}')

    # Temperature  and derivative of lnT
    ax[1][0].plot(cref.radius,cref.temperature,'yo')
    ax[1][0].plot(radius1,temperature)
    ax[1][0].set_xlabel('Radius')
    ax[1][0].set_title('Temperature')

    ax[1][1].plot(cref.radius, cref.dlnT,'yo')
    ax[1][1].plot(radius1, dlnt)
    ax[1][1].set_xlabel('Radius')
    ax[1][1].set_title('Log temperature gradient')

    # dS/dr
    ax[2][1].plot(cref.radius, cref.dsdr,'yo')
    ax[2][1].plot(radius1, dsdr)
    ax[2][1].set_xlabel('Radius')
    ax[2][1].set_title('Entropy gradient')

    # Buoyancy, Heating
    # Note that you must build the buoyancy term from the functions/constants
    ax[3][1].plot(cref.radius, cref.functions[:,1]*cref.constants[1],'yo')
    ax[3][1].plot(radius1, gravity*density*Ra)
    ax[3][1].set_xlabel('Radius')
    ax[3][1].set_title('Buoyancy')

    # Note that the output heating (cref.heating) is hprofile/density/temperature
    ax[3][2].plot(cref.radius, cref.heating,'yo')
    ax[3][2].plot(radius1, hprofile_nd/density/temperature)
    ax[3][2].set_xlabel('Radius')
    ax[3][2].set_title('Heating')

    plt.tight_layout()
```

(continues on next page)

(continued from previous page)

```
    plt.show()
```

[ ]:

[ ]:

An example for custom reference states from MESA

This script will take a MESA stellar evolution profile and convert it into a format that can be read in as a custom reference state in Rayleigh. You will need the `rayleigh_diagnostics.py`, `reference_tools.py`, and `mesa.py` files. You will also need a suitable MESA profile file, such as `profile_mesa.data`.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate as spi
import scipy.integrate as spint
import scipy.signal as spsig
import sys, os

sys.path.insert(0, os.path.abspath('../../'))

import post_processing.rayleigh_diagnostics as rd
import post_processing.reference_tools as rt
import mesa
%matplotlib inline
```

[ ]:
```
def interp(r, v):
    prad = p.rmid[::-1] * mesa.rsol
    #You can also use 10**p.logR[::-1] or p.radius[::-1] instead of rmid[::-1],␣
↪but rmid is the most accurate choice
    return np.interp(r, prad, v[::-1])
```

Set the `work_dir` variable to the location of the Python files listed above and MESA profile you would like to use.

[ ]:
```
work_dir = './'
sys.path.append(work_dir)
```

[ ]:
```
p = mesa.profile('profile_mesa.data')
```

Choose a suitable number of radial grid points. They do not need to be regularly spaced. You should err on the side of high resolution since Rayleigh's Chebyshev domains have very fine grid spacing at the top and bottom of the domain.

[ ]:
```
nr = 5000
r0 = 5.1e10 # in cm
r1 = 6.8e10 # in cm
radius = np.linspace(r0, r1, nr)
```

From the MESA model, Rayleigh will need the density, buoyancy function $\rho g/C_P$, temperature, vicoscity, thermal diffusion, electrical resistivity (for magnetic cases), heating profile (for cases with $Q \neq 0$), entropy gradient (for cases with reference state advection). Note that MESA radial indicies start at the bottom, while Rayleigh radial indicies start at the top.

[ ]:
```
r_MESA = p.rmid*mesa.rsol
density = interp(radius, 10**p.logRho)
temperature = interp(radius, 10**p.logT)
grav = interp(radius, p.grav)
cp = interp(radius, p.cp)
buoy = density * grav / cp
nu = 1e14 * np.ones_like(radius)
kappa = 1e14 * np.ones_like(radius)
eta = 1e14 * np.ones_like(radius)
hprofile = np.zeros_like(radius)
dsdr = np.zeros_like(radius)
```

WARNING You should be very careful how you think about the entropy gradient when moving from MESA to Rayleigh due to the differing equations of state. Here we have chosen to simply set the reference state entropy gradient to zero and let the convection establish it's own entropy gradient. This may or may not be satisfactory for your application.

ANOTHER WARNING You should be very careful with radiative luminosity and/or nuclear energy generation. There are a number of ways to compute the heating functions you need. For this example, we have chosen to simply compute the luminosity profile needed for flux balance if the convective transport matches the values from MESA.

[ ]:
```
q_rad = -np.gradient((p.luminosity - p.conv_L_div_L*p.luminosity)*mesa.solarlum,␣
↪r_MESA)/(4.0*np.pi*r_MESA**2)
heatingp = interp(radius, q_rad)
luminosity = np.trapz(4.0*np.pi*radius**2*heatingp, radius)
hprofile = heatingp/luminosity
```

Plot the density from MESA and the newly interpolated density that will be fed into Rayleigh to make sure they are consistent. We also plot the heating profile as a sanity check.

[ ]:
```
plt.plot(radius, density,'or')
plt.plot(p.rmid[::-1] * mesa.rsol, 10**p.logRho[::-1], '-b')
plt.xlabel('Radius (cm)')
plt.ylabel(r'Desnity (g/cm$^3$)')
plt.xlim([r0,r1])
plt.ylim([np.min(density), np.max(density)])
```

```
[ ]: plt.plot(radius, luminosity*hprofile, '-r')
     plt.xlabel('Radius (cm)')
     plt.ylabel(r'$Q$ (erg/(cm$^3$ s))')
```

Now create the data structure that will be written to a file that Rayleigh can read, and then load in the needed radial functions.

```
[ ]: my_ref = rt.equation_coefficients(radius)

     my_ref.set_function(density,'density')
     my_ref.set_function(buoy,'buoy')
     my_ref.set_constant(1.0, 'buoy_fact')
     my_ref.set_function(nu,'nu')
     my_ref.set_constant(1.0, 'visc_fact')
     my_ref.set_function(temperature,'temperature')
     my_ref.set_function(kappa,'kappa')
     my_ref.set_constant(1.0, 'diff_fact')
     my_ref.set_constant(1.0, 'p_fact')
     my_ref.set_function(hprofile,'heating')
     my_ref.set_constant(1.0, 'luminosity')
     my_ref.set_function(eta,'eta')
     my_ref.set_constant(1.0, 'resist_fact')
     my_ref.set_function(dsdr,'ds_dr')

     my_ref.set_constant(luminosity,'luminosity')
     print(my_ref.constants)
```

```
[ ]: file_write='cref_from_MESA.dat'
     my_ref.write(file_write)
```

Now you can use this file to run a Rayleigh model. Once your Rayleigh model has run you can use the `equation_coefficients` file to check how your specified reference state looks when transfered into Rayleigh.

```
[ ]: #Once you're run for one time step, set have_run = True
     radius1 = radius
     gravity = grav
     try:
         cref = rd.PDE_Coefficients()
         have_run = True
     except:
         have_run = False

     if (have_run):

         fig, ax = plt.subplots(ncols=3,nrows=3, figsize=(9,3*3))
         # Density variables
```

(continues on next page)

(continued from previous page)

```
    ax[0][0].plot(cref.radius,cref.density,'yo')
    ax[0][0].plot(radius1,density)
    ax[0][0].set_xlabel('Radius')
    ax[0][0].set_title('Density')

    ax[0][1].plot(cref.radius, cref.nu,'yo')
    ax[0][1].plot(radius1, nu)
    ax[0][1].set_xlabel('Radius')
    ax[0][1].set_title(r'$\nu$')

    ax[0][2].plot(cref.radius,cref.kappa,'yo')
    ax[0][2].plot(radius1,kappa)
    ax[0][2].set_xlabel('Radius')
    ax[0][2].set_title(r'$\kappa$')

    ax[1][1].plot(cref.radius,cref.temperature,'yo')
    ax[1][1].plot(radius1,temperature)
    ax[1][1].set_xlabel('Radius')
    ax[1][1].set_title('Temperature')

    ""
    #Activate this if your case is magnetic
    ax[1][0].plot(cref.radius, cref.eta,'yo')
    ax[1][0].plot(radius1, eta)
    ax[1][0].set_xlabel('Radius')
    ax[1][0].set_title(r'$\eta$')
    ""

    ax[2][1].plot(cref.radius, cref.dsdr,'yo')
    ax[2][1].plot(radius1, dsdr)
    ax[2][1].set_xlabel('Radius')
    ax[2][1].set_title('Log entropy gradient')

    ax[1][2].plot(cref.radius, cref.functions[:,1]*cref.constants[1],'yo')
    ax[1][2].plot(radius1, gravity*density/cp)
    ax[1][2].set_xlabel('Radius')
    ax[1][2].set_title('Gravity')

    ax[2][0].plot(cref.radius, cref.heating*cref.rho*cref.T,'yo')
    ax[2][0].plot(radius1, hprofile*luminosity)
    ax[2][0].set_xlabel('Radius')
    ax[2][0].set_title('Heating')

    plt.tight_layout()
    plt.show()
```

[ ]:

1.3.7 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 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.

By default, magnetic boundary conditions are set to match to a potential field at each boundary.

By default, the thermal anomoly 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.

Generic Boundary Conditions

Boundary conditions for temperature, T, and the magnetic poloidal potential, C, may also be set using generic spectral input. As with initial conditions (see *Generic Initial Conditions*) this is done by generating a generic input file using the **rayleigh_spectral_input.py** script. The file output from this script can then be applied using:

- **fix_Tvar_top** and **T_top_file** (overrides **T_top** for a constant boundary condition) to set a fixed upper T boundary condition
- **fix_dTdr_top** and **dTdr_top_file** (overrides **dTdr_top**) to set a fixed upper T gradient boundary condition
- **fix_Tvar_bottom** and **T_bottom_file** (overrides **T_bottom**) to set a fixed lower T boundary condition
- **fix_dTdr_bottom** and **dTdr_bottom_file** (overrides **dTdr_bottom**) to set a fixed lower T gradient boundary condition
- **fix_poloidal_top** and **C_top_file** (overrides **impose_dipole_field**) to set a fixed upper C boundary condition
- **fix_poloidal_bottom** and **C_bottom_file** (overrides **impose_dipole_field**) to set a fixed lower C boundary condition

For example, to set a C boundary condition on both boundaries with modes (l,m) = (1,0) and (1,1) set to pre-calculated values run:

```
rayleigh_spectral_input.py -m 1 0 2.973662220170157 -m 1 1 0.5243368809294343+0.
↪j -o ctop_init_bc
rayleigh_spectral_input.py -m 1 0 8.496177771914736 -m 1 1 1.4981053740840984+0.
↪j -o cbottom_init_bc
```

which will generate generic spectral input files *ctop_init_bc* and *cbottom_init_bc*. Set these to be used as the boundary conditions in *main_input* using:

```
&Boundary_Conditions_Namelist
fix_poloidalfield_top = .true.
fix_poloidalfield_bottom = .true.
C_top_file = 'ctop_init_bc'
C_bottom_file = 'cbottom_init_bc'
/
```

This can be seen being applied in *tests/generic_input*.

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 :

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

where γ is a normalization constant defined such that

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

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

$$\hat{\rho}\hat{T}\frac{\partial \Theta}{\partial t} = \gamma.$$

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

Note: If internal heating is used in combination with **fix_dTdr_top**, then the value of $\partial\Theta/\partial r$ at the upper boundary is set by Rayleigh. Any value for **dTdr_top** specified in main_input is ignored. This is done to ensure consistency with the internal heating and any flux passing through the lower boundary due to the use of a fixed-flux condition. To override this behavior, set **adjust_dTdr_top** to .false. in the **Boundary_Conditions** namelist.

1.3.8 Output Controls

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). Here we will focus on generating basic output, but we refer the user to the section plotting and *Output Quantity Codes* for more information.

Rayleigh can compute the quantities listed in *Output Quantity Codes* in a variety of averages, slices, and spectra, which are collectively called data products. These are sorted by Rayleigh into directories as follows:

- G_Avgs: The quantity is averaged over the entire simulation volume.
- Shell_Avgs: The quantity is averaged over each spherical shell and output as a function of radius.
- AZ_Avgs: The quantity is averaged over longitude and output as a function of radius and latitude.
- Shell_Slices: The quantity at the specified radii is output as a function of latitude and longitude.
- Equatorial_Slices: The quantity at the specified latitudes is output as a function of radius and longitude.
- Meridional_Slices: The quantity at the specified longitudes is output as a function of radius and latitude.
- Spherical_3D: The qunatity over the entire domain. Careful – these files can be quite large.
- Shell_Spectra: The quantity's spherical harmonic coefficents at the specified radii.
- Point_Probes: The quantity at a specified radius, latitude, and longtiude.
- SPH_Mode_Sampels: The quantity's spherical harmonic coefficents at the specified radii and ℓ.

In addition Rayleigh can output *Checkpoints*, which are the data required to restart Rayleigh and will be discussed in detail in *Checkpointing*, and *Timings*, which contain information about the performance of the run.

Output in Rayleigh is controled through the *io_controls_namelist*. For each of the data products listed, the output is specified using the following pattern:

- _values: The quantity codes desired (seperated by commas)
- _frequency: The frequency in iterations those quantities will be output.
- _nrec: Number of records to be combined into a single file.
- _levels: Radial indicies at which the quantities will be output.
- _indices: Latitudinal indicies at which the quantities will be output.
- _ell: The spherical harmonic degree at which the quantities will be output.
- _r, _theta, _phi: The radial, latitudinal, and longitudinal indicies at which the quantities will be output.

For example, if you wanted to output shell slice data for quantities 1, 2, 10, and 2711 at radial indicices 2 and 54 every 100 iterations and have 4 records per file, you would set

```
shellslice_levels    = 2,54
shellslice_values    = 1,2,10,2711
shellslice_frequency = 100
shellslice_nrec      = 4
```

Files output in this way will have the filename of their iteration.

1.3.9 Transport coefficents

Transport coefficients (viscosity, thermal diffusivity) are specified in the transport namelist.

Table. Anelastic Transport.

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.

Variable [Default value]	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

1.3.10 Examples from Recent Publications

A Solar-like Case

This is the main_input file from Case 39 from:

Hindman, Bradley W., Nicholas A. Featherstone, and Keith Julien. 2020. "Morphological Classification of the Convective Regimes in Rotating Stars." The Astrophysical Journal 898 (2): 120. https://doi.org/10.3847/1538-4357/ab9ec2.

```
&problemsize_namelist
n_r = 64
n_theta = 192
nprow = 32
npcol = 32
rmin = 5.0d10
rmax = 6.5860209d10
/
&numerical_controls_namelist
/
```

(continues on next page)

(continued from previous page)

```
&physical_controls_namelist
rotation  = .true.
magnetism = .false.
/
&temporal_controls_namelist
max_time_step = 1000.0d0
max_iterations = 5000000
checkpoint_interval = 50000
quicksave_interval = 10000
num_quicksaves = 4
cflmin = 0.4d0
cflmax = 0.6d0
/
&io_controls_namelist
/
&output_namelist
!shellslice_levels    = 16,32,48,64,80,96,112
!shellslice_values    = 1                                                 ! Codes␣
↪needed for standard output routines
shellslice_levels    = 8,16,24,32,40,48,56,64,72,80,88,96,104,112,120
shellslice_values    = 1,2,3,301,302,303,304,305,306,307,308,309,401,501,502,
↪2701,2702,2703,2704,2705,2706,2707,2708,2709,2710,2711
shellslice_frequency = 10000
shellslice_nrec      = 1

!shellspectra_values    = 1,2,3                                            ! Codes␣
↪needed for standard output routines
shellspectra_levels    = 16,32,48,64,80,96,112
shellspectra_values    = 1,2,3,301,302,303,304,305,306,307,308,309,401,501,502,
↪503,504,2701,2702,2703,2704,2705,2706,2707,2708,2709,2710,2711
shellspectra_frequency = 10000
shellspectra_nrec      = 1

!azavg_values    = 1,2,3,201,202                                          ! Codes␣
↪needed for standard output routines
azavg_values    = 1,2,3,201,202,401,405,409,501,502,1433,1455,1470,1923,1935,
↪1943,2701,2702,2703,2704,2705,2706,2707,2708,2709,2710,2711,2712,2713,2714,2715
azavg_frequency = 1000
azavg_nrec = 10

!shellavg_values    = 1,2,3,501,502,1433,1455,1470,1923,1935              ! Codes␣
↪needed for standard output routines
shellavg_values    = 1,2,3,401,405,409,501,502,1433,1455,1470,1923,1935,2701,
↪2702,2703,2704,2705,2706,2707,2708,2709,2710,2711,2712,2713,2714,2715
shellavg_frequency = 100
shellavg_nrec = 100
```

(continues on next page)

(continued from previous page)

```
!globalavg_values = 401,402,403,404,405,406,407,408,409,410,411,412      ! Codes␣
↪needed for standard output routines
globalavg_values = 401,402,403,404,405,406,407,408,409,410,411,412,413,417,421,
↪2701,2702,2703,2704,2705,2706,2707
globalavg_frequency = 100
globalavg_nrec = 100

!equatorial_values    = 1,3                                              ! Codes␣
↪needed for standard output routines
equatorial_values    = 1,2,3,4,5,6,201,203,301,302,303,304,305,306,307,308,309,
↪401,501,502,503,504,2701,2702,2703,2704,2705,2706,2707,2708,2709,2710,2711
equatorial_frequency = 10000
equatorial_nrec      = 1

full3d_values = 4
full3d_frequency = 9000000
/

&Boundary_Conditions_Namelist
no_slip_boundaries = .false.
strict_L_Conservation = .false.
dtdr_bottom = 0.0d0
T_Top    = 0.0d0
T_Bottom = 851225.7d0
fix_tvar_top = .true.
fix_tvar_bottom = .false.
fix_dtdr_bottom = .true.
/
&Initial_Conditions_Namelist
init_type = 7
magnetic_init_type = -1
mag_amp = 1.0d0
temp_amp = 1.0d1
temp_w = 0.01d4
!restart_iter = 0    ! restart from latest checkpoint of any flavor
/
&Test_Namelist
/
&Reference_Namelist
reference_type = 2
heating_type = 1
luminosity = 3.846d33
poly_n = 1.5d0
poly_Nrho = 3.0d0
poly_mass = 1.98891D33
```

(continues on next page)

(continued from previous page)

```
poly_rho_i = 0.18053428d0
pressure_specific_heat = 3.5d8
angular_velocity = 5.74d-6   ! Sidereal period of 12.7 days (twice the sidereal␣
↪Carrington rate)
/
&Transport_Namelist
nu_top    = 4.d12
kappa_top = 4.d12
/
```

1.4 Running Rayleigh

After setting up a custom *main_input* file, now it is time to run the new model. This section focuses on the basics of running a Rayleigh model.

1.4.1 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:

$$\mathrm{N} = \mathrm{npcol} \times \mathrm{nprow}.$$

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
```

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 = nprow $\times$ 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

$$N_\ell = \frac{2}{3} N_\theta$$

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

1.4.2 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 *table_checkpoints*). 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 *table_checkpoints*.

Table. Checkpoints.

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

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

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 **quicksaves**. This section discusses how to generate and restart from both types of checkpoints.

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.

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.

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 5,000:

```
&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 restart_iter:

```
&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.

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.

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.

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.

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.

1.4.3 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.

An orderly shutdown of Rayleigh can be manually triggered by creating a file with the name set in **terminate_file** (i.e., running the command *touch terminate* in the default setting). If the file is found, Rayleigh will stop after the next time step and write a checkpoint file. The existence of **terminate_file** is checked every **terminate_check_interval** iterations. The check can be switched off completely by setting **terminate_check_interval** to -1. Both of these options are set in the **io_controls_namelist**. With the appropriate job script this feature can be used to easily restart the code with new settings without losing the current allocation in the queuing system. A **terminate_file** left over from a previous run is automatically deleted when the code starts.

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 $t_{\rm CFL}$ 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 { $\Delta_t \geq \mathrm{cflmax} \times t_{\rm CFL}$ } THEN { Δ_t is set to $\mathrm{cflmax} \times t_{\rm CFL}$ }.
- IF { $\Delta_t \leq \mathrm{cflmin} \times t_{\rm CFL}$ } THEN { Δ_t is set to $\mathrm{cflmax} \times t_{\rm CFL}$ }.
- IF{ $t_{\rm CFL} \geq$ max_time_step } THEN { Δ_t is set to max_time_step }
- IF{ $t_{\rm CFL} \leq$ min_time_step } THEN { Rayleigh Exits }

The default values for these variables are:

```
&temporal_controls_namelist
max_iterations = 1000000
```

(continues on next page)

(continued from previous page)

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

1.4.4 The Log File

Section needs to be written.

1.4.5 I/O Control

Some aspects of Rayleigh's I/O can be controlled through variables found in the io_controls namelist.

I/O Format Controls

By default, integer output is reported with 8 digits and padded with leading zeros. This includes integer iteration numbers reported to stdout at each timestep and integer-number filenames created through diagnostics and checkpointing output. If desired, the number of digits may be controlled through the **integer_output_digits** variable. When reading in a Checkpoint created with a different number of digits, set the **integer_input_digits** variable to an appropriate value.

At several points in the code, floating-point output is sent to stdout. This output is formatted using scienific notation, with three digits to the right of the decimal place. The number of digits after the decimal can be controlled through the **decimal_places** variable.

As an example, the following combination of inputs

```
&temporal_controls_namelist
checkpoint_interval=10
/
&io_controls_namelist
integer_output_digits=5
integer_input_digits=3
decimal_places=5
/
&initial_conditions_namelist
init_type=-1
restart_iter=10
/
```

would restart from checkpoint files with the prefix formatted as:

```
Checkpoints/010_grid_etc.
```

It would generate status line, shell_slice output, and checkpoints formatted as:

```
Iteration:  00033   DeltaT:  1.00000E-04   Iter/sec:  2.68500E+00
Shell_Slices/00020
Checkpoints/00020_grid_etc.
```

Developer's Note: The format codes generated through the values of these three variables are declared (with descriptive comments) in Controls.F90. For integer variables that may take on a negative value, additional format codes with one extra digit (for the negative sign) are also provided.

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 path 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 be 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 directed to normal stdout.

To save on disk space for logs of very long runs, the number of status outputs can be reduced by specifying **statusline_interval** in the **io_controls_namelist**. This causes only every n-th status line to be written.

1.4.6 Run Termination

Section needs to be written.

1.5 Running at Scale

1.5.1 Getting Ready for Large Runs

Need text here.

1.5.2 Modules and Queues

Need text here.

1.5.3 Example Configurations

Need text here.

1.5.4 Sample Jobscripts

Need text here.

1.5.5 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 mulitple, 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
&problemsize_namelist
 n_r = 128
 n_theta = 192
 nprow = 16
 npcol = 16
/

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

user@machinename ~runs/ensemble $ head run3/main_input
&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 §*I/O Control*). 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.

1.6 Analyzing Output

As discussed in *Output Controls*,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. We refer the user to the diagnostics plotting notebook, located at Rayleigh/post_processing/Diagnostic_Plotting.ipynb. An html version is provided *here.*

1.6.1 The Lookup Table (LUT)

Rayleigh has on the order of 1,000 possible diagnostic quantities available to the user. As discussed in the examples above, the user specifies which diagnostic outputs to compute by providing the appropriate quantity codes in the input file. Internally, Rayleigh uses the quantity codes similarly to array indices. The purpose of the lookup table is to map the quantity code to the correct position in the output data array, you should never assume the quantities will be output in any particular order. The user may have only requested two quantity codes, for example, 1 and 401. The output data array will be of size 2 along the axis corresponding to the quantities. The lookup table could map 401 to the first entry and 1 to the second entry.

The standard way to interact with the lookup table is to know the quantity code and explicitly use it. Here we describe an alternative method. Each quantity code entry (*Output Quantity Codes*) has an equation, a code, and a name. There are some python scripts in the post_processing directory that allow you to use the name, instead of the code, when interacting with the lookup table:

- lut.py
- generate_mapping.py
- lut_shortcuts.py

The lut.py file is the main user-interface and contains various utility routines, including functions to convert between codes and names. The generate_mapping.py file is responsible for generating the mapping between codes and names. The lut_shortcuts.py allows users to define their own mapping, allowing a conversion from a user-defined name to the desired quantity code. The lut_shortcuts.py file does not exist in the source code, it must be generated by the user; an example shortcuts file can be found in the post_processing/lut_shortcuts.py.example file. The fastest way to start using shortcuts is to copy the example file:

```
cd /path/to/Rayleigh
cd post_processing/
cp lut_shortcuts.py.example lut_shortcuts.py
```

and then make edits to the new lut_shortcuts.py file.

The mapping has already been generated and is stored in the lut_mapping.py file. For developers or anyone wanting to re-generate the mapping, use the generate_mapping.py file:

```
python generate_mapping.py /path/to/Rayleigh
```

This will parse the Rayleigh directory tree and generate the standard mapping between quantity codes and their associated names stored in the new file lut_mapping.py. Only quantity codes that are defined within the Rayleigh source tree will be included. Rayleigh does not need to be compiled before generating the mapping.

If a user has a custom directory where output diagnostics are defined, the above command will not capture

the custom diagnostic codes. To include custom quantities, the user must generate the mapping themselvese with the generate_mapping.py file:

```
python generate_mapping.py /path/to/Rayleigh/ --custom-dir=/path/to/custom/
```

Note that the Rayleigh directories are identical between the two calls, the only addition is the custom-dir flag. This command will generate a new mapping stored in the file lut_mapping_custom.py and will include all of the standard output quantities as well as the custom diagnostics.

Without using this mapping technique, plotting something like the kinetic energy could appear as:

```
from rayleigh_diagnostics import G_Avgs, build_file_list

files = build_file_list(0, 10000000, path='G_Avgs')
g = G_Avgs(filename=files[0], path='')

ke_code = g.lut[401] # must use quantity code in lookup table

ke = g.data[:, ke_code] # extract KE as a function of time
```

With the newly generated mapping, the above code could be rewritten as:

```
from rayleigh_diagnostics import G_Avgs, build_file_list

from lut import lookup # <-- import helper function from main interface

files = build_file_list(0, 10000000, path='G_Avgs')
g = G_Avgs(filename=files[0], path='')

ke_code = g.lut[lookup('kinetic_energy')] # use quantity *name* in lookup table

ke = g.data[:, ke_code] # extract KE as a function of time, same as before
```

There is one drawback to using the quantity names: the naming scheme is somewhat random and they can be quite long strings. This is where the lut_shortcuts.py can be very useful. This allows users to define their own names to use in the mapping. These are defined in the lut_shortcuts.py file and always take the form:

```
shortcuts['custom_name'] = 'rayleigh_name'
```

where custom_name is defined by the user, and rayleigh_name is the quantity name that Rayleigh uses. The main dictionary must be named 'shortcuts'. With an entry like:

```
shortcuts['ke'] = 'kinetic_energy'
```

the above example for extracting the kinetic energy is even more simple:

```
from rayleigh_diagnostics import G_Avgs, build_file_list

from lut import lookup # <-- import helper function from main interface

```

(continues on next page)

(continued from previous page)

```
files = build_file_list(0, 10000000, path='G_Avgs')
g = G_Avgs(filename=files[0], path='')

ke_code = g.lut[lookup('ke')] # user defined *name* in lookup table

ke = g.data[:, ke_code] # extract KE as a function of time, same as before
```

1.6.2 Plotting Examples

Note: Please note this notebook has not been updated since the conversion to online documentation (July 2019). The Rayleigh/doc directory has been reorganized. A pdf version of the document can be created by the user through the website.

Nick Featherstone (January, 2018)

NOTE: This document can be viewed in PDF or HTML (recommended) form. It can also be run as an interactive Jupyter notebook.

The HTML and PDF versions are located in Rayleigh/doc/Diagnostic_Plotting.{html,pdf}
The Jupyter notebook is located in Rayleigh/post_processing/Diagnostic_Plotting.ipynb
Standalone Python example scripts for each output type may also found in Rayleigh/post_processing/

Contents

I. Running a Benchmark with Sample Output

Before you can plot data, you will need to generate data. The code samples in this notebook assume that you have run the model described by the input file found in:

rayleigh/input_examples/benchmark_diagnostics_input

This input file instructs *Rayleigh* to run the Christensen et al. (2001) hydrodynamic (case 0) benchmark. Running this model with the prescribed outputs will generate approximately 70 MB of data.

To run this model: 1. Create a directory for your simulation (e.g., **mkdir my_test_run**) 2. Copy the input file: **cp rayleigh/input_examples/benchmark_diagnostics_input my_test_run/main_input** 3. Copy or soft-link the *rayleigh* executable: **cp Rayleigh/bin/rayleigh.opt my_test_run/.** 4. Run the code: **mpiexec -np N ./rayleigh -nprow n -npcol m** (choose values of {N,n,m} such that n x m = N)

The code will run for 40,000 timesteps, or four viscous diffusion times. While it runs, *Rayleigh* will perform an in-situ analysis of the accuracy benchmark. Reports are written once every 1,000 time steps and are stored in the *Benchmark_Reports* subdirectory. Examine file 00030000 and ensure that you see similar results to those below. Your exact numbers may differ slightly, but all quantities should be under 1% difference.

Observable	Measured	Suggested	% Difference	Std. Dev.
Kinetic Energy	58.347827	58.348000	-0.000297	0.000000
Temperature	0.427424	0.428120	-0.162460	0.000101
Vphi	-10.119483	-10.157100	-0.370356	0.013835
Drift Frequency	0.183016	0.182400	0.337630	0.007295

If necessary, copy the data to the system on which you intend to conduct your analysis. Before you can plot, you will need to configure your Python environment.

II. Configuring Your Python Environment

Rayleigh comes packaged with a Python library (*rayleigh_diagnostics.py*) that provides data structures and methods associated with each type of diagnostic output in Rayleigh. This library relies on Numpy and is compatible with Python 3.x or 2.x (The *print* function is imported from the **future** module).

If you wish to follow along with the plotting examples described in this document, you will need to have the Numpy and Matplotlib Python packages installed. The following versions of these packages were used when creating these examples: * Matplotlib v2.0.2 * Numpy v1.13.1

Unless you are experienced at installing and managing Python packages, I recommend setting up a virtual environment for Python using Conda. You may also install the required packages manually, but the advantage of this approach is that you maintain an entirely separate version of Python and related packages for this project. Below are directions for setting up a Python/Conda environment with Intel-optimized Python packages on a Linux system (Mac and Windows work similarly).

Conda Installation on Linux Systems

Step 1: Download the appropriate Miniconda installation script from https://conda.io/miniconda.html (choose Python 3.x)

Step 2: Make the shell script executable via: **chmod +x Miniconda3-latest-Linux-x86_64.sh** (or similar script name)

Step 3: Run the installation script: **./Miniconda3-latest-Linux-x86_64.sh**

NOTE: The default installation directory is your home directory. This is also where Python packages for your Conda environments will be installed. Avoid installing to a disk with limited space (user home directories on HPC systems are often limited to a few GB).

NOTE: Unless you have a specific reason not to do so, answer "yes" to the question concerning prepending to PATH.

Step 5: Update your Conda: **conda update conda**

Step 6: Add the Intel Conda channel: **conda config –add channels intel**

Step 7: Create a virtual environment for Intel's Conda distribution: **conda create -n idp intelpython3_full python=3**

NOTE: In this case, *idp* will be your virtual environment name. You are free to pick an alternative when running conda create.

NOTE: A number of Python packages will be downloaded, including Numpy and Matplotlib. The process may appear to hang at the last step. Be patient.

Step 8: Activate your virtual environment: **source activate idp**

Step 9: Verify your installation. Type **python** and then type the following commands at the prompt: 1. import numpy 2. import matplotlib

If those commands worked without error, you may close Python (type **exit()**). You can revert to your native environment by typing **source deactivate** (or just close the terminal). Whenever you wish to access your newly-installed Python, type **source activate idp** first, before running python.

Preparing to Plot

All examples in this document rely on the rayleigh_diagnostics module. This module is located in Rayleigh/-post_processing, along with several standalone scripts copies from the individual sections of this document. For example, the script **plot_G_Avgs.py** contains the code from section IV below. All python files you wish to use will need to reside in either your run directory (recommended) or a directory within your PYTHON-PATH.

We suggest copying all python files to your my_test_run directory: 1. cp Rayleigh/post_processing/*.py my_test_run/. 2. cp Rayleigh/post_processing/*.ipynb my_test_run/.

The Jupyter Notebook

This document resides in three places: 1. Rayleigh/doc/Diagnostic_Plotting.pdf 2. Rayleigh/doc/Diagnostic_Plotting.html 3. Rayleigh/post_processing/Diagnostic_Plotting.ipynb

The third file is a Jupyter notebook file. This source code was used to generate the html and pdf documents. The notebook is designed to be run from within a Rayleigh simulation directory. If you wish to follow along interactively, copy the Jupyter notebook file from Rayleigh/post_processing/ into your Rayleigh simulation directory (step 2 from *Preparing to Plot*). You can run the file in Jupyter via: 1. source activate idp 2. jupyter notebook (from within your my_test_run directory) 3. select Diagnostic_Plotting.ipynb in the file menu that presents itself.

When finished: 1. To close the notebook, type **ctrl+c** and enter "yes" when prompted to shut down the notebook server. 2. type **source deactivate**

III. Overview of Diagnostics in Rayleigh

Rayleigh's diagnostics package facilitates the in-situ analysis of a simulation using a variety of sampling methods. Each sampling method may be applied to a unique set of sampled quantities. Sampling methods are hereafter referred to as *output types* and sampled quantities as *output variables*.

Files of each output type are stored in a similarly-named subdirectory within the *Rayleigh* simulation directory. Output files are numbered by the time step of the final data record stored in the file. Output behavior for each simulation is controlled through the *main_input* file. For each output type, the user specifies the output variables, cadence, records-per-file, and other properties by modifying the appropriate variables in the **output_namelist** section of *main_input*.

Basic Output Control

Each output type in *Rayleigh* has at least three namelist variables that govern its behavior:

{OutputType}_values: comma-separated list of menu codes corresponding to the desired output variables

{OutputType}_frequency: integer value that determines how often this type of output is performed

{OutputType}_nrec: integer value that determines how many records are stored in each output file.

All possible output variables and their associated menu codes are described in **rayleigh/doc/rayleigh_output_variables.pdf** You may find it useful to have that document open while following along with examples in this notebook.

As an example of how these variables work, suppose that we want to occasionally output equatorial cuts (output type) of temperature, kinetic energy density, and radial velocity (output variables). At the same time, we might wish to dump full-volume averages (output type) of kinetic and magnetic energy (output variables) with a higher cadence. In that case, something similar to the following would appear in main_input:

globalavg_values = 401, 1101
globalavg_frequency = 50
globalavg_nrec = 100

equatorial_values = 1, 401, 501

equatorial_frequency = 2500

equatorial_nrec = 2

This tells *Rayleigh* to output full-volume-averages of kinetic energy density (value code 401) and magnetic energy density (value code 1101) once every 50 time steps, with 100 records per file. Files are named based on the time step number of their final record. As a result, information from time steps 50, 100, 150, …, 4950, 5000 will be stored in the file named *G_Avgs/00005000*. Time steps 5050 through 10,000 will stored in *G_Avgs/00010000*, and so on.

For the equatorial cuts, *Rayleigh* will output radial velocity (code 1), the kinetic energy density (code 401) and temperature (code 501) in the equatorial plane once every 2,500 time steps, storing two time steps per file. Data from time steps 2,500 and 5,000 will be stored in *Equatorial_Slices/00005000*. Data from time steps 7,500 and 10,000 will be stored in *Equatorial_Slices/00010000* , and so on.

This general organizational scheme for output was adapted from that developed by Thomas Clune for the ASH code.

Positional Output Control

Many of *Rayleigh's* output types allow the user to specify a set of gridpoints at which to sample the simulation. A user can, for example, output spherical surfaces sampled at arbitrary radii, or a meridional plane sampled at a specific longitude. This behavior is controlled through additional namelist variables; we refer to these variables as positional specifiers. In the sections that follow, positional specifiers associated with a given output type, if any, will be defined.

Positional specifiers are either *indicial* or *normalized*. In the *main_input* file, indicial specifiers can be assigned a comma-separated list of grid indices on which to perform the output. For example,

shellslice_levels = 1, 32, 64, 128

instructs *Rayleigh* to output shell slices at { radius[1], radius[32], radius[64], radius[128]}. Note that radius[1] is the outer boundary.

While useful in some situations, specifying indices can lead to confusion if a simulations resolution needs to be changed at some point during a model's evolution. For example if the radial grid initially had 128 points, index 128 would correspond to the lower boundary. If the resolution were to double, index 128 would correspond to mid-shell.

For this reason, all positional specifiers may also be written in normalized form. Instead of integers, the normalized specifier is assigned a comma separated list of real values in the range [0,1]. The value of zero corresponds to the lowest-value grid coordinate (e.g., the inner radial boundary or theta=0 pole). The value 1 corresponds to the maximal coordinate (e.g., the outer radial boundary or theta=pi pole). A value of 0.5 corresponds to mid-domain. Normalized coordinates are indicated by adding *_nrm* to the indicial specifier's name. For example,

shellslice_levels_nrm= 0, 0.5, 0.95

instructs *Rayleigh* to output shell slices at the lower boundary, mid-shell, and slightly below the upper boundary. *Rayleigh* does not interpolate, but instead picks the grid coordinate closest to each specified normalized coordinate.

We recommend using normalized coordinates to avoid inconsistencies between restarts. They also overcome difficulties associated with the non-uniform nature of the radial and theta grids wherein grid points cluster near the boundaries.

Positional Ranges Ranges of coordinates can be specified using shorthand, if desired. The inclusive coordinate range [X,Y] is indicated by a positive/negative number pair appearing in the indicial or normalized coordinate list. Multiple ranges can be specified within a list. For example,

shellslice_levels = 1,10,-15, 16, 20,-25, 128

would instruct *Rayleigh* to output shell slices at radial indices = { 1, 10, 11, 12, 13, 14, 15, 16, 20, 21, 22, 23, 24, 25, 128}

Similarly,

shellslice_levels_nrm = 0,-0.5, 1.0

instructs *Rayleigh* to output shells at all radii in the lower half of the domain, and at the outer boundary.

IV. Global Averages

Summary: Full-volume averages of requested output variables over the full, spherical shell

Subdirectory: G_Avgs

main_input prefix: globalavg

Python Class: G_Avgs

Additional Namelist Variables:
None

Before proceeding, ensure that you have copied Rayleigh/post_processing/rayleigh_diagnostics.py to your simulation directory. This Python module is required for reading Rayleigh output into Python.

Examining the *main_input* file, we see that the following output values have been denoted for the Global Averages (see *rayleigh_output_variables.pdf* for the mathematical formulae):

Menu Code	Description
401	Full Kinetic Energy Density (KE)
402	KE (radial motion)
403	KE (theta motion)
404	KE (phi motion)
405	Mean Kinetic Energy Density (MKE)
406	MKE (radial motion)
407	MKE (theta motion)
408	MKE (phi motion)
409	Fluctuating Kinetic Energy Density (FKE)
410	FKE (radial motion)
411	FKE (theta motion)
412	FKE (phi motion)

In the example that follows, we will plot the time-evolution of these different contributions to the kinetic energy budget. We begin with the following preamble:

[ ]:
```
%matplotlib inline
from rayleigh_diagnostics import G_Avgs, build_file_list
import matplotlib.pyplot as plt
import numpy
```

The preamble for each plotting example will look similar to that above. We import the numpy and matplotlib.pyplot modules, aliasing the latter to *plt*. We also import two items from *rayleigh_diagnostics*: a helper function *build_file_list* and the *GlobalAverage* class.

The *G_Avgs* class is the Python class that corresponds to the full-volume averages stored in the *G_Avgs* subdirectory of each Rayleigh run.

We will use the build_file_list function in many of the examples that follow. It's useful when processing a time series of data, as opposed to a single snapshot. This function accepts three parameters: a beginning time step, an ending time step, and a subdirectory (path). It returns a list of all files found in that directory that lie within the inclusive range [beginning time step, ending time step]. The file names are prepended with the subdirectory name, as shown below.

[ ]:
```
# Build a list of all files ranging from iteration 0 million to 1 million
files = build_file_list(0,1000000,path='G_Avgs')
print(files)
```

We can create an instance of the G_Avgs class by initializing it with a filename. The optional keyword parameter *path* is used to specify the directory. If *path* is not specified, its value will default to the subdirectory name associated with the datastructure (*G_Avgs* in this instance).

Each class was programmed with a **docstring** describing the class attributes. Once you created an instance of a rayleigh_diagnostics class, you can view its attributes using the help function as shown below.

[ ]:
```
a = G_Avgs(filename=files[0],path='')  # Here, files[0]='G_Avgs/00010000'
#a= G_Avgs(filename='00010000') would yield an equivalent result
help(a)
```

Examining the docstring, we see a few important attributes that are common to the other outputs discussed in this document: 1. niter – the number of time steps in the file 2. nq – the number of output variables stored in the file 3. qv – the menu codes for those variables 4. vals – the actual data 5. time – the simulation time corresponding to each output dump

The first step in plotting a time series is to collate the data.

[ ]:
```
# Loop over all files and concatenate their data into a single array
nfiles = len(files)
for i,f in enumerate(files):
    a = G_Avgs(filename=f,path='')
    if (i == 0):
        nq = a.nq
        niter = a.niter
```

(continues on next page)

(continued from previous page)

```
        gavgs = numpy.zeros((niter*nfiles,nq),dtype='float64')
        iters = numpy.zeros(niter*nfiles,dtype='int32')
        time = numpy.zeros(niter*nfiles,dtype='float64')
    i0 = i*niter
    i1 = (i+1)*niter
    gavgs[i0:i1,:] = a.vals
    time[i0:i1] = a.time
    iters[i0:i1] = a.iters
```

The Lookup Table (LUT)

The next step in the process is to identify where within the *gavgs* array our deisired output variables reside. Every Rayleigh file object possesses a lookup table (lut). The lookup table is a python list used to identify the index within the vals array where a particular menu code resides. For instance, the menu code for the theta component of the velocity is 2. The location of v_theta in the vals array is then stored in lut[2].

Note that you should never assume that output variables are stored in any particular order. Moreover, the lookup table is unique to each file and is likely to change during a run if you modify the output variables in between restarts. When running the benchmark, we kept a consistent set of outputs throughout the entirety of the run. This means that the lookup table did not change between outputs and that we can safely use the final file's lookup table (or any other file's table) to reference our data.

Plotting Kinetic Energy

Let's examine the different contributions to the kinetic energy density in our models. Before we can plot, we should use the lookup table to identify the location of each quantity we are interested in plotting.

[ ]:
```
#The indices associated with our various outputs are stored in a lookup table
#as part of the GlobalAverage data structure.  We define several variables to
#hold those indices here:

lut = a.lut
ke  = lut[401]  # Kinetic Energy (KE)
rke = lut[402]  # KE associated with radial motion
tke = lut[403]  # KE associated with theta motion
pke = lut[404]  # KE associated with azimuthal motion

#We also grab some energies associated with the mean (m=0) motions
mke  = lut[405]
mrke = lut[406]  # KE associated with mean radial motion
mtke = lut[407]  # KE associated with mean theta motion
mpke = lut[408]  # KE associated with mean azimuthal motion
```

(continues on next page)

(continued from previous page)

```
#We also output energies associated with the fluctuating/nonaxisymmetric
#motions (e.g., v- v_{m=0})
fke  = lut[409]
frke = lut[410]  # KE associated with mean radial motion
ftke = lut[411]  #KE associated with mean theta motion
fpke = lut[412]  # KE associated with mean azimuthal motion
```

To begin with, let's plot the total, mean, and fluctuating kinetic energy density during the initial transient phase, and then during the equilibrated phase.

[ ]:
```
sizetuple=(10,3)
fig, ax = plt.subplots(ncols=2, figsize=sizetuple)
ax[0].plot(time, gavgs[:,ke], label='KE')
ax[0].plot(time, gavgs[:,mke],label='MKE')
ax[0].plot(time, gavgs[:,fke], label='FKE')
ax[0].legend(loc='center right', shadow=True)
ax[0].set_xlim([0,0.2])
ax[0].set_title('Equilibration Phase')
ax[0].set_xlabel('Time')
ax[0].set_ylabel('Energy')

ax[1].plot(time, gavgs[:,ke], label='KE')
ax[1].plot(time, gavgs[:,mke], label = 'MKE')
ax[1].plot(time,gavgs[:,fke],label='FKE')
ax[1].legend(loc='center right', shadow=True)
ax[1].set_title('Entire Time-Trace')
ax[1].set_xlabel('Time')
ax[1].set_ylabel('Energy')

saveplot = False # Plots appear in the notebook and are not written to disk (set␣
↪to True to save to disk)
savefile = 'energy_trace.pdf'  #Change .pdf to .png if pdf conversion gives␣
↪issues
plt.tight_layout()
plt.show()
```

We can also look at the energy associated with each velocity component. Note that we log scale in the last plot. There is very little mean radial or theta kinetic energy; it is mostly phi energy.

[ ]:
```
sizetuple=(5,10)
xlims=[0,0.2]
fig, ax = plt.subplots(ncols=1, nrows=3, figsize=sizetuple)
ax[0].plot(time, gavgs[:,ke], label='KE')
ax[0].plot(time, gavgs[:,rke],label='RKE')
ax[0].plot(time, gavgs[:,tke], label='TKE')
```

(continues on next page)

(continued from previous page)

```
ax[0].plot(time, gavgs[:,pke], label='PKE')
ax[0].legend(loc='center right', shadow=True)
ax[0].set_xlim(xlims)
ax[0].set_title('Total KE Breakdown')
ax[0].set_xlabel('Time')
ax[0].set_ylabel('Energy')

ax[1].plot(time, gavgs[:,fke],  label='FKE')
ax[1].plot(time, gavgs[:,frke], label='FRKE')
ax[1].plot(time, gavgs[:,ftke], label='FTKE')
ax[1].plot(time, gavgs[:,fpke], label='FPKE')
ax[1].legend(loc='center right', shadow=True)
ax[1].set_xlim(xlims)
ax[1].set_title('Fluctuating KE Breakdown')
ax[1].set_xlabel('Time')
ax[1].set_ylabel('Energy')

ax[2].plot(time, gavgs[:,mke],  label='MKE')
ax[2].plot(time, gavgs[:,mrke], label='MRKE')
ax[2].plot(time, gavgs[:,mtke], label='MTKE')
ax[2].plot(time, gavgs[:,mpke], label='MPKE')
ax[2].legend(loc='lower right', shadow=True)
ax[2].set_xlim(xlims)
ax[2].set_title('Mean KE Breakdown')
ax[2].set_xlabel('Time')
ax[2].set_ylabel('Energy')
ax[2].set_yscale('log')

plt.tight_layout()
plt.show()
```

V. Shell Averages

Summary: Spherical averages of requested output variables. Each output variable is stored as a 1-D function of radius.

Subdirectory: Shell_Avgs

main_input prefix: shellavg

Python Class: Shell_Avgs

Additional Namelist Variables:

None

The Shell-Averaged outputs are useful for examining how quantities vary as a function of radius. They are

particularly useful for examining the distribution of energy as a function of radius, or the heat flux balance established by the system.

Examining the *main_input* file, we see that the following output values have been denoted for the Shell Averages (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity
501	Temperature Perturbation
1438	Radial Convective Heat Flux
1468	Radial Conductive Heat Flux

In the example that follows, we will plot the spherically-averaged velocity field as a function of radius, the mean temperature profile, and the radial heat flux. We begin with a preamble similar to that used for the Global Averages. Using the help function, we see that the Shell_Avgs data structure is similar to that of the G_Avgs. There are three important differences: * There is a radius attribute (necessary if we want to plot anything vs. radius) * The dimensionality of the values array has changed; radial index forms the first dimension. * The second dimension of the values array has a length of 4. In addition to the spherical mean, the 1st, 2nd and 3rd moments are stored in indices 0,1,2, and 3 respectively.

[ ]:
```
from rayleigh_diagnostics import Shell_Avgs, build_file_list
import matplotlib.pyplot as plt
import numpy

# Build a list of all files ranging from iteration 0 million to 1 million
files = build_file_list(0,1000000,path='Shell_Avgs')
a = Shell_Avgs(filename=files[0], path='')
help(a)
```

While it can be useful to look at instaneous snapshots of Shell Averages, it's often useful to examine these outputs in a time-averaged sense. Let's average of all 200 snapshots in the last file that was output. We could average over data from multiple files, but since the benchmark run achieves a nearly steady state, a single file will do in this case.

[ ]:
```
nfiles = len(files)

nr = a.nr
nq = a.nq
nmom = 4
niter = a.niter
radius = a.radius
savg=numpy.zeros((nr,nmom,nq),dtype='float64')
for i in range(niter):
    savg[:,:,:] += a.vals[:,:,:,i]
savg = savg*(1.0/niter)
```

(continues on next page)

(continued from previous page)

```
lut = a.lut
vr = lut[1]         # Radial Velocity
vtheta = lut[2]     # Theta Velocity
vphi = lut[3]       # Phi Velocity
thermal = lut[501] # Temperature


eflux = lut[1440]  # Convective Heat Flux (radial)
cflux = lut[1470]  # Conductive Heat Flux (radial)
```

Velocity vs. Radius

Next, we plot the mean velocity field, and its first moment, as a function of radius. Notice that the radial and theta velocity components have a zero spherical mean. Since we are running an incompressible model, this is a good sign!

[ ]:
```
sizetuple = (7,7)
fig, ax = plt.subplots(nrows=2, ncols =1, figsize=sizetuple)

ax[0].plot(radius,savg[:,0,vr],label=r'$v_r$')
ax[0].plot(radius,savg[:,0,vtheta], label=r'$v_\theta$')
ax[0].plot(radius,savg[:,0,vphi], label=r'$v_\phi$')
ax[0].legend(shadow=True,loc='lower right')
ax[0].set_xlabel('Radius')
ax[0].set_ylabel('Velocity')
ax[0].set_title('Spherically-Averaged Velocity Components')

ax[1].plot(radius,savg[:,1,vr],label=r'$v_r$')
ax[1].plot(radius,savg[:,1,vtheta], label=r'$v_\theta$')
ax[1].plot(radius,savg[:,1,vphi], label=r'$v_\phi$')
ax[1].legend(shadow=True,loc='upper left')
ax[1].set_xlabel('Radius')
ax[1].set_ylabel('Velocity')
ax[1].set_title('Velocity Components: First Spherical Moment')


plt.tight_layout()
plt.show()
```

Radial Temperature Profile

We might also look at temperature ...

[ ]:
```
fig, ax = plt.subplots()

ax.plot(radius,savg[:,0,thermal],label='Temperature (mean)')
ax.plot(radius,savg[:,1,thermal]*10, label='Temperature (standard dev.)')
ax.legend(shadow=True,loc='upper right')
ax.set_xlabel('Radius')
ax.set_ylabel('Temperature')
ax.set_title('Radial Temperature Profile')


plt.show()
```

Heat Flux Contributions

We can also examine the balance between convective and conductive heat flux. In this case, before plotting these quantities as a function of radius, we normalize them by the surface area of the sphere to form a luminosity.

[ ]:
```
fpr=4.0*numpy.pi*radius*radius
elum = savg[:,0,eflux]*fpr
clum = savg[:,0,cflux]*fpr
tlum = elum+clum
fig, ax = plt.subplots()
ax.plot(radius,elum,label='Convection')
ax.plot(radius,clum, label='Conduction')
ax.plot(radius,tlum, label='Total')
ax.set_title('Flux Balance')
ax.set_ylabel(r'Energy Flux ($\times 4\pi r^2$)')
ax.set_xlabel('Radius')
ax.legend(shadow=True)
plt.tight_layout()
plt.show()
```

VI. Azimuthal Averages

Summary: Azimuthal averages of requested output variables. Each output variable is stored as a 2-D function of radius and latitude.

Subdirectory: AZ_Avgs

main_input prefix: azavg

Python Class: AZ_Avgs

Additional Namelist Variables:

None

Azimuthally-Averaged outputs are particularly useful for examining a system's mean flows (i.e., differential rotation and meridional circulation).

Examining the *main_input* file, we see that the following output values have been denoted for the Azimuthal Averages (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity
201	Radial Mass Flux
202	Theta Mass Flux
501	Temperature Perturbation

In the example that follows, we demonstrate how to plot azimuthal averages, including how to generate streamlines of mass flux. Note that since the benchmark is Boussinesq, our velocity and mass flux fields are identical. This is not the case when running an anelastic simulation.

We begin with the usual preamble and also import two helper routines used for displaying azimuthal averages.

Examining the data structure, we see that the vals array is dimensioned to account for latitudinal variation, and that we have new attributes costheta and sintheta used for referencing locations in the theta direction.

[ ]:
```
from rayleigh_diagnostics import AZ_Avgs, build_file_list, plot_azav,␣
↪streamfunction
import matplotlib.pyplot as plt
import pylab
import numpy
#from azavg_util import *
files = build_file_list(30000,40000,path='AZ_Avgs')
az = AZ_Avgs(files[0],path='')
help(az)
```

Before creating our plots, let's time-average over the last two files that were output (thus sampling the equilibrated phase).

[ ]:
```
nfiles = len(files)
tcount=0
for i in range(nfiles):
    az=AZ_Avgs(files[i],path='')
```

(continues on next page)

(continued from previous page)

```
    if (i == 0):
        nr = az.nr
        ntheta = az.ntheta
        nq = az.nq
        azavg=numpy.zeros((ntheta,nr,nq),dtype='float64')

    for j in range(az.niter):
        azavg[:,:,:] += az.vals[:,:,:,j]
        tcount+=1
azavg = azavg*(1.0/tcount)  # Time steps were uniform for this run, so a simple␣
↪average will suffice

lut = az.lut
vr = azavg[:,:,lut[1]]
vtheta = azavg[:,:,lut[2]]
vphi = azavg[:,:,lut[3]]
rhovr = azavg[:,:,lut[201]]
rhovtheta = azavg[:,:,lut[202]]
temperature = azavg[:,:,lut[501]]
radius = az.radius
costheta = az.costheta
sintheta = az.sintheta
```

Before we render, we need to do some quick post-processing: 1. Remove the spherical mean temperature from the azimuthal average. 2. Convert v_phi into omega 3. Compute the magnitude of the mass flux vector 4. Compute stream function associated with the mass flux field

[ ]:
```
#Subtrace the ell=0 component from temperature at each radius
for i in range(nr):
    temperature[:,i]=temperature[:,i] - numpy.mean(temperature[:,i])

#Convert v_phi to an Angular velocity
omega=numpy.zeros((ntheta,nr))
for i in range(nr):
    omega[:,i]=vphi[:,i]/(radius[i]*sintheta[:])

#Generate a streamfunction from rhov_r and rhov_theta
psi = streamfunction(rhovr,rhovtheta,radius,costheta,order=0)
#contours of mass flux are overplotted on the streamfunction PSI
rhovm = numpy.sqrt(rhovr**2+rhovtheta**2)*numpy.sign(psi)
```

Finally, we render the azimuthal averages.

NOTE: If you want to save any of these figures, you can mimic the saveplot logic at the bottom of this example.

[ ]:
```
#   We do a single row of 3 images
#   Spacing is default spacing set up by subplot
figdpi=300
sizetuple=(5.5*3,3*3)


tsize = 20     # title font size
cbfsize = 10   # colorbar font size
fig, ax = plt.subplots(ncols=3,figsize=sizetuple,dpi=figdpi)
plt.rcParams.update({'font.size': 14})

#temperature
#ax1 = f1.add_subplot(1,3,1)
units = '(nondimensional)'
plot_azav(fig,ax[0],temperature,radius,costheta,sintheta,mycmap='RdYlBu_r',
↪boundsfactor = 2,
          boundstype='rms', units=units, fontsize = cbfsize)
ax[0].set_title('Temperature',fontsize=tsize)

#Differential Rotation
#ax1 = f1.add_subplot(1,3,2)
units = '(nondimensional)'
plot_azav(fig,ax[1],omega,radius,costheta,sintheta,mycmap='RdYlBu_r',
↪boundsfactor = 1.5,
          boundstype='rms', units=units, fontsize = cbfsize)
ax[1].set_title(r'$\omega$',fontsize=tsize)

#Mass Flux
#ax1 = f1.add_subplot(1,3,3)
units = '(nondimensional)'
plot_azav(fig,ax[2],psi,radius,costheta,sintheta,mycmap='RdYlBu_r',boundsfactor␣
↪= 1.5,
          boundstype='rms', units=units, fontsize = cbfsize, underlay = rhovm)
ax[2].set_title('Mass Flux',fontsize = tsize)

saveplot=False
if (saveplot):
    p.savefig(savefile)
else:
    plt.show()
```

VII. Simulation Slices

VII.1 Equatorial Slices

Summary: 2-D profiles of selected output variables in the equatorial plane.

Subdirectory: Equatorial_Slices

main_input prefix: equatorial

Python Class: Equatorial_Slices

Additional Namelist Variables:
None

The equatorial-slice output type allows us to examine how the fluid properties vary in longitude and radius.

Examining the *main_input* file, we see that the following output values have been denoted for the Equatorial Slices (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity

In the example that follows, we demonstrate how to create a 2-D plot of radial velocity in the equatorial plane (at a single time step).

We begin with the usual preamble. Examining the data structure, we see that the *vals* array is dimensioned to account for longitudinal variation, and that we have the new coordinate attribute *phi*.

[ ]:
```
from rayleigh_diagnostics import Equatorial_Slices
import numpy
import matplotlib.pyplot as plt
from matplotlib import ticker, font_manager
istring = '00040000'
es = Equatorial_Slices(istring)
tindex =1 # Grab second time index from this file
help(es)
```

[ ]:
```
################################
# Equatorial Slice
#Set up the grid

remove_mean = True # Remove the m=0 mean
nr = es.nr
nphi = es.nphi
r = es.radius/numpy.max(es.radius)
```

(continues on next page)

(continued from previous page)

```
phi = numpy.zeros(nphi+1,dtype='float64')
phi[0:nphi] = es.phi
phi[nphi] = numpy.pi*2  # For display purposes, it is best to have a redunant␣
↪data point at 0,2pi

#We need to generate a cartesian grid of x-y coordinates (both X & Y are 2-D)
radius_matrix, phi_matrix = numpy.meshgrid(r,phi)
X = radius_matrix * numpy.cos(phi_matrix)
Y = radius_matrix * numpy.sin(phi_matrix)

qindex = es.lut[1] # radial velocity
field = numpy.zeros((nphi+1,nr),dtype='float64')
field[0:nphi,:] =es.vals[:,:,qindex,tindex]
field[nphi,:] = field[0,:]  #replicate phi=0 values at phi=2pi

#remove the mean if desired (usually a good idea, but not always)
if (remove_mean):
    for i in range(nr):
        the_mean = numpy.mean(field[:,i])
        field[:,i] = field[:,i]-the_mean

#Plot
sizetuple=(8,5)
fig, ax = plt.subplots(figsize=(8,8))
tsize = 20     # title font size
cbfsize = 10   # colorbar font size
img = ax.pcolormesh(X,Y,field,cmap='jet')
ax.axis('equal')  # Ensure that x & y axis ranges have a 1:1 aspect ratio
ax.axis('off')    # Do not plot x & y axes

# Plot bounding circles
ax.plot(r[nr-1]*numpy.cos(phi), r[nr-1]*numpy.sin(phi), color='black')  # Inner␣
↪circle
ax.plot(r[0]*numpy.cos(phi), r[0]*numpy.sin(phi), color='black')  # Outer circle

ax.set_title(r'$v_r$', fontsize=20)

#colorbar ...
cbar = plt.colorbar(img,orientation='horizontal', shrink=0.5, aspect = 15, ax=ax)
cbar.set_label('nondimensional')

tick_locator = ticker.MaxNLocator(nbins=5)
cbar.locator = tick_locator
cbar.update_ticks()
cbar.ax.tick_params(labelsize=cbfsize)   #font size for the ticks
```

(continues on next page)

(continued from previous page)

```
t = cbar.ax.xaxis.label
t.set_fontsize(cbfsize)  # font size for the axis title


plt.tight_layout()
plt.show()
```

VII.2 Meridional Slices

Summary: 2-D profiles of selected output variables sampled in meridional planes.

Subdirectory: Meridional_Slices

main_input prefix: meridional

Python Class: Meridional_Slices

Additional Namelist Variables:

- meridional_indices (indicial) : indices along longitudinal grid at which to output meridional planes.
- meridional_indices_nrm (normalized) : normalized longitudinal grid coordinates at which to output

The meridional-slice output type allows us to examine how the fluid properties vary in latitude and radius.

Examining the *main_input* file, we see that the following output values have been denoted for the Meridional Slices (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity

In the example that follows, we demonstrate how to create a 2-D plot of radial velocity in a meridional plane. The procedure is similar to that used to plot an azimuthal average.

We begin with the usual preamble and import the *plot_azav* helper function. Examining the data structure, we see that it is similar to the AZ_Avgs data structure. The *vals* array possesses an extra dimension relative to its AZ_Avgs counterpart to account for the multiple longitudes that may be output, we see attributes *phi* and *phi_indices* have been added to reference the longitudinal grid.

[ ]:
```
####################################
#  Meridional Slice
from rayleigh_diagnostics import Meridional_Slices, plot_azav
import numpy
import matplotlib.pyplot as plt
from matplotlib import ticker, font_manager
# Read the data

istring = '00040000'
```

(continues on next page)

(continued from previous page)

```
ms = Meridional_Slices(istring)
tindex =1 # All example quantities were output with same cadence.  Grab second␣
↪time-index from all.
help(ms)
```

[ ]:

```
radius = ms.radius
costheta = ms.costheta
sintheta = ms.sintheta
phi_index = 0  # We only output one Meridional Slice
vr_ms = ms.vals[phi_index,:,:,ms.lut[1],tindex]
units = 'nondimensional'

# Plot
sizetuple=(8,5)
fig, ax = plt.subplots(figsize=(8,8))
tsize = 20     # title font size
cbfsize = 10   # colorbar font size
ax.axis('equal')  # Ensure that x & y axis ranges have a 1:1 aspect ratio
ax.axis('off')    # Do not plot x & y axes
plot_azav(fig,ax,vr_ms,radius,costheta,sintheta,mycmap='RdYlBu_r',boundsfactor =␣
↪4.5,
          boundstype='rms', units=units, fontsize = cbfsize)
ax.set_title('Radial Velocity',fontsize=tsize)
plt.tight_layout()
plt.show()
```

VII.3 Shell Slices

Summary: 2-D, spherical profiles of selected output variables sampled in at discrete radii.

Subdirectory: Shell_Slices

main_input prefix: shellslice

Python Class: Shell_Slices

Additional Namelist Variables:

- shellslice_levels (indicial) : indices along radial grid at which to output spherical surfaces.
- shellslice_levels_nrm (normalized) : normalized radial grid coordinates at which to output spherical surfaces.

The shell-slice output type allows us to examine how the fluid properties vary on spherical surfaces.

Examining the *main_input* file, we see that the following output values have been denoted for the Shell Slices (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity

In the example that follows, we demonstrate how to create a 2-D plot of the radial velocity on a Cartesian, lat-lon grid.

Plotting on a lat-lon grid is straightforward and illustrated below. The shell-slice data structure is also displayed via the help() function in the example below and contains information needed to define the spherical grid for plotting purposes.

[ ]:
```
#####################################
#  Shell Slice
from rayleigh_diagnostics import Shell_Slices
import numpy
import matplotlib.pyplot as plt
from matplotlib import ticker, font_manager
# Read the data

istring = '00040000'
ss = Shell_Slices(istring)
help(ss)
ntheta = ss.ntheta
nphi = ss.nphi
costheta = ss.costheta
theta = numpy.arccos(costheta)

#help(ss)
tindex =1 # All example quantities were output with same cadence.  Grab second␣
↪time-index from all.
rindex = 0 # only output one radius
sizetuple=(8,8)

vr = ss.vals[:,:,rindex,ss.lut[1],tindex]
fig, ax = plt.subplots(figsize=sizetuple)


img = plt.imshow(numpy.transpose(vr), extent=[0,360,-90,90])
ax.set_xlabel( 'Longitude')
ax.set_ylabel( 'Latitude')
ax.set_title(  'Radial Velocity')

plt.tight_layout()
plt.show()
```

By running the cell below, we can plot different output quantities on a spherical surface. In the example shown here, we plot all three velocity components (u_r, u_θ and u_ϕ) projected onto a spherical surface. For

demonstration purposes, we illustrate each velocity component using different colormaps, at different latitudinal centers of vantage point etc. (for more details see comments within cell below as well as the Jupyter notebook titled "plot_shells.ipynb").

Note that in order to successfully run the cell below, you also need to have the orthographic projection code "projection.py" within the same directory/folder as this notebook.

[ ]:
```
import numpy
import matplotlib.pyplot as plt
from matplotlib import gridspec
from rayleigh_diagnostics import Shell_Slices
from projection import plot_ortho

# This plots various data from a single shell_slice file.
# 3 diferent plots in 1 row and 3 columns are created.
# It's easy to hack this to work with multiple files

s1=Shell_Slices('00040000')
data = numpy.zeros((s1.nphi,s1.ntheta),dtype='float64')
costheta = s1.costheta
nrows=2
ncols=2
pltin = 9  # Size of each subimage in inches (will be square)


# number of rows and columns
nrow=1
ncol=3

#We use gridspec to set up a grid.  We actually have nrow*2 rows, with every␣
↪other
#row being a 'spacer' row that's 10% the height of the main rows.
#This was the simplest way I could come up with the have the color bars appear␣
↪nicely.
fig = plt.figure(constrained_layout=False, figsize=(pltin*ncol,pltin*nrow*1.1))
spec = gridspec.GridSpec(ncols=ncol, nrows=nrow*2, figure=fig, height_ratios=[1,.
↪1]*nrow, width_ratios=[1]*ncol)



plt.rcParams.update({'font.size': 16})
#quantities codes to plot -- here all three velocity components
qi =  [1,2, 3]
nm = [r'u$_r$', r'u$_\theta$', r'u$_\phi$']

qinds = [qi, qi]  # Quantity codes to plot
names = [nm, nm]  # Names for labeling
```

(continues on next page)

(continued from previous page)

```
lv = [[1]*ncol , [2]*ncol]  # Shell levels to plot (top row is level 1, bottom␣
↪row is level 2)


style1=['-','--',':']
style2=['-', '-', '-']
styles = [style1, style2]  # Line style of grid lines

gwidth1=[0.5 , 1 , 1.5]
gwidth2=[1,1,1]
gwidths = [gwidth1, gwidth2] # width of grid lines for each image (Default: True)

hwidths1=[2.5,2.5,2.5]  # Width of the horizon line or each image (Default: 2)
hwidths2=[2,2,2]
hwidths=[hwidths1,hwidths2]

cmaps1 = ["RdYlBu_r", "seismic", 'PiYG']  # A color table for each image␣
↪(Default: RdYlBu_r)
cmaps2 = ["RdYlBu_r"]*4
cmaps = [cmaps1, cmaps1]

pgrids1 = [True, True, True]
pgrids2 = [True, True, True]
pgrids  = [pgrids1, pgrids2] # Plot grids, or not for each image (Default: True)

latcens1 = [60, 45, 15]
latcens = [latcens1, latcens1]   # Latitudinal center of vantage point (Default:␣
↪45 N)

loncens1 = [0,0,0]
loncens2 = [30,30,30]  # Longitudinal center of vantage point (Default: 0)
loncens = [loncens1,loncens2]

############################################################
# If the grid is plotted, the number of latitude lines
# for the grid can be controlled via the nlats keyword.
# Default: 9
# Note that if nlats is even, the equator will not be drawn
nlats1 = [3,5,7]
nlats2 = [4,6,8]
nlats = [nlats1, nlats1]

#############################################################################
# Similarly, the nlons keyword can be used to control longitude lines
# More precisely, it controls the number of MERIDIANS (great circles) drawn
# Default:  8
```

(continues on next page)

(continued from previous page)

```
nlons1 = [4,8,12]
nlons2 = [4,12,16]
nlons = [nlons1,nlons1]

#Longitude grid-lines can be drawn in one of two ways:
# 1)  Completely to the pole (polar_style = 'polar')
# 2)  Truncated at the last line of latitue drawn (polar_style = 'truncated')
# Default:  "truncated"
pstyle1 = ['truncated', 'polar', 'truncated']
pstyle = [pstyle1, pstyle1]


#################################################################
# We can also control the way in which the image is saturated
# via the scale_type keyword.   There are three possibilities:
# 1) scale_type=['rms', a], where a is of type 'float'
#    In this instance, the image bounds are -a*rms(data), +a*rms(data)
# 2) scale_type = ['abs', a]
#    In this instance, the image bounds are -a*abs(data), +a*abs(data)
# 3) scale_type= ['force', [a,b]]
#    In this instance, the image bounds are a,b
# 4) scale_type = [None,None]
#    In this instance, the image bounds are min(image), max(image)
# Default:  [None,None]
# Note that rms and abs are taken over projected image values, not input data
# (you only see half the data in the image)

scale_type1 = [['rms',2.0 ], [None,None], ['abs', 0.5]]
scale_type2 = [['force', [-1500,1500]], ['force',[-10000,10000]], ['rms',2.5]]
scale_types = [scale_type1, scale_type1]

# Number of pixels across each projected, interpolated image
# 768 is the default and seems to do a reasonable job
nyzi = 768



for j in range(ncol):
    for i in range(nrow):

        data[:,:] = s1.vals[:,:,lv[i][j],s1.lut[qinds[i][j]],0]

        row_ind = 2*i # skip over space allowed for color bars
        col_ind = j

        print("ROW/COL: ", row_ind, col_ind)
```

(continues on next page)

(continued from previous page)

```
        ax    = fig.add_subplot(spec[row_ind,col_ind])
        cspec = spec[row_ind+1,col_ind]
        caxis=None

        plot_ortho(data,s1.costheta,fig,ax,caxis, hwidth=hwidths[i][j],␣
↪gridstyle=styles[i][j],
                   gridwidth=gwidths[i][j], nyz=nyzi, colormap=cmaps[i][j],
                   plot_grid=pgrids[i][j], latcen=latcens[i][j], loncen=␣
↪loncens[i][j],
                   pole_style=pstyle[i][j], nlats = nlats[i][j],scale_type=scale_
↪types[i][j])
        ptitle=names[i][j]+"   (r_index = "+str(lv[i][j])+")"
        ax.set_title(ptitle)


# You can save the plot as a figure
#plt.savefig('flows.pdf')
```

VIII. Spherical Harmonic Spectra

Summary: Spherical Harmonic Spectra sampled at discrete radii.

Subdirectory: Shell_Spectra

main_input prefix: shellspectra

Python Classes:

- Shell_Spectra : Complete data structure associated with Shell_Spectra outputs.
- PowerSpectrum : Reduced data structure – contains power spectrum of velocity and/or magnetic fields only.

Additional Namelist Variables:

- shellspectra_levels (indicial) : indices along radial grid at which to output spectra.
- shellspectra_levels_nrm (normalized) : normalized radial grid coordinates at which to output spectra.

The shell-spectra output type allows us to examine the spherical harmonic decomposition of output variables at discrete radii.

Examining the *main_input* file, we see that the following output values have been denoted for the Shell Spectra (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity

Spherical harmonic spectra can be read into Python using either the **Shell_Spectra** or **PowerSpectrum** classes.

The **Shell_Spectra** class provides the full complex spectra, as a function of degree ell and azimuthal order m, for each specified output variable. It possesses an attribute named *lpower* that contains the associated power for each variable, along with its m=0 contributions separated and removed.

The **Power_Spectrum** class can be used to read a Shell_Spectra file and quickly generate a velocity or magnetic-field power spectrum. For this class to work correctly, your file must contain all three components of either the velocity or magnetic field. Other variables are ignored (use Shell_Spectrum's lpower for those).

We illustrate how to use these two classes below. As usual, we call the help() function to display the docstrings that describe the different data structures embodied by each class.

[ ]:
```
import matplotlib.pyplot as plt
from matplotlib import ticker
import numpy
from rayleigh_diagnostics import Shell_Spectra, Power_Spectrum
istring = '00040000'


tind = 0
rind = 0
#help(ss)

vpower = Power_Spectrum(istring)
help(vpower)
power = vpower.power

fig, ax = plt.subplots(nrows=3, figsize=(6,6))
ax[0].plot(power[:,rind,tind,0])
ax[0].set_xlabel(r'Degree $\ell$')
ax[0].set_title('Velocity Power (total)')


ax[1].plot(power[:,rind,tind,1])
ax[1].set_xlabel(r'Degree $\ell$')
ax[1].set_title('Velocity Power (m=0)')

ax[2].plot(power[:,rind,tind,2])
ax[2].set_xlabel(r'Degree $\ell$')
ax[2].set_title('Velocity Power ( total - {m=0} )')

plt.tight_layout()
plt.show()

fig, ax = plt.subplots()
ss = Shell_Spectra(istring)
help(ss)
mmax = ss.mmax
lmax = ss.lmax
```

(continues on next page)

(continued from previous page)

```
power_spectrum = numpy.zeros((lmax+1,mmax+1),dtype='float64')

for i in range(1,4):   # i takes on values 1,2,3
    qind=ss.lut[i]
    complex_spectrum = ss.vals[:,:,rind,qind,tind]
    power_spectrum = power_spectrum+numpy.real(complex_spectrum)**2 + numpy.
↪imag(complex_spectrum)**2

power_spectrum = numpy.transpose(power_spectrum)

tiny = 1e-6
img=ax.imshow(numpy.log10(power_spectrum+tiny), origin='lower')
ax.set_ylabel('Azimuthal Wavenumber m')
ax.set_xlabel(r'Degree $\ell$')
ax.set_title('Velocity Power Spectrum')

#colorbar ...
cbar = plt.colorbar(img) # ,shrink=0.5, aspect = 15)
cbar.set_label('Log Power')

tick_locator = ticker.MaxNLocator(nbins=5)
cbar.locator = tick_locator
cbar.update_ticks()
cbar.ax.tick_params()   #font size for the ticks


plt.show()
```

IX. Point Probes

Summary: Point-wise sampling of desired output variables.

Subdirectory: Point_Probes

main_input prefix: point_probe

Python Class: Point_Probes

Additional Namelist Variables:

- point_probe_r : radial indices for point-probe output
- point_probe_theta : theta indices for point-probe output
- point_probe_phi : phi indices for point-probe output
- point_probe_r_nrm : normalized radial coordinates for point-probe output
- point_probe_theta_nrm : normalized theta coordinates for point-probe output

- point_probe_phi_nrm : normalized phi coordinates for point-probe output
- point_probe_cache_size : number of time-samples to save before accessing the disk

Point-probes allow us to sample a simulation at an arbitrary set of points. This output type serves two purposes: 1. It provides an analog to laboratory measurements where slicing and averaging are difficult, but taking high-time-cadence using (for example) thermistors is common-practice. 2. It provides an alternative method of slicing a model (for when equatorial, meridional, or shell slices do yield the desired result).

IX.1 Specifying Point-Probe Locations

Point-probe locations are indicated by specifying a grid. The user does not supply a set of ordered coordinates (r,theta,phi). Instead, the user specifies nodes on the grid using the namelist variables described above. Examples follow.

Example 1: 4-point Coarse Grid

point_probe_r_nrm = 0.25, 0.5
point_probe_theta_nrm = 0.5
point_probe_phi_nrm = 0.2, 0.8

This example would produce point probes at the four coordinates { (0.25, 0.5, 0.2), (0.25, 0.5, 0.8), (0.5, 0.5, 0.2), (0.5,0.5,0.8) } (r,theta,phi; normalized coordinates).

Example 2: "Ring" in Phi

point_probe_r_nrm = 0.5
point_probe_theta_nrm = 0.5
point_probe_phi_nrm = 0.0, -1.0

This example describes a ring in longitude, sampled at mid-shell, in the equatorial plane. We have made use of the positional range feature here by indicating normalized phi coordinates of 0.0, -1.0. Rayleigh intreprets this as an instruction to sample all phi coordinates.

** Example 3: 2-D Surface in (r,phi) **

point_probe_r_nrm = 0, -1.0
point_probe_theta_nrm = 0.25
point_probe_phi_nrm = 0, -1.0

This example uses the positional range feature along with normalized coordinates to generate a 2-D slice in r-phi at theta = 45 degrees (theta_nrm = 0.25). Using the syntax 0,-1.0 instructs *Rayleigh* to grab all r and phi coordinates.

** Example 4: 3-D Meridional "Wedges" **

point_probe_r_nrm = 0.0, -1.0
point_probe_theta_nrm = 0.0, -1.0
point_probe_phi_nrm = 0.20, -0.30, 0.7, -0.8

This example generates two 3-D wedges described by all r,theta points and all longitudes in the ranges [72 deg, 108 deg] and [252 deg, 288 deg].

IX.2 Point-Probe Caching

When performing sparse spatial sampling using point-probes, it may be desireable to output with a high-time cadence. As this may cause disk-access patterns characterized by frequent, small writes, the point-probes are programmed with a caching feature. This feature is activated by specifing the **point_probe_cache_size** variable in the output namelist.

This variable determines how many time-samples are saved in memory before a write is performed. Its default value is 1, which means that the disk is accessed with a frequency of **point_probe_frequency**. If the cache size is set to 10 (say), then samples are still peformed at **point_probe_frequency** but they are only written to disk after 10 have been collected in memory.

NOTE: Be sure that **point_probe_cache_size** divides evenly into **point_probe_nrec**.

IX.3 Example: Force-Balance with Point Probes

Our example input file specifies a coarse, six-point grid. Examining the *main_input* file, we see that all variables necessary to examine the force balance in each direction have been specified. (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity
1201	Radial Advection (v dot grad v)
1202	Theta Advection
1203	Phi Advection
1216	Buoyancy Force (ell=0 component subtracted)
1219	Radial Coriolis Force
1220	Theta Coriolis Force
1221	Phi Coriolis Force
1228	Radial Viscous Force
1229	Theta Viscous Force
1230	Phi Viscous Force

Note that the pressure force appears to be missing. This is not an oversight. The diagnostic nature of the Pressure equation in incompressible/anelastic models, coupled with the second-order Crank-Nicolson time-stepping scheme, means that the pressure field can exhibit an even/odd sawtoothing in time. The *effective* pressure force (as implemented through the Crank-Nicolson scheme) is always a weighted average over two

time steps **and is always well-resolved in time**.

When sampling at regular intervals as we have here, if we directly sample the pressure force, we will sample either the high or low end of the sawtooth envelope, and the force balance will be off by a large factor. The easiest fix is to output the velocity field and compute its time derivative. This, in tandem with the sum of all other forces, can be used to calculate the effective pressure as a post-processing step. The (undesireable) alternative is to output once every time step and compute the effective pressure using the Crank-Nicolson weighting.

We demonstrate how to compute the effective pressure force via post-processing in the example below.

[ ]:
```
from rayleigh_diagnostics import Point_Probes,  build_file_list
import numpy
from matplotlib import pyplot as plt

#Decide which direction you want to look at (set direction = {radial,theta, or␣
↪phi})
#This is used to determine the correct quantity codes below
radial = 0
theta  = 1
phi    = 2
direction=radial
# Build a list of all files ranging from iteration 0 million to 1 million
files = build_file_list(0,1000000,path='Point_Probes')
nfiles = len(files)-1


for i in range(nfiles):
    pp = Point_Probes(files[i],path='')
    if (i == 0):
        nphi = pp.nphi
        ntheta = pp.ntheta
        nr = pp.nr
        nq = pp.nq
        niter = pp.niter
        vals=numpy.zeros( (nphi,ntheta,nr,nq,niter*nfiles),dtype='float64')
        time=numpy.zeros(niter*nfiles,dtype='float64')
    vals[:,:,:,:, i*niter:(i+1)*niter] = pp.vals
    time[i*niter:(i+1)*niter]=pp.time
istring='00040000' # iteration to examine
help(pp)
#####################################################
# We choose the coordinate indices **within**
# the Point-Probe array that we want to examine
# These indices start at zero and run to n_i-1
# where n_i is the number of points sampled in
# the ith direction
```

(continues on next page)

(continued from previous page)

```
# Use help(pp) after loading the Point-Probe file
# to see the Point-Probe class structure

pind = 0           # phi-index to examine
rind = 0           # r-index to examine
tind = 0           # theta-index to examine


pp   = Point_Probes(istring)
lut  = pp.lut

nt   = pp.niter


#########################################################################
#  Grab velocity from the point probe data
u  = vals[pind,0,rind,pp.lut[1+direction],:]
dt=time[1]-time[0]


#########################################################################
# Use numpy to compute time-derivative of u
# (necessary to compute a smooth effective pressure without outputing every␣
↪timestep)

#Depending on Numpy version, gradient function takes either time (array) or dt␣
↪(scalar)
try:
    dudt = numpy.gradient(u,time)
except:
    dt = time[1]-time[0]  # Assumed to be constant...
    dudt = numpy.gradient(u,dt)



#########################################################################
#      Forces (modulo pressure)
# Note the minus sign for advection.  Advective terms are output as u dot grad u,
↪ not -u dot grad u
advec = -vals[ pind, tind, rind, lut[1201 + direction], :]
cor   = vals[ pind, tind, rind, lut[1219 + direction], :]
visc  = vals[ pind, tind, rind, lut[1228 + direction], :]
forces = visc+cor+advec
if (direction == radial):
    buoy  = vals[ pind, tind, rind, lut[1216], :]
    forces = forces+buoy
```

(continues on next page)

(continued from previous page)

```
##########################################3
# Construct effective pressure force
pres = dudt-forces
forces = forces+pres
#############################################################
# Set up the plot
yfsize='xx-large'  # size of y-axis label

ustrings = [r'u_r', r'u_\theta', r'u_\phi']
ustring=ustrings[direction]
dstring = r'$\frac{\partial '+ustring+'}{\partial t}$'
fstrings = [r'$\Sigma\,F_r$' , r'$\Sigma\,F_\theta$' , r'$\Sigma\,F_\phi$' ]
fstring = fstrings[direction]
diff_string = dstring+' - '+fstring

pstring = 'pressure'
cstring = 'coriolis'
vstring = 'viscous'
bstring = 'buoyancy'
fig, axes = plt.subplots(nrows=2, figsize=(7*2.54, 9.6))
ax0 = axes[0]
ax1 = axes[1]


########################################
# Upper: dur/dt and F_total
#mpl.rc('xtick', labelsize=20) --- still trying to understand xtick label size etc.
#mpl.rc('ytick', labelsize=20)

ax0.plot(time,forces, label = fstring)
ax0.plot(time,pres,label=pstring)
ax0.plot(time,cor,label=cstring)
ax0.plot(time,visc,label=vstring)
if (direction == radial):
    ax0.plot(time,buoy,label=bstring)
ax0.set_xlabel('Time', size=yfsize)

ax0.set_ylabel('Acceleration', size=yfsize)
ax0.set_title('Equilibration Phase',size=yfsize)
ax0.set_xlim([0,0.1])
leg0 = ax0.legend(loc='upper right', shadow=True, ncol = 1, fontsize=yfsize)

########################################
# Lower:  Numpy Gradient Approach
```

(continues on next page)

(continued from previous page)

```
ax1.plot(time,forces,label=fstring)
ax1.plot(time,pres,label=pstring)
ax1.plot(time,cor,label=cstring)
ax1.plot(time,visc,label=vstring)
if (direction == radial):
    ax1.plot(time,buoy,label=bstring)
ax1.set_title('Late Evolution',size=yfsize)
ax1.set_xlabel('Time',size=yfsize)
ax1.set_ylabel('Acceleration', size =yfsize)
ax1.set_xlim([0.2,4])
leg1 = ax1.legend(loc='upper right', shadow=True, ncol = 1, fontsize=yfsize)


plt.tight_layout()
plt.show()
```

X. Modal Outputs

Summary: Spherical Harmonic Spectral Coefficients sampled at discrete radii and degree ell.

Subdirectory: SPH_Modes

main_input prefix: sph_mode

Python Classes: SPH_Modes

Additional Namelist Variables:

- sph_mode_levels (indicial) : indices along radial grid at which to output spectral coefficients.
- sph_mode_levels_nrm (normalized) : normalized radial grid coordinates at which to output spectral coefficients.
- sph_mode_ell : Comma-separated list of spherical harmonic degree ell to output.

The Modal output type allows us to output a restricted set of complex spherical harmonic coefficients at discrete radii. For each specified ell-value, all associated azimuthal wavenumbers are output.

This output can be useful for storing high-time-cadence spectral data for a few select modes. In the example below, we illustrate how to read in this output type, and we plot the temporal variation of the real and complex components of radial velocity for mode ell = 4, m = 4.

Examining the *main_input* file, we see that the following output values have been denoted for the Shell Spectra (see *rayleigh_output_variables.pdf* for mathematical formulae):

Menu Code	Description
1	Radial Velocity
2	Theta Velocity
3	Phi Velocity

We also see that ell=2,4,8 have been selected in the *main_input* file, leading to power at the following modes:

ell-value	m-values
2	0,1,2
4	0,1,2,3,4
8	0,1,2,3,4,5,6,7,8

[ ]:
```
from rayleigh_diagnostics import SPH_Modes, build_file_list
import matplotlib.pyplot as plt
import numpy

qind = 1  # Radial velocity
rind = 0  # First radius stored in file


files = build_file_list(0,1000000,path='SPH_Modes')
nfiles = len(files)
for i in range(nfiles):
    spm = SPH_Modes(files[i],path='')
    if (i == 0):
        nell = spm.nell
        nr = spm.nr
        nq = spm.nq
        niter = spm.niter
        lvals = spm.lvals
        max_ell = numpy.max(lvals)
        nt = niter*nfiles
        vr = spm.lut[qind]
        vals=numpy.zeros( (max_ell+1,nell,nr,nq,nt),dtype='complex64')
        time=numpy.zeros(nt,dtype='float64')
    vals[:,:,:,:, i*niter:(i+1)*niter] = spm.vals
    time[i*niter:(i+1)*niter]=spm.time
help(spm)
#########################################################3
# Print some information regarding the bookkeeping
print('...........')
print(' Contents')
print('  nr = ', nr)
print('  nq = ', nq)
print('  nt = ', nt)
for i in range(nell):
    lstring=str(lvals[i])
    estring = 'Ell='+lstring+' Complex Amplitude : vals[0:'+lstring+','+str(i)+',
↪0:nr-1,0:nq-1,0:nt-1]'
    print(estring)
print(' First dimension is m-value.')
print('...........')
```

(continues on next page)

(continued from previous page)

```
#####################################
# Create a plot of the ell=4, m=4 real and imaginary amplitudes
radius = spm.radius[rind]
lfour_mfour = vals[4,1,rind,vr,:]
fig, ax = plt.subplots()
ax.plot(time,numpy.real(lfour_mfour), label='real part')
ax.plot(time,numpy.imag(lfour_mfour), label='complex part')
ax.set_xlabel('Time')
ax.set_ylabel('Amplitude')
rstring = "{0:4.2f}".format(radius)
ax.set_title(r'Radial Velocity ( $\ell=4$ , m=4, radius='+rstring+' ) ')
ax.legend(shadow=True)
ax.set_xlim([0.5,4.0])
plt.show()
```

[ ]:

1.6.3 3-D Visualization with VAPOR

Rayleigh's Spherical_3D data can be visualized using volume rendering software such as Paraview or VAPoR .

The following video walks through the process of formatting Rayleigh data for VAPoR. You can do this with your own data or with the sample data referenced in the video. That data can now be found here .

https://www.youtube.com/embed/U-SgJYoX3q8

1.6.4 Common Diagnostics

1.7 Contributing to Rayleigh

Rayleigh is a community project that lives by the participation of its members — i.e., including you! It is our goal to build an inclusive and participatory community so we are happy that you are interested in participating!

1.7.1 Getting started with git and GitHub

GitHub provides a helpful guide on the process of contributing to an open-source project here.

1.7.2 Asking and answering questions about Rayleigh

The Rayleigh community maintains an active forum hosted by CIG here.

1.7.3 Bug reports

It is a great help to the community if you report any bugs that you may find. We keep track of all open issues related to Rayleigh here.

Please follow these simple instructions before opening a new bug report:

- Do a quick search in the list of open and closed issues for a duplicate of your issue.
- Do a google search in the archived mailing list discussions for a duplicate of your issue by searching for

  ```
  your search term site:http://lists.geodynamics.org/pipermail/geodyn/
  ```

- If you did not find an answer, open a new issue and explain your problem in as much detail as possible.
- Attach as much as possible of the following information to your issue:
  - a minimal parameter file that reproduces the issue,
  - the error message you saw on your screen,
  - any information that helps us understand why you think this is a bug, and how to reproduce it.

1.7.4 Making Rayleigh better

Rayleigh is a community project, and we are encouraging all kinds of contributions. Much appreciated contributions are new examples (cookbooks, tests, or benchmarks), extended documentation (every paragraph helps), and in particular fixing typos or updating outdated documentation. Obviously, we also encourage contributions to the core functionality in any form! If you consider making a larger contribution to the core functionality, please open a new issue first, to discuss your idea with one of the maintainers. This allows us to give you early feedback and prevents you from spending much time on a project that might already be planned, or that conflicts with other plans.

To make a change to Rayleigh you should:

- Create a fork (through GitHub) of the code base.
- Create a separate branch (sometimes called a feature branch) on which you do your modifications.
- You can propose that your branch be merged into the Rayleigh code by opening a pull request. This will give others a chance to review your code.

If you want to modify the documentation and preview your changes locally, you can find instructions for compiling the documentation in the INSTALL file.

We follow the philosophy that no pull request (independent of the author) is merged without a review from one other member of the community, and approval of one of the maintainers. This applies to maintainers as

well as to first-time contributors. We know that a review can be a daunting process, but pledge to keep all comments friendly and supportive! We are as interested in making Rayleigh better as you are!

While this seems very formal, keeping all of the code review in one place makes it easier to coordinate changes to the code (and there are usually several people making changes to the code at once). Please do not hesitate to ask questions about the workflow on the mailing list if you are not sure what to do.

If you add new Fortran files or change the module structure of Rayleigh, the dependencies in the makefile have to be updated. This is done by running `make fdeps` from the main repository directory, which modifies the file `src/Makefile.fdeps`. Commit this file along with your changes. You need the `makedepf90` tool on your development machine to perform this update. `makedepf90` is available in most package managers.

This is a placeholder for a paragraph about coding conventions

If you are new to the project then we will work with you to ensure your contributions are formatted with this style, so please do not think of it as a road block if you would like to contribute some code.

1.7.5 Acknowledgment of contributions

While we are grateful for every contribution, there are also several official ways how your contribution will be acknowledged by the Rayleigh community:

- Every commit that was merged into the Rayleigh repository will make you part of the growing group of Rayleigh contributors.
- If you contributed a significant part of the manual (such as a new cookbook, benchmark, or subsection), you will be listed as one of the contributing authors of the manual.
- Regularly, the Principal Developers of Rayleigh come together and discuss based on the contributions of the last years who should be invited to join the group of Principal Developers. Criteria that *Principal Developers* should match are:
  - A profound understanding of Rayleigh's structure and vision,
  - A proven willingness to further the project's goals and help other users,
  - Significant contributions to Rayleigh (not necessarily only source code, also mailing list advice, documentation, benchmarks, tutorials),
  - Regular and active contributions to Rayleigh for more than one year, not restricted to user meetings.

  The group of current Principal Developers is listed in the AUTHORS file in the main repository.

1.7.6 License

Rayleigh is published under the GPL v3 or newer; while you will retain copyright on your contributions, all changes to the code must be provided under this common license.

1.8 Troubleshooting

If you have questions that go beyond this manual, there are a number of resources:

- For questions on the source code of Rayleigh, portability, installation, new or existing features, etc., use the Rayleigh forum at <https://community.geodynamics.org/c/rayleigh>.
- In case of more general questions about mantle convection, you can ask on the CIG mantle convection forum at <https://community.geodynamics.org/c/dynamo>.
- If you have specific questions about Rayleigh that are not suitable for public and archived forums, you can contact the primary developers as listed at <https://github.com/geodynamics/Rayleigh/#more-information>.

1.8.1 Compiling Errors

Need text here.

1.8.2 Segmentation Fault Crashes

Need test here.

1.8.3 Timestep Crashes

Need text here.

1.8.4 Numerical Ringing

Need text here.

1.9 References

1.10 Under Development

1.10.1 Arbitrary Scalar Fields

Rayleigh can solve for additional active, χ_{a_i}, (coupled to the momentum equation through buoyancy) or passive, χ_{p_i}, scalar fields (where i can range up to 50 for each type of scalar). Both types of field follow a simple advection-diffusion equation:

$$\frac{\partial \chi_{a,p_i}}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \chi_{a,p_i} = 0 \tag{1.37}$$

The number of each type of field can be set using, e.g.:

```
&physical_controls_namelist
 n_active_scalars = 2
 n_passive_scalars = 2
/
```

Other model parameters follow the same convention as temperature but using the prefix *chi_a* or *chi_p* for active and passive scalars respectively.

See *tests/chi_scalar* for example input files.

CHAPTER

TWO

CITING RAYLEIGH

We ask that you cite the appropriate references if you publish results that were obtained in some part using Rayleigh. Receiving citations for Rayleigh is important to demonstrate the relevance of our work to our funding agencies and is a matter of fairness to all the developers that have donated their effort and time to make Rayleigh what it is today.

Please cite the code as: Featherstone, Nicholas A., Edelmann, Philipp V. F., Gassmoeller, Rene, Matilsky, Loren I., Orvedahl, Ryan J., & Wilson, Cian R. (2022). Rayleigh Version 1.1.0 (v1.1.0). Zenodo. https://doi.org/10.5281/zenodo.6522806

To cite other versions of the code, please see: https://geodynamics.org/resources/rayleigh/howtocite

```
@Software{featherstone_et_al_2022,
      author = "{Featherstone}, N.~A. and {Edelmann}, P.~V.~F. and {Gassmoeller},
↪ R. and {Matilsky}, L.~I. and {Orvedahl}, R.~J. and {Wilson}, C.~R.",
      title="Rayleigh 1.1.0",
      year="2022",
      organization="",
      optkeywords="Rayleigh",
      doi="http://doi.org/10.5281/zenodo.6522806",
      opturl="https://doi.org/10.5281/zenodo.6522806"}
```

Please also cite the following references:

Featherstone, N.A.; Hindman, B.W. (2016), The spectral amplitude of stellar convection and its scaling in the high-rayleigh-number regime, *The Astrophysical Journal*, 818 (1) , 32, DOI: 10.3847/0004-637X/818/1/32

Matsui, H. et al., 2016, Performance benchmarks for a next generation numerical dynamo model, *Geochem., Geophys., Geosys.*, 17,1586 DOI: 10.1002/2015GC006159

```
@Article{,
author = "Featherstone, N.A. and Hindman, B.W.",
title="The Spectral Amplitude Of Stellar Convection And Its Scaling In The High-
↪Rayleigh-Number Regime",
year="2016",
journal="The Astrophysical Journal",
volume="818",
number="1",
pages="32",
```

(continues on next page)

(continued from previous page)

```
optkeywords="Rayleigh",
issn="1538-4357",
doi="http://doi.org/10.3847/0004-637X/818/1/32",
opturl="http://stacks.iop.org/0004-637X/818/i=1/a=32?key=
crossref.a90f82507dd0eeb7a6e7562d1e4b0210"}

@Article{Matsui_etal_2016,
author = "Matsui, H. and Heien, E. and Aubert, J. and Aurnou, J.M. and Avery, M.␣
↪and Brown, B. and Buffett, B.A. and Busse, F. and Christensen, U.R. and Davies,
↪ C.J. and Featherstone, N. and Gastine, T. and Glatzmaier, G.A. and Gubbins, D.
↪ and Guermond, J.-L. and Hayashi, Y.-Y. and Hollerbach, R. and Hwang, L.J. and␣
↪Jackson, A. and Jones, C.A. and Jiang, W. and Kellogg, L.H. and Kuang, W. and␣
↪Landeau, M. and Marti, P.H. and Olson, P. and Ribeiro, A. and Sasaki, Y. and␣
↪Schaeffer, N. and Simitev, R.D. and Sheyko, A. and Silva, L. and Stanley, S.␣
↪and Takahashi, F. and Takehiro, S.-ichi and Wicht, J. and Willis, A.P.",
title="Performance benchmarks for a next generation numerical dynamo model",
year="2016",
journal="Geochemistry, Geophysics, Geosystems",
volume="17",
number="5",
pages="1586-1607",
optkeywords="Calypso",
issn="1525-2027",
doi="http://doi.org/10.1002/2015GC006159",
opturl="http://doi.wiley.com/10.1002/2015GC006159"
}
```

Rayleigh's development is supported by the National Science Foundation through the Dynamo Working Group of the Computational Infrastructure for Geodynamics (CIG, https://geodynamics.org/groups/dynamo).

Please acknowledge CIG support in your work as follows:

> **Note:** Rayleigh is hosted and receives support from the Computational Infrastructure for Geodynamics (CIG) which is supported by the National Science Foundation awards NSF-0949446, NSF-1550901 and NSF-2149126.

2.1 Publishing

Open research statements are now a common requirement when publishing research. These support reuse, validation, and citation and often take the form of *Data availability, Data access, Code availability, Open Research*, and *Software availability* statements. We recommend depositing input files that allow your published research to be reproduced and output model data in support of your research outcomes and figures. In addition, consider depositing model files that may be reused by others.

Remember to cite software and data in your text as well as in your Data Availability or similar statement.

Files should be deposited in an approved repository.

Additional information on *Publishing <https://geodynamics.org/software/software-bp/software-publishing* is available on the CIG website.

2.1.1 Data

Input parameters

- Main_input, *_input
- Data files for custom: * profiles, * boundary conditions, * generic initial conditions, * reference states (coefficients)
- Basic simulation information e.g. grid, job

Model output

Data products/checkpoints for the cases used in your publication.

2.1.2 Repository

The Rayleigh Simulation Library (RSL), a repository for accessing published Rayleigh datasets has been established using the Open Science Framework (OSF) at the University of Colorado Boulder. For more information on this repository and preparing your datasets see the RSL home page: https://osf.io/j275z/

2.1.3 Template

> We use Rayleigh version number (Featherstone et al., XXXX; Featherstone and Hindman, 2016, Matsui et al., 2016) which is available for download through its software landing page https://geodynamics.org/resources/rayleigh or from Zenodo<insert PID>. Model data necessary to reproduce these results including <insert a description> can be downloaded from Zenodo <insert PID> (Authors, YYYY).
>
> Featherstone, N.A.; Hindman, B.W. (2016), The spectral amplitude of stellar convection and its scaling in the high-rayleigh-number regime, The Astrophysical Journal, 818 (1) , 32, DOI: 10.3847/0004-637X/818/1/32
>
> Matsui, H. et al., 2016, Performance benchmarks for a next generation numerical dynamo model, Geochem., Geophys., Geosys., 17,1586 DOI: 10.1002/2015GC006159
>
> Authors (ZZZZ), ...

Where XXXX refers to the appropriate year of the software version cited and Authors (ZZZZ) is the citation to the data.

IOP <https://publishingsupport.iopscience.iop.org/iop-publishing-standard-data-policy/> (The Astrophysical Journal) r
The data that support the findings of this study are openly available at the following URL/DOI: [insert web link or DOI to the data].

See above or https://geodynamics.org/resources/rayleigh/howtocite for the citation to the version used.

2.1.4 Published examples

https://doi.org/10.5281/zenodo.7117668

2.2 Acknowledging

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: [Gla84], [GCE+99]

Glatzmaier, G.A., 1984, Numerical simulations of stellar convective dynamos. I. the model and method, *J. Comp. Phys.*, 55(3), 461-484. ISSN 0021-9991, doi:10.1016/0021-9991(84)90033-0.

Clune, T.C., Elliott, J.R., Miesch, M.S.,Toomre, J., and Glatzmaier, G.A., 1999, Computational aspects of a code to study rotating turbulent convection in spherical shells, Parallel Comp., 25, 361-380.

CHAPTER

THREE

ACCESSING AND SHARING MODEL DATA

3.1 Accessing Available Datasets

Rayleigh model configurations and checkpoint data of several publications can be accessed here:

- https://osf.io/j275z/
- https://osf.io/qbt32/
- https://doi.org/10.5281/zenodo.7117668

A large number of Rayleigh simulations were generated as part of the INCITE project "Frontiers in Planetary and Stellar Magnetism Through High Performance Computing" supported by the Department of Energy and the Computational Infrastructure for Geodynamics. Results and publications of this project can be found here:

- https://geodynamics.org/groups/dynamo/frontiers
- https://incite.readthedocs.io/en/latest/

3.2 Sharing Your Rayleigh Data

If you want to share your Rayleigh data, please follow our established best practics:

Under construction.

CHAPTER

FOUR

RESEARCH ENABLED BY RAYLEIGH

4.1 Research Projects

Under construction.

One of the research projects using Rayleigh: https://geodynamics.org/groups/dynamo/frontiers.

4.2 Video Gallery

4.2.1 Rotating

https://www.youtube.com/embed/3iRggdo3i0I https://www.youtube.com/embed/1KArtuLYUmY https://www.youtube.com/embed/OUICRNiFhpU

4.3 Publications

A list of publications using the Rayleigh code.

4.3.1 Software Citation

List: [Fea18a], [Fea18b]

4.3.2 Publications by Year

2019

List: [BM19]

2018

List: [KMB18], [MXF+18], [OCFH18]

2017

blank

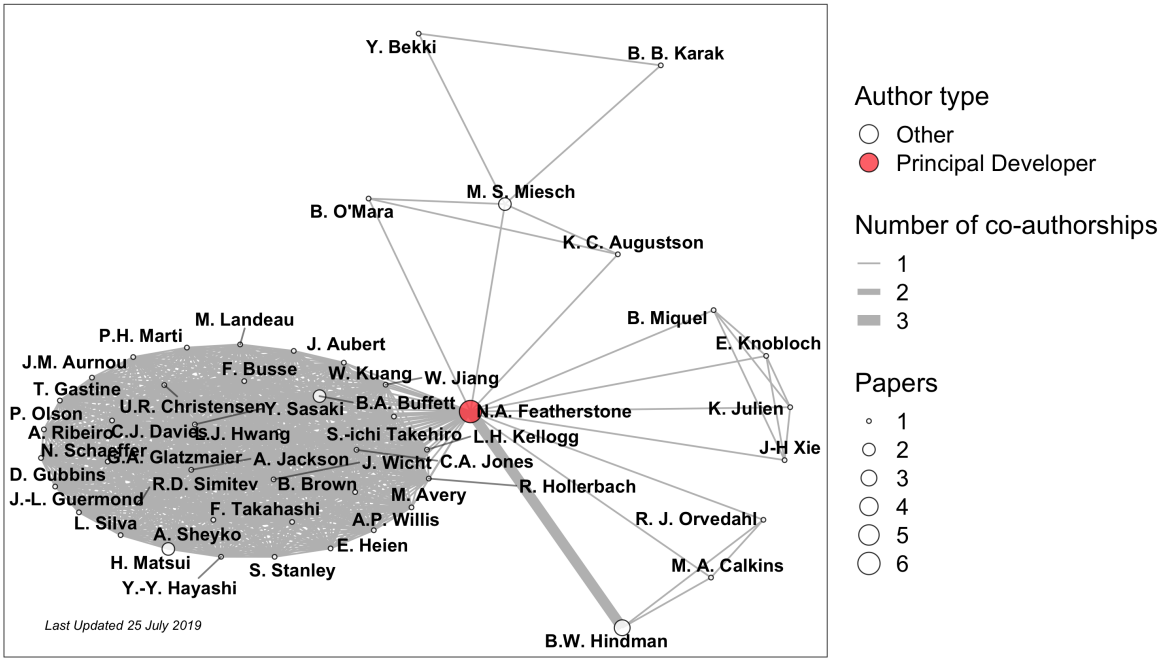
2016

List: [FH16], [MHA+16], [OMaraMFA16]

4.3.3 Co-Author Network

Network diagram illustrates the relationship between authors of the above publications.

Rayleigh Co-author Relationships



CHAPTER

FIVE

QUICK REFERENCE

5.1 Input parameters

This page provides a quick reference for all supported namelist variables in the main_input file.

5.1.1 Problemsize

This namelist is used to specify the grid.

n_r Number of radial points in model grid

rmin Radius of the inner domain boundary, $r_{\min}$

rmax Radius of the outer domain boundary, $r_{\max}$

aspect_ratio $r_{\min}/r_{\max}$

shell_depth $r_{\max} - r_{\min}$

n_theta Number of theta points in the model grid, N_θ

l_max Truncation degree $\ell_{\max}$ used in the spherical harmonic expansion

n_l $\ell_{\max} + 1$

nprow Number of MPI ranks within each row of the 2-D process grid

npcol Number of MPI ranks within each column of the 2-D process grid

ncheby Comma-separated list indicating number of Chebyshev polynomials used in each radial subdomain (e.g., 16, 32, 16). Default: n_r [single domain]

dealias_by Comma-separated list indicating number of Chebyshev modes dealiased to zero. Default is 2/3 ncheby.

domain_bounds The domain bounds defining each Chebyshev subdomain

n_uniform_domains Number of uniformly-sized Chebyshev domains spanning the depth of the shell. Default: 1

uniform_bounds When set to .true., each chebyshev subdomain will possess the same radial extent. Default: .false.

dr_weights Comma-separated list of of real-valued numbers that defines the relative weighting of grid-point spacing between subregions of uniform grid spacing when working with finite-differences and a nonuniform mesh. If left unspecified, a uniform grid spanning from rmin to rmax will be employed. dr_weights should contain the same number of elements as nr_count. Additional details may be found

here.

nr_count Comma-separated list of integer numbers that defines the number of radial points within each region of uniform grid spacing. nr_count must contain the same number of elements as dr_weights. When nr_count and dr_weights are specified, any value of n_r specified in main_input is ignored, and n_r is instead set to SUM(nr_count). Details are provided *here*.

radial_grid_file String variable indicating the name of a grid-description file. When specified, and when finite-difference mode is active, Rayleigh will use the contents of this file to define the radial grid. Instructions for generating this file in the proper format are provided *here*.

rescale_radial_grid Logical variable. When set to .true., the contents of *radial_grid_file* will be rescaled so that *rmin* and *rmax* coincide with any values specified in *main_input*. Default value = .false.

5.1.2 Numerical Controls

This namelist provides access to Rayleigh's run-time optimization options.

band_solve For use with models employing either a finite-difference scheme or at least three Chebyshev domains in radius. In those models, the rows of the normally dense matrices used in the Crank-Nicolson scheme may be rearranged into a banded or block-banded form for finite-difference and Chebyshev methods respectively. Setting this variable to .true. will perform this rearrangement, and Rayleigh will execute a band, rather than dense, solve during each timestep. Using the band-solve approach can help save memory and may yield performance gains. No benefit is gained for models using one or two Chebyshev domains. The default behavior is to use a dense solve (band_solve = .false.).

static_transpose When set to .true., buffer space used during Rayleigh's transposes is allocated once at runtime. The default behavior (static_tranpose=.false.) is to allocate and deallocate buffer space during each transpose. On some machines, avoiding this cycle of allocation/deallocation has led to minor performance improvements.

static_config When set to .true., sphericalbuffer configurations (e.g., p3a, s2b) are allocated once at runtime. The default behavior (static_config=.false.) is to save memory by deallocating memory associated with the prior configuration space following a transpose. If memory is not an issue, this may lead to minor performance improvements on some systems.

pad_alltoall When set to .true., transpose buffers are padded throughout with zeros to enforce uniform message size, and a standard alltoall is used for each transpose. The default behavior (pad_alltoall=.false.) uses alltoallv and variable message sizes. Depending on the underlying alltoall algorithms in the MPI implementation used, performance my differ between these two approaches.

chebyshev When set to .true. (the default setting), a Chebyshev collocation scheme will be employed in radius. When set to .false., a 4th-order finite-difference scheme will instead be employed for the interior points, and 2nd-order finite differences will be applied at the inner and outer radial boundaries.

5.1.3 Physical Controls

This namelist controls the physical effects used in a Rayleigh simulation.

magnetism When set to .true., the MHD approximation is employed. The default (magnetism=.false.) is to omit the effects of magnetism.

nonlinear When set to .false., all nonlinear terms are omitted in the model. The default (nonlinear=.true.) is to include those terms.

momentum_advection When set to .false., $\boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{v} = 0$. This flag is primarily for debugging purposes. The default value is .true.

inertia When set to .false., the material derivative of velocity is omitted ($\frac{D\boldsymbol{v}}{Dt} = 0$). This option is primarily intended for mantle convection models. The default value is .true.

rotation When set to .true., the Coriolis term is included in the momentum equation. The default behavior is to omit rotation in a Rayleigh model (rotation = .false.).

lorentz_forces Set this debugging/development flag to .false. to disable the Lorentz force. Default value is .true., but this flag is ignored entirely when magnetism = .false.

viscous_heating Determines whether viscous heating is included in the thermal energy equation. Default value is .true. Note that the user-supplied value of this variable is ignored entirely for Boussinesq models run with reference_type = 1. In those models, viscous_heating is set to .false.

ohmic_heating Determines whether ohmic heating is included in the thermal energy equation. Default value is .true. Note that the user-supplied value of this variable is ignored entirely for Boussinesq models run with reference_type = 1. In those models, ohmic_heating is set to .false.

advect_reference_state Determines whether the reference-state entropy is advected. The default is .true. When set to .false., the $v_r \frac{\partial \overline{S}}{\partial r}$ term is omitted in the thermal energy equation. Note that this variable has no impact on models with an adiabatic background state.

benchmark_mode When set to a positive value in the interval [1,4], an accuracy benchmark will be performed. The default is 0 (no benchmarking). Boussinesq benchmarks are peformed for values of 1 (nonmagnetic) and 2 (magnetic). Anelastic benchmarks are performed if benchmark_mode has a value of 3 (nonmagnetic) or 4 (magnetic).

benchmark_integration_interval Determines the interval (in timesteps) between successive benchmark snapshot analyses.

benchmark_report_interval Determines the interval (in timesteps) between successive benchmark report outputs. Each output contains an average over all benchmark snapshot analyses performed since the previous report.

5.1.4 Temporal Controls

This namelist controls timing, time-stepping, and checkpointing in Rayleigh.

alpha_implicit Determines the value of α used in the Crank-Nicolson semi-implicit time-stepping scheme employed for linear terms. The default value is 0.5, which ensures second-order accuracy of the algorithm. A value of 1 (0) describes a fully implicit (explicit) algorithm.

max_iterations Maximum number of timesteps for which to evolve a single instance of Rayleigh before exiting the program. Note that this value does not describe the maximum number of timesteps a model can be run for. Instead, it determines the maximum number of timesteps Rayleigh will run for during a given session (i.e. following a single call to mpiexec/mpirun). The default value is 1,000,000.

max_time_minutes Maximum walltime (in minutes) for which to run a single instance of Rayleigh before exiting. As with max_iterations, this is specific to a given Rayleigh session. Default is 10^8 minutes (essentially, unlimited).

max_simulated_time The maximum time, in simulation units, for which to evolve a Rayleigh model. Restarting a model that has already reached this limit will result in running for a single time step before exiting. The default is effectively unlimited, with a value of 10^{20}.

save_last_timestep When set to .true. (default), Rayleigh will checkpoint before exiting normally. Note that this generally occurs when the maximum time or iterations is reached. This does not apply when a job is terminated by the MPI job scheduler.

checkpoint_interval Number of iterations between successive checkpoint outputs. Default value is -1 (no checkpointing).

check_frequency (deprecated) Same as checkpoint_interval.

quicksave_interval Number of iterations between successive quicksave outputs. Default value is -1 (no quicksaves).

num_quicksaves Number of quicksave slots (i.e., rapid, rolling checkpoint folders) to use for a given simulation. Default value is 3.

quicksave_minutes Time in minutes between successive quicksaves. If this variable is set to a positive value (default is -1), the value of quicksave_interval will be ignored.

max_time_step The maximum allowed time step. This value will respected even when if the CFL constraint admits a larger time-step size. Default value is 1.0.

min_time_step The minimum allowable time step. If the CFL contraint forces a time-step size that falls below this value, Rayleigh will exit.

cflmin Used for adaptive timestep control. Rayleigh ensures that the time-step size never falls below $cflmin \times t_{CFL}$, where t_{CFL} is the minimum timestep allowed by the CFL constraint. The default value is 0.4.

clfmax Used for adaptive timestep control. Rayleigh ensures that the time-step size never exceeds $\mathrm{cflmax} \times t_{\mathrm{CFL}}$, where t_{CFL} is the minimum timestep allowed by the CFL constraint. The default value is 0.6.

new_iteration If desired, a simulation's iteration numbers may be reset upon restarting from a checkpoint. Set this value to the new iteration number to use (must be greater than zero), and the old iteration number contained in the checkpoint file will ignored. The default value is 0.

5.1.5 IO Controls

This namelist provides various options to control Rayleigh's input and output cadence and structure.

stdout_file If desired, set this variable to the name of a file to which Rayleigh's text output is redirected. This can be useful for monitoring run progress and time-step size on systems that otherwise don't produce the text output until a run has complete. The default value is 'nofile,' which indicates that Rayleigh should not redirect stdout to a file.

stdout_flush_interval Number of lines to cache before writing to the stdout_file if used. This prevents excessive disk access while a model is evolving. The default value if 50.

jobinfo_file Set this variable to the name of a file, generated during Rayleigh's initialization, that contains the values assigned to each namelist variable, along with compiler and Git hash information. The default filename is 'jobinfo.txt'

terminate_file The name of a file that, if found in the top-level simulation directory, indicates Rayleigh should terminate execution. This can be useful when trying to exit a run cleanly before the scheduled wall time runs out. The default filename is 'terminate'.

terminate_check_interval Number of iterations between successive checks for the presence of the job termination file. The default value is 50.

statusline_interval Number of iterations between successive outputs to sdout indicating time step number and size. The default value is 1, so that iteration number and time-step size are printed during every time step.

outputs_per_row Determines the number of process columns that particpate in MPI-IO during checkpointing and diagnostic outputs. Acceptable values fall in the range [1,nprow], with a default value of 1.

integer_output_digits Number of digits to use for all integer-based filenames (e.g., G_Avgs/00000001). The default value is 8.

integer_input_digits Number of digits for integer-based checkpoint names to be read during a restart. The default value is 8.

decimal_places Number of digits to use after then decimal point for those portions of Rayleigh's text output that displayed in scientific notation. The default value is 3.

5.1.6 Output

This namelist is described in extensive detail in Rayleigh/post_processing/Diagnostic_Plotting.ipynb. Please see that document for a discussion of these namelist variables and the general structure of Rayleigh's output.

5.1.7 Boundary Conditions

This namelist provides those options necessary to determine the boundary conditions employed in a Rayleigh model.

fix_tvar_top Logical flag indicating whether thermal variable (T,S) should be fixed on the upper boundary. Default = .true.

fix_tvar_bottom Logical flag indicating whether thermal variable (T,S) should be fixed on the lower boundary. Default = .true.

fix_dtdr_top Logical flag indicating whether the radial derivative of thermal variable (T,S) should be fixed on the upper boundary. Default = .false.

fix_dtdr_bottom Logical flag indicating whether the radial derivative of thermal variable (T,S) should be fixed on the lower boundary. Default = .false.

T_top Value of thermal variable (T,S) at the upper boundary. Default = 0.

T_bottom Value of thermal variable (T,S) at the lower boundary. Default = 1.

dTdr_top Value of radial derivative of thermal variable (T,S) at the upper boundary. Default = 0.

dTdr_bottom Value of radial derivative of thermal variable (T,S) at the lower boundary. Default = 0.

adjust_dTdr_top Logical flag indicating that dTdr_top should be set based on the values of heating_integral (or luminosity) and the value of dTdr_bottom. Default value is .false. When .true., this flag only has an effect when fix_dtdr_top = .true. and heating_type > 0. When active, dTdr_top is set such that the integrated flux passing through the upper boundary is equal to the sum of those due to internal heating and any flux passing through the lower boundary due to fixed dTdr_bottom.

no_slip_top When .true., a no-slip condition on the horizontal velocity field is enforced at the upper boundary. Default = .false.

no_slip_bottom When .true., a no-slip condition on the horizontal velocity field is enforced at the lower boundary. Default = .false.

stress_free_top When .true., a stress-free condition on the horizontal velocity field is enforced at the upper boundary. Default = .true.

stress_free_bottom When .true., a stress-free condition on the horizontal velocity field is enforced at the lower boundary. Default = .true.

no_slip_boundaries When .true., both no_slip_top and no_slip_bottom are set to .false. Default = .false.

strict_L_Conservation In some cases, typically rotating models employing MHD or thick shells, angular momentum can leak into/out of the domain even when using stree-free boundaries. When .true., this flag replaces the upper boundary condition with an integral constraint on the $\ell = 1$ toroidal streamfunction that enforces strict conservation of angular momentum. Note that the upper boundary is neither stress-free nor no-slip in this case. Default = .false.

T_top_file Generic-input file containing a custom, fixed (T,S) upper boundary condition.

T_bottom_file Generic-input file containing a custom, fixed (T,S) lower boundary condition.

dTdr_top_file Generic-input file containing a custom, fixed ($\partial T/\partial r$, $\partial S/\partial r$) upper boundary condition.

dTdr_bottom_file Generic-input file containing a custom, fixed ($\partial T/\partial r$, $\partial S/\partial r$) lower boundary condition.

C_top_file Generic-input file containing a custom upper boundary condition for the poloidal flux function C.

C_bottom_file Generic-input file containing a custom lower boundary condition for the poloidal flux function C.

5.1.8 Initial Conditions

All variables necessary to initialize velocity, temperature, pressure, and magnetic field are supplied here.

init_type

Integer value indicating how nonmagnetic variables should be initialized.

- type -1: Restart from a checkpoint
- type 1: Hydro Boussinesq benchmark init (Christensen et al. 2001). The temperature field is initialized with an $\ell = 4$, m=4 perturbation on top of a conductive profile. Velocity/pressure are zero.
- type 6: Hydro anelastic benchmark init (Jones et al. 2011). The entropy field is initialized with an $\ell = 19$, m=19 and $\ell = 1$, m=1 perturbation on top of a conductive profile. Velocity/pressure are zero.
- type 7: A randomized temperature/entropy field is initialized. Velocity and pressure are set to zero.
- type 8: Velocity, entropy/temperature, and pressure are initialized to zero, or if an associated filename is provided, they are initialized using the generic input interface.

magnetic_init_type

Integer value indicating how magnetic field should be initialized.

- type -1: Initialize magnetic field from a checkpoint.
- type 1: Magnetic initialization for Christensen et al. (2001), case 1. The poloidal flux function is initialized using an $\ell = 1, m = 0$ mode. THe toroidal flux function is initialized with an $\ell = 2, m = 0$ mode.
- type 7: The poloidal and toroidal flux functions are initialized to randomized values.
- type 8: The poloidal and toroidal flux functions are intialized to zero, and then if a corresponding generic input file is specified, their initial state is read from that file.

restart_iter Iteration number indicating the checkpoint to restart from when init_type and magnetic_init_type equal 1.

temp_amp Amplitude of randomized temperature/entropy perturbations to use with init_type = 7.

mag_amp Amplitude of randomized magnetic perturbations to use with magnetic_init_type = 7.

t_init_file Name of generic input file that, if init_type=8, will be used to initialize temperature/entropy.

p_init_file Name of generic input file that, if init_type=8, will be used to initialize pressure.

w_init_file Name of generic input file that, if init_type=8, will be used to initialize the poloidal stream function *W*.

z_init_file Name of generic input file that, if init_type=8, will be used to initialize the toroidal stream function *Z*.

c_init_file Name of generic input file that, if init_type=8, will be used to initialize the poloidal stream function *C*.

a_init_file Name of generic input file that, if init_type=8, will be used to initialize the toroidal stream function *A*.

rescale_velocity Logical variable indicating that the velocity field should be rescaled upon restart. Default = .false.

velocity_scale Factor by which to rescale the velocity field upon restart.

rescale_pressure Logical variable indicating that the pressure field should be rescaled upon restart. Default = .false.

pressure_scale Factor by which to rescale the pressure field upon restart.

rescale_tvar Logical variable indicating that the temperature/entropy field should be rescaled upon restart. Default = .false.

tvar_scale Factor by which to rescale the temperature/entropy field upon restart.

rescale_bfield Logical variable indicating that the magnetic field should be rescaled upon restart. Default = .false.

bfield_scale Factor by which to rescale the magnetic field upon restart.

5.1.9 Reference

This namelist provides options to control the properties of Rayleigh's background state.

reference_type

Determines the fluid approximation and background state used by Rayleigh.

- type 1: Boussinesq + nondimensional
- type 2: Anelastic + polytropic background state (dimensional)
- type 3: Anelastic + polytropic background state (non-dimensional)
- type 4: Custom reference-state (read from file)

poly_n The polytropic index used to describe the background state for reference types 2 and 3.

poly_Nrho Number of density scaleheights spanning the interval $r_{\rm min} \leq r \leq r_{\rm max}$ for reference types 2 and 3.

poly_mass Mass interior to $r_{\rm min}$, used in defining the polytropic reference state for reference types 2 and 3.

poly_rho_i Specifies the value of density at the inner boundary $r = r_{\rm min}$ for the polytropic reference states of reference types 2 and 3.

pressure_specific_heat Determines the value of the specific heat at constant pressure, $c_{\rm p}$ for reference types 2 and 3.

heating_type

Integer value that determines the form of the internal heating function $Q(r)$. The default value is 0, which indi

- type 1: $Q(r) \propto \overline{\rho}(r)\overline{T}(r)$.
- type 4: $Q(r)$ is a constant function of radius.

heating_integral Determines the heating normalization L, defined such that $L = 4\pi \int_{r_{\rm min}}^{r_{\rm max}} Q(r) r^2 dr$.

luminosity Same as heating_integral. If both are specified, the value of heating_integral will be used.

angular_velocity Determines the frame rotation rate Ω for rotating models employing reference type 2.

rayleigh_number Sets the value of the Rayleigh number Ra for reference type 1.

ekman_number Sets the value of the Ekman number Ek for reference types 1 and 3.

prandtl_number Sets the value of the Prandtl number Pr for reference types 1 and 3.

prandtl_number Sets the value of the magnetic Prandtl number Pm for reference types 1 and 3.

dissipation_number Sets the value of the dissipationg number Di for reference type 3.

modified_rayleigh_number Sets the value of the modified Rayleigh number Ra^* for reference type 3.

gravity_power Specifies the value of *n* (real number) used to determine the radial variation of gravitational acceleration g in reference type 1, where $g \propto \left(\frac{r}{r_{\max}}\right)^n$.

ra_constants Indicates the desired value of specified constant coefficients when reading the value from main_input instead of from a custom-refernce file. For use with override_constants or override_constant flags. Syntax is:

```
&Reference_Namelist
 ...
 ra_constants( 2) = 1.0
 ra_constants(10) = 14.0
 ...
/
```

with_custom_constants Comma separated list of integers indicating which constant coefficients should be read from a custom-refernce file when with_custom_reference is true.

with_custom_functions Comma separated list of integers indicating which non-constant coefficients should be read from a custom-refernce file when with_custom_reference is true.

with_custom_reference Logical flag that indicates some constant and non-constant coefficients should be read from a custom-reference file and used to overwrite those values otherwise assigned for reference_Types 1–3. Default value is .false.

custom_reference_file Name of file from which to read custom-reference-state information when using reference_type 4 or when augmenting reference types 1–3.

override_constants When true, ALL constant coefficients specified in the custom-reference file will be ignored, and those specified in main_input will be used instead. Constant coefficients not specified in main_input will be assigned a value of zero. Default value is .false.

override_constant Indicates that particular constant coefficients, rather than all, should be overridden using main_input values when using reference_type 4. Multiple constant overrides can be specified, one per line, with the syntax:

```
&Reference_Namelist
 ...
 override_constant( 2) = T
 override_constant(10) = T
 ...
/
```

5.1.10 Transport

This namelist enables control of Rayleigh's diffusivities.

{nu,kappa,eta}_type

Determines the radial profile of the associated diffusion coefficient.

- type 1 : no radial variation
- type 2 : diffusivity profile varies as ρ^n for some real number *n*.
- type 3 : diffusivity profile is read from a custom-reference-state file

{nu,kappa,eta}_top

Specifies the value of the associated diffusion coefficient at the upper boundary. This is primarily used for dime

- reference_type 1: $\nu_{\rm top} = 1$, $\kappa_{\rm top} = 1/{\rm Pr}$, $\eta_{\rm top} = 1/{\rm Pm}$
- reference_type 3: $\nu_{\rm top} = {\rm Ek}$, $\kappa_{\rm top} = {\rm Ek}/{\rm Pr}$, $\eta_{\rm top} = {\rm Ek}/{\rm Pm}$

{nu,kappa,eta}_power Denotes the value of the exponent *n* in the ρ^n variation associated with diffusion type 2.

hyperdiffusion

Set this to variable to .true. to enable hyperdiffusion. The default value is .false. When active, diffusivities are n

- $\{\nu, \kappa, \eta\} \rightarrow \{\nu, \kappa, \eta\} \left(1 + \alpha \left(\frac{\ell - 1}{\ell_{\rm max} - 1}\right)^{\beta}\right)$

hyperdiffusion_alpha Determines the value of α when hyper diffusion is active.

hyperdiffusion_beta Determines the value of β when hyper diffusion is active.

5.2 Output Quantity Codes

5.2.1 Velocity Field

v_r	1	v_r
v_θ	2	v_theta
v_ϕ	3	v_phi
v_r'	4	vp_r
v_θ'	5	vp_theta
v_ϕ'	6	vp_phi
$\overline{v_r}$	7	vm_r
$\overline{v_\theta}$	8	vm_theta
$\overline{v_\phi}$	9	vm_phi
$\frac{\partial v_r}{\partial r}$	10	dv_r_dr
$\frac{\partial v_\theta}{\partial r}$	11	dv_theta_dr
$\frac{\partial v_\phi}{\partial r}$	12	dv_phi_dr

continues on next page

Table 1 – continued from previous page

$\frac{\partial v_r'}{\partial r}$	13	dvp_r_dr
$\frac{\partial v_\theta'}{\partial r}$	14	dvp_theta_dr
$\frac{\partial v_\phi'}{\partial r}$	15	dvp_phi_dr
$\frac{\partial \overline{v_r}}{\partial r}$	16	dvm_r_dr
$\frac{\partial \overline{v_\theta}}{\partial r}$	17	dvm_theta_dr
$\frac{\partial \overline{v_\phi}}{\partial r}$	18	dvm_phi_dr
$\frac{\partial v_r}{\partial \theta}$	19	dv_r_dt
$\frac{\partial v_\theta}{\partial \theta}$	20	dv_theta_dt
$\frac{\partial v_\phi}{\partial \theta}$	21	dv_phi_dt
$\frac{\partial v_r'}{\partial \theta}$	22	dvp_r_dt
$\frac{\partial v_\theta'}{\partial \theta}$	23	dvp_theta_dt
$\frac{\partial v_\phi'}{\partial \theta}$	24	dvp_phi_dt
$\frac{\partial \overline{v_r}}{\partial \theta}$	25	dvm_r_dt
$\frac{\partial \overline{v_\theta}}{\partial \theta}$	26	dvm_theta_dt
$\frac{\partial \overline{v_\phi}}{\partial \theta}$	27	dvm_phi_dt
$\frac{\partial v_r}{\partial \phi}$	28	dv_r_dp
$\frac{\partial v_\theta}{\partial \phi}$	29	dv_theta_dp
$\frac{\partial v_\phi}{\partial \phi}$	30	dv_phi_dp
$\frac{\partial v_r'}{\partial \phi}$	31	dvp_r_dp
$\frac{\partial v_\theta'}{\partial \phi}$	32	dvp_theta_dp
$\frac{\partial v_\phi'}{\partial \phi}$	33	dvp_phi_dp
$\frac{\partial \overline{v_r}}{\partial \phi}$	34	dvm_r_dp
$\frac{\partial \overline{v_\theta}}{\partial \phi}$	35	dvm_theta_dp
$\frac{\partial \overline{v_\phi}}{\partial \phi}$	36	dvm_phi_dp
$\frac{1}{r}\frac{\partial v_r}{\partial \theta}$	37	dv_r_dtr
$\frac{1}{r}\frac{\partial v_\theta}{\partial \theta}$	38	dv_theta_dtr
$\frac{1}{r}\frac{\partial v_\phi}{\partial \theta}$	39	dv_phi_dtr
$\frac{1}{r}\frac{\partial v_r'}{\partial \theta}$	40	dvp_r_dtr
$\frac{1}{r}\frac{\partial v_r'}{\partial \theta}$	41	dvp_theta_dtr
$\frac{1}{r}\frac{\partial v_r'}{\partial \theta}$	42	dvp_phi_dtr
$\frac{1}{r}\frac{\partial \overline{v_r}}{\partial \theta}$	43	dvm_r_dtr
$\frac{1}{r}\frac{\partial \overline{v_\theta}}{\partial \theta}$	44	dvm_theta_dtr
$\frac{1}{r}\frac{\partial \overline{v_\phi}}{\partial \theta}$	45	dvm_phi_dtr
$\frac{1}{r\sin\theta}\frac{\partial v_r}{\partial \phi}$	46	dv_r_dprs
$\frac{1}{r\sin\theta}\frac{\partial v_\theta}{\partial \phi}$	47	dv_theta_dprs
$\frac{1}{r\sin\theta}\frac{\partial v_\phi}{\partial \phi}$	48	dv_phi_dprs
$\frac{1}{r\sin\theta}\frac{\partial v_r'}{\partial \phi}$	49	dvp_r_dprs
$\frac{1}{r\sin\theta}\frac{\partial v_\theta'}{\partial \phi}$	50	dvp_theta_dprs

continues on next page

Table 1 – continued from previous page

$\frac{1}{r\sin\theta}\frac{\partial v'_\phi}{\partial\phi}$	51	dvp_phi_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{v_r}}{\partial\phi}$	52	dvm_r_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{v_\theta}}{\partial\phi}$	53	dvm_theta_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{v_\phi}}{\partial\phi}$	54	dvm_phi_dprs
$\frac{\partial^2 v_r}{\partial r^2}$	55	dv_r_d2r
$\frac{\partial^2 v_\theta}{\partial r^2}$	56	dv_theta_d2r
$\frac{\partial^2 v_\phi}{\partial r^2}$	57	dv_phi_d2r
$\frac{\partial^2 v'_r}{\partial r^2}$	58	dvp_r_d2r
$\frac{\partial^2 v'_\theta}{\partial r^2}$	59	dvp_theta_d2r
$\frac{\partial^2 v'_\phi}{\partial r^2}$	60	dvp_phi_d2r
$\frac{\partial^2 \overline{v_r}}{\partial r^2}$	61	dvm_r_d2r
$\frac{\partial^2 \overline{v_\theta}}{\partial r^2}$	62	dvm_theta_d2r
$\frac{\partial^2 \overline{v_\phi}}{\partial r^2}$	63	dvm_phi_d2r
$\frac{\partial^2 v_r}{\partial \theta^2}$	64	dv_r_d2t
$\frac{\partial^2 v_\theta}{\partial \theta^2}$	65	dv_theta_d2t
$\frac{\partial^2 v_\phi}{\partial \theta^2}$	66	dv_phi_d2t
$\frac{\partial^2 v'_r}{\partial \theta^2}$	67	dvp_r_d2t
$\frac{\partial^2 v'_\theta}{\partial \theta^2}$	68	dvp_theta_d2t
$\frac{\partial^2 v'_\phi}{\partial \theta^2}$	69	dvp_phi_d2t
$\frac{\partial^2 \overline{v_r}}{\partial \theta^2}$	70	dvm_r_d2t
$\frac{\partial^2 \overline{v_\theta}}{\partial \theta^2}$	71	dvm_theta_d2t
$\frac{\partial^2 \overline{v_\phi}}{\partial \theta^2}$	72	dvm_phi_d2t
$\frac{\partial^2 v_r}{\partial \phi^2}$	73	dv_r_d2p
$\frac{\partial^2 v_\theta}{\partial \phi^2}$	74	dv_theta_d2p
$\frac{\partial^2 v_\phi}{\partial \phi^2}$	75	dv_phi_d2p
$\frac{\partial^2 v'_r}{\partial \phi^2}$	76	dvp_r_d2p
$\frac{\partial^2 v'_\theta}{\partial \phi^2}$	77	dvp_theta_d2p
$\frac{\partial^2 v'_\phi}{\partial \phi^2}$	78	dvp_phi_d2p
$\frac{\partial^2 \overline{v_r}}{\partial \phi^2}$	79	dvm_r_d2p
$\frac{\partial^2 \overline{v_\theta}}{\partial \phi^2}$	80	dvm_theta_d2p
$\frac{\partial^2 \overline{v_\phi}}{\partial \phi^2}$	81	dvm_phi_d2p
$\frac{\partial^2 v_r}{\partial r\partial \theta}$	82	dv_r_d2rt
$\frac{\partial^2 v_\theta}{\partial r\partial \theta}$	83	dv_theta_d2rt
$\frac{\partial^2 v_\phi}{\partial r\partial \theta}$	84	dv_phi_d2rt
$\frac{\partial^2 v'_r}{\partial r\partial \theta}$	85	dvp_r_d2rt
$\frac{\partial^2 v'_\theta}{\partial r\partial \theta}$	86	dvp_theta_d2rt

continues on next page

Table 1 – continued from previous page

$\frac{\partial^2 v'_\phi}{\partial r \partial \theta}$	87	dvp_phi_d2rt
$\frac{\partial^2 \overline{v_r}}{\partial r \partial \theta}$	88	dvm_r_d2rt
$\frac{\partial^2 \overline{v_\theta}}{\partial r \partial \theta}$	89	dvm_theta_d2rt
$\frac{\partial^2 \overline{v_\phi}}{\partial r \partial \theta}$	90	dvm_phi_d2rt
$\frac{\partial^2 v_r}{\partial r \partial \phi}$	91	dv_r_d2rp
$\frac{\partial^2 v_\theta}{\partial r \partial \phi}$	92	dv_theta_d2rp
$\frac{\partial^2 v_\phi}{\partial r \partial \phi}$	93	dv_phi_d2rp
$\frac{\partial^2 v'_r}{\partial r \partial \phi}$	94	dvp_r_d2rp
$\frac{\partial^2 v'_\theta}{\partial r \partial \phi}$	95	dvp_theta_d2rp
$\frac{\partial^2 v'_\phi}{\partial r \partial \phi}$	96	dvp_phi_d2rp
$\frac{\partial^2 \overline{v_r}}{\partial r \partial \phi}$	97	dvm_r_d2rp
$\frac{\partial^2 \overline{v_\theta}}{\partial r \partial \phi}$	98	dvm_theta_d2rp
$\frac{\partial^2 \overline{v_\phi}}{\partial r \partial \phi}$	99	dvm_phi_d2rp
$\frac{\partial^2 v_r}{\partial \theta \partial \phi}$	100	dv_r_d2tp
$\frac{\partial^2 v_\theta}{\partial \theta \partial \phi}$	101	dv_theta_d2tp
$\frac{\partial^2 v_\phi}{\partial \theta \partial \phi}$	102	dv_phi_d2tp
$\frac{\partial^2 v'_r}{\partial \theta \partial \phi}$	103	dvp_r_d2tp
$\frac{\partial^2 v'_\theta}{\partial \theta \partial \phi}$	104	dvp_theta_d2tp
$\frac{\partial^2 v'_\phi}{\partial \theta \partial \phi}$	105	dvp_phi_d2tp
$\frac{\partial^2 \overline{v_r}}{\partial \theta \partial \phi}$	106	dvm_r_d2tp
$\frac{\partial^2 \overline{v_\theta}}{\partial \theta \partial \phi}$	107	dvm_theta_d2tp
$\frac{\partial^2 \overline{v_\phi}}{\partial \theta \partial \phi}$	108	dvm_phi_d2tp

5.2.2 Mass Flux

$\mathrm{f}_1 v_r$	201	rhov_r
$\mathrm{f}_1 v_\theta$	202	rhov_theta
$\mathrm{f}_1 v_\phi$	203	rhov_phi
$\mathrm{f}_1 v'_r$	204	rhovp_r
$\mathrm{f}_1 v'_\theta$	205	rhovp_theta
$\mathrm{f}_1 v'_\phi$	206	rhovp_phi
$\mathrm{f}_1 \overline{v_r}$	207	rhovm_r
$\mathrm{f}_1 \overline{v_\theta}$	208	rhovm_theta
$\mathrm{f}_1 \overline{v_\phi}$	209	rhovm_phi

5.2.3 Vorticity

ω_r	301	vort_r
ω_θ	302	vort_theta
ω_ϕ	303	vort_phi
ω_r'	304	vortp_r
ω_θ'	305	vortp_theta
ω_ϕ'	306	vortp_phi
$\overline{\omega_r}$	307	vortm_r
$\overline{\omega_\theta}$	308	vortm_theta
$\overline{\omega_\phi}$	309	vortm_phi
$\boldsymbol{\omega} \cdot \boldsymbol{\omega}$	310	enstrophy
$\boldsymbol{\omega}' \cdot \overline{\boldsymbol{\omega}}$	311	enstrophy_pm
$\overline{\boldsymbol{\omega}} \cdot \overline{\boldsymbol{\omega}}$	312	enstrophy_mm
$\boldsymbol{\omega}' \cdot \boldsymbol{\omega}'$	313	enstrophy_pp
ω_r^2	314	vort_r_sq
ω_θ^2	315	vort_theta_sq
ω_ϕ^2	316	vort_phi_sq
$\omega_r'^2$	317	vortp_r_sq
$\omega_\theta'^2$	318	vortp_theta_sq
$\omega_\phi'^2$	319	vortp_phi_sq
$\overline{\omega_r}^2$	320	vortm_r_sq
$\overline{\omega_\theta}^2$	321	vortm_theta_sq
$\overline{\omega_\phi}^2$	322	vortm_phi_sq
Z	323	zstream
$v_r\omega_r$	324	kin_helicity_r
$v_\theta\omega_\theta$	325	kin_helicity_theta
$v_\phi\omega_\phi$	326	kin_helicity_phi
$v_r'\omega_r'$	327	kin_helicity_pp_r
$v_\theta'\omega_\theta'$	328	kin_helicity_pp_theta
$v_\phi'\omega_\phi'$	329	kin_helicity_pp_phi
$\overline{v_r\omega_r}$	330	kin_helicity_mm_r
$\overline{v_\theta\omega_\theta}$	331	kin_helicity_mm_theta
$\overline{v_\phi\omega_\phi}$	332	kin_helicity_mm_phi
$\overline{v_r}\omega_r'$	333	kin_helicity_mp_r
$\overline{v_\theta}\omega_\theta'$	334	kin_helicity_mp_theta
$\overline{v_\phi}\omega_\phi'$	335	kin_helicity_mp_phi
$v_r'\overline{\omega_r}$	336	kin_helicity_pm_r
$v_\theta'\overline{\omega_\theta}$	337	kin_helicity_pm_theta
$v_\phi'\overline{\omega_\phi}$	338	kin_helicity_pm_phi
$v \cdot \omega$	339	kin_helicity
$v' \cdot \omega'$	340	kin_helicity_pp
$\overline{v} \cdot \overline{\omega}$	341	kin_helicity_mm
$\overline{v} \cdot \omega'$	342	kin_helicity_mp

continues on next page

Table 2 – continued from previous page

$v' \cdot \overline{\omega}$	343	kin_helicity_pm

5.2.4 Kinetic Energy

$\frac{1}{2}\mathrm{f}_1 \boldsymbol{v}^2$	401	kinetic_energy
$\frac{1}{2}\mathrm{f}_1 v_r^2$	402	radial_ke
$\frac{1}{2}\mathrm{f}_1 v_\theta^2$	403	theta_ke
$\frac{1}{2}\mathrm{f}_1 v_\phi^2$	404	phi_ke
$\frac{1}{2}\mathrm{f}_1 \overline{\boldsymbol{v}}^2$	405	mkinetic_energy
$\frac{1}{2}\mathrm{f}_1 \overline{v_r}^2$	406	radial_mke
$\frac{1}{2}\mathrm{f}_1 \overline{v_\theta}^2$	407	theta_mke
$\frac{1}{2}\mathrm{f}_1 \overline{v_\phi}^2$	408	phi_mke
$\frac{1}{2}\mathrm{f}_1 \boldsymbol{v}'^2$	409	pkinetic_energy
$\frac{1}{2}\mathrm{f}_1 v_r'^2$	410	radial_pke
$\frac{1}{2}\mathrm{f}_1 v_\theta'^2$	411	theta_pke
$\frac{1}{2}\mathrm{f}_1 v_\phi'^2$	412	phi_pke
$\boldsymbol{v}^2$	413	vsq
$v_r{}^2$	414	radial_vsq
$v_\theta{}^2$	415	theta_vsq
$v_\phi{}^2$	416	phi_vsq
$\overline{\boldsymbol{v}}^2$	417	mvsq
$\overline{v_r}^2$	418	radial_mvsq
$\overline{v_\theta}^2$	419	theta_mvsq
$\overline{v_\phi}^2$	420	phi_mvsq
$\boldsymbol{v}'^2$	421	pvsq
$v_r'^2$	422	radial_pvsq
$v_\theta'^2$	423	theta_pvsq
$v_\phi'^2$	424	phi_pvsq

5.2.5 Thermal Variables

Θ	501	entropy
P	502	pressure
Θ'	503	entropy_p
P'	504	pressure_p
$\overline{\Theta}$	505	entropy_m
$\overline{P}$	506	pressure_m
$\frac{\partial \Theta}{\partial r}$	507	entropy_dr
$\frac{\partial P}{\partial r}$	508	pressure_dr
$\frac{\partial \Theta'}{\partial r}$	509	entropy_p_dr
$\frac{\partial P'}{\partial r}$	510	pressure_p_dr

continues on next page

Table 3 – continued from previous page

$\frac{\partial \overline{\Theta}}{\partial r}$	511	entropy_m_dr
$\frac{\partial \overline{P}}{\partial r}$	512	pressure_m_dr
$\frac{\partial \overline{\Theta}}{\partial \theta}$	513	entropy_dtheta
$\frac{\partial \overline{P}}{\partial \theta}$	514	pressure_dtheta
$\frac{\partial \Theta'}{\partial \theta}$	515	entropy_p_dtheta
$\frac{\partial P'}{\partial \theta}$	516	pressure_p_dtheta
$\frac{\partial \overline{\Theta}}{\partial \theta}$	517	entropy_m_dtheta
$\frac{\partial \overline{P}}{\partial \theta}$	518	pressure_m_dtheta
$\frac{\partial \overline{\Theta}}{\partial \phi}$	519	entropy_dphi
$\frac{\partial \overline{P}}{\partial \phi}$	520	pressure_dphi
$\frac{\partial \Theta'}{\partial \phi}$	521	entropy_p_dphi
$\frac{\partial P'}{\partial \phi}$	522	pressure_p_dphi
$\frac{\partial \overline{\Theta}}{\partial \phi}$	523	entropy_m_dphi
$\frac{\partial \overline{P}}{\partial \phi}$	524	pressure_m_dphi
$\frac{1}{r}\frac{\partial \Theta}{\partial \theta}$	525	entropy_dtr
$\frac{1}{r}\frac{\partial P}{\partial \theta}$	526	pressure_dtr
$\frac{1}{r}\frac{\partial \Theta'}{\partial \theta}$	527	entropy_p_dtr
$\frac{1}{r}\frac{\partial P'}{\partial \theta}$	528	pressure_p_dtr
$\frac{1}{r}\frac{\partial \overline{\Theta}}{\partial \theta}$	529	entropy_m_dtr
$\frac{1}{r}\frac{\partial \overline{P}}{\partial \theta}$	530	pressure_m_dtr
$\frac{1}{r\sin\theta}\frac{\partial \Theta}{\partial \phi}$	531	entropy_dprs
$\frac{1}{r\sin\theta}\frac{\partial P}{\partial \phi}$	532	pressure_dprs
$\frac{1}{r\sin\theta}\frac{\partial \Theta'}{\partial \phi}$	533	entropy_p_dprs
$\frac{1}{r\sin\theta}\frac{\partial P'}{\partial \phi}$	534	pressure_p_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{\Theta}}{\partial \phi}$	535	entropy_m_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{P}}{\partial \phi}$	536	pressure_m_dprs
$\frac{\partial^2 \Theta}{\partial r^2}$	537	entropy_d2r
$\frac{\partial^2 P}{\partial r^2}$	538	pressure_d2r
$\frac{\partial^2 \Theta'}{\partial r^2}$	539	entropy_p_d2r
$\frac{\partial^2 P'}{\partial r^2}$	540	pressure_p_d2r
$\frac{\partial^2 \overline{\Theta}}{\partial r^2}$	541	entropy_m_d2r
$\frac{\partial^2 \overline{P}}{\partial r^2}$	542	pressure_m_d2r
$\frac{\partial^2 \Theta}{\partial \theta^2}$	543	entropy_d2t
$\frac{\partial^2 P}{\partial \theta^2}$	544	pressure_d2t
$\frac{\partial^2 \Theta'}{\partial \theta^2}$	545	entropy_p_d2t
$\frac{\partial^2 P'}{\partial \theta^2}$	546	pressure_p_d2t
$\frac{\partial^2 \overline{\Theta}}{\partial \theta^2}$	547	entropy_m_d2t
$\frac{\partial^2 \overline{P}}{\partial \theta^2}$	548	pressure_m_d2t
$\frac{\partial^2 \Theta}{\partial \phi^2}$	549	entropy_d2p

continues on next page

Table 3 – continued from previous page

$\frac{\partial^2 P}{\partial \phi^2}$	550	pressure_d2p
$\frac{\partial^2 \Theta'}{\partial \phi^2}$	551	entropy_p_d2p
$\frac{\partial^2 P'}{\partial \phi^2}$	552	pressure_p_d2p
$\frac{\partial^2 \overline{\Theta}}{\partial \phi^2}$	553	entropy_m_d2p
$\frac{\partial^2 \overline{P}}{\partial \phi^2}$	554	pressure_m_d2p
$\frac{\partial^2 \Theta}{\partial r \partial \theta}$	555	entropy_d2rt
$\frac{\partial^2 P}{\partial r \partial \theta}$	556	pressure_d2rt
$\frac{\partial^2 \Theta'}{\partial r \partial \theta}$	557	entropy_p_d2rt
$\frac{\partial^2 P'}{\partial r \partial \theta}$	558	pressure_p_d2rt
$\frac{\partial^2 \overline{\Theta}}{\partial r \partial \theta}$	559	entropy_m_d2rt
$\frac{\partial^2 \overline{P}}{\partial r \partial \theta}$	560	pressure_m_d2rt
$\frac{\partial^2 \Theta}{\partial r \partial \phi}$	561	entropy_d2rp
$\frac{\partial^2 P}{\partial r \partial \phi}$	562	pressure_d2rp
$\frac{\partial^2 \Theta'}{\partial r \partial \phi}$	563	entropy_p_d2rp
$\frac{\partial^2 P'}{\partial r \partial \phi}$	564	pressure_p_d2rp
$\frac{\partial^2 \overline{\Theta}}{\partial r \partial \phi}$	565	entropy_m_d2rp
$\frac{\partial^2 \overline{P}}{\partial r \partial \phi}$	566	pressure_m_d2rp
$\frac{\partial^2 \Theta}{\partial \theta \partial \phi}$	567	entropy_d2tp
$\frac{\partial^2 P}{\partial \theta \partial \phi}$	568	pressure_d2tp
$\frac{\partial^2 \Theta'}{\partial \theta \partial \phi}$	569	entropy_p_d2tp
$\frac{\partial^2 P'}{\partial \theta \partial \phi}$	570	pressure_p_d2tp
$\frac{\partial^2 \overline{\Theta}}{\partial \theta \partial \phi}$	571	entropy_m_d2tp
$\frac{\partial^2 \overline{P}}{\partial \theta \partial \phi}$	572	pressure_m_d2tp
$\frac{\partial}{\partial r}\left(\frac{P}{\hat{\rho}}\right)$	573	rhopressure_dr
$\frac{\partial}{\partial r}\left(\frac{P'}{\hat{\rho}}\right)$	574	rhopressurep_dr
$\frac{\partial}{\partial r}\left(\frac{\overline{P}}{\hat{\rho}}\right)$	575	rhopressurem_dr

5.2.6 Thermal Energy

$\mathrm{f_1 f_4 \Theta}$	701	thermal_energy_full
$\mathrm{f_1 f_4 \Theta}$	702	thermal_energy_p
$\mathrm{f_1 f_4 \overline{\Theta}}$	703	thermal_energy_m
$c_P \hat{\rho} T$	704	enthalpy_full
$c_P \hat{\rho} T'$	705	enthalpy_p
$c_P \hat{\rho} \overline{T}$	706	enthalpy_m
$\left(\mathrm{f_1 f_4 \Theta}\right)^2$	707	thermal_energy_sq
$\left(\mathrm{f_1 f_4 \Theta}\right)^2$	708	thermal_energyp_sq
$\left(\mathrm{f_1 f_4 \overline{\Theta}}\right)^2$	709	thermal_energym_sq
$\left(c_P \hat{\rho} T\right)^2$	710	enthalpy_sq
$\left(c_P \hat{\rho} T'\right)^2$	711	enthalpyp_sq
$\left(c_P \hat{\rho} \overline{T}\right)^2$	712	enthalpym_sq

5.2.7 Magnetic Field

B_r	801	b_r
B_θ	802	b_theta
B_ϕ	803	b_phi
B'_r	804	bp_r
B'_θ	805	bp_theta
B'_ϕ	806	bp_phi
$\overline{B_r}$	807	bm_r
$\overline{B_\theta}$	808	bm_theta
$\overline{B_\phi}$	809	bm_phi
$\frac{\partial B_r}{\partial r}$	810	db_r_dr
$\frac{\partial B_\theta}{\partial r}$	811	db_theta_dr
$\frac{\partial B_\phi}{\partial r}$	812	db_phi_dr
$\frac{\partial B'_r}{\partial r}$	813	dbp_r_dr
$\frac{\partial B'_\theta}{\partial r}$	814	dbp_theta_dr
$\frac{\partial B'_\phi}{\partial r}$	815	dbp_phi_dr
$\frac{\partial \overline{B_r}}{\partial r}$	816	dbm_r_dr
$\frac{\partial \overline{B_\theta}}{\partial r}$	817	dbm_theta_dr
$\frac{\partial \overline{B_\phi}}{\partial r}$	818	dbm_phi_dr
$\frac{\partial B_r}{\partial \theta}$	819	db_r_dt
$\frac{\partial B_\theta}{\partial \theta}$	820	db_theta_dt
$\frac{\partial B_\phi}{\partial \theta}$	821	db_phi_dt
$\frac{\partial B'_r}{\partial \theta}$	822	dbp_r_dt
$\frac{\partial B'_\theta}{\partial \theta}$	823	dbp_theta_dt

continues on next page

Table 4 – continued from previous page

$\frac{\partial B'_\phi}{\partial \theta}$	824	dbp_phi_dt
$\frac{\partial \overline{B}_r}{\partial \theta}$	825	dbm_r_dt
$\frac{\partial \overline{B}_\theta}{\partial \theta}$	826	dbm_theta_dt
$\frac{\partial \overline{B}_\phi}{\partial \theta}$	827	dbm_phi_dt
$\frac{\partial B'_r}{\partial \phi}$	828	db_r_dp
$\frac{\partial B'_\theta}{\partial \phi}$	829	db_theta_dp
$\frac{\partial B'_\phi}{\partial \phi}$	830	db_phi_dp
$\frac{\partial B'_r}{\partial \phi}$	831	dbp_r_dp
$\frac{\partial B'_\theta}{\partial \phi}$	832	dbp_theta_dp
$\frac{\partial B'_\phi}{\partial \phi}$	833	dbp_phi_dp
$\frac{\partial \overline{B}_r}{\partial \phi}$	834	dbm_r_dp
$\frac{\partial \overline{B}_\theta}{\partial \phi}$	835	dbm_theta_dp
$\frac{\partial \overline{B}_\phi}{\partial \phi}$	836	dbm_phi_dp
$\frac{1}{r}\frac{\partial B_r}{\partial \theta}$	837	db_r_dtr
$\frac{1}{r}\frac{\partial B_\theta}{\partial \theta}$	838	db_theta_dtr
$\frac{1}{r}\frac{\partial B_\phi}{\partial \theta}$	839	db_phi_dtr
$\frac{1}{r}\frac{\partial B'_r}{\partial \theta}$	840	dbp_r_dtr
$\frac{1}{r}\frac{\partial B'_r}{\partial \theta}$	841	dbp_theta_dtr
$\frac{1}{r}\frac{\partial B'_r}{\partial \theta}$	842	dbp_phi_dtr
$\frac{1}{r}\frac{\partial \overline{B}_r}{\partial \theta}$	843	dbm_r_dtr
$\frac{1}{r}\frac{\partial \overline{B}_\theta}{\partial \theta}$	844	dbm_theta_dtr
$\frac{1}{r}\frac{\partial \overline{B}_\phi}{\partial \theta}$	845	dbm_phi_dtr
$\frac{1}{r\sin\theta}\frac{\partial B_r}{\partial \phi}$	846	db_r_dprs
$\frac{1}{r\sin\theta}\frac{\partial B_\theta}{\partial \phi}$	847	db_theta_dprs
$\frac{1}{r\sin\theta}\frac{\partial B_\phi}{\partial \phi}$	848	db_phi_dprs
$\frac{1}{r\sin\theta}\frac{\partial B'_r}{\partial \phi}$	849	dbp_r_dprs
$\frac{1}{r\sin\theta}\frac{\partial B'_\theta}{\partial \phi}$	850	dbp_theta_dprs
$\frac{1}{r\sin\theta}\frac{\partial B'_\phi}{\partial \phi}$	851	dbp_phi_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{B}_r}{\partial \phi}$	852	dbm_r_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{B}_\theta}{\partial \phi}$	853	dbm_theta_dprs
$\frac{1}{r\sin\theta}\frac{\partial \overline{B}_\phi}{\partial \phi}$	854	dbm_phi_dprs
$\frac{\partial^2 B_r}{\partial r^2}$	855	db_r_d2r
$\frac{\partial^2 B_\theta}{\partial r^2}$	856	db_theta_d2r
$\frac{\partial^2 B_\phi}{\partial r^2}$	857	db_phi_d2r
$\frac{\partial^2 B'_r}{\partial r^2}$	858	dbp_r_d2r
$\frac{\partial^2 B'_\theta}{\partial r^2}$	859	dbp_theta_d2r

continues on next page

Table 4 – continued from previous page

$\frac{\partial^2 B'_\phi}{\partial r^2}$	860	dbp_phi_d2r
$\frac{\partial^2 B_r}{\partial r^2}$	861	dbm_r_d2r
$\frac{\partial^2 B_\theta}{\partial r^2}$	862	dbm_theta_d2r
$\frac{\partial^2 B_\phi}{\partial r^2}$	863	dbm_phi_d2r
$\frac{\partial^2 B_r}{\partial \theta^2}$	864	db_r_d2t
$\frac{\partial^2 B_\theta}{\partial \theta^2}$	865	db_theta_d2t
$\frac{\partial^2 B_\phi}{\partial \theta^2}$	866	db_phi_d2t
$\frac{\partial^2 B'_r}{\partial \theta^2}$	867	dbp_r_d2t
$\frac{\partial^2 B'_\theta}{\partial \theta^2}$	868	dbp_theta_d2t
$\frac{\partial^2 B'_\phi}{\partial \theta^2}$	869	dbp_phi_d2t
$\frac{\partial^2 B_r}{\partial \theta^2}$	870	dbm_r_d2t
$\frac{\partial^2 B_\theta}{\partial \theta^2}$	871	dbm_theta_d2t
$\frac{\partial^2 B_\phi}{\partial \theta^2}$	872	dbm_phi_d2t
$\frac{\partial^2 B_r}{\partial \phi^2}$	873	db_r_d2p
$\frac{\partial^2 B_\theta}{\partial \phi^2}$	874	db_theta_d2p
$\frac{\partial^2 B_\phi}{\partial \phi^2}$	875	db_phi_d2p
$\frac{\partial^2 B'_r}{\partial \phi^2}$	876	dbp_r_d2p
$\frac{\partial^2 B'_\theta}{\partial \phi^2}$	877	dbp_theta_d2p
$\frac{\partial^2 B'_\phi}{\partial \phi^2}$	878	dbp_phi_d2p
$\frac{\partial^2 B_r}{\partial \phi^2}$	879	dbm_r_d2p
$\frac{\partial^2 B_\theta}{\partial \phi^2}$	880	dbm_theta_d2p
$\frac{\partial^2 B_\phi}{\partial \phi^2}$	881	dbm_phi_d2p
$\frac{\partial^2 B_r}{\partial r \partial \theta}$	882	db_r_d2rt
$\frac{\partial^2 B_\theta}{\partial r \partial \theta}$	883	db_theta_d2rt
$\frac{\partial^2 B_\phi}{\partial r \partial \theta}$	884	db_phi_d2rt
$\frac{\partial^2 B'_r}{\partial r \partial \theta}$	885	dbp_r_d2rt
$\frac{\partial^2 B'_\theta}{\partial r \partial \theta}$	886	dbp_theta_d2rt
$\frac{\partial^2 B'_\phi}{\partial r \partial \theta}$	887	dbp_phi_d2rt
$\frac{\partial^2 B_r}{\partial r \partial \theta}$	888	dbm_r_d2rt
$\frac{\partial^2 B_\theta}{\partial r \partial \theta}$	889	dbm_theta_d2rt
$\frac{\partial^2 B_\phi}{\partial r \partial \theta}$	890	dbm_phi_d2rt
$\frac{\partial^2 B_r}{\partial r \partial \phi}$	891	db_r_d2rp
$\frac{\partial^2 B_\theta}{\partial r \partial \phi}$	892	db_theta_d2rp
$\frac{\partial^2 B_\phi}{\partial r \partial \phi}$	893	db_phi_d2rp
$\frac{\partial^2 B'_r}{\partial r \partial \phi}$	894	dbp_r_d2rp
$\frac{\partial^2 B'_\theta}{\partial r \partial \phi}$	895	dbp_theta_d2rp

continues on next page

Table 4 – continued from previous page

$\frac{\partial^2 B'_\phi}{\partial r \partial \phi}$	896	dbp_phi_d2rp
$\frac{\partial^2 \overline{B}_r}{\partial r \partial \phi}$	897	dbm_r_d2rp
$\frac{\partial^2 \overline{B}_\theta}{\partial r \partial \phi}$	898	dbm_theta_d2rp
$\frac{\partial^2 \overline{B}_\phi}{\partial r \partial \phi}$	899	dbm_phi_d2rp
$\frac{\partial^2 B_r}{\partial \theta \partial \phi}$	900	db_r_d2tp
$\frac{\partial^2 B_\theta}{\partial \theta \partial \phi}$	901	db_theta_d2tp
$\frac{\partial^2 B_\phi}{\partial \theta \partial \phi}$	902	db_phi_d2tp
$\frac{\partial^2 B'_r}{\partial \theta \partial \phi}$	903	dbp_r_d2tp
$\frac{\partial^2 B'_\theta}{\partial \theta \partial \phi}$	904	dbp_theta_d2tp
$\frac{\partial^2 B'_\phi}{\partial \theta \partial \phi}$	905	dbp_phi_d2tp
$\frac{\partial^2 \overline{B}_r}{\partial \theta \partial \phi}$	906	dbm_r_d2tp
$\frac{\partial^2 \overline{B}_\theta}{\partial \theta \partial \phi}$	907	dbm_theta_d2tp
$\frac{\partial^2 \overline{B}_\phi}{\partial \theta \partial \phi}$	908	dbm_phi_d2tp

5.2.8 Current Density

$\mathcal{J}_r$	1001	j_r
$\mathcal{J}'_r$	1002	jp_r
$\overline{\mathcal{J}}_r$	1003	jm_r
$\mathcal{J}_\theta$	1004	j_theta
$\mathcal{J}'_\theta$	1005	jp_theta
$\overline{\mathcal{J}}_\theta$	1006	jm_theta
$\mathcal{J}_\phi$	1007	j_phi
$\mathcal{J}'_\phi$	1008	jp_phi
$\overline{\mathcal{J}}_\phi$	1009	jm_phi
$\boldsymbol{\mathcal{J}} \cdot \boldsymbol{\mathcal{J}}$	1010	j_sq
$\boldsymbol{\mathcal{J}'} \cdot \boldsymbol{\mathcal{J}'}$	1011	jp_sq
$\overline{\boldsymbol{\mathcal{J}}} \cdot \overline{\boldsymbol{\mathcal{J}}}$	1012	jm_sq
$\overline{\boldsymbol{\mathcal{J}}} \cdot \boldsymbol{\mathcal{J}'}$	1013	jpm_sq
$\left(\mathcal{J}_r\right)^2$	1014	j_r_sq
$\left(\mathcal{J}'_r\right)^2$	1015	jp_r_sq
$\left(\overline{\mathcal{J}}_r\right)^2$	1016	jm_r_sq
$\left(\mathcal{J}_\theta\right)^2$	1017	j_theta_sq
$\left(\mathcal{J}'_\theta\right)^2$	1018	jp_theta_sq
$\left(\overline{\mathcal{J}}_\theta\right)^2$	1019	jm_theta_sq
$\left(\mathcal{J}_\phi\right)^2$	1020	j_phi_sq
$\left(\mathcal{J}'_\phi\right)^2$	1021	jp_phi_sq
$\left(\overline{\mathcal{J}}_\phi\right)^2$	1022	jm_phi_sq

5.2.9 Magnetic Energy

$\frac{1}{2}c_4\boldsymbol{B}^2$	1101	magnetic_energy
$\frac{1}{2}c_4 B_r^2$	1102	radial_me
$\frac{1}{2}c_4 B_\theta^2$	1103	theta_me
$\frac{1}{2}c_4 B_\phi^2$	1104	phi_me
$\frac{1}{2}c_4\overline{\boldsymbol{B}}^2$	1105	mmagnetic_energy
$\frac{1}{2}c_4\overline{B_r}^2$	1106	radial_mme
$\frac{1}{2}c_4\overline{B_\theta}^2$	1107	theta_mme
$\frac{1}{2}c_4\overline{B_\phi}^2$	1108	phi_mme
$\frac{1}{2}c_4\boldsymbol{B'}^2$	1109	pmagnetic_energy
$\frac{1}{2}c_4 B_r'^2$	1110	radial_pme
$\frac{1}{2}c_4 B_\theta'^2$	1111	theta_pme
$\frac{1}{2}c_4 B_\phi'^2$	1112	phi_pme

5.2.10 Momentum Equation

$\mathrm{f}_1\left[\boldsymbol{v}\cdot\boldsymbol{\nabla v}\right]_r$	1201	v_grad_v_r
$\mathrm{f}_1\left[\boldsymbol{v}\cdot\boldsymbol{\nabla v}\right]_\theta$	1202	v_grad_v_theta
$\mathrm{f}_1\left[\boldsymbol{v}\cdot\boldsymbol{\nabla v}\right]_\phi$	1203	v_grad_v_phi
$\mathrm{f}_1\left[\boldsymbol{v'}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_r$	1204	vp_grad_vm_r
$\mathrm{f}_1\left[\boldsymbol{v'}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_\theta$	1205	vp_grad_vm_theta
$\mathrm{f}_1\left[\boldsymbol{v'}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_\phi$	1206	vp_grad_vm_phi
$\mathrm{f}_1\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla v'}\right]_r$	1207	vm_grad_vp_r
$\mathrm{f}_1\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla v'}\right]_\theta$	1208	vm_grad_vp_theta
$\mathrm{f}_1\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla v'}\right]_\phi$	1209	vm_grad_vp_phi
$\mathrm{f}_1\left[\boldsymbol{v'}\cdot\boldsymbol{\nabla v'}\right]_r$	1210	vp_grad_vp_r
$\mathrm{f}_1\left[\boldsymbol{v'}\cdot\boldsymbol{\nabla v'}\right]_\theta$	1211	vp_grad_vp_theta
$\mathrm{f}_1\left[\boldsymbol{v'}\cdot\boldsymbol{\nabla v'}\right]_\phi$	1212	vp_grad_vp_phi
$\mathrm{f}_1\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_r$	1213	vm_grad_vm_r
$\mathrm{f}_1\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_\theta$	1214	vm_grad_vm_theta
$\mathrm{f}_1\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_\phi$	1215	vm_grad_vm_phi
$c_2\mathrm{f}_2\Theta$	1216	buoyancy_force
$c_2\mathrm{f}_2\Theta'$	1217	buoyancy_pforce
$c_2\mathrm{f}_2\overline{\Theta}$	1218	buoyancy_mforce
$-c_1\mathrm{f}_1\left[\hat{\boldsymbol{z}}\times\boldsymbol{v}\right]_r$	1219	Coriolis_Force_r
$-c_1\mathrm{f}_1\left[\hat{\boldsymbol{z}}\times\boldsymbol{v}\right]_\theta$	1220	Coriolis_Force_theta
$-c_1\mathrm{f}_1\left[\hat{\boldsymbol{z}}\times\boldsymbol{v}\right]_\phi$	1221	Coriolis_Force_phi
$-c_1\mathrm{f}_1\left[\hat{\boldsymbol{z}}\times\boldsymbol{v'}\right]_r$	1222	Coriolis_pForce_r
$-c_1\mathrm{f}_1\left[\hat{\boldsymbol{z}}\times\boldsymbol{v'}\right]_\theta$	1223	Coriolis_pForce_theta
$-c_1\mathrm{f}_1\left[\hat{\boldsymbol{z}}\times\boldsymbol{v'}\right]_\phi$	1224	Coriolis_pForce_phi
$-c_1\mathrm{f}_1\left[\hat{\boldsymbol{z}}\times\overline{\boldsymbol{v}}\right]_r$	1225	Coriolis_mForce_r

continues on next page

Table 5 – continued from previous page

$-c_1 \mathrm{f}_1 \left[\hat{\boldsymbol{z}} \times \overline{\boldsymbol{v}}\right]_\theta$	1226	Coriolis_mForce_theta
$-c_1 \mathrm{f}_1 \left[\hat{\boldsymbol{z}} \times \overline{\boldsymbol{v}}\right]_\phi$	1227	Coriolis_mForce_phi
$c_5 \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}}\right]_r$	1228	viscous_Force_r
$c_5 \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}}\right]_\theta$	1229	viscous_Force_theta
$c_5 \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}}\right]_\phi$	1230	viscous_Force_phi
$c_5 \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}}'\right]_r$	1231	viscous_pForce_r
$c_5 \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}}'\right]_\theta$	1232	viscous_pForce_theta
$c_5 \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}}'\right]_\phi$	1233	viscous_pForce_phi
$c_5 \left[\boldsymbol{\nabla} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]_r$	1234	viscous_mForce_r
$c_5 \left[\boldsymbol{\nabla} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]_\theta$	1235	viscous_mForce_theta
$c_5 \left[\boldsymbol{\nabla} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]_\phi$	1236	viscous_mForce_phi
$-c_3 \mathrm{f}_1 \frac{\partial}{\partial r}\left(\frac{P}{\mathrm{f}_1}\right)$	1237	pressure_Force_r
$-c_3 \frac{1}{r}\frac{\partial P}{\partial \theta}$	1238	pressure_Force_theta
$-c_3 \frac{1}{r \sin\theta}\frac{\partial P}{\partial \phi}$	1239	pressure_Force_phi
$-c_3 \mathrm{f}_1 \frac{\partial}{\partial r}\left(\frac{P'}{\mathrm{f}_1}\right)$	1240	pressure_pForce_r
$-c_3 \frac{1}{r}\frac{\partial P'}{\partial \theta}$	1241	pressure_pForce_theta
$-c_3 \frac{1}{r \sin\theta}\frac{\partial P'}{\partial \phi}$	1242	pressure_pForce_phi
$-c_3 \mathrm{f}_1 \frac{\partial}{\partial r}\left(\frac{\overline{P}}{\mathrm{f}_1}\right)$	1243	pressure_mForce_r
$-c_3 \frac{1}{r}\frac{\partial \overline{P}}{\partial \theta}$	1244	pressure_mForce_theta
$-c_3 \frac{1}{r \sin\theta}\frac{\partial \overline{P}}{\partial \phi}$	1245	pressure_mForce_phi
$c_2 \mathrm{f}_2 \Theta_{00}$	1246	buoyancy_force_ell0
$-c_3 \mathrm{f}_1 \frac{\partial}{\partial r}\left(\frac{P_{00}}{\mathrm{f}_1}\right)$	1247	pressure_force_ell0_r
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}) \times \boldsymbol{B}\right]_r$	1248	j_cross_b_r
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}) \times \boldsymbol{B}\right]_\theta$	1249	j_cross_b_theta
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}) \times \boldsymbol{B}\right]_\phi$	1250	j_cross_b_phi
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}') \times \overline{\boldsymbol{B}}\right]_r$	1251	jp_cross_bm_r
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}') \times \overline{\boldsymbol{B}}\right]_\theta$	1252	jp_cross_bm_theta
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}') \times \overline{\boldsymbol{B}}\right]_\phi$	1253	jp_cross_bm_phi
$c_4 \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \boldsymbol{B}'\right]_r$	1254	jm_cross_bp_r
$c_4 \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \boldsymbol{B}'\right]_\theta$	1255	jm_cross_bp_theta
$c_4 \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \boldsymbol{B}'\right]_\phi$	1256	jm_cross_bp_phi
$c_4 \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \overline{\boldsymbol{B}}\right]_r$	1257	jm_cross_bm_r
$c_4 \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \overline{\boldsymbol{B}}\right]_\theta$	1258	jm_cross_bm_theta
$c_4 \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \overline{\boldsymbol{B}}\right]_\phi$	1259	jm_cross_bm_phi
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}') \times \boldsymbol{B}'\right]_r$	1260	jp_cross_bp_r
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}') \times \boldsymbol{B}'\right]_\theta$	1261	jp_cross_bp_theta
$c_4 \left[(\boldsymbol{\nabla} \times \boldsymbol{B}') \times \boldsymbol{B}'\right]_\phi$	1262	jp_cross_bp_phi

5.2.11 Thermal Energy Equation

$\mathrm{f}_1\mathrm{f}_4\boldsymbol{v}\cdot\boldsymbol{\nabla}\Theta$	1401	rhotv_grad_s
$\mathrm{f}_1\mathrm{f}_4\boldsymbol{v'}\cdot\boldsymbol{\nabla}\Theta'$	1402	rhotvp_grad_sp
$\mathrm{f}_1\mathrm{f}_4\boldsymbol{v'}\cdot\boldsymbol{\nabla}\overline{\Theta}$	1403	rhotvp_grad_sm
$\mathrm{f}_1\mathrm{f}_4\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\overline{\Theta}$	1404	rhotvm_grad_sm
$\mathrm{f}_1\mathrm{f}_4\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\Theta'$	1405	rhotvm_grad_sp
$\mathrm{f}_1\mathrm{f}_4 v_r\frac{\partial\Theta}{\partial r}$	1406	rhotvr_grad_s
$\mathrm{f}_1\mathrm{f}_4 v_r'\frac{\partial\Theta'}{\partial r}$	1407	rhotvpr_grad_sp
$\mathrm{f}_1\mathrm{f}_4 v_r'\frac{\partial\overline{\Theta}}{\partial r}$	1408	rhotvpr_grad_sm
$\mathrm{f}_1\mathrm{f}_4\overline{v_r}\frac{\partial\overline{\Theta}}{\partial r}$	1409	rhotvmr_grad_sm
$\mathrm{f}_1\mathrm{f}_4\overline{v_r}\frac{\partial\Theta'}{\partial r}$	1410	rhotvmr_grad_sp
$\mathrm{f}_1\mathrm{f}_4\frac{v_\theta}{r}\frac{\partial\Theta}{\partial\theta}$	1411	rhotvt_grad_s
$\mathrm{f}_1\mathrm{f}_4\frac{v_\theta'}{r}\frac{\partial\Theta'}{\partial\theta}$	1412	rhotvpt_grad_sp
$\mathrm{f}_1\mathrm{f}_4\frac{v_\theta'}{r}\frac{\partial\overline{\Theta}}{\partial\theta}$	1413	rhotvpt_grad_sm
$\mathrm{f}_1\mathrm{f}_4\frac{\overline{v_\theta}}{r}\frac{\partial\overline{\Theta}}{\partial\theta}$	1414	rhotvmt_grad_sm
$\mathrm{f}_1\mathrm{f}_4\frac{\overline{v_\theta}}{r}\frac{\partial\Theta'}{\partial\theta}$	1415	rhotvmt_grad_sp
$\mathrm{f}_1\mathrm{f}_4\frac{v_\phi}{r\sin\theta}\frac{\partial\Theta}{\partial\phi}$	1416	rhotvp_grad_s
$\mathrm{f}_1\mathrm{f}_4\frac{v_\phi'}{r\sin\theta}\frac{\partial\Theta'}{\partial\phi}$	1417	rhotvpp_grad_sp
$\mathrm{f}_1\mathrm{f}_4\frac{v_\phi'}{r\sin\theta}\frac{\partial\overline{\Theta}}{\partial\phi}$	1418	rhotvpp_grad_sm
$\mathrm{f}_1\mathrm{f}_4\frac{\overline{v_\phi}}{r\sin\theta}\frac{\partial\overline{\Theta}}{\partial\phi}$	1419	rhotvmp_grad_sm
$\mathrm{f}_1\mathrm{f}_4\frac{\overline{v_\phi}}{r\sin\theta}\frac{\partial\Theta'}{\partial\phi}$	1420	rhotvmp_grad_sp
$-c_6\boldsymbol{\nabla}\cdot\boldsymbol{F}_{cond}$	1421	s_diff
$-c_6\boldsymbol{\nabla}\cdot\boldsymbol{F'}_{cond}$	1422	sp_diff
$-c_6\boldsymbol{\nabla}\cdot\overline{\boldsymbol{F}}_{cond}$	1423	sm_diff
$c_6\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5\left(\frac{\partial^2\Theta}{\partial r^2}+\frac{\partial\Theta}{\partial r}\left[\frac{2}{r}+\frac{\mathrm{d}}{\mathrm{dr}}\ln\left\{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5\right\}\right]\right)$	1424	s_diff_r
$c_6\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5\left(\frac{\partial^2\Theta'}{\partial r^2}+\frac{\partial\Theta'}{\partial r}\left[\frac{2}{r}+\frac{\mathrm{d}}{\mathrm{dr}}\ln\left\{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5\right\}\right]\right)$	1425	sp_diff_r
$c_6\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5\left(\frac{\partial^2\overline{\Theta}}{\partial r^2}+\frac{\partial\overline{\Theta}}{\partial r}\left[\frac{2}{r}+\frac{\mathrm{d}}{\mathrm{dr}}\ln\left\{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5\right\}\right]\right)$	1426	sm_diff_r
$c_6\frac{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5}{r^2}\left(\frac{\partial^2\Theta}{\partial\theta^2}+\cot\theta\frac{\partial s}{\partial\theta}\right)$	1427	s_diff_theta
$c_6\frac{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5}{r^2}\left(\frac{\partial^2\Theta'}{\partial\theta^2}+\cot\theta\frac{\partial\Theta'}{\partial\theta}\right)$	1428	sp_diff_theta
$c_6\frac{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5}{r^2}\left(\frac{\partial^2\overline{\Theta}}{\partial\theta^2}+\cot\theta\frac{\partial\overline{\Theta}}{\partial\theta}\right)$	1429	sm_diff_theta
$c_6\frac{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5}{r^2\sin^2\theta}\frac{\partial^2\Theta}{\partial\phi^2}$	1430	s_diff_phi
$c_6\frac{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5}{r^2\sin^2\theta}\frac{\partial^2\Theta'}{\partial\phi^2}$	1431	sp_diff_phi
$c_6\frac{\mathrm{f}_1\mathrm{f}_4\mathrm{f}_5}{r^2\sin^2\theta}\frac{\partial^2\overline{\Theta}}{\partial\phi^2}$	1432	sm_diff_phi
$F_Q(r)$	1433	vol_heat_flux
$\mathrm{f}_6(r)$	1434	vol_heating
$c_5\Phi(r,\theta,\phi)$	1435	visc_heating
$\mathrm{f}_7c_4\left(\boldsymbol{\mathcal{J}}\cdot\boldsymbol{\mathcal{J}}\right)$	1436	ohmic_heat
$\mathrm{f}_7c_4\left(\boldsymbol{\mathcal{J}'}\cdot\boldsymbol{\mathcal{J}'}\right)$	1437	ohmic_heat_pp

continues on next page

Table 6 – continued from previous page

$\mathrm{f}_7 c_4 \left(\overline{\mathcal{J}} \cdot \overline{\mathcal{J}}\right)$	1438	ohmic_heat_pm
$\mathrm{f}_7 c_4 \left(\overline{\mathcal{J}} \cdot \mathcal{J}'\right)$	1439	ohmic_heat_mm
$\mathrm{f}_1 \mathrm{f}_4 v_r \Theta$	1440	rhot_vr_s
$\mathrm{f}_1 \mathrm{f}_4 v_r' \Theta'$	1441	rhot_vrp_sp
$\mathrm{f}_1 \mathrm{f}_4 v_r' \overline{\Theta}$	1442	rhot_vrp_sm
$\mathrm{f}_1 \mathrm{f}_4 \overline{v_r} \Theta'$	1443	rhot_vrm_sp
$\mathrm{f}_1 \mathrm{f}_4 \overline{v_r}\, \overline{\Theta}$	1444	rhot_vrm_sm
$\mathrm{f}_1 \mathrm{f}_4 v_\theta \Theta$	1445	rhot_vt_s
$\mathrm{f}_1 \mathrm{f}_4 v_\theta' \Theta'$	1446	rhot_vtp_sp
$\mathrm{f}_1 \mathrm{f}_4 v_\theta' \overline{\Theta}$	1447	rhot_vtp_sm
$\mathrm{f}_1 \mathrm{f}_4 \overline{v_\theta} \Theta'$	1448	rhot_vtm_sp
$\mathrm{f}_1 \mathrm{f}_4 \overline{v_\theta}\, \overline{\Theta}$	1449	rhot_vtm_sm
$\mathrm{f}_1 \mathrm{f}_4 v_\phi \Theta$	1450	rhot_vp_s
$\mathrm{f}_1 \mathrm{f}_4 v_\phi' \Theta'$	1451	rhot_vpp_sp
$\mathrm{f}_1 \mathrm{f}_4 v_\phi' \overline{\Theta}$	1452	rhot_vpp_sm
$\mathrm{f}_1 \mathrm{f}_4 \overline{v_\phi} \Theta'$	1453	rhot_vpm_sp
$\mathrm{f}_1 \mathrm{f}_4 \overline{v_\phi}\, \overline{\Theta}$	1454	rhot_vpm_sm
$c_P \mathrm{f}_1 v_r T$	1455	enth_flux_r
$c_P \mathrm{f}_1 v_\theta T$	1456	enth_flux_theta
$c_P \mathrm{f}_1 v_\phi T$	1457	enth_flux_phi
$c_P \mathrm{f}_1 v_r' T'$	1458	enth_flux_rpp
$c_P \mathrm{f}_1 v_\theta' T'$	1459	enth_flux_thetapp
$c_P \mathrm{f}_1 v_\phi' T'$	1460	enth_flux_phipp
$c_P \mathrm{f}_1 v_r' \overline{T}$	1461	enth_flux_rpm
$c_P \mathrm{f}_1 v_\theta' \overline{T}$	1462	enth_flux_thetapm
$c_P \mathrm{f}_1 v_\phi' \overline{T}$	1463	enth_flux_phipm
$c_P \mathrm{f}_1 \overline{v_r}\, T'$	1464	enth_flux_rmp
$c_P \mathrm{f}_1 \overline{v_\theta}\, T'$	1465	enth_flux_thetamp
$c_P \mathrm{f}_1 \overline{v_\phi}\, T'$	1466	enth_flux_phimp
$c_P \mathrm{f}_1 \overline{v_r}\, \overline{T}$	1467	enth_flux_rmm
$c_P \mathrm{f}_1 \overline{v_\theta}\, \overline{T}$	1468	enth_flux_thetamm
$c_P \mathrm{f}_1 \overline{v_\phi}\, \overline{T}$	1469	enth_flux_phimm
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{\partial \Theta}{\partial r}$	1470	cond_flux_r
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{1}{r} \frac{\partial \Theta}{\partial \theta}$	1471	cond_flux_theta
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{1}{r \sin\theta} \frac{\partial \Theta}{\partial \phi}$	1472	cond_flux_phi
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{\partial \Theta'}{\partial r}$	1473	cond_fluxp_r
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{1}{r} \frac{\partial \Theta'}{\partial \theta}$	1474	cond_fluxp_theta
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{1}{r \sin\theta} \frac{\partial \Theta'}{\partial \phi}$	1475	cond_fluxp_phi
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{\partial \overline{\Theta}}{\partial r}$	1476	cond_fluxm_r
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{1}{r} \frac{\partial \overline{\Theta}}{\partial \theta}$	1477	cond_fluxm_theta
$-c_6 \mathrm{f}_1 \mathrm{f}_4 \mathrm{f}_5 \frac{1}{r \sin\theta} \frac{\partial \overline{\Theta}}{\partial \phi}$	1478	cond_fluxm_phi
$\mathrm{f}_1 \mathrm{f}_4 v_r \mathrm{f}_{14}$	1479	ref_advec
$\mathrm{f}_1 \mathrm{f}_4 v_r' \mathrm{f}_{14}$	1480	ref_advec_p

continues on next page

Table 6 – continued from previous page

$\mathrm{f}_1\mathrm{f}_4\overline{v_r}\mathrm{f}_{14}$	1481	ref_advec_m

5.2.12 Induction Equation

$[\boldsymbol{B}\cdot\boldsymbol{\nabla}\boldsymbol{v}]_r$	1601	induct_shear_r
$-(\boldsymbol{\nabla}\cdot\boldsymbol{v})\,B_r$	1602	induct_comp_r
$-[\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{B}]_r$	1603	induct_advec_r
$[\boldsymbol{\nabla}\times(\boldsymbol{v}\times\boldsymbol{B})]_r$	1604	induct_r
$-c_7\,[\boldsymbol{\nabla}\times(\mathrm{f}_7\boldsymbol{\nabla}\times\boldsymbol{B})]_r$	1605	induct_diff_r
$[\boldsymbol{B}\cdot\boldsymbol{\nabla}\boldsymbol{v}]_\theta$	1606	induct_shear_theta
$-(\boldsymbol{\nabla}\cdot\boldsymbol{v})\,B_\theta$	1607	induct_comp_theta
$-[\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{B}]_\theta$	1608	induct_advec_theta
$[\boldsymbol{\nabla}\times(\boldsymbol{v}\times\boldsymbol{B})]_\theta$	1609	induct_theta
$-c_7\,[\boldsymbol{\nabla}\times(\mathrm{f}_7\boldsymbol{\nabla}\times\boldsymbol{B})]_\theta$	1610	induct_diff_theta
$[\boldsymbol{B}\cdot\boldsymbol{\nabla}\boldsymbol{v}]_\phi$	1611	induct_shear_phi
$-(\boldsymbol{\nabla}\cdot\boldsymbol{v})\,B_\phi$	1612	induct_comp_phi
$-[\boldsymbol{v}\cdot\boldsymbol{\nabla}\boldsymbol{B}]_\phi$	1613	induct_advec_phi
$[\boldsymbol{\nabla}\times(\boldsymbol{v}\times\boldsymbol{B})]_\phi$	1614	induct_phi
$-c_7\,[\boldsymbol{\nabla}\times(\mathrm{f}_7\boldsymbol{\nabla}\times\boldsymbol{B})]_\phi$	1615	induct_diff_phi
$\left[\overline{\boldsymbol{B}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_r$	1616	induct_shear_vmbm_r
$-\left(\boldsymbol{\nabla}\cdot\overline{\boldsymbol{v}}\right)\overline{B_r}$	1617	induct_comp_vmbm_r
$-\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{B}}\right]_r$	1618	induct_advec_vmbm_r
$\left[\boldsymbol{\nabla}\times\left(\overline{\boldsymbol{v}}\times\overline{\boldsymbol{B}}\right)\right]_r$	1619	induct_vmbm_r
$-c_7\left[\boldsymbol{\nabla}\times\left(\mathrm{f}_7\boldsymbol{\nabla}\times\overline{\boldsymbol{B}}\right)\right]_r$	1620	induct_diff_bm_r
$\left[\overline{\boldsymbol{B}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_\theta$	1621	induct_shear_vmbm_theta
$-\left(\boldsymbol{\nabla}\cdot\overline{\boldsymbol{v}}\right)\overline{B_\theta}$	1622	induct_comp_vmbm_theta
$-\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{B}}\right]_\theta$	1623	induct_advec_vmbm_theta
$\left[\boldsymbol{\nabla}\times\left(\overline{\boldsymbol{v}}\times\overline{\boldsymbol{B}}\right)\right]_\theta$	1624	induct_vmbm_theta
$-c_7\left[\boldsymbol{\nabla}\times\left(\mathrm{f}_7\boldsymbol{\nabla}\times\overline{\boldsymbol{B}}\right)\right]_\theta$	1625	induct_diff_bm_theta
$\left[\overline{\boldsymbol{B}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}\right]_\phi$	1626	induct_shear_vmbm_phi
$-\left(\boldsymbol{\nabla}\cdot\overline{\boldsymbol{v}}\right)\overline{B_\phi}$	1627	induct_comp_vmbm_phi
$-\left[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{B}}\right]_\phi$	1628	induct_advec_vmbm_phi
$\left[\boldsymbol{\nabla}\times\left(\overline{\boldsymbol{v}}\times\overline{\boldsymbol{B}}\right)\right]_\phi$	1629	induct_vmbm_phi
$-c_7\left[\boldsymbol{\nabla}\times\left(\mathrm{f}_7\boldsymbol{\nabla}\times\overline{\boldsymbol{B}}\right)\right]_\phi$	1630	induct_diff_bm_phi
$[\boldsymbol{B'}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}]_r$	1631	induct_shear_vmbp_r
$-\left(\boldsymbol{\nabla}\cdot\overline{\boldsymbol{v}}\right)B'_r$	1632	induct_comp_vmbp_r
$-[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\boldsymbol{B'}]_r$	1633	induct_advec_vmbp_r
$[\boldsymbol{\nabla}\times(\overline{\boldsymbol{v}}\times\boldsymbol{B'})]_r$	1634	induct_vmbp_r
$-c_7\,[\boldsymbol{\nabla}\times(\mathrm{f}_7\boldsymbol{\nabla}\times\boldsymbol{B'})]_r$	1635	induct_diff_bp_r
$[\boldsymbol{B'}\cdot\boldsymbol{\nabla}\overline{\boldsymbol{v}}]_\theta$	1636	induct_shear_vmbp_theta
$-\left(\boldsymbol{\nabla}\cdot\overline{\boldsymbol{v}}\right)B'_\theta$	1637	induct_comp_vmbp_theta
$-[\overline{\boldsymbol{v}}\cdot\boldsymbol{\nabla}\boldsymbol{B'}]_\theta$	1638	induct_advec_vmbp_theta

continues on next page

Table 7 – continued from previous page

$[\boldsymbol{\nabla} \times (\overline{\boldsymbol{v}} \times \boldsymbol{B'})]_\theta$	1639	induct_vmbp_theta
$-c_7 [\boldsymbol{\nabla} \times (\mathrm{f}_7 \boldsymbol{\nabla} \times \boldsymbol{B'})]_\theta$	1640	induct_diff_bp_theta
$[\boldsymbol{B'} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{v}}]_\phi$	1641	induct_shear_vmbp_phi
$-\left(\overline{\boldsymbol{\nabla}} \cdot \overline{\boldsymbol{v}}\right) B'_\phi$	1642	induct_comp_vmbp_phi
$-[\overline{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \boldsymbol{B'}]_\phi$	1643	induct_advec_vmbp_phi
$[\boldsymbol{\nabla} \times (\overline{\boldsymbol{v}} \times \boldsymbol{B'})]_\phi$	1644	induct_vmbp_phi
$-c_7 [\boldsymbol{\nabla} \times (\mathrm{f}_7 \boldsymbol{\nabla} \times \boldsymbol{B'})]_\phi$	1645	induct_diff_bp_phi
$\left[\overline{\boldsymbol{B}} \cdot \boldsymbol{\nabla} \boldsymbol{v'}\right]_r$	1646	induct_shear_vpbm_r
$-\left(\overline{\boldsymbol{\nabla}} \cdot \boldsymbol{v'}\right) \overline{B_r}$	1647	induct_comp_vpbm_r
$-\left[\boldsymbol{v'} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{B}}\right]_r$	1648	induct_advec_vpbm_r
$\left[\boldsymbol{\nabla} \times \left(\boldsymbol{v'} \times \overline{\boldsymbol{B}}\right)\right]_r$	1649	induct_vpbm_r
$\left[\overline{\boldsymbol{B}} \cdot \boldsymbol{\nabla} \boldsymbol{v'}\right]_\theta$	1650	induct_shear_vpbm_theta
$-\left(\overline{\boldsymbol{\nabla}} \cdot \boldsymbol{v'}\right) \overline{B_\theta}$	1651	induct_comp_vpbm_theta
$-\left[\boldsymbol{v'} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{B}}\right]_\theta$	1652	induct_advec_vpbm_theta
$\left[\boldsymbol{\nabla} \times \left(\boldsymbol{v'} \times \overline{\boldsymbol{B}}\right)\right]_\theta$	1653	induct_vpbm_theta
$\left[\overline{\boldsymbol{B}} \cdot \boldsymbol{\nabla} \boldsymbol{v'}\right]_\phi$	1654	induct_shear_vpbm_phi
$-\left(\overline{\boldsymbol{\nabla}} \cdot \boldsymbol{v'}\right) \overline{B_\phi}$	1655	induct_comp_vpbm_phi
$-\left[\boldsymbol{v'} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{B}}\right]_\phi$	1656	induct_advec_vpbm_phi
$\left[\boldsymbol{\nabla} \times \left(\boldsymbol{v'} \times \overline{\boldsymbol{B}}\right)\right]_\phi$	1657	induct_vpbm_phi
$[\boldsymbol{B'} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]_r$	1658	induct_shear_vpbp_r
$-\left(\boldsymbol{\nabla} \cdot \boldsymbol{v'}\right) B'_r$	1659	induct_comp_vpbp_r
$-[\boldsymbol{v'} \cdot \boldsymbol{\nabla} \boldsymbol{B'}]_r$	1660	induct_advec_vpbp_r
$[\boldsymbol{\nabla} \times (\boldsymbol{v'} \times \boldsymbol{B'})]_r$	1661	induct_vpbp_r
$[\boldsymbol{B'} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]_\theta$	1662	induct_shear_vpbp_theta
$-\left(\boldsymbol{\nabla} \cdot \boldsymbol{v'}\right) B'_\theta$	1663	induct_comp_vpbp_theta
$-[\boldsymbol{v'} \cdot \boldsymbol{\nabla} \boldsymbol{B'}]_\theta$	1664	induct_advec_vpbp_theta
$[\boldsymbol{\nabla} \times (\boldsymbol{v'} \times \boldsymbol{B'})]_\theta$	1665	induct_vpbp_theta
$[\boldsymbol{B'} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]_\phi$	1666	induct_shear_vpbp_phi
$-\left(\boldsymbol{\nabla} \cdot \boldsymbol{v'}\right) B'_\phi$	1667	induct_comp_vpbp_phi
$-[\boldsymbol{v'} \cdot \boldsymbol{\nabla} \boldsymbol{B'}]_\phi$	1668	induct_advec_vpbp_phi
$[\boldsymbol{\nabla} \times (\boldsymbol{v'} \times \boldsymbol{B'})]_\phi$	1669	induct_vpbp_phi

5.2.13 Angular Momentum Equation

$r\sin\theta \mathrm{f}_1 \left[\boldsymbol{v'} \cdot \boldsymbol{\nabla v'}\right]_\phi$	1801	samom_advec_pp
$r\sin\theta \mathrm{f}_1 \left[\overline{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{v}}\right]_\phi$	1802	samom_advec_mm
$-c_1 \mathrm{f}_1 r\sin\theta \left(\cos\theta\, v_\theta + \sin\theta\, v_r\right)$	1803	samom_coriolis
$r\sin\theta \left[\boldsymbol{\nabla} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]_\phi$	1804	samom_diffusion
$r\sin\theta c_4 \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \overline{\boldsymbol{B}}\right]_\phi$	1805	samom_lorentz_mm
$r\sin\theta c_4 \left[\left(\boldsymbol{\nabla} \times \boldsymbol{B'}\right) \times \boldsymbol{B'}\right]_\phi$	1806	samom_lorentz_pp
$\mathrm{f}_1 r\sin\theta v'_r v'_\phi$	1807	famom_fluct_r
$\mathrm{f}_1 r\sin\theta v'_\theta v'_\phi$	1808	famom_fluct_theta
$\mathrm{f}_1 r\sin\theta\, \overline{v_r}\ \overline{v_\phi}$	1809	famom_dr_r
$\mathrm{f}_1 r\sin\theta\, \overline{v_\theta}\ \overline{v_\phi}$	1810	famom_dr_theta
$\frac{c_1}{2} \mathrm{f}_1 r^2 \sin^2\theta\, \overline{v_r}$	1811	famom_mean_r
$\frac{c_1}{2} \mathrm{f}_1 r^2 \sin^2\theta\, \overline{v_\theta}$	1812	famom_mean_theta
$\mathrm{f}_1 \nu \sin\theta \left(v_\phi - r\frac{\partial\, \overline{v_\phi}}{\partial r}\right)$	1813	famom_diff_r
$\mathrm{f}_1 \nu \left(\cos\theta\, \overline{v_\phi} - \sin\theta \frac{\partial\, \overline{v_\phi}}{\partial \theta}\right)$	1814	famom_diff_theta
$-r\sin\theta c_4\, B'_r\, B'_\phi$	1815	famom_maxstr_r
$-r\sin\theta c_4\, B'_\theta\, B'_\phi$	1816	famom_maxstr_theta
$-r\sin\theta c_4\, \overline{B_r}\, \overline{B_\phi}$	1817	famom_magtor_r
$-r\sin\theta c_4\, \overline{B_\theta}\, \overline{B_\phi}$	1818	famom_magtor_theta
$\mathrm{f}_1 r\sin\theta v_\phi$	1819	amom_z
$\mathrm{f}_1 r(-\sin\theta v_\phi - \cos\phi v_\theta)$	1820	amom_x
$\mathrm{f}_1 r(-\cos\theta v_\phi + \cos\phi v_\theta)$	1821	amom_y
$\mathrm{f}_1 r\sin\theta v'_\phi$	1822	amomp_z
$\mathrm{f}_1 r(-\sin\theta v'_\phi - \cos\phi v'_\theta)$	1823	amomp_x
$\mathrm{f}_1 r(-\cos\theta v'_\phi + \cos\phi v'_\theta)$	1824	amomp_y
$\mathrm{f}_1 r\sin\theta \overline{v_\phi}$	1825	amomm_z
$\mathrm{f}_1 r(-\sin\theta \overline{v_\phi} - \cos\phi \overline{v'_\theta}$	1826	amomm_x
$\mathrm{f}_1 r(-\cos\theta \overline{v_\phi} + \cos\phi \overline{v_\theta})$	1827	amomm_y

5.2.14 Kinetic Energy Equation

$-c_3 \mathrm{f}_1 \boldsymbol{v} \cdot \boldsymbol{\nabla} \left(\frac{P}{\mathrm{f}_1}\right)$	1901	press_work
$-c_3 \mathrm{f}_1 \boldsymbol{v'} \cdot \boldsymbol{\nabla} \left(\frac{P'}{\mathrm{f}_1}\right)$	1902	press_work_pp
$-c_3 \mathrm{f}_1 \overline{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \left(\frac{\overline{P}}{\mathrm{f}_1}\right)$	1903	press_work_mm
$c_2 v_r \mathrm{f}_2 \Theta$	1904	buoy_work
$c_2 v'_r \mathrm{f}_2 \Theta'$	1905	buoy_work_pp
$c_2 \overline{v_r} \mathrm{f}_2 \overline{\Theta}$	1906	buoy_work_mm
$c_5 \boldsymbol{v} \cdot \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}}\right]$	1907	visc_work
$c_5 \boldsymbol{v'} \cdot \left[\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{D}'}\right]$	1908	visc_work_pp

continues on next page

Table 8 – continued from previous page

$c_5 \overline{\boldsymbol{v}} \cdot \left[\boldsymbol{\nabla} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]$	1909	visc_work_mm
$\mathrm{f}_1 \boldsymbol{v} \cdot [\boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{v}]$	1910	advec_work
$\mathrm{f}_1 \boldsymbol{v'} \cdot [\boldsymbol{v'} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]$	1911	advec_work_ppp
$\mathrm{f}_1 \overline{\boldsymbol{v}} \cdot [\boldsymbol{v'} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]$	1912	advec_work_mpp
$\mathrm{f}_1 \boldsymbol{v'} \cdot [\overline{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]$	1913	advec_work_pmp
$\mathrm{f}_1 \boldsymbol{v'} \cdot [\boldsymbol{v'} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{v}}]$	1914	advec_work_ppm
$\mathrm{f}_1 \overline{\boldsymbol{v}} \cdot [\overline{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{v}}]$	1915	advec_work_mmm
$c_4 \boldsymbol{v} \cdot [(\boldsymbol{\nabla} \times \boldsymbol{B}) \times \boldsymbol{B}]$	1916	mag_work
$c_4 \boldsymbol{v'} \cdot [(\boldsymbol{\nabla} \times \boldsymbol{B'}) \times \boldsymbol{B'}]$	1918	mag_work_ppp
$c_4 \overline{\boldsymbol{v}} \cdot [(\boldsymbol{\nabla} \times \boldsymbol{B'}) \times \boldsymbol{B'}]$	1919	mag_work_mpp
$c_4 \boldsymbol{v'} \cdot \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \boldsymbol{B'}\right]$	1920	mag_work_pmp
$c_4 \boldsymbol{v'} \cdot \left[(\boldsymbol{\nabla} \times \boldsymbol{B'}) \times \overline{\boldsymbol{B}}\right]$	1921	mag_work_ppm
$c_4 \overline{\boldsymbol{v}} \cdot \left[\left(\boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right) \times \overline{\boldsymbol{B}}\right]$	1922	mag_work_mmm
$\frac{1}{2} \mathrm{f}_1 v_r \, v^2$	1923	ke_flux_radial
$\frac{1}{2} \mathrm{f}_1 v_\theta \, v^2$	1924	ke_flux_theta
$\frac{1}{2} \mathrm{f}_1 v_\phi \, v^2$	1925	ke_flux_phi
$\frac{1}{2} \mathrm{f}_1 \, \overline{v_r} \, \overline{v}^2$	1926	mke_mflux_radial
$\frac{1}{2} \mathrm{f}_1 \, \overline{v_\theta} \, \overline{v}^2$	1927	mke_mflux_theta
$\frac{1}{2} \mathrm{f}_1 \, \overline{v_\phi} \, \overline{v}^2$	1928	mke_mflux_phi
$\frac{1}{2} \mathrm{f}_1 \, \overline{v_r} \, v'^2$	1929	pke_mflux_radial
$\frac{1}{2} \mathrm{f}_1 \, \overline{v_\theta} \, v'^2$	1930	pke_mflux_theta
$\frac{1}{2} \mathrm{f}_1 \, \overline{v_\phi} \, v'^2$	1931	pke_mflux_phi
$\frac{1}{2} \mathrm{f}_1 \, v'_r \, v'^2$	1932	pke_pflux_radial
$\frac{1}{2} \mathrm{f}_1 \, v'_\theta \, v'^2$	1933	pke_pflux_theta
$\frac{1}{2} \mathrm{f}_1 \, v'_\phi \, v'^2$	1934	pke_pflux_phi
$c_5 \left[\boldsymbol{v} \cdot \boldsymbol{\mathcal{D}}\right]_r$	1935	visc_flux_r
$c_5 \left[\boldsymbol{v} \cdot \boldsymbol{\mathcal{D}}\right]_\theta$	1936	visc_flux_theta
$c_5 \left[\boldsymbol{v} \cdot \boldsymbol{\mathcal{D}}\right]_\phi$	1937	visc_flux_phi
$c_5 \left[\boldsymbol{v'} \cdot \boldsymbol{\mathcal{D}'}\right]_r$	1938	visc_fluxpp_r
$c_5 \left[\boldsymbol{v'} \cdot \boldsymbol{\mathcal{D}'}\right]_\theta$	1939	visc_fluxpp_theta
$c_5 \left[\boldsymbol{v'} \cdot \boldsymbol{\mathcal{D}'}\right]_\phi$	1940	visc_fluxpp_phi
$c_5 \left[\overline{\boldsymbol{v}} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]_r$	1941	visc_fluxmm_r
$c_5 \left[\overline{\boldsymbol{v}} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]_\theta$	1942	visc_fluxmm_theta
$c_5 \left[\overline{\boldsymbol{v}} \cdot \overline{\boldsymbol{\mathcal{D}}}\right]_\phi$	1943	visc_fluxmm_phi
$-c_3 v_r P$	1944	press_flux_r
$-c_3 v_\theta P$	1945	press_flux_theta
$-c_3 v_\phi P$	1946	press_flux_phi
$-c_3 v'_r P'$	1947	press_fluxpp_r
$-c_3 v'_\theta P'$	1948	press_fluxpp_theta
$-c_3 v'_\phi P'$	1949	press_fluxpp_phi
$-c_3 \overline{v_r} \, \overline{P}$	1950	press_fluxmm_r
$-c_3 \overline{v_\theta} \, \overline{P}$	1951	press_fluxmm_theta
$-c_3 \overline{v_\phi} \, \overline{P}$	1952	press_fluxmm_phi

continues on next page

Table 8 – continued from previous page

−−	1953	production_shear_ke
−−	1954	production_shear_pke
−−	1955	production_shear_mke

5.2.15 Magnetic Energy Equation

$[(\boldsymbol{v} \times \boldsymbol{B} - \eta \boldsymbol{\mathcal{J}}) \times \boldsymbol{B}]_r$	2001	ecrossb_r
$[(\boldsymbol{v} \times \boldsymbol{B} - \eta \boldsymbol{\mathcal{J}}) \times \boldsymbol{B}]_\theta$	2002	ecrossb_theta
$[(\boldsymbol{v} \times \boldsymbol{B} - \eta \boldsymbol{\mathcal{J}}) \times \boldsymbol{B}]_\phi$	2003	ecrossb_phi
$[(\boldsymbol{v'} \times \boldsymbol{B'} - \eta \boldsymbol{\mathcal{J}'}) \times \boldsymbol{B'}]_r$	2004	ecrossb_ppp_r
$[(\boldsymbol{v'} \times \boldsymbol{B'} - \eta \boldsymbol{\mathcal{J}'}) \times \boldsymbol{B'}]_\theta$	2005	ecrossb_ppp_theta
$[(\boldsymbol{v'} \times \boldsymbol{B'} - \eta \boldsymbol{\mathcal{J}'}) \times \boldsymbol{B'}]_\phi$	2006	ecrossb_ppp_phi
$\left[\left(\overline{\boldsymbol{v}} \times \overline{\boldsymbol{B}} - \eta \overline{\boldsymbol{\mathcal{J}}}\right) \times \overline{\boldsymbol{B}}\right]_r$	2007	ecrossb_mmm_r
$\left[\left(\overline{\boldsymbol{v}} \times \overline{\boldsymbol{B}} - \eta \overline{\boldsymbol{\mathcal{J}}}\right) \times \overline{\boldsymbol{B}}\right]_\theta$	2008	ecrossb_mmm_theta
$\left[\left(\overline{\boldsymbol{v}} \times \overline{\boldsymbol{B}} - \eta \overline{\boldsymbol{\mathcal{J}}}\right) \times \overline{\boldsymbol{B}}\right]_\phi$	2009	ecrossb_mmm_phi
$\left[(\boldsymbol{v'} \times \boldsymbol{B'}) \times \overline{\boldsymbol{B}}\right]_r$	2010	ecrossb_ppm_r
$\left[(\boldsymbol{v'} \times \boldsymbol{B'}) \times \overline{\boldsymbol{B}}\right]_\theta$	2011	ecrossb_ppm_theta
$\left[(\boldsymbol{v'} \times \boldsymbol{B'}) \times \overline{\boldsymbol{B}}\right]_\phi$	2012	ecrossb_ppm_phi
$\left[\left(\boldsymbol{v'} \times \overline{\boldsymbol{B}}\right) \times \boldsymbol{B'}\right]_r$	2013	ecrossb_pmp_r
$\left[\left(\boldsymbol{v'} \times \overline{\boldsymbol{B}}\right) \times \boldsymbol{B'}\right]_\theta$	2014	ecrossb_pmp_theta
$\left[\left(\boldsymbol{v'} \times \overline{\boldsymbol{B}}\right) \times \boldsymbol{B'}\right]_\phi$	2015	ecrossb_pmp_phi
$[(\overline{\boldsymbol{v}} \times \boldsymbol{B'}) \times \boldsymbol{B'}]_r$	2016	ecrossb_mpp_r
$[(\overline{\boldsymbol{v}} \times \boldsymbol{B'}) \times \boldsymbol{B'}]_\theta$	2017	ecrossb_mpp_theta
$[(\overline{\boldsymbol{v}} \times \boldsymbol{B'}) \times \boldsymbol{B'}]_\phi$	2018	ecrossb_mpp_phi
$\boldsymbol{B} \cdot [\boldsymbol{\nabla} \times (\boldsymbol{v} \times \boldsymbol{B})]$	2019	induct_work
$\boldsymbol{B'} \cdot [\boldsymbol{\nabla} \times (\boldsymbol{v'} \times \boldsymbol{B'})]$	2020	induct_work_ppp
$\boldsymbol{B'} \cdot \left[\boldsymbol{\nabla} \times \left(\boldsymbol{v'} \times \overline{\boldsymbol{B}}\right)\right]$	2021	induct_work_ppm
$\boldsymbol{B'} \cdot [\boldsymbol{\nabla} \times (\overline{\boldsymbol{v}} \times \boldsymbol{B'})]$	2022	induct_work_pmp
$\overline{\boldsymbol{B}} \cdot [\boldsymbol{\nabla} \times (\boldsymbol{v'} \times \boldsymbol{B'})]$	2023	induct_work_mpp
$\overline{\boldsymbol{B}} \cdot \left[\boldsymbol{\nabla} \times \left(\overline{\boldsymbol{v}} \times \overline{\boldsymbol{B}}\right)\right]$	2024	induct_work_mmm
$\boldsymbol{B} \cdot [\boldsymbol{B} \cdot \boldsymbol{\nabla} \boldsymbol{v}]$	2025	ishear_work
$-\boldsymbol{B} \cdot [\boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{B}]$	2026	iadvec_work
$-\boldsymbol{B} \cdot (\boldsymbol{\nabla} \cdot \boldsymbol{v})\, \boldsymbol{B}$	2027	icomp_work
$\boldsymbol{B'} \cdot \left[\overline{\boldsymbol{B}} \cdot \boldsymbol{\nabla} \boldsymbol{v'}\right]$	2028	ishear_work_pmp
$-\boldsymbol{B'} \cdot [\overline{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \boldsymbol{B'}]$	2029	iadvec_work_pmp
$-\boldsymbol{B'} \cdot (\boldsymbol{\nabla} \cdot \overline{\boldsymbol{v}})\, \boldsymbol{B'}$	2030	icomp_work_pmp
$\boldsymbol{B'} \cdot [\boldsymbol{B'} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{v}}]$	2031	ishear_work_ppm
$-\boldsymbol{B'} \cdot \left[\boldsymbol{v'} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{B}}\right]$	2032	iadvec_work_ppm
$-\boldsymbol{B'} \cdot (\boldsymbol{\nabla} \cdot \boldsymbol{v'})\, \overline{\boldsymbol{B}}$	2033	icomp_work_ppm
$\overline{\boldsymbol{B}} \cdot \left[\overline{\boldsymbol{B}} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{v}}\right]$	2034	ishear_work_mmm
$-\overline{\boldsymbol{B}} \cdot \left[\overline{\boldsymbol{v}} \cdot \boldsymbol{\nabla} \overline{\boldsymbol{B}}\right]$	2035	iadvec_work_mmm
$-\overline{\boldsymbol{B}} \cdot (\boldsymbol{\nabla} \cdot \overline{\boldsymbol{v}})\, \overline{\boldsymbol{B}}$	2036	icomp_work_mmm

continues on next page

Table 9 – continued from previous page

$\overline{\boldsymbol{B}} \cdot [\boldsymbol{B'} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]$	2037	ishear_work_mpp
$-\overline{\boldsymbol{B}} \cdot [\boldsymbol{v'} \cdot \boldsymbol{\nabla} \boldsymbol{B'}]$	2038	iadvec_work_mpp
$-\overline{\boldsymbol{B}} \cdot (\boldsymbol{\nabla} \cdot \boldsymbol{v'}) \boldsymbol{B'}$	2039	icomp_work_mpp
$\boldsymbol{B'} \cdot [\boldsymbol{B'} \cdot \boldsymbol{\nabla} \boldsymbol{v'}]$	2040	ishear_work_ppp
$-\boldsymbol{B'} \cdot [\boldsymbol{v'} \cdot \boldsymbol{\nabla} \boldsymbol{B'}]$	2041	iadvec_work_ppp
$-\boldsymbol{B'} \cdot (\boldsymbol{\nabla} \cdot \boldsymbol{v'}) \boldsymbol{B'}$	2042	icomp_work_ppp
$-c_7 \boldsymbol{B} \cdot [\boldsymbol{\nabla} \times (\mathrm{f}_7 \boldsymbol{\nabla} \times \boldsymbol{B})]$	2043	idiff_work
$-c_7 \boldsymbol{B'} \cdot [\boldsymbol{\nabla} \times (\mathrm{f}_7 \boldsymbol{\nabla} \times \boldsymbol{B'})]$	2044	idiff_work_pp
$-c_7 \overline{\boldsymbol{B}} \cdot \left[\boldsymbol{\nabla} \times \left(\mathrm{f}_7 \boldsymbol{\nabla} \times \overline{\boldsymbol{B}}\right)\right]$	2045	idiff_work_mm

5.2.16 Turbulent Kinetic Energy Generation

$\mathrm{f}_2 \Theta' {v'}_r$	2701	production_buoyant_pKE
$-\mathrm{f}_1 v'_i v'_j \overline{e}_{ij}$	2702	production_shear2_pKE
$2\mathrm{f}_1\mathrm{f}_3 \left[e'_{ij} e'_{ij} - (\boldsymbol{\nabla} \cdot \boldsymbol{v'})^2 / 3\right]$	2703	dissipation_viscous_pKE
$-\boldsymbol{\nabla} \cdot (P' \boldsymbol{v'})$	2704	transport_pressure_pKE
$\boldsymbol{\nabla} \cdot (\boldsymbol{\mathcal{D}'} \cdot \boldsymbol{v'})$	2705	transport_viscous_pKE
$-\boldsymbol{\nabla} \cdot \left(\frac{1}{2}\mathrm{f}_1 {v'}^2 \boldsymbol{v'}\right)$	2706	transport_turbadvect_pKE
$-\boldsymbol{\nabla} \cdot \left(\frac{1}{2}\mathrm{f}_1 {v'}^2 \overline{\boldsymbol{v}}\right)$	2707	transport_meanadvect_pKE
$P' v'_r$	2708	rflux_pressure_pKE
$-\left[\boldsymbol{\mathcal{D}'} \cdot \boldsymbol{v'}\right]_r$	2709	rflux_viscous_pKE
$\frac{1}{2}\mathrm{f}_1 {v'}^2 v'_r$	2710	rflux_turbadvect_pKE
$\frac{1}{2}\mathrm{f}_1 {v'}^2 \overline{v}_r$	2711	rflux_meanadvect_pKE
$P' v'_\theta$	2712	thetaflux_pressure_pKE
$-\left[\boldsymbol{\mathcal{D}'} \cdot \boldsymbol{v'}\right]_\theta$	2713	thetaflux_viscous_pKE
$\frac{1}{2}\mathrm{f}_1 {v'}^2 v'_\theta$	2714	thetaflux_turbadvect_pKE
$\frac{1}{2}\mathrm{f}_1 {v'}^2 \overline{v}_\theta$	2715	thetaflux_meanadvect_pKE
$P' v'_\phi$	2716	phiflux_pressure_pKE
$-\left[\boldsymbol{\mathcal{D}'} \cdot \boldsymbol{v'}\right]_\phi$	2717	phiflux_viscous_pKE
$\frac{1}{2}\mathrm{f}_1 {v'}^2 v'_\phi$	2718	phiflux_turbadvect_pKE
$\frac{1}{2}\mathrm{f}_1 {v'}^2 \overline{v}_\phi$	2719	phiflux_meanadvect_pKE

5.2.17 Axial Field

v_z	2801	v_z
$\overline{v_z}$	2802	vm_z
v_z'	2803	vp_z
$\frac{\partial v_z}{\partial z}$	2804	dvzdz
$\overline{\frac{\partial v_z}{\partial z}}$	2805	dvzdz_m
$\frac{\partial v_z}{\partial z}'$	2806	dvzdz_p
ω_z	2807	vort_z
$\overline{\omega_z}$	2808	vortm_z
ω_z'	2809	vortp_z
$v_z\omega_z$	2810	kin_helicity_z
$\overline{v_z}\overline{\omega_z}$	2811	kin_helicity_z_mm
$v_z'\omega_z'$	2812	kin_helicity_z_pp
$\overline{v_z}\omega_z'$	2813	kin_helicity_z_mp
$v_z'\overline{\omega_z}$	2814	kin_helicity_z_pm
B_z	2815	B_z
$\overline{B_z}$	2816	Bm_z
B_z'	2817	Bp_z
$\mathcal{J}_z$	2818	J_z
$\overline{\mathcal{J}_z}$	2819	Jm_z
$\mathcal{J}_z'$	2820	Jp_z
$\frac{\partial \Theta}{\partial z}$	2821	dTvardz
$\overline{\frac{\partial \Theta}{\partial z}}$	2822	dTvardz_m
$\frac{\partial \Theta}{\partial z}'$	2823	dTvardz_p
$\frac{\partial \bar{P}}{\partial z}$	2824	dPdz
$\overline{\frac{\partial P}{\partial z}}$	2825	dPdz_m
$\frac{\partial \bar{P}'}{\partial z}$	2826	dPdz_p

Note that we use the shorthand $\mathcal{J}$ to denote the curl of $\boldsymbol{B}$, namely $\mathcal{J} \equiv \boldsymbol{\nabla} \times \boldsymbol{B}$.

CHAPTER

SIX

GETTING HELP

For questions on the source code of Rayleigh, portability, installation, new or existing features, etc., use the Rayleigh forum. This is also the place where we announce our regular user calls. For a more direct contact you can also join our Slack channel.

BIBLIOGRAPHY

[BMW+10] M. Breuer, A. Manglik, J. Wicht, T. Trumper, H. Harder, and U. Hansen1. Thermochemically driven convection in a rotating spherical shell. *Geophysical Journal International*, 183:150–162, 2010. URL: https://academic.oup.com/gji/article/183/1/150/592285, doi:http://doi.org/10.1111/j.1365-246X.2010.04722.x.

[CAC+01] U.R. Christensen, J. Aubert, P. Cardin, E. Dormy, S. Gibbons, G.A. Glatzmaier, E. Grote, Y. Honkura, C. Jones, M. Kono, M. Matsushima, A. Sakuraba, F. Takahashi, A. Tilgner, J. Wicht, and K. Zhang. A numerical dynamo benchmark. *Physics of the Earth and Planetary Interiors*, 128(1):25 – 34, 2001. Dynamics and Magnetic Fields of the Earth's and Planetary Interiors. URL: http://www.sciencedirect.com/science/article/pii/S0031920101002758, doi:https://doi.org/10.1016/S0031-9201(01)00275-8.

[GCE+99] G. A. Glatzmaier, T. C. Clune, J. R. Elliott, M. S. Miesch, and J. Toomre. Computational aspects of a code to study rotating turbulent convection in spherical shells. *Parallel Comput.*, 25(4):361–380, April 1999. URL: http://dx.doi.org/10.1016/S0167-8191(99)00009-5, doi:10.1016/S0167-8191(99)00009-5.

[Gla84] Gary A Glatzmaier. Numerical simulations of stellar convective dynamos. i. the model and method. *Journal of Computational Physics*, 55(3):461 – 484, 1984. URL: http://www.sciencedirect.com/science/article/pii/0021999184900330, doi:https://doi.org/10.1016/0021-9991(84)90033-0.

[JBB+11] C.A. Jones, P. Boronski, A.S. Brun, G.A. Glatzmaier, T. Gastine, M.S. Miesch, and J. Wicht. Anelastic convection-driven dynamo benchmarks. *Icarus*, 216(1):120 – 135, 2011. URL: http://www.sciencedirect.com/science/article/pii/S0019103511003319, doi:https://doi.org/10.1016/j.icarus.2011.08.014.

[Fea18a] N. Featherstone. Rayleigh 0.9.1. 2018. doi:http://doi.org/10.5281/zenodo.1236565.

[Fea18b] N. Featherstone. Rayleigh version 0.9.0. 2018. doi:http://doi.org/10.5281/zenodo.1158290.

[BM19] B. Buffett and H. Matsui. Equatorially trapped waves in earth\textquoteright s core. *Geophysical Journal International*, 218(2):1210–1225, 2019. doi:10.1093/gji/ggz233.

[KMB18] B. B. Karak, M. Miesch, and Y. Bekki. Consequences of high effective prandtl number on solar differential rotation and convective velocity. *Physics of Fluids*, 30(4):046602, 2018. doi:10.1063/1.5022034.

[MXF+18] B. Miquel, J-H Xie, N. Featherstone, K. Julien, and E. Knobloch. Equatorially trapped convection in a rapidly rotating shallow shell. *Physical Review Fluids*, 2018. doi:10.1103/PhysRevFluids.3.053801.

[OCFH18] R. J. Orvedahl, M. A. Calkins, N. A. Featherstone, and B. W. Hindman. Prandtl-number effects in high-rayleigh-number spherical convection. *The Astrophysical Journal*, 856(1):13, 2018. doi:10.3847/1538-4357/aaaeb5.

[FH16] N.A. Featherstone and B.W. Hindman. The spectral amplitude of stellar convection and its scaling in the high-rayleigh-number regime. *The Astrophysical Journal*, 818(1):32, 2016. doi:http://doi.org/10.3847/0004-637X/818/1/32.

[MHA+16] H. Matsui, E. Heien, J. Aubert, J.M. Aurnou, M. Avery, B. Brown, B.A. Buffett, F. Busse, U.R. Christensen, C.J. Davies, N. Featherstone, T. Gastine, G.A. Glatzmaier, D. Gubbins, J.-L. Guermond, Y.-Y. Hayashi, R. Hollerbach, L.J. Hwang, A. Jackson, C.A. Jones, W. Jiang, L.H. Kellogg, W. Kuang, M. Landeau, P.H. Marti, P. Olson, A. Ribeiro, Y. Sasaki, N. Schaeffer, R.D. Simitev, A. Sheyko, L. Silva, S. Stanley, F. Takahashi, S.-ichi Takehiro, J. Wicht, and A.P. Willis. Performance benchmarks for a next generation numerical dynamo model. *Geochemistry, Geophysics, Geosystems*, 17(5):1586–1607, 2016. doi:http://doi.org/10.1002/2015GC006159.

[OMaraMFA16] B. O'Mara, M. S. Miesch, N. A. Featherstone, and K. C. Augustson. Velocity amplitudes in global convection simulations: the role of the prandtl number and near-surface driving. *Advances in Space Research*, 2016. doi:10.1016/j.asr.2016.03.038.