Documentation of MagIC

Release 6.0

The MagIC dev team

Mar 09, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Foreword

MagIC is a numerical code that can simulate fluid dynamics in a spherical shell. MagIC solves for the Navier-Stokes equation including Coriolis force, optionally coupled with an induction equation for Magneto-Hydro Dynamics (MHD), a temperature (or entropy) equation and an equation for chemical composition under both the anelastic and the Boussinesq approximations.

MagIC uses Chebyshev polynomials or finite difference in the radial direction and spherical harmonic decomposition in the azimuthal and latitudinal directions. MagIC supports several Implicit-Explicit time schemes where the nonlinear terms and the Coriolis force are treated explicitly, while the remaining linear terms are treated implicitly.

MagIC is written in Fortran and designed to be used on supercomputing clusters. It thus relies on a hybrid parallelisation scheme using both OpenMP and MPI. Postprocessing functions written in python (requiring matplotlib and scipy) are also provided to allow a useful data analysis.

MagIC is a free software. It can be used, modified and redistributed under the terms of the GNU GPL v3 licence.

1.2 Giving credit

In case you intend to publish scientific results obtained with the MagIC code or present them in a conference, we (the developers of MagIC) kindly ask to be acknowledged with a reference to the website https://magic-sph.github.io/ or https://github.com/magic-sph/magic.

We also suggest to give appropriate reference to one or several of the following papers:

- Boussinesq equations: Wicht (2002, PEPI, 132, 281-302)
- Anelastic equations: Gastine & Wicht (2012, Icarus, 219, 28-442)
- Boussinesq benchmark: Christensen et al. (2001, PEPI, 128, 25-34)
- Benchmark for double diffusive convection: Breuer et al. (2010, GJI, 183, 150-162)
- Anelastic benchmark: Jones et al. (2011, Icarus, 216, 120-135)
- In case you use the SHTns library for the spherical harmonics transforms (MagIC 5.3 or later), please also cite: Schaeffer (2013, GGG, 14, 751-758)

See also:

A (tentative) comprehensive list of the publications that have been produced to date (May 2019) using **MagIC** is accessible here. To date, more than **100 publications** have been-accepted in more than 10 different peer-reviewed journals: PEPI (22), Icarus (11), E&PSL (7), GJI (8), A&A (6), GRL (4), JFM (6), GAFD (3), Nature (2), etc.

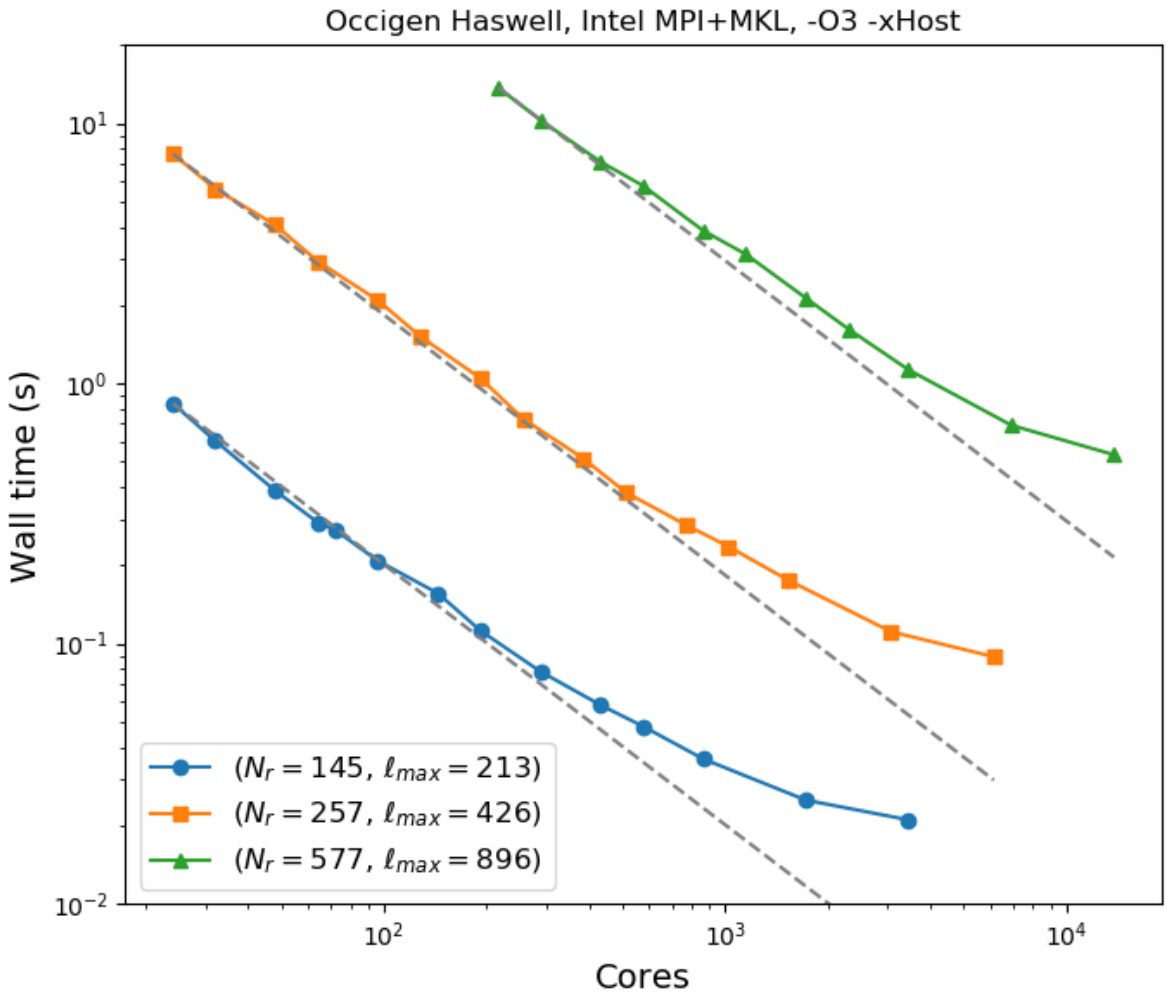

Fig. 1: Mean walltime of the MagIC code on the supercomputer Occigen versus number of cores for a Boussinesq dynamo model computed at three different numerical resolutions $(N_r, \ell_{\max})$. The dashed grey lines show the ideal scalings.

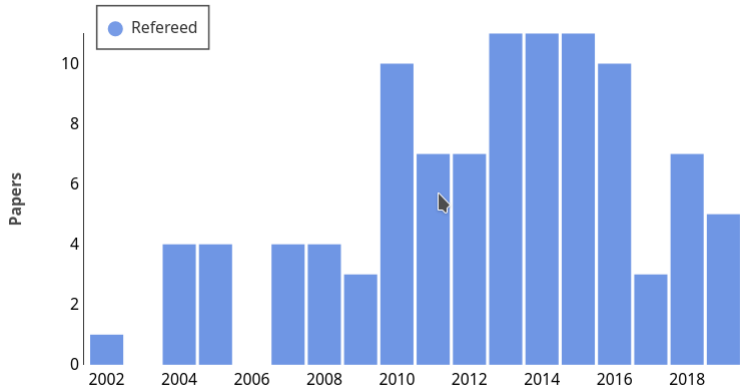

Fig. 2: Number of peer-reviewed publications produced using **MagIC**

CHAPTER

TWO

GET MAGIC AND RUN IT

2.1 Download the code

You can download a snapshot of the code from the Git repository using

```
$ git clone https://github.com/magic-sph/magic.git
```

In case you already have an account on github.com and uploaded a public SSH key on it, you could then rather use SSH:

```
$ git clone ssh://git@github.com/magic-sph/magic.git
```

2.2 Setting up the environment variables

Although not mandatory, it is strongly recommended to correctly source the environment variables of the MagIC code. It will ensure a smoother usage of the post-processing *python classes* and allow to run the *auto-test suite*. To do that, just go to the root directory of the MagIC code (`magic`) and source `sourceme` file that corresponds to your `$SHELL` environment variable.

In case you use bash, ksh or zsh, just use:

```
$ source sourceme.sh
```

In case you use csh or tcsh, rather use

```
$ source sourceme.csh
```

You can make sure that the environment variables have been correctly sourced by typing:

```
$ echo $MAGIC_HOME
$ echo $PYTHONPATH
```

If you don't want to `source sourceme.[c]sh` on each session, you can add the following into your `.bash_profile` (or `.profile` or `.zprofile` or `.cshrc`):

```
$ source whereverYouCheckedOut/magic/sourceme.sh
```

To get started, you then need to compile the code.

2.3 Install SHTns (recommended)

SHTns is a an open-source library for the Spherical Harmonics transforms. It is significantly faster than the native transforms implemented in MagIC, and it is hence **recommended** (though not mandatory) to install it. To install the library, first define a C compiler

```
$ export CC= gcc
```

or

```
$ export CC= icc
```

Then make sure a FFT library such FFTW or the MKL is installed on the target machine. Then make use of the install script

```
cd $MAGIC_HOME/bin
./install-shtns.sh
```

or install it manually after downloading and extracting the latest version here

```
./configure --enable-openmp --enable-ishioka --enable-magic-layout --prefix=$HOME/
↪local
```

if FFTW is used or

```
./configure --enable-openmp --enable-ishioka --enable-magic-layout --prefix=$HOME/
↪local --enable-mkl
```

if the MKL is used. Possible additional options may be required depending on the machine (check the website). Then compile and install the library

```
make
make install
```

2.4 Setting up compiler options and compiling

The **recommended way of compiling MagIC** is to use the build system CMake , if available on your platform. Otherwise, a backup solution is provided via the manual edition of a `Makefile`.

2.4.1 Generic compiling options

For both build systems (CMake or make), several build options can be toggled using the following available options:

- `ARCH` Set it to '64' for 64 bit architecture or to '32' for 32 bit architecture
- `PRECISION` Set it to 'dble' for double-precision calculations or to 'sngl' for single-precision calculations
- `OUT_PREC` Set it to 'dble' for double-precision in binary outputs or to 'sngl' for single precision
- `USE_MPI` Set to `yes` to use MPI, set it to `no` if you want a serial version of the code .
- `USE_OMP` Set it to `yes` to use the hybrid version of the code, or to `no` for a pure MPI (or serial) version.
- `USE_PRECOND` Set to `yes` to perform some pre-conditioning of the matrices.

- `USE_FFTLIB` This option lets you select the library you want to use for Fast Fourier Transforms. This can be set to 'JW', 'FFTW' or 'MKL'. 'JW' refers to the built-in library by **J** ohannes **W** icht, FFTW refers to the Fastest Fourier Transform in the West, while 'MKL' refers to the Intel Math Kernel Library. Use 'JW' if you don't have Intel MKL installed.
- `USE_DCTLIB` This option lets you select the library you want to use for Discrete Cosine Transforms. This can be set to 'JW', 'FFTW' or 'MKL'.
- `USE_LAPACKLIB` This option allows you to select the library you want to use for LU factorisation. This can be set to 'JW', 'MKL', 'LIBFLAME' or 'LAPACK'. 'LIBFLAME' refers to the AMD dense matrix solvers libflame.
- `USE_SHTNS` Set to `yes` to use SHTns library for spherical harmonics transforms. The helper script `install-shtns.sh` is available in the `bin` directory to help installing SHTns.
- `CMAKE_BUILD_TYPE` Set to `Debug` to enable the full debug flags.

Warning: MagIC cannot run with openMP alone, therefore a configuration of the form `USE_MPI=no`, `USE_OMP=yes` will be overwritten to force `USE_OMP=no`

2.4.2 Using `CMake` (recommended)

CMake is a powerful tool that can automatically detects and finds the best appropriate configuration for your platform. To use it, you just need to create a directory where you want to build the sources. For instance:

```
$ mkdir $MAGIC_HOME/build
$ cd $MAGIC_HOME/build
```

In a second step, you might want to specify your C and Fortran compilers (in case you skip this step, CMake will look for compilers for you but it might pick up another compiler as the one you might have wanted). For instance, in case you want to use the Intel compilers, you can export the following environment variables

```
$ export FC=mpiifort
$ export CC=icc
```

for bash/ksh/zsh users and

```
$ setenv FC=mpiifort
$ setenv CC=mpiicc
```

for csh/tcsh users. At this stage you should be ready to build the code. If you simply use:

```
$ cmake .. -DUSE_SHTNS=yes
```

CMake will try to use the best options available on your machine (for instance it will try to locate and link the Intel Math Kernel Library). Otherwise you can pass the aforementioned available options to CMake using the generic form *-DOPTION=value*. For instance, in case you want to make use of the built-in libraries of MagIC and want to disable OpenMP, simply use

```
$ cmake .. -DUSE_OMP=no -DUSE_FFTLIB=JW -DUSE_LAPACKLIB=JW
```

Once you're happy with your configuration, just compile the code:

```
$ make -j
```

The executable `magic.exe` should have been produced in the local directory.

If you want to recompile the code from scratch do

```
$ make clean
```

to remove all the files generated by the compiler.

Once the executable is built, you are now ready to run your first production run!

2.4.3 Using `make` (backup solution)

In case CMake is not available on your platform, it is still possible to compile the code directly. Go to the directory where the source files of MagIC are contained

```
$ cd $MAGIC_HOME/src
```

Select compiler

Edit the file named `Makefile` using your favourite editor and set a suitable compiler for your platform using the variable: `COMPILER = value`. The possible options are `intel`, `gnu` or `portland` compilers.

List of default compilers

Compiler Option	Normal	With MPI
intel	ifort, icc	mpiifort, mpiicc
gnu	gfortran, gcc	mpif90, mpicc
portland	pgf95, pgcc	mpif90, mpicc

Warning: In case you want to use intel but `mpiifort` and `mpiicc` are not available, you may also need to adapt the variables `COMP_MPFC` and `COMP_MPCC`.

Select compiling options

You can also modify the different compiling options by editing the values of the various parameters defined in the first lines of the `Makefile`. For instance, in case you want to make use of the built-in libraries and want to disable OpenMP, just define

```
USE_OMP=no
USE_FFTLIB=JW
USE_LAPACKLIB=JW
```

MPI_INCPATH

This variable sets the path for your MPI header file `mpif.h`. This is in general useless if you already use the MPI wrappers such as `mpiifort` or `mpif90` to compile the code. It might be however required to define this path for some compiler configurations: `MPI_INCPATH` is usually `/usr/include` or `/usr/include/mpi` and should be found by the `Makefile` automatically thanks to the command `mpif90 --showme:incdirs`. In case this doesn't work, you may need to specify this variable manually in the `Makefile`. On supercomputing clusters, this variable is in general not used.

Other compilers

If your available compilers are different from the options provided in the `Makefile`, then just create a new profile for your desired compiler by changing the options `COMP_FC` and `COMP_CC` for serial fortran and C compilers and `COMP_MPFC` and `COMP_MPCC` for the possible MPI wrappers.

Once you've set up your compiling options compile the code using

```
$ make -j
```

The compiler should then produce an executable named `magic.exe`.

If you want to recompile the code from scratch do

```
$ make clean
```

to remove all the files generated by the compiler.

Once the executable is built, you are now ready to run your first production run!

2.5 Preparing a production run

After building the executable, use one of the namelists provided in the `$MAGIC_HOME/samples` directory (called `input.nml`), adapt it to your physical problem (see *here* for an exhaustive description of the possible options) and run **MagIC** as follows:

- Running a serial version of the code (`USE_MPI=no` and `USE_OMP=no`):

```
$ ./magic.exe input.nml
```

- Running the code without OpenMP (`USE_MPI=yes` and `USE_OMP=no`) with `<n_mpi>` MPI ranks:

```
$ mpiexec -n <n_mpi> ./magic.exe input.nml
```

- Running the hybrid code (`USE_MPI=yes` and `USE_OMP=yes`) with `<n_mpi>` MPI ranks and `<n_omp>` OpenMP threads:

```
$ export OMP_NUM_THREAD = <n_omp>
$ export KMP_AFFINITY=verbose,granularity=core,compact,1
$ mpiexec -n <n_mpi> ./magic.exe input.nml
```

Note that the *n_r_max-1* must be a multiple of `<n_mpi>`, where *n_r_max* is the number of radial grid points (see *here*).

CHAPTER

THREE

FORMULATION OF THE (MAGNETO)-HYDRODYNAMICS PROBLEM

The general equations describing thermal convection and dynamo action of a rotating compressible fluid are the starting point from which the Boussinesq or the anelastic approximations are developed. In MagIC, we consider a spherical shell rotating about the vertical z axis with a constant angular velocity Ω. Equations are solve in the corotating system.

The conservation of momentum is formulated by the Navier-Stokes equation

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\boldsymbol{\nabla}\,\boldsymbol{u}\right)=-\boldsymbol{\nabla}p+\frac{1}{\mu_0}(\boldsymbol{\nabla}\times\boldsymbol{B})\times\boldsymbol{B}+\rho\boldsymbol{g}-2\rho\boldsymbol{\Omega}\times\boldsymbol{u}+\boldsymbol{\nabla}\cdot\mathsf{S}, \tag{3.1}$$

where $\boldsymbol{u}$ is the velocity field, $\boldsymbol{B}$ the magnetic field, and p a modified pressure that includes centrifugal forces. S corresponds to the rate-of-strain tensor given by:

$$S_{ij}=2\nu\rho\left[e_{ij}-\frac{1}{3}\delta_{ij}\,\boldsymbol{\nabla}\cdot\boldsymbol{u}\right],$$
$$e_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right).$$

Convection is driven by buoyancy forces acting on variations in density ρ. These variations have a dynamical part formulated by the continuity equation describing the conservation of mass:

$$\frac{\partial\rho}{\partial t}=-\boldsymbol{\nabla}\cdot\rho\boldsymbol{u}. \tag{3.2}$$

In addition an equation of state is required to formulate the thermodynamic density changes. For example the relation

$$\frac{1}{\rho}\partial\rho=-\alpha\partial T+\beta\partial p-\delta\partial\xi \tag{3.3}$$

describes density variations caused by variations in temperature T, pressure p, and composition ξ. The latter contribution needs to be considered for describing the effects of light elements released from a growing solid iron core in a so-called double diffusive approach.

To close the system we also have to formulate the dynamic changes of entropy, pressure, and composition. The evolution equation for pressure can be derived from the Navier-Stokes equation, as will be further discussed below. For entropy variations we use the so-called energy or heat equation

$$\rho T\left(\frac{\partial s}{\partial t}+\boldsymbol{u}\cdot\boldsymbol{\nabla}s\right)=\boldsymbol{\nabla}\cdot(k\boldsymbol{\nabla}T)+\Phi_\nu+\frac{\lambda}{\mu_0}\left(\boldsymbol{\nabla}\times\boldsymbol{B}\right)^2+\epsilon, \tag{3.4}$$

where Φ_ν corresponds to the viscous heating expressed by

$$\Phi_\nu=2\rho\left[e_{ij}e_{ji}-\frac{1}{3}\left(\boldsymbol{\nabla}\cdot\boldsymbol{u}\right)^2\right]$$

Note that we use here the summation convention over the indices i and j. The second last term on the right hand side is the Ohmic heating due to electric currents. The last term is a volumetric sink or source term that can describe various

effects, for example radiogenic heating, the mixing-in of the light elements or, when radially dependent, potential variations in the adiabatic gradient (see below). For chemical composition, we finally use

$$\rho\left(\frac{\partial \xi}{\partial t}+\boldsymbol{u}\cdot\boldsymbol{\nabla}\xi\right)=\boldsymbol{\nabla}\cdot(k_\xi\boldsymbol{\nabla}\xi)+\epsilon_\xi, \tag{3.5}$$

The induction equation is obtained from the Maxwell equations (ignoring displacement current) and Ohm's law (neglecting Hall effect):

$$\frac{\partial \boldsymbol{B}}{\partial t}=\boldsymbol{\nabla}\times(\boldsymbol{u}\times\boldsymbol{B}-\lambda\boldsymbol{\nabla}\times\boldsymbol{B}). \tag{3.6}$$

When the magnetic diffusivity λ is homogeneous this simplifies to the commonly used form

$$\frac{\partial \boldsymbol{B}}{\partial t}=\boldsymbol{\nabla}\times(\boldsymbol{u}\times\boldsymbol{B})+\lambda\boldsymbol{\Delta}\boldsymbol{B}. \tag{3.7}$$

The physical properties determining above equations are rotation rate Ω, the kinematic viscosity ν, the magnetic permeability μ_0, gravity $\boldsymbol{g}$, thermal conductivity k, Fick's conductiviy k_ξ, magnetic diffusivity λ. The latter connects to the electrical conductivity σ via $\lambda=1/(\mu_0\sigma)$. The thermodynamics properties appearing in (3.3) are the thermal expansivity at constant pressure (and composition)

$$\alpha=-\frac{1}{\rho}\left(\frac{\partial\rho}{\partial T}\right)_{p,\xi}, \tag{3.8}$$

the compressibility at constant temperature

$$\beta=\frac{1}{\rho}\left(\frac{\partial\rho}{\partial p}\right)_{T,\xi}$$

and an equivalent parameter δ for the dependence of density on composition:

$$\delta=-\frac{1}{\rho}\left(\frac{\partial\rho}{\partial \xi}\right)_{p,T}, \tag{3.9}$$

3.1 The reference state

The convective flow and the related processes including magnetic field generation constitute only small disturbances around a background or reference state. In the following we denote the background state with a tilde and the disturbance we are interested in with a prime. Formally we will solve equations in first order of a smallness parameters ϵ which quantified the ratio of convective disturbances to background state:

$$\epsilon\sim\frac{T'}{\tilde{T}}\sim\frac{p'}{\tilde{p}}\sim\frac{\rho'}{\tilde{\rho}}\sim\ldots\ll 1. \tag{3.10}$$

The background state is hydrostatic, i.e. obeys the simple force balance

$$\nabla\tilde{p}=\tilde{\rho}\tilde{\boldsymbol{g}}. \tag{3.11}$$

Convective motions are supposed to be strong enough to provide homogeneous entropy (and composition). The reference state is thus adiabatic and its gradients can be expressed in terms of the pressure gradient (3.11):

$$\frac{\nabla\tilde{T}}{\tilde{T}}=\frac{1}{\tilde{T}}\left(\frac{\partial T}{\partial p}\right)_s\nabla p=\frac{\alpha}{c_p}\tilde{\boldsymbol{g}}, \tag{3.12}$$

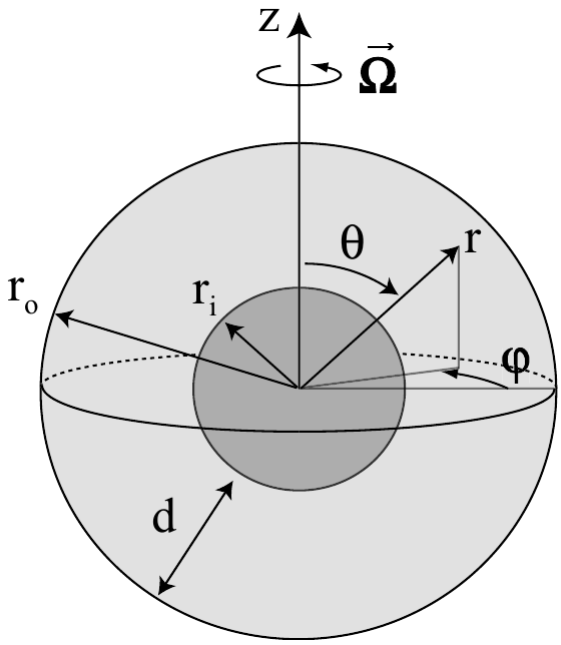

Fig. 1: Sketch of the spherical shell model and its system of coordinate.

$$\frac{\nabla\tilde{\rho}}{\tilde{\rho}} = \frac{1}{\tilde{\rho}}\left(\frac{\partial\rho}{\partial p}\right)_s \nabla p = \beta\tilde{\rho}\tilde{\boldsymbol{g}}. \tag{3.13}$$

The reference state obviously dependence only on radius. Dimensionless numbers quantifying the temperature and density gradients are called dissipation number Di and compressibility parameter Co respectively:

$$Di = \frac{\alpha d}{c_p}\tilde{g},$$

and

$$Co = d\beta\tilde{\rho}\tilde{g}.$$

Here d is a typical length scale, for example the shell thickness of the problem. The dissipation number is something like an inverse temperature scale hight while the compressibility parameters is an inverse density scale hight. The ratio of both numbers also helps to quantify the relative impact of temperature and pressure on density variations:

$$\frac{\alpha\nabla T}{\beta\nabla\rho} \approx \alpha\tilde{T}\frac{Di}{Co}. \tag{3.14}$$

As an example we demonstrate how to derive the first order continuity equation here. Using $\rho = \tilde{\rho} + \rho'$ in (3.2) leads to

$$\frac{\partial\tilde{\rho}}{\partial t} + \frac{\partial\rho'}{\partial t} = -\boldsymbol{\nabla}\cdot(\tilde{\rho}\boldsymbol{u}) - \boldsymbol{\nabla}\cdot(\rho'\boldsymbol{u}).$$

The zero order term vanishes since the background density is considered static (or actually changing very slowly on very long time scales). The second term in the right hand side is obviously of second order. The ratio of the remaining two terms can be estimated to also be of first order in ϵ, meaning that the time derivative of ρ is actually also of second order:

$$\frac{[\partial\rho/\partial t]}{[\boldsymbol{\nabla}\cdot\rho\boldsymbol{u}]} \approx \frac{\rho'}{\tilde{\rho}} \approx \epsilon \ .$$

Square brackets denote order of magnitude estimates here. We have used the fact that the reference state is static and assume time scale of changes are comparable (or slower) ρ' than the time scales represented by u and that length

scales associated to the gradient operator are not too small. We can then neglect local variations in ρ' which means that sound waves are filtered out. This first order continuity equation thus simply reads:

$$\boldsymbol{\nabla} \cdot (\tilde{\rho}\boldsymbol{u}) = 0. \tag{3.15}$$

This defines the so-called anelastic approximation where sound waves are filtered out by neglecting the local time derivative of density. This approximation is justified when typical velocities are sufficiently smaller than the speed of sound.

3.2 Boussinesq approximation

For Earth the dissipation number and the compressibility parameter are around 0.2 when temperature and density jump over the whole liquid core are considered. This motivates the so called Boussinesq approximation where Di and Co are assumed to vanish. The continuity equation (3.2) then simplifies further:

$$\frac{1}{\tilde{\rho}}\boldsymbol{\nabla} \cdot \tilde{\rho}\boldsymbol{u} = \frac{\boldsymbol{u}}{\tilde{\rho}} \cdot \nabla\tilde{\rho} + \nabla \cdot \boldsymbol{u} \approx \nabla \cdot \boldsymbol{u} = 0.$$

When using typical number for Earth, (3.14) becomes 0.05 so that pressure effects on density may be neglected. The first order Navier-Stokes equation (after to zero order hydrostatic reference solution has been subtracted) then reads:

$$\tilde{\rho}\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla}\, \boldsymbol{u}\right) = -\boldsymbol{\nabla} p' - 2\rho\boldsymbol{\Omega} \times \boldsymbol{u} + \alpha\tilde{g}_o T' \frac{\boldsymbol{r}}{r_o} + \delta\tilde{g}_o \xi' \frac{\boldsymbol{r}}{r_o} + \frac{1}{\mu_0}(\boldsymbol{\nabla} \times \boldsymbol{B}) \times \boldsymbol{B} + \tilde{\rho}\nu\Delta\boldsymbol{u}. \tag{3.16}$$

Here u and B are understood as first order disturbances and p' is the first order non-hydrostatic pressure and T' the super-adiabatic temperature and ξ the super-adiabatic chemical composition. Above we have adopted a simplification of the buoyancy term. In the Boussinesq limit with vanishing Co and a small density difference between a solid inner and a liquid outer core a linear gravity dependence provides a reasonable approximation:

$$\tilde{\boldsymbol{g}} = \tilde{g}_o \frac{\boldsymbol{r}}{r_o},$$

where we have chosen the gravity $\tilde{g}_o$ at the outer boundary radius r_o as reference.

The first order energy equation becomes

$$\tilde{\rho}\left(\frac{\partial T'}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} T'\right) = \kappa\Delta T' + \epsilon, \tag{3.17}$$

where we have assumed a homogeneous k and neglected viscous and Ohmic heating which can be shown to scale with Di as we discuss below. Furthermore, we have used the simple relation

$$\partial s \approx \frac{\tilde{\rho}c_p}{\tilde{T}}\partial T,$$

defined the thermal diffusivity

$$\kappa = \frac{k}{\tilde{\rho}c_p},$$

and adjusted the definition of ϵ. Finally the first order equation for chemical composition becomes

$$\tilde{\rho}\left(\frac{\partial \xi'}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla}\xi'\right) = \kappa_\xi\Delta\xi' + \epsilon_\xi, \tag{3.18}$$

where we have assumed a homogeneous k_ξ and adjusted the definition of ϵ_ξ.

MagIC solves a dimensionless form of the differential equations. Time is scaled in units of the viscous diffusion time d^2/ν, length in units of the shell thickness d, temperature in units of the temperature drop $\Delta T = T_o - T_i$ over the

shell, composition in units of the composition drop $\Delta\xi = \xi_o - \xi_i$ over the shell and magnetic field in units $(\mu\lambda\tilde{\rho}\Omega)^{1/2}$. Technically the transition to the dimensionless form is achieved by the substitution

$$r \to r\, d, \quad t \to (d^2/\nu)\, t, \quad T \to \Delta T\, T, \quad \xi \to \Delta\xi\, \xi, \quad B \to (\mu\lambda\tilde{\rho}\Omega)^{1/2}\, B$$

where r stands for any length. The next step then is to collect the physical properties as a few possible characteristic dimensionless numbers. Note that many different scalings and combinations of dimensionless numbers are possible. For the Navier-Stokes equation in the Boussinesq limit MagIC uses the form:

$$\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\,\boldsymbol{u}\right) = -\boldsymbol{\nabla}p' - \frac{2}{E}\boldsymbol{e_z}\times\boldsymbol{u} + \frac{Ra}{Pr}T'\frac{\boldsymbol{r}}{r_o} + \frac{Ra_\xi}{Sc}\xi'\frac{\boldsymbol{r}}{r_o} + \frac{1}{EPm}(\boldsymbol{\nabla}\times\boldsymbol{B})\times\boldsymbol{B} + \Delta\boldsymbol{u}, \tag{3.19}$$

where $\boldsymbol{e}_z$ is the unit vector in the direction of the rotation axis and the meaning of the pressure disturbance p' has been adjusted to the new dimensionless equation form.

3.3 Anelastic approximation

The anelastic approximation adopts the simplified continuity (3.15). The background state can be specified in different ways, for example by providing profiles based on internal models and/or ab initio simulations. We will assume a polytropic ideal gas in the following.

3.3.1 Analytical solution in the limit of an ideal gas

In the limit of an ideal gas which follows $\tilde{p} = \tilde{\rho}\tilde{T}$ and has $\alpha = 1/\tilde{T}$, one directly gets:

$$\begin{aligned}\frac{d\tilde{T}}{dr} &= -Di\,\tilde{g}(r),\\ \tilde{\rho} &= \tilde{T}^{1/(\gamma-1)},\end{aligned}$$

where $\gamma = c_p/c_v$. Note that we have moved to a dimensionless formulations here, where all quantities have been normalized with their outer boundary values and Di refers to the respective outer boundary value. If we in addition make the assumption of a centrally-condensed mass in the center of the spherical shell of radius $r \in [r_i, r_o]$, i.e. $g \propto, 1/r^2$, this leads to

$$\begin{aligned}\tilde{T}(r) &= Di\frac{r_o^2}{r} + (1 - Di\, r_o),\\ \tilde{\rho}(r) &= \tilde{T}^m,\\ Di &= \frac{r_i}{r_o}\left(\exp\frac{N_\rho}{m} - 1\right),\end{aligned}$$

where $N_\rho = \ln(\tilde{\rho}_i/\tilde{\rho}_o)$ is the number of density scale heights of the reference state and $m = 1/(\gamma-1)$ is the polytropic index.

Warning:

- The relationship between N_ρ and the dissipation number Di directly depends on the gravity profile. The formula above is only valid when $g \propto 1/r^2$.
- In this formulation, when you change the polytropic index m, you also change the nature of the fluid you're modelling since you accordingly modify $\gamma = c_p/c_v$.

3.3.2 Anelastic MHD equations

In the most general formulation, all physical properties defining the background state may vary with depth. Specific reference values must then be chosen to provide a unique dimensionless formulations and we typically chose outer boundary values here. The exception is the magnetic diffusivity where we adopt the inner boundary value instead. The motivation is twofold: (i) it allows an easier control of the possible continuous conductivity value in the inner core; (ii) it is a more natural choice when modelling gas giants planets which exhibit a strong electrical conductivity decay in the outer layer.

The time scale is then the viscous diffusion time d^2/ν_o where ν_o is the kinematic viscosity at the outer boundary. Magnetic field is expressed in units of $(\rho_o\mu_0\lambda_i\Omega)^{1/2}$, where ρ_o is the density at the outer boundary and λ_i is the magnetic diffusivity at the **inner** boundary.

This leads to the following sets of dimensionless equations:

$$\left(\frac{\partial \boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u}\right)=-\boldsymbol{\nabla}\left(\frac{p'}{\tilde{\rho}}\right)-\frac{2}{E}\boldsymbol{e_z}\times\boldsymbol{u}+\frac{Ra}{Pr}\tilde{g}\,s'\,\boldsymbol{e_r}+\frac{Ra_\xi}{Sc}\tilde{g}\,\xi'\,\boldsymbol{e_r}+\frac{1}{Pm\,E\,\tilde{\rho}}\left(\boldsymbol{\nabla}\times\boldsymbol{B}\right)\times\boldsymbol{B}+\frac{1}{\tilde{\rho}}\boldsymbol{\nabla}\cdot\mathsf{S}, \tag{3.20}$$

$$\boldsymbol{\nabla}\cdot\tilde{\rho}\boldsymbol{u}=0, \tag{3.21}$$

$$\boldsymbol{\nabla}\cdot\boldsymbol{B}=0, \tag{3.22}$$

$$\tilde{\rho}\left(\frac{\partial \xi'}{\partial t}+\boldsymbol{u}\cdot\boldsymbol{\nabla}\xi'\right)=\frac{1}{Sc}\boldsymbol{\nabla}\cdot\left(\kappa_\xi(r)\tilde{\rho}\boldsymbol{\nabla}\xi'\right) \tag{3.23}$$

$$\frac{\partial \boldsymbol{B}}{\partial t}=\boldsymbol{\nabla}\times\left(\boldsymbol{u}\times\boldsymbol{B}\right)-\frac{1}{Pm}\boldsymbol{\nabla}\times\left(\lambda(r)\,\boldsymbol{\nabla}\times\boldsymbol{B}\right). \tag{3.24}$$

Here $\tilde{g}$ and $\tilde{\rho}$ are the normalized radial gravity and density profiles that reach one at the outer boundary.

3.3.3 Entropy equation and turbulent diffusion

The entropy equation usually requires an additional assumption in most of the existing anelastic approximations. Indeed, if one simply expands Eq. (3.4) with the classical temperature diffusion an operator of the form:

$$\epsilon\,\boldsymbol{\nabla}\cdot\left(K\boldsymbol{\nabla}T'\right)+\boldsymbol{\nabla}\cdot\left(K\boldsymbol{\nabla}\tilde{T}\right),$$

will remain the right-hand side of the equation. At first glance, there seems to be a $1/\epsilon$ factor between the first term and the second one, which would suggest to keep only the second term in this expansion. However, for astrophysical objects which exhibit strong convective driving (and hence large Rayleigh numbers), the diffusion of the adiabatic background is actually very small and may be comparable or even smaller in magnitude than the ϵ terms representing the usual convective perturbations. For the Earth core for instance, if one assumes that the typical temperature fluctuations are of the order of 1 mK and the temperature contrast between the inner and outer core is of the order of 1000 K, then $\epsilon\sim 10^{-6}$. The ratio of the two terms can thus be estimated as

$$\epsilon\frac{T'/\delta^2}{T/d^2}, \tag{3.25}$$

where d is the thickness of the inner core and δ is the typical thermal boundary layer thickness. This ratio is exactly one when $\delta=1$ m, a plausible value for the Earth inner core.

In numerical simulations however, the over-estimated diffusivities restrict the computational capabilities to much lower Rayleigh numbers. As a consequence, the actual boundary layers in a global DNS will be much thicker and the ratio (3.25) will be much smaller than unity. The second terms will thus effectively acts as a radial-dependent heat source or sink that will drive or hinder convection. This is one of the physical motivation to rather introduce a **turbulent diffusivity** that will be approximated by

$$\kappa\tilde{\rho}\tilde{T}\boldsymbol{\nabla}s,$$

where κ is the turbulent diffusivity. **Entropy diffusion is assumed to dominate over temperature diffusion in turbulent flows**.

The choice of the entropy scale to non-dimensionalize Eq. (3.4) also depends on the nature of the boundary conditions: it can be simply the entropy contrast over the layer Δs when the entropy is held constant at both boundaries, or $d\,(ds/dr)$ when flux-based boundary conditions are employed. We will restrict to the first option in the following, but keep in mind that depending on your setup, the entropy reference scale (and thus the Rayleigh number definition) might change.

$$\tilde{\rho}\tilde{T}\left(\frac{\partial s'}{\partial t}+\boldsymbol{u}\cdot\boldsymbol{\nabla}s'\right)=\frac{1}{Pr}\boldsymbol{\nabla}\cdot\left(\kappa(r)\tilde{\rho}\tilde{T}\boldsymbol{\nabla}s'\right)+\frac{Pr\,Di}{Ra}\Phi_\nu+\frac{Pr\,Di}{Pm^2\,E\,Ra}\lambda(r)\left(\boldsymbol{\nabla}\times\boldsymbol{B}\right)^2, \tag{3.26}$$

A comparison with (3.20) reveals meaning of the different non-dimensional numbers that scale viscous and Ohmic heating. The fraction Pr/Ra simply expresses the ratio of entropy and flow in the Navier-Stokes equation, while the additional factor $1/EPm$ reflects the scale difference of magnetic field and flow. Then remaining dissipation number Di then expresses the relative importance of viscous and Ohmic heating compared to buoyancy and Lorentz force in the Navier-Stokes equation. For small Di both heating terms can be neglected compared to entropy changes due to advection, an limit that is used in the Boussinesq approximation.

3.4 Dimensionless control parameters

The equations (3.20)-(3.26) are governed by four dimensionless numbers: the Ekman number

$$E=\frac{\nu}{\Omega d^2}, \tag{3.27}$$

the thermal Rayleigh number

$$Ra=\frac{\alpha_o g_o T_o d^3\Delta s}{c_p\kappa_o\nu_o}, \tag{3.28}$$

the compositional Rayleigh number

$$Ra_\xi=\frac{\delta_o g_o d^3\Delta\xi}{\kappa_\xi\nu_o}, \tag{3.29}$$

the Prandtl number

$$Pr=\frac{\nu_o}{\kappa_o}, \tag{3.30}$$

the Schmidt number

$$Sc=\frac{\nu_o}{\kappa_\xi}, \tag{3.31}$$

and the magnetic Prandtl number

$$Pm=\frac{\nu_o}{\lambda_i}. \tag{3.32}$$

In addition to these four numbers, the reference state is controlled by the geometry of the spherical shell given by its radius ratio

$$\eta=\frac{r_i}{r_o}, \tag{3.33}$$

and the background density and temperature profiles, either controlled by Di or by N_ρ and m.

In the Boussinesq approximation all physical properties are assumed to be homogeneous and we can drop the sub-indices o and i except for gravity. Moreover, the Rayleigh number can be expressed in terms of the temperature jump across the shell:

$$Ra = \frac{\alpha g_o d^3 \Delta T}{\kappa \nu}. \tag{3.34}$$

See also:

In MagIC, those control parameters can be adjusted in the *&phys_param* section of the input namelist.

Variants of the non-dimensional equations and control parameters result from different choices for the fundamental scales. For the length scale often r_o is chosen instead of d. Other natural scales for time are the magnetic or the thermal diffusion time, or the rotation period. There are also different options for scaling the magnetic field strength. The prefactor of two, which is retained in the Coriolis term in (3.20), is often incorporated into the definition of the Ekman number.

See also:

Those references timescales and length scales can be adjusted by several input parameters in the *&control* section of the input namelist.

3.4.1 Usual diagnostic quantities

Characteristic properties of the solution are usually expressed in terms of non-dimensional diagnostic parameters. In the context of the geodynamo for instance, the two most important ones are the magnetic Reynolds number Rm and the Elsasser number Λ. Usually the rms-values of the velocity u_{rms} and of the magnetic field B_{rms} inside the spherical shell are taken as characteristic values. The magnetic Reynolds number

$$Rm = \frac{u_{rms} d}{\lambda_i}$$

can be considered as a measure for the flow velocity and describes the ratio of advection of the magnetic field to magnetic diffusion. Other characteristic non-dimensional numbers related to the flow velocity are the (hydrodynamic) Reynolds number

$$Re = \frac{u_{rms} d}{\nu_o},$$

which measures the ratio of inertial forces to viscous forces, and the Rossby number

$$Ro = \frac{u_{rms}}{\Omega d},$$

a measure for the ratio of inertial to Coriolis forces.

$$\Lambda = \frac{B_{rms}^2}{\mu_0 \lambda_i \rho_o \Omega}$$

measures the ratio of Lorentz to Coriolis forces and is equivalent to the square of the non-dimensional magnetic field strength in the scaling chosen here.

See also:

The time-evolution of these diagnostic quantities are stored in the *par.TAG* file produced during the run of MagIC.

3.5 Boundary conditions and treatment of inner core

3.5.1 Mechanical conditions

In its simplest form, when modelling the geodynamo, the fluid shell is treated as a container with rigid, impenetrable, and co-rotating walls. This implies that within the rotating frame of reference all velocity components vanish at r_o and r_i. In case of modelling the free surface of a gas giant planets or a star, it is preferable to rather replace the condition of zero horizontal velocity by one of vanishing viscous shear stresses (the so-called free-slip condition).

Furthermore, even in case of modelling the liquid iron core of a terrestrial planet, there is no a priori reason why the inner core should necessarily co-rotate with the mantle. Some models for instance allow for differential rotation of the inner core and mantle with respect to the reference frame. The change of rotation rate is determined from the net torque. Viscous, electromagnetic, and torques due to gravitational coupling between density heterogeneities in the mantle and in the inner core contribute.

See also:

The mechanical boundary conditions can be adjusted with the parameters *ktopv* and *kbotv* in the *&phys_param* section of the input namelist.

3.5.2 Magnetic boundary conditions and inner core conductivity

When assuming that the fluid shell is surrounded by electrically insulating regions (inner core and external part), the magnetic field inside the fluid shell matches continuously to a potential field in both the exterior and the interior regions. Alternative magnetic boundary conditions (like cancellation of the horizontal component of the field) are also possible.

Depending on the physical problem you want to model, treating the inner core as an insulator is not realistic either, and it might instead be more appropriate to assume that it has the same electrical conductivity as the fluid shell. In this case, an equation equivalent to (3.24) must be solved for the inner core, where the velocity field simply describes the solid body rotation of the inner core with respect to the reference frame. At the inner core boundary a continuity condition for the magnetic field and the horizontal component of the electrical field apply.

See also:

The magnetic boundary conditions can be adjusted with the parameters *ktopb* and *kbotb* in the *&phys_param* section of the input namelist.

3.5.3 Thermal boundary conditions and distribution of buoyancy sources

In many dynamo models, convection is simply driven by an imposed fixed super-adiabatic entropy contrast between the inner and outer boundaries. This approximation is however not necessarily the best choice, since for instance, in the present Earth, convection is thought to be driven by a combination of thermal and compositional buoyancy. Sources of heat are the release of latent heat of inner core solidification and the secular cooling of the outer and inner core, which can effectively be treated like a heat source. The heat loss from the core is controlled by the convecting mantle, which effectively imposes a condition of fixed heat flux at the core-mantle boundary on the dynamo. The heat flux is in that case spatially and temporally variable.

See also:

The thermal boundary conditions can be adjusted with the parameters *ktops* and *kbots* in the *&phys_param* section of the input namelist.

3.5.4 Chemical composition boundary conditions

They are treated in a very similar manner as the thermal boundary conditions

See also:

The boundary conditions for composition can be adjusted with the parameters *ktopxi* and *kbotxi* in the *&phys_param* section of the input namelist.

CHAPTER

FOUR

NUMERICAL TECHNIQUE

MagIC is a pseudo-spectral MHD code. This numerical technique was originally developed by P. Gilman and G. Glatzmaier for the spherical geometry. In this approach the unknowns are expanded into complete sets of functions in radial and angular directions: **Chebyshev polynomials or Finite differences in the radial direction** and **spherical harmonic functions in the azimuthal and latitudinal directions**. This allows to express all partial derivatives analytically. Employing orthogonality relations of spherical harmonic functions and using collocation in radius then lead to algebraic equations that are integrated in time with a **mixed implicit/explicit time stepping scheme**. The nonlinear terms and the Coriolis force are evaluated in the physical (or grid) space rather than in spectral space. Although this approach requires costly numerical transformations between the two representations (from spatial to spectral using Legendre and Fourier transforms), the resulting decoupling of all spherical harmonic modes leads to a net gain in computational speed. Before explaining these methods in more detail, we introduce the poloidal/toroidal decomposition.

4.1 Poloidal/toroidal decomposition

Any vector $\boldsymbol{v}$ that fulfills $\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$, i.e. a so-called *solenoidal field*, can be decomposed in a poloidal and a toroidal part W and Z, respectively

$$\boldsymbol{v} = \boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times W\, \boldsymbol{e_r}) + \boldsymbol{\nabla} \times Z\, \boldsymbol{e_r}.$$

Three unknown vector components are thus replaced by two scalar fields, the poloidal potential W and the toroidal potential Z. This decomposition is unique, aside from an arbitrary radial function $f(r)$ that can be added to W or Z without affecting $\boldsymbol{v}$.

In the anelastic approximation, such a decomposition can be used for the mass flux $\tilde{\rho}\boldsymbol{u}$ and the magnetic field $\boldsymbol{B}$. This yields

$$\boxed{\begin{aligned} \tilde{\rho}\boldsymbol{u} &= \boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times W\, \boldsymbol{e_r}) + \boldsymbol{\nabla} \times Z\, \boldsymbol{e_r}\,, \\ \boldsymbol{B} &= \boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times g\, \boldsymbol{e_r}) + \boldsymbol{\nabla} \times h\, \boldsymbol{e_r}\,. \end{aligned}} \tag{4.1}$$

The two scalar potentials of a divergence free vector field can be extracted from its radial component and the radial component of its curl using the fact that the toroidal field has not radial component:

$$\begin{aligned} \boldsymbol{e_r} \cdot \tilde{\rho}\boldsymbol{u} &= -\Delta_H\, W, \\ \boldsymbol{e_r} \cdot (\boldsymbol{\nabla} \times \boldsymbol{u}) &= -\Delta_H\, Z, \end{aligned} \tag{4.2}$$

where the operator Δ_H denotes the horizontal part of the Laplacian:

$$\Delta_H = \frac{1}{r^2 \sin\theta} \frac{\partial}{\partial \theta} \left(\sin\theta \frac{\partial}{\partial \theta} \right) + \frac{1}{r^2 \sin^2\theta} \frac{\partial^2}{\partial^2 \phi}. \tag{4.3}$$

The equation (4.1) can be expanded in spherical coordinates. The three components of $\tilde{\rho}\boldsymbol{u}$ are given by

$$\tilde{\rho}\boldsymbol{u} = -(\Delta_H W)\,\boldsymbol{e_r} + \left(\frac{1}{r}\frac{\partial^2 W}{\partial r\partial\theta} + \frac{1}{r\sin\theta}\frac{\partial Z}{\partial\phi}\right)\boldsymbol{e_\theta} + \left(\frac{1}{r\sin\theta}\frac{\partial^2 W}{\partial r\partial\phi} - \frac{1}{r}\frac{\partial Z}{\partial\theta}\right)\boldsymbol{e_\phi}, \tag{4.4}$$

while the curl of $\tilde{\rho}\boldsymbol{u}$ is expressed by

$$\begin{aligned}\boldsymbol{\nabla}\times\tilde{\rho}\boldsymbol{u} = &-(\Delta_H Z)\,\boldsymbol{e_r} + \left[-\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\left(\frac{\partial^2}{\partial r^2}+\Delta_H\right)W + \frac{1}{r}\frac{\partial^2 Z}{\partial r\partial\theta}\right]\boldsymbol{e_\theta}\\ &+\left[\frac{1}{r}\frac{\partial}{\partial\theta}\left(\frac{\partial^2}{\partial r^2}+\Delta_H\right)W + \frac{1}{r\sin\theta}\frac{\partial^2 Z}{\partial r\partial\phi}\right]\boldsymbol{e_\phi},\end{aligned} \tag{4.5}$$

Using the horizontal part of the divergence operator

$$\boldsymbol{\nabla}_H = \frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\sin\theta\;\boldsymbol{e}_\theta + \frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\;\boldsymbol{e}_\phi$$

above expressions can be simplified to

$$\tilde{\rho}\boldsymbol{u} = -\Delta_H\;\boldsymbol{e_r}\;W + \boldsymbol{\nabla}_H\frac{\partial}{\partial r}\;W + \boldsymbol{\nabla}_H\times\boldsymbol{e}_r\;Z$$

and

$$\nabla\times\tilde{\rho}\boldsymbol{u} = -\Delta_H\;\boldsymbol{e}_r\;Z + \boldsymbol{\nabla}_H\frac{\partial}{\partial r}\;Z - \boldsymbol{\nabla}_H\times\Delta_H\boldsymbol{e}_r\;W\quad.$$

Below we will use the fact that the horizontal components of the poloidal field depend on the radial derivative of the poloidal potential.

4.2 Spherical harmonic representation

Spherical harmonic functions Y_ℓ^m are a natural choice for the horizontal expansion in colatitude θ and longitude ϕ:

$$Y_\ell^m(\theta,\phi) = P_\ell^m(\cos\theta)\,e^{im\phi},$$

where ℓ and m denote spherical harmonic degree and order, respectively, P_ℓ^m is an associated Legendre function. Different normalization are in use. Here we adopt a complete normalization so that the orthogonality relation reads

$$\int_0^{2\pi} d\,\phi\int_0^\pi \sin\theta\,d\theta\;Y_\ell^m(\theta,\phi)\,Y_{\ell'}^{m'}(\theta,\phi)\;=\;\delta_{\ell\ell'}\delta^{mm'}. \tag{4.6}$$

This means that

$$Y_\ell^m(\theta,\phi) = \left(\frac{(2\ell+1)}{4\pi}\frac{(\ell-|m|)!}{(\ell+|m|)!}\right)^{1/2}P_\ell^m(\cos\theta)\,e^{im\phi},$$

As an example, the spherical harmonic representation of the magnetic poloidal potential $g(r,\theta,\phi)$, truncated at degree and order ℓ_{max}, then reads

$$g(r,\theta,\phi) = \sum_{\ell=0}^{\ell_{max}}\sum_{m=-\ell}^{\ell} g_{\ell m}(r)\,Y_\ell^m(\theta,\phi), \tag{4.7}$$

with

$$g_{\ell m}(r) = \frac{1}{\pi}\int_0^\pi d\theta\sin\theta\;g_m(r,\theta)\;P_\ell^m(\cos\theta), \tag{4.8}$$

$$g_m(r,\theta) = \frac{1}{2\pi}\int_0^{2\pi} d\phi\; g(r,\theta,\phi)\; e^{-im\phi}. \tag{4.9}$$

The potential $g(r,\theta,\phi)$ is a real function so that $g^\star_{\ell m}(r) = g_{\ell,-m}(r)$, where the asterisk denotes the complex conjugate. Thus, only coefficients with $m \geq 0$ have to be considered. The same kind of expansion is made for the toroidal magnetic potential, the mass flux potentials, pressure, entropy (or temperature) and chemical composition.

The equations (4.8) and (4.9) define a two-step transform from the longitude/latitude representation to the spherical harmonic representation $(r,\theta,\phi) \longrightarrow (r,\ell,m)$. The equation (4.7) formulates the inverse procedure $(r,\ell,m) \longrightarrow (r,\theta,\phi)$. Fast-Fourier transforms are employed in the longitudinal direction, requiring (at least) $N_\phi = 2\ell_{max}+1$ evenly spaced grid points ϕ_i. MagIC relies on the Gauss-Legendre quadrature for evaluating the integral (4.8)

$$g_{\ell m}(r) = \frac{1}{N_\theta}\sum_{j=1}^{N_\theta} w_j\, g_m(r,\theta_j)\; P_\ell^m(\cos\theta_j),$$

where θ_j are the N_θ Gaussian quadrature points defining the latitudinal grid, and w_j are the respective weights. Pre-stored values of the associated Legendre functions at grid points θ_j in combination with a FFT in ϕ provide the inverse transform (4.7). Generally, $N_\phi = 2N_\theta$ is chosen, which provides isotropic resolution in the equatorial region. Choosing $\ell_{max} = [\min(2N_\theta, N_\phi) - 1]/3$ prevents aliasing errors.

See also:

In MagIC, the Legendre functions are defined in the subroutine `plm_theta`. The Legendre transforms from spectral to grid space are computed in the module `legendre_spec_to_grid`, while the backward transform (from grid space to spectral space) are computed in the module `legendre_grid_to_spec`. The fast Fourier transforms are computed in the module `fft`.

4.2.1 Special recurrence relations

The action of a horizontal Laplacian (4.3) on spherical harmonics can be analytically expressed by

$$\boxed{\Delta_H Y_\ell^m = -\frac{\ell(\ell+1)}{r^2}\, Y_\ell^m\,.} \tag{4.10}$$

They are several useful recurrence relations for the Legendre polynomials that will be further employed to compute Coriolis forces and the θ and ϕ derivatives of advection and Lorentz forces. Four of these operators are used in **MagIC**. The first one is defined by

$$\vartheta_1 = \frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\sin^2\theta = \sin\theta\frac{\partial}{\partial\theta} + 2\cos\theta\,.$$

The action of this operator on a Legendre polynomials is given by

$$\vartheta_1 = (\ell+2)\, c_{\ell+1}^m\, P_{\ell+1}^m(\cos\theta) - (\ell-1)\, c_\ell^m\, P_{\ell-1}^m(\cos\theta),$$

where c_ℓ^m is defined by

$$c_\ell^m = \sqrt{\frac{(\ell+m)(\ell-m)}{(2\ell-1)(2\ell+1)}}\,. \tag{4.11}$$

How is that implemented in the code? Let's assume we want the spherical harmonic contribution of degree ℓ and order *m* for the expression

$$\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta\, f(\theta))\,.$$

In order to employ the operator ϑ_1 for the derivative, we thus define a new function

$$F(\theta) = f(\theta)/\sin\theta\,,$$

so that

$$\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}[\sin\theta\, f(\theta)] = \vartheta_1 F(\theta)\,.$$

Expanding $F(\theta)$ in Legendre polynomials and using the respective orthogonality relation we can then map out the required contribution in the following way:

$$\boxed{\int_0^\pi d\theta\, \sin\theta\, P_\ell^m \vartheta_1 \sum_{\ell'} F_{\ell'}^m P_{\ell'}^m = (\ell+1)\, c_\ell^m\, F_{\ell-1}^m - \ell\, c_{\ell+1}^m\, F_{\ell+1}^m\,.} \tag{4.12}$$

Here, we have assumed that the Legendre functions are completely normalized such that

$$\int_0^\pi d\theta\, \sin\theta\, P_\ell^m P_{\ell'}^m = \delta_{\ell\ell'}\,.$$

See also:

This operator is defined in the module `horizontal_data` by the variables `dTheta1S` for the first part of the right-hand side of (4.12) and `dTheta1A` for the second part.

The second operator used to formulate colatitude derivatives is

$$\vartheta_2 = \sin\theta\frac{\partial}{\partial\theta}\,.$$

The action of this operator on the Legendre polynomials reads

$$\vartheta_2 P_\ell^m(\cos\theta) = \ell\, c_{\ell+1}^m\, P_{\ell+1}^m(\cos\theta) - (\ell+1)\, c_\ell^m\, P_{\ell-1}^m(\cos\theta)\,,$$

so that

$$\boxed{\int_0^\pi d\theta\, \sin\theta\, P_\ell^m \vartheta_2 \sum_{\ell'} f_{\ell'}^m P_{\ell'}^m = (\ell-1)\, c_\ell^m\, f_{\ell-1}^m - (\ell+2)\, c_{\ell+1}^m\, f_{\ell+1}^m\,.} \tag{4.13}$$

See also:

This operator is defined in the module `horizontal_data` by the variables `dTheta2S` for the first part of the right-hand side of (4.13) and `dTheta2A` for the second part.

The third combined operator is defined by:

$$\vartheta_3 = \sin\theta\frac{\partial}{\partial\theta} + \cos\theta\, L_H\,,$$

where $-L_H/r^2 = \Delta_H$.

Acting with ϑ_3 on a Legendre function gives:

$$\vartheta_3 P_\ell^m(\cos\theta) = \ell(\ell+1)\, c_{\ell+1}^m\, P_{\ell+1}^m(\cos\theta) + (\ell-1)(\ell+1)\, c_\ell^m\, P_{\ell-1}^m(\cos\theta)\,,$$

which results into:

$$\boxed{\int_0^\pi d\theta\, \sin\theta\, P_\ell^m \vartheta_3 \sum_{\ell'} f_{\ell'}^m P_{\ell'}^m = (\ell-1)(\ell+1)\, c_\ell^m\, f_{\ell-1}^m + \ell(\ell+2)\, c_{\ell+1}^m\, f_{\ell+1}^m\,.} \tag{4.14}$$

See also:

This operator is defined in the module `horizontal_data` by the variables `dTheta3S` for the first part of the right-hand side of (4.14) and `dTheta3A` for the second part.

The fourth (and last) combined operator is defined by:

$$\vartheta_4 = \frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\sin^2\theta\, L_H = \vartheta 1\, L_H\,,$$

Acting with ϑ_3 on a Legendre function gives:

$$\vartheta_4 P_\ell^m(\cos\theta) = \ell(\ell+1)(\ell+2)\, c_{\ell+1}^m\, P_{\ell+1}^m(\cos\theta) - \ell(\ell-1)(\ell+1)\, c_\ell^m\, P_{\ell-1}^m(\cos\theta)\,,$$

which results into:

$$\boxed{\int_0^\pi d\theta\, \sin\theta\, P_\ell^m \vartheta_4 \sum_{\ell'} f_{\ell'}^m P_{\ell'}^m = \ell(\ell-1)(\ell+1)\, c_\ell^m\, f_{\ell-1}^m - \ell(\ell+1)(\ell+2)\, c_{\ell+1}^m\, f_{\ell+1}^m\,.} \tag{4.15}$$

See also:

This operator is defined in the module `horizontal_data` by the variables `dTheta4S` for the first part of the right-hand side of (4.15) and `dTheta4A` for the second part.

4.3 Radial representation

In MagIC, the radial dependencies are expanded into complete sets of functions: the Chebyshev polynomials $\mathcal{C}(x)$. The polynomial of degree n is defined by

$$\mathcal{C}_n(x) = \cos\left[n\, \arccos(x)\right] \quad -1 \le x \le 1\,.$$

When truncating at degree N, the radial expansion of the poloidal magnetic potential reads

$$g_{\ell m}(r) = \sum_{n=0}^{N} g_{\ell m n}\, \mathcal{C}_n(r), \tag{4.16}$$

with

$$g_{\ell m n} = \frac{2-\delta_{n0}}{\pi}\int_{-1}^{1} \frac{dx\, g_{\ell m}(r(x))\, \mathcal{C}_n(x)}{\sqrt{1-x^2}}. \tag{4.17}$$

The Chebyshev definition space $(-1 \le x \le 1)$ is then linearly mapped onto a radius range $(r_i \le r \le r_o)$ by

$$x(r) = 2\frac{r-r_i}{r_o-r_i} - 1. \tag{4.18}$$

In addition, nonlinear mapping can be defined to modify the radial dependence of the grid-point density.

When choosing the N_r extrema of $\mathcal{C}_{N_r-1}$ as radial grid points,

$$x_k = \cos\left(\pi\frac{(k-1)}{N_r-1}\right) \quad , \quad k = 1, 2, \ldots, N_r, \tag{4.19}$$

the values of the Chebyshev polynomials at these points are simply given by the cosine functions:

$$\mathcal{C}_{nk} = \mathcal{C}_n(x_k) = \cos\left(\pi\frac{n(k-1)}{N_r-1}\right).$$

This particular choice has two advantages. For one, the grid points become denser toward the inner and outer radius and better resolve potential thermal and viscous boundary layers. In addition, FFTs can be employed to switch between grid representation (4.16) and Chebyshev representations (4.17), rendering this procedure a fast-Chebyshev transform. Choosing $N_r > N$ provides radial dealiasing.

See also:

The Chebyshev (Gauss-Lobatto) grid is defined in the module `chebyshev_polynoms_mod`. The cosine transforms are computed in the modules `cosine_transform_even` and `fft_fac_mod`.

4.4 Spectral equations

We have now introduced the necessary tools for deriving the spectral equations. Taking the **radial components** of the Navier-Stokes equation and the induction equation provides the equations for the poloidal potentials $W(r,\theta,\phi)$ and $g(r,\theta,\phi)$. The **radial component of the curl** of these equations provides the equations for the toroidal counterparts $Z(r,\theta,\phi)$ and $h(r,\theta,\phi)$. The pressure remains an additional unknown. Hence one more equation involving $W_{\ell mn}$ and $p_{\ell mn}$ is required. It is obtained by taking the **horizontal divergence** of the Navier-Stokes equation.

Expanding all potentials in spherical harmonics and Chebyshev polynomials, multiplying with $Y_\ell^{m\star}$, and integrating over spherical surfaces (while making use of the orthogonality relation (4.6) results in equations for the coefficients $W_{\ell mn}$, $Z_{\ell mn}$, $g_{\ell mn}$, $h_{\ell mn}$, $P_{\ell mn}$ and $s_{\ell mn}$, respectively.

4.4.1 Equation for the poloidal potential W

The temporal evolution of W is obtained by taking $\boldsymbol{e_r}\cdot$ of each term entering the Navier-Stokes equation. For the time-derivative, one gets using (4.2):

$$\tilde{\rho}\boldsymbol{e_r}\cdot\left(\frac{\partial \boldsymbol{u}}{\partial t}\right)=\frac{\partial}{\partial t}(\boldsymbol{e_r}\cdot\tilde{\rho}\boldsymbol{u})=-\Delta_H\frac{\partial W}{\partial t}.$$

For the viscosity term, one gets

$$\begin{aligned}\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\cdot\mathsf{S}=&-\nu\,\Delta_H\left[\frac{\partial^2 W}{\partial r^2}+\left\{2\frac{d\ln\nu}{dr}-\frac{1}{3}\frac{d\ln\tilde{\rho}}{dr}\right\}\frac{\partial W}{\partial r}\right.\\&\left.-\left\{-\Delta_H+\frac{4}{3}\left(\frac{d^2\ln\tilde{\rho}}{dr^2}+\frac{d\ln\nu}{dr}\frac{d\ln\tilde{\rho}}{dr}+\frac{1}{r}\left[3\frac{d\ln\nu}{dr}+\frac{d\ln\tilde{\rho}}{dr}\right]\right)\right\}W\right],\end{aligned}$$

Note: In case of a constant kinematic viscosity, the $d\ln\nu/dr$ terms vanish. If in addition,the background density is constant, the $d\ln\tilde{\rho}/dr$ terms also vanish. In that Boussinesq limit, this viscosity term would then be simplified as

$$\boldsymbol{e_r}\cdot\Delta\boldsymbol{u}=-\Delta_H\left[\frac{\partial^2 W}{\partial r^2}+\Delta_H W\right].$$

Using Eq. (4.10) then allows to finally write the time-evolution equation for the poloidal potential $W_{\ell mn}$:

$$\begin{aligned}E\,\frac{\ell(\ell+1)}{r^2}&\left[\left\{\frac{\partial}{\partial t}+\nu\,\frac{\ell(\ell+1)}{r^2}+\frac{4}{3}\nu\left(\frac{d^2\ln\tilde{\rho}}{dr^2}+\frac{d\ln\nu}{dr}\frac{d\ln\tilde{\rho}}{dr}+\frac{1}{r}\left[3\frac{d\ln\nu}{dr}+\frac{d\ln\tilde{\rho}}{dr}\right]\right)\right\}\mathcal{C}_n\right.\\&\qquad-\nu\left\{2\frac{d\ln\nu}{dr}-\frac{1}{3}\frac{d\ln\tilde{\rho}}{dr}\right\}\mathcal{C}_n'\\&\left.\qquad-\nu\,\mathcal{C}_n''\right]\qquad W_{\ell mn}\\&\qquad+\left[\mathcal{C}_n'-\frac{d\ln\tilde{\rho}}{dr}\mathcal{C}_n\right]\qquad P_{\ell mn}\\&\qquad-\left[\frac{Ra\,E}{Pr}\,\tilde{\rho}\,g(r)\right]\mathcal{C}_n\qquad s_{\ell mn}\\&\qquad-\left[\frac{Ra_\xi\,E}{Sc}\,\tilde{\rho}\,g(r)\right]\mathcal{C}_n\qquad \xi_{\ell mn}\\&=\mathcal{N}_{\ell m}^W=\int d\Omega\,Y_\ell^{m\star}\,\mathcal{N}^W=\int d\Omega\,Y_\ell^{m\star}\,\boldsymbol{e_r}\cdot\boldsymbol{F}\,.\end{aligned}\tag{4.20}$$

Here, $d\Omega$ is the spherical surface element. We use the summation convention for the Chebyshev index n. The radial derivatives of Chebyshev polynomials are denoted by primes.

See also:

The exact computation of the linear terms of (4.20) are coded in the subroutines `get_wpMat`

4.4.2 Equation for the toroidal potential Z

The temporal evolution of Z is obtained by taking the radial component of the curl of the Navier-Stokes equation (i.e. $\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\times$). For the time derivative, one gets using (4.2):

$$\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\times\left(\frac{\partial\tilde{\rho}\boldsymbol{u}}{\partial t}\right)=\frac{\partial}{\partial t}(\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\times\tilde{\rho}\boldsymbol{u})=-\frac{\partial}{\partial t}(\Delta_H Z)=-\Delta_H\frac{\partial Z}{\partial t}\,.$$

The pressure gradient, one has

$$\boldsymbol{\nabla}\times\left[\tilde{\rho}\boldsymbol{\nabla}\left(\frac{p'}{\tilde{\rho}}\right)\right]=\boldsymbol{\nabla}\tilde{\rho}\times\boldsymbol{\nabla}\left(\frac{p'}{\tilde{\rho}}\right)+\underbrace{\tilde{\rho}\boldsymbol{\nabla}\times\left[\boldsymbol{\nabla}\left(\frac{p'}{\tilde{\rho}}\right)\right]}_{=0}\,.$$

This expression has no component along $\boldsymbol{e_r}$, as a consequence, there is no pressure gradient contribution here. The gravity term also vanishes as $\boldsymbol{\nabla}\times(\tilde{\rho}g(r)\boldsymbol{e_r})$ has no radial component.

$$\begin{aligned}\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\times[\boldsymbol{\nabla}\cdot\mathsf{S}]=&-\nu\,\Delta_H\left[\frac{\partial^2 Z}{\partial r^2}+\left(\frac{d\ln\nu}{dr}-\frac{d\ln\tilde{\rho}}{dr}\right)\frac{\partial Z}{\partial r}\right.\\&\left.-\left(\frac{d\ln\nu}{dr}\frac{d\ln\tilde{\rho}}{dr}+\frac{2}{r}\frac{d\ln\nu}{dr}+\frac{d^2\ln\tilde{\rho}}{dr^2}+\frac{2}{r}\frac{d\ln\tilde{\rho}}{dr}-\Delta_H\right)Z\right].\end{aligned}$$

Note: Once again, this viscous term can be greatly simplified in the Boussinesq limit:

$$\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\times(\Delta\boldsymbol{u})=-\Delta_H\left[\frac{\partial^2 Z}{\partial r^2}+\Delta_H Z\right]\,.$$

Using Eq. (4.10) then allows to finally write the time-evolution equation for the poloidal potential $Z_{\ell mn}$:

$$\begin{aligned}&E\,\frac{\ell(\ell+1)}{r^2}\left[\left\{\frac{\partial}{\partial t}+\nu\,\frac{\ell(\ell+1)}{r^2}+\nu\left(\frac{d\ln\nu}{dr}\frac{d\ln\tilde{\rho}}{dr}+\frac{2}{r}\frac{d\ln\nu}{dr}+\frac{d^2\ln\tilde{\rho}}{dr^2}+\frac{2}{r}\frac{d\ln\tilde{\rho}}{dr}\right)\right\}\mathcal{C}_n\right.\\&\qquad\qquad-\nu\left(\frac{d\ln\nu}{dr}-\frac{d\ln\tilde{\rho}}{dr}\right)\mathcal{C}_n'\\&\qquad\qquad\left.-\nu\,\mathcal{C}_n''\right]\quad Z_{\ell mn}\\&=\mathcal{N}^Z_{\ell m}=\int d\Omega\,Y_\ell^{m\star}\,\mathcal{N}^Z=\int d\Omega\,Y_\ell^{m\star}\,\boldsymbol{e_r}\cdot(\boldsymbol{\nabla}\times\boldsymbol{F})\,.\end{aligned}\tag{4.21}$$

See also:

The exact computation of the linear terms of (4.21) are coded in the subroutines `get_zMat`

4.4.3 Equation for pressure P

The evolution of equation for pressure is obtained by taking the horizontal divergence (i.e. $\boldsymbol{\nabla}_H \cdot$) of the Navier-Stokes equation. This operator is defined such that

$$\boldsymbol{\nabla}_H \cdot \boldsymbol{a} = r \sin \frac{\partial (\sin\theta\, a_\theta)}{\partial \theta} + r \sin \frac{\partial a_\phi}{\partial \phi}.$$

This relates to the total divergence via:

$$\boldsymbol{\nabla} \cdot \boldsymbol{a} = \frac{1}{r^2}\frac{\partial (r^2 a_r)}{\partial r} + \boldsymbol{\nabla}_H \cdot \boldsymbol{a}.$$

The time-derivative term is thus expressed by

$$\begin{aligned}
\boldsymbol{\nabla}_H \cdot \left(\tilde{\rho} \frac{\partial \boldsymbol{u}}{\partial t} \right) &= \frac{\partial}{\partial t} \left[\boldsymbol{\nabla}_H \cdot (\tilde{\rho} \boldsymbol{u}) \right], \\
&= \frac{\partial}{\partial t} \left[\boldsymbol{\nabla} \cdot (\tilde{\rho} \boldsymbol{u}) - \frac{1}{r^2}\frac{\partial (r^2 \tilde{\rho} u_r)}{\partial r} \right], \\
&= -\frac{\partial}{\partial t} \left[\frac{\partial (\tilde{\rho} u_r)}{\partial r} + \frac{2 \tilde{\rho} u_r}{r} \right], \\
&= \frac{\partial}{\partial t} \left[\frac{\partial (\Delta_H W)}{\partial r} + \frac{2}{r} \Delta_H W \right], \\
&= \Delta_H \frac{\partial}{\partial t} \left(\frac{\partial W}{\partial r} \right).
\end{aligned}$$

We note that the gravity term vanishes since $\boldsymbol{\nabla}_H \cdot (\tilde{\rho} g(r) \boldsymbol{e_r}) = 0$. Concerning the pressure gradient, one has

$$-\boldsymbol{\nabla}_H \cdot \left[\tilde{\rho} \boldsymbol{\nabla} \left(\frac{p'}{\tilde{\rho}} \right) \right] = -\left\{ \boldsymbol{\nabla} \cdot \left[\tilde{\rho} \boldsymbol{\nabla} \left(\frac{p'}{\tilde{\rho}} \right) \right] - \frac{1}{r^2}\frac{\partial}{\partial r} \left[r^2 \tilde{\rho} \frac{\partial}{\partial r} \left(\frac{p'}{\tilde{\rho}} \right) \right] \right\} = -\Delta_H\, p'.$$

The viscosity term then reads

$$\begin{aligned}
\boldsymbol{\nabla}_H \cdot (\boldsymbol{\nabla} \cdot \mathsf{S}) =& \nu\, \Delta_H \left[\frac{\partial^3 W}{\partial r^3} + \left(\frac{d \ln \nu}{dr} - \frac{d \ln \tilde{\rho}}{dr} \right) \frac{\partial^2 W}{\partial r^2} \right. \\
&- \left[\frac{d^2 \ln \tilde{\rho}}{dr^2} + \frac{d \ln \nu}{dr}\frac{d \ln \tilde{\rho}}{dr} + \frac{2}{r} \left(\frac{d \ln \nu}{dr} + \frac{d \ln \tilde{\rho}}{dr} \right) - \Delta_H \right] \frac{\partial W}{\partial r} \\
&\left. - \left(\frac{2}{3}\frac{d \ln \tilde{\rho}}{dr} + \frac{2}{r} + \frac{d \ln \nu}{dr} \right) \Delta_H\, W \right].
\end{aligned}$$

Note: Once again, this viscous term can be greatly simplified in the Boussinesq limit:

$$\boldsymbol{\nabla}_H \cdot (\Delta \boldsymbol{u}) = -\Delta_H \left[\frac{\partial^3 W}{\partial r^3} + \Delta_H \frac{\partial W}{\partial r} - \frac{2}{r} \Delta_H\, W \right].$$

Using Eq. (4.10) then allows to finally write the equation for the pressure $P_{\ell mn}$:

$$\begin{aligned}
& E\,\frac{\ell(\ell+1)}{r^2}\left[-\nu\left(\frac{2}{3}\frac{d\ln\tilde{\rho}}{dr}+\frac{2}{r}+\frac{d\ln\nu}{dr}\right)\frac{\ell(\ell+1)}{r^2}\,\mathcal{C}_n\right. \\
& \left\{\frac{\partial}{\partial t}+\nu\,\frac{\ell(\ell+1)}{r^2}+\nu\left[\frac{d^2\ln\tilde{\rho}}{dr^2}+\frac{d\ln\nu}{dr}\frac{d\ln\tilde{\rho}}{dr}+\frac{2}{r}\left(\frac{d\ln\nu}{dr}+\frac{d\ln\tilde{\rho}}{dr}\right)\right]\right\}\mathcal{C}'_n \\
& -\nu\left(\frac{d\ln\nu}{dr}-\frac{d\ln\tilde{\rho}}{dr}\right)\mathcal{C}''_n \\
& \left.-\nu\,\mathcal{C}'''_n\right] \quad W_{\ell mn} \\
& +\left[\frac{\ell(\ell+1)}{r^2}\right]\mathcal{C}_n \quad P_{\ell mn} \\
& =\mathcal{N}^P_{\ell m}=-\int d\Omega\,Y_\ell^{m\star}\,\mathcal{N}^P=-\int d\Omega\,Y_\ell^{m\star}\,\boldsymbol{\nabla}_H\cdot\boldsymbol{F}\,.
\end{aligned} \tag{4.22}$$

See also:

The exact computation of the linear terms of (4.22) are coded in the subroutines `get_wpMat`

Note: We note that the terms on the left hand side of (4.20), (4.21) and (4.22) resulting from the viscous term, the pressure gradient, the buoyancy term, and the explicit time derivative completely decouple in spherical harmonic degree and order.

The terms that do not decouple, namely Coriolis force, Lorentz force and advection of momentum, are collected on the right-hand side of (4.20), (4.21) and (4.22) into the forcing term $\boldsymbol{F}$:

$$\boldsymbol{F}=-2\,\tilde{\rho}\,\boldsymbol{e_z}\times\boldsymbol{u}-E\,\tilde{\rho}\,\boldsymbol{u}\cdot\boldsymbol{\nabla}\,\boldsymbol{u}+\frac{1}{Pm}\,(\boldsymbol{\nabla}\times\boldsymbol{B})\times\boldsymbol{B}\,. \tag{4.23}$$

Resolving $\boldsymbol{F}$ into potential functions is not required. Its numerical evaluation is discussed *below*.

4.4.4 Equation for entropy s

The equation for the entropy (or temperature in the Boussinesq limit) is given by

$$\begin{aligned}
& \frac{1}{Pr}\left[\left(Pr\frac{\partial}{\partial t}+\kappa\,\frac{\ell(\ell+1)}{r^2}\right)\mathcal{C}_n\right. \\
& -\kappa\left(\frac{d\ln\kappa}{dr}+\frac{d\ln\tilde{\rho}}{dr}++\frac{dln\tilde{T}}{dr}+\frac{2}{r}\right)\mathcal{C}'_n \\
& \left.-\kappa\,\mathcal{C}''_n\right] \quad s_{\ell mn} \\
& =\mathcal{N}^S_{\ell m}=\int d\Omega\,Y_\ell^{m\star}\,\mathcal{N}^S=\int d\Omega\,Y_\ell^{m\star}\left[-\boldsymbol{u}\cdot\boldsymbol{\nabla}s+\frac{Pr\,Di}{Ra}\frac{1}{\tilde{\rho}\tilde{T}}\left(\Phi_\nu+\frac{\lambda}{Pm^2\,E}\,j^2\right)\right]\,.
\end{aligned} \tag{4.24}$$

In this expression, $j=\boldsymbol{\nabla}\times\boldsymbol{B}$ is the current. Once again, the numerical evaluation of the right-hand-side (i.e. the non-linear terms) is discussed *below*.

See also:

The exact computation of the linear terms of (4.24) are coded in the subroutines `get_sMat`

4.4.5 Equation for chemical composition ξ

The equation for the chemical composition is given by

$$\begin{aligned} &\frac{1}{Sc}\left[\left(Sc\frac{\partial}{\partial t}+\kappa_\xi\frac{\ell(\ell+1)}{r^2}\right)\mathcal{C}_n\right. \\ &\quad -\kappa_\xi\left(\frac{d\ln\kappa_\xi}{dr}+\frac{d\ln\tilde{\rho}}{dr}++\frac{2}{r}\right)\mathcal{C}'_n \\ &\quad \left.-\kappa_\xi\,\mathcal{C}''_n\right] \quad \xi_{\ell mn} \\ &= \mathcal{N}^\xi_{\ell m} = \int d\Omega\, Y_\ell^{m\star}\,\mathcal{N}^\xi = \int d\Omega\, Y_\ell^{m\star}\,\left[-\boldsymbol{u}\cdot\boldsymbol{\nabla}\xi\right]. \end{aligned} \tag{4.25}$$

Once again, the numerical evaluation of the right-hand-side (i.e. the non-linear term) is discussed *below*.

See also:

The exact computation of the linear terms of (4.25) are coded in the subroutines `get_xiMat`

4.4.6 Equation for the poloidal magnetic potential g

The equation for the poloidal magnetic potential is the radial component of the dynamo equation since

$$\boldsymbol{e_r}\cdot\left(\frac{\partial\boldsymbol{B}}{\partial t}\right)=\frac{\partial}{\partial t}(\boldsymbol{e_r}\cdot\boldsymbol{B})=-\Delta_H\frac{\partial g}{\partial t}.$$

The spectral form then reads

$$\begin{aligned} &\frac{\ell(\ell+1)}{r^2}\left[\left(\frac{\partial}{\partial t}+\frac{1}{Pm}\lambda\frac{\ell(\ell+1)}{r^2}\right)\mathcal{C}_n\right. \\ &\quad \left.-\frac{1}{Pm}\lambda\,\mathcal{C}''_n\right] \quad g_{\ell mn} \\ &= \mathcal{N}^g_{\ell m} = \int d\Omega\, Y_\ell^{m\star}\,\mathcal{N}^g = \int d\Omega\, Y_\ell^{m\star}\,\boldsymbol{e_r}\cdot\boldsymbol{D}\,. \end{aligned} \tag{4.26}$$

See also:

The exact computation of the linear terms of (4.26) are coded in the subroutines `get_bMat`

4.4.7 Equation for the toroidal magnetic potential h

The equation for the toroidal magnetic field coefficient reads

$$\begin{aligned} &\frac{\ell(\ell+1)}{r^2}\left[\left(\frac{\partial}{\partial t}+\frac{1}{Pm}\lambda\frac{\ell(\ell+1)}{r^2}\right)\mathcal{C}_n\right. \\ &\quad -\frac{1}{Pm}\frac{d\lambda}{dr}\mathcal{C}'_n \\ &\quad \left.-\frac{1}{Pm}\lambda\,\mathcal{C}''_n\right] \quad h_{\ell mn} \\ &= \mathcal{N}^h_{\ell m} = \int d\Omega\, Y_\ell^{m\star}\,\mathcal{N}^h = \int d\Omega\, Y_\ell^{m\star}\,\boldsymbol{e_r}\cdot(\boldsymbol{\nabla}\times\boldsymbol{D})\,. \end{aligned} \tag{4.27}$$

See also:

The exact computation of the linear terms of (4.27) are coded in the subroutines `get_bMat`

Note: We note that the terms on the left hand side of (4.26) and (4.27) resulting from the magnetic diffusion term and the explicit time derivative completely decouple in spherical harmonic degree and order.

The dynamo term does not decouple:

$$\boldsymbol{D} = \boldsymbol{\nabla} \times (\boldsymbol{u} \times \boldsymbol{B}) \ . \tag{4.28}$$

We have now derived a full set of equations (4.20), (4.21), (4.22), (4.24), (4.26) and (4.27), each describing the evolution of a single spherical harmonic mode of the six unknown fields (assuming that the terms on the right hand side are given). Each equation couples $N+1$ Chebyshev coefficients for a given spherical harmonic mode (ℓ, m). Typically, a collocation method is employed to solve for the Chebyshev coefficients. This means that the equations are required to be exactly satisfied at $N-1$ grid points defined by the equations (4.18) and (4.19). Excluded are the points $r = r_i$ and $r = r_o$, where the *boundary conditions* provide additional constraints on the set of Chebyshev coefficients.

4.5 Time-stepping schemes

Implicit time stepping schemes theoretically offer increased stability and allow for larger time steps. However, fully implicit approaches have the disadvantage that the nonlinear-terms couple all spherical harmonic modes. The potential gain in computational speed is therefore lost at higher resolution, where one very large matrix has to be dealt with rather than a set of much smaller ones. Similar considerations hold for the Coriolis force, one of the dominating forces in the system and therefore a prime candidate for implicit treatment. However, the Coriolis term couples modes (ℓ, m, n) with $(\ell+1, m, n)$ and $(\ell-1, m, n)$ and also couples poloidal and toroidal flow potentials. An implicit treatment of the Coriolis term therefore also results in a much larger (albeit sparse) inversion matrix.

We consequently adopt in **MagIC** a mixed implicit/explicit algorithm. The general differential equation in time can be written in the form

$$\frac{\partial}{\partial t} y = \mathcal{I}(y,t) + \mathcal{E}(y,t), \quad y(t_o) = y_o \ . \tag{4.29}$$

where $\mathcal{I}$ denotes the terms treated in an implicit time step and $\mathcal{E}$ the terms treated explicitly, i.e. the nonlinear and Coriolis contributions. In MagIC, two families of time-stepping schemes are supported: IMEX multistep and IMEX multistage methods.

First of all, the IMEX multistep methods correpond to time schemes where the solution results from the combination of several previous steps (such as for instance the Crank-Nicolson/Adams-Bashforth scheme). In this case, a general k-step IMEX multistep scheme reads

$$\left(I - b_o^{\mathcal{I}} \delta t \, \mathcal{I}\right) y_{n+1} = \sum_{j=1}^{k} a_j y_{n+1-j} + \delta t \sum_{j=1}^{k} \left(b_j^{\mathcal{E}} \mathcal{E}_{n+1-j} + b_j^{\mathcal{I}} \mathcal{I}_{n+1-j}\right) \ ,$$

where I is the identity matrix. The vectors $\boldsymbol{a}$, $\boldsymbol{b}^{\mathcal{E}}$ and $\boldsymbol{b}^{\mathcal{I}}$ correspond to the weighting factors of an IMEX multistep scheme. For instance, the commonly-used second-order scheme assembled from the combination of a Crank-Nicolson for the implicit terms and a second-order Adams-Bashforth for the explicit terms (hereafter CNAB2) corresponds to the following vectors: $\boldsymbol{a} = (1, 0)$, $\boldsymbol{b}^{\mathcal{I}} = (1/2, 1/2)$ and $\boldsymbol{b}^{\mathcal{E}} = (3/2, -1/2)$ for a constant timestep size δt.

In addition to CNAB2, MagIC supports several semi-implicit backward differentiation schemes of second, third and fourth order that are known to have good stability properties (heareafter SBDF2, SBDF3 and SBDF4), a modified CNAB2 from Ascher et al. (1995) (termed MODCNAB) and the CNLF scheme (combination of Crank-Nicolson and Leap-Frog for the explicit terms).

MagIC also supports several IMEX Runge-Kutta multistage methods, frequently called DIRK, an acronym that stands for *Diagonally Implicit Runge Kutta*. For such schemes, the equation (4.29) is time-advanced from t_n to t_{n+1} by solving ν sub-stages

$$(I - a_{ii}^{\mathcal{I}} \delta t \mathcal{I}) y_i = y_n + \delta t \sum_{j=1}^{i-1} \left(a_{i,j}^{\mathcal{E}} \mathcal{E}_j + a_{i,j}^{\mathcal{I}} \mathcal{I}_j \right), \quad 1 \leq i \leq \nu,$$

where y_i is the intermediate solution at the stage i. The matrices $a_{i,j}^{\mathcal{E}}$ and $a_{i,j}^{\mathcal{I}}$ constitute the so-called Butcher tables that correspond to a DIRK scheme. MagIC supports several second and third order schemes: ARS222 and ARS443 from Ascher et al. (1997), LZ232 from Liu and Zou (2006), PC2 from Jameson et al. (1981) and BPR353 from Boscarino et al. (2013).

In the code the equation (4.29) is formulated for each unknown spectral coefficient (expect pressure) of spherical harmonic degree ℓ and order m and for each radial grid point r_k. Because non-linear terms and the Coriolis force are treated explicitly, the equations decouple for all spherical harmonic modes. The different radial grid points, however, couple via the Chebychev modes and form a linear algebraic system of equations that can be solved with standard methods for the different spectral contributions.

For example the respective system of equations for the poloidal magnetic potential g time advanced with a CNAB2 reads

$$\left(\mathcal{A}_{kn} - \frac{1}{2} \delta t \, \mathcal{I}_{kn} \right) g_{\ell mn}(t + \delta t) = \left(\mathcal{A}_{kn} + \frac{1}{2} \delta t \, \mathcal{I}_{kn} \right) g_{\ell mn}(t) + \frac{3}{2} \delta t \, \mathcal{E}_{k\ell m}(t) - \frac{1}{2} \delta t \, \mathcal{E}_{k\ell m}(t - \delta t) \tag{4.30}$$

with

$$\mathcal{A}_{kn} = \frac{\ell(\ell+1)}{r_k^2} \mathcal{C}_{nk} \ ,$$

$$\mathcal{I}_{kn} = \frac{\ell(\ell+1)}{r_k^2} \frac{1}{Pm} \left(\mathcal{C}''_{nk} - \frac{\ell(\ell+1)}{r_k^2} \mathcal{C}_{nk} \right) \ ,$$

and $\mathcal{C}_{nk} = \mathcal{C}_n(r_k)$. $\mathcal{A}_{kn}$ is a matrix that converts the poloidal field modes $g_{\ell mn}$ to the radial magnetic field $B_r(r_k, \ell, m)$ for a given spherical harmonic contribution.

Here k and n number the radial grid points and the Chebychev coefficients, respectively. Note that the Einstein sum convention is used for Chebychev modes n.

$\mathcal{I}_{kn}$ is the matrix describing the implicit contribution which is purely diffusive here. Neither $\mathcal{A}_{kn}$ nor $\mathcal{I}_{kn}$ depend on time but the former needs to be updated when the time step δt is changed. The only explicit contribution is the nonlinear dynamo term

$$\mathcal{E}_{k\ell m}(t) = \mathcal{N}^g_{k\ell m} = \int d\Omega \, Y_\ell^{m\star} \, \boldsymbol{e_r} \cdot \boldsymbol{D}(t, r_k, \theta, \phi) \ .$$

$\mathcal{E}_{k\ell m}$ is a one dimensional vector for all spherical harmonic combinations ℓm.

Courant's condition offers a guideline concerning the value of δt, demanding that δt should be smaller than the advection time between two grid points. Strong Lorentz forces require an additional stability criterion that is obtained by replacing the flow speed by Alfvén's velocity in a modified Courant criterion. The explicit treatment of the Coriolis force requires that the time step is limited to a fraction of the rotation period, which may be the relevant criterion at low Ekman number when flow and magnetic field remain weak. Non-homogeneous grids and other numerical effects generally require an additional safety factor in the choice of δt.

4.6 Coriolis force and nonlinear terms

4.6.1 Nonlinear terms entering the equation for W

The nonlinear term $\mathcal{N}^W$ that enters the equation for the poloidal potential (4.20) contains the radial component of the advection, the Lorentz force and Coriolis force. In spherical coordinate, the two first contributions read:

$$\tilde{\rho}\left(\boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u}\right)=\begin{Bmatrix}\mathcal{A}_r\\ \mathcal{A}_\theta\\ \mathcal{A}_\phi\end{Bmatrix}=\begin{Bmatrix}-\tilde{\rho}\,E\left(u_r\dfrac{\partial u_r}{\partial r}+\dfrac{u_\theta}{r}\dfrac{\partial u_r}{\partial\theta}+\dfrac{u_\phi}{r\sin\theta}\dfrac{\partial u_r}{\partial\phi}-\dfrac{u_\theta^2+u_\phi^2}{r}\right)+\dfrac{1}{Pm}\left(j_\theta\,B_\phi-j_\phi\,B_\theta\right)\,,\\ -\tilde{\rho}\,E\left(u_r\dfrac{\partial u_\theta}{\partial r}+\dfrac{u_\theta}{r}\dfrac{\partial u_\theta}{\partial\theta}+\dfrac{u_\phi}{r\sin\theta}\dfrac{\partial u_\theta}{\partial\phi}+\dfrac{u_r u_\theta}{r}-\dfrac{\cos\theta}{r\sin\theta}u_\phi^2\right)+\dfrac{1}{Pm}\left(j_\phi\,B_r-j_r\,B_\phi\right)\,,\\ -\tilde{\rho}\,E\left(u_r\dfrac{\partial u_\phi}{\partial r}+\dfrac{u_\theta}{r}\dfrac{\partial u_\phi}{\partial\theta}+\dfrac{u_\phi}{r\sin\theta}\dfrac{\partial u_\phi}{\partial\phi}+\dfrac{u_r u_\phi}{r}+\dfrac{\cos\theta}{r\sin\theta}u_\theta u_\phi\right)+\dfrac{1}{Pm}\left(j_r\,B_\theta-j_\theta\,B_r\right)\,,\end{Bmatrix} \tag{4.31}$$

The Coriolis force can be expressed as a function of the potentials W and Z using (4.4)

$$2\tilde{\rho}\boldsymbol{e_r}\cdot\left(\boldsymbol{u}\times\boldsymbol{e_z}\right)=2\sin\theta\,\tilde{\rho}u_\phi=\frac{2}{r}\left(\frac{\partial^2 W}{\partial r\partial\phi}-\sin\theta\frac{\partial Z}{\partial\theta}\right)\,.$$

The nonlinear terms that enter the equation for the poloidal potential (4.20) thus reads:

$$\mathcal{N}^W=\frac{2}{r}\left(\frac{\partial^2 W}{\partial r\partial\phi}-\sin\theta\frac{\partial Z}{\partial\theta}\right)+\mathcal{A}_r\,.$$

The θ-derivative entering the radial component of the Coriolis force is thus the operator ϑ_2 defined in (4.12). Using the recurrence relation, one thus finally gets in spherical harmonic space:

$$\boxed{\mathcal{N}^W_{\ell m}=\frac{2}{r}\left[im\frac{\partial W^m_\ell}{\partial r}-(\ell-1)c^m_\ell Z^m_{\ell-1}+(\ell+2)c^m_{\ell+1}Z^m_{\ell+1}\right]+\mathcal{A}_r{}^m_\ell\,.} \tag{4.32}$$

To get this expression, we need to first compute $\mathcal{A}_r$ in the physical space. This term is computed in the subroutine `get_nl` in the module `grid_space_arrays_mod`. $\mathcal{A}_r$ is then transformed to the spectral space by using a Legendre and a Fourier transform to produce $\mathcal{A}_r{}^m_\ell$.

See also:

The final calculations of (4.32) are done in the subroutine `get_td`.

4.6.2 Nonlinear terms entering the equation for Z

The nonlinear term $\mathcal{N}^Z$ that enters the equation for the toroidal potential (4.21) contains the radial component of the curl of the advection and Coriolis force. The Coriolis force can be rewritten as a function of W and Z:

$$\begin{aligned}\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\times\left[(2\tilde{\rho}\boldsymbol{u})\times\boldsymbol{e_z}\right]&=2\boldsymbol{e_r}\cdot\left[(\boldsymbol{e_z}\cdot\boldsymbol{\nabla})(\tilde{\rho}\boldsymbol{u})\right]\,,\\ &=2\left[\cos\theta\frac{\partial(\tilde{\rho}u_r)}{\partial r}-\frac{\sin\theta}{r}\frac{\partial(\tilde{\rho}u_r)}{\partial\theta}+\frac{\tilde{\rho}u_\theta\sin\theta}{r}\right]\,,\\ &=2\left[-\cos\theta\frac{\partial}{\partial r}(\Delta_H W)+\frac{\sin\theta}{r}\frac{\partial}{\partial\theta}(\Delta_H W)+\frac{\sin\theta}{r^2}\frac{\partial^2 W}{\partial r\partial\theta}+\frac{1}{r^2}\frac{\partial Z}{\partial\phi}\right]\,.\end{aligned}$$

Using the ϑ operators defined in (4.12)-(4.15) then allows to rewrite the Coriolis force in the following way:

$$\boldsymbol{e_r}\cdot\boldsymbol{\nabla}\times\left[(2\tilde{\rho}\boldsymbol{u})\times\boldsymbol{e_z}\right]=\frac{2}{r^2}\left(\vartheta_3\frac{\partial W}{\partial r}-\frac{1}{r}\vartheta_4 W+\frac{\partial Z}{\partial\phi}\right)\,. \tag{4.33}$$

The contributions of nonlinear advection and Lorentz forces that enter the equation for the toroidal potential are written this way:

$$\frac{1}{r\sin\theta}\left[\frac{\partial(\sin\theta\mathcal{A}_\phi)}{\partial\theta}-\frac{\partial\mathcal{A}_\theta}{\partial\phi}\right].$$

To make use of the recurrence relations (4.12)-(4.15), the actual strategy is to follow the following steps:

1. Compute the quantities $\mathcal{A}_\phi/r\sin\theta$ and $\mathcal{A}_\theta/r\sin\theta$ in the physical space. In the code, this step is computed in the subroutine `get_nl` in the module `grid_space_arrays_mod`.
2. Transform $\mathcal{A}_\phi/r\sin\theta$ and $\mathcal{A}_\theta/r\sin\theta$ to the spectral space (thanks to a Legendre and a Fourier transform). In MagIC, this step is computed in the modules `legendre_grid_to_spec` and `fft`. After this step $\mathcal{A}t_\ell^m$ and $\mathcal{A}p_\ell^m$ are defined.
3. Calculate the colatitude and theta derivatives using the recurrence relations:

$$\vartheta_1\,\mathcal{A}p_\ell^m-\frac{\partial\mathcal{A}t_\ell^m}{\partial\phi}. \tag{4.34}$$

Using (4.33) and (4.34), one thus finally gets

$$\begin{aligned}\mathcal{N}_{\ell m}^Z=&\frac{2}{r^2}\left[(\ell-1)(\ell+1)\,c_\ell^m\,\frac{\partial W_{\ell-1}^m}{\partial r}+\ell(\ell+2)\,c_{\ell+1}^m\,\frac{\partial W_{\ell+1}^m}{\partial r}\right.\\&\left.-\frac{\ell(\ell-1)(\ell+1)}{r}\,c_\ell^m\,W_{\ell-1}^m+\frac{\ell(\ell+1)(\ell+2)}{r}\,c_{\ell+1}^m\,W_{\ell+1}^m+im\,Z_\ell^m\right]\\&+(\ell+1)\,c_\ell^m\,\mathcal{A}p_{\ell-1}^m-\ell\,c_{\ell+1}^m\,\mathcal{A}p_{\ell+1}^m-im\,\mathcal{A}t_\ell^m.\end{aligned} \tag{4.35}$$

See also:

The final calculations of (4.35) are done in the subroutine `get_td`.

4.6.3 Nonlinear terms entering the equation for P

The nonlinear term $\mathcal{N}^P$ that enters the equation for the pressure (4.22) contains the horizontal divergence of the advection and Coriolis force. The Coriolis force can be rewritten as a function of W and Z:

$$\begin{aligned}\boldsymbol{\nabla}_H\cdot[(2\tilde\rho\boldsymbol{u})\times\boldsymbol{e_z}]&=2\boldsymbol{e_z}\cdot[\boldsymbol{\nabla}\times(\tilde\rho\boldsymbol{u})]-\left(\frac{\partial}{\partial r}+\frac{2}{r}\right)[\boldsymbol{e_r}\cdot(2\tilde\rho\boldsymbol{u}\times\boldsymbol{e_z})],\\&=-2\cos\theta\,\Delta_H Z-2\sin\theta\left[-\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\left(\frac{\partial^2}{\partial r^2}+\Delta_H\right)W+\frac{1}{r}\frac{\partial^2 Z}{\partial r\partial\theta}\right]\\&\quad-\left(\frac{\partial}{\partial r}+\frac{2}{r}\right)[2\sin\theta\tilde\rho u_\phi],\\&=2\left[\frac{1}{r}\left(\Delta_H+\frac{\partial^2}{\partial r^2}\right)\frac{\partial W}{\partial\phi}-\cos\theta\Delta_H Z-\frac{\sin\theta}{r}\frac{\partial^2 Z}{\partial r\partial\theta}\right]\\&\quad-\left(\frac{\partial}{\partial r}+\frac{2}{r}\right)\left[\frac{2}{r}\left(\frac{\partial^2 W}{\partial r\partial\phi}-\sin\theta\frac{\partial Z}{\partial\theta}\right)\right],\\&=2\left(\frac{\Delta_H}{r}\frac{\partial W}{\partial\phi}-\frac{1}{r^2}\frac{\partial^2 W}{\partial\phi\partial r}-\cos\theta\Delta_H\,Z+\frac{\sin\theta}{r^2}\frac{\partial Z}{\partial\theta}\right).\end{aligned}$$

Using the ϑ operators defined in (4.14)-(4.15) then allows to rewrite the Coriolis force in the following way:

$$\boldsymbol{\nabla}_H\cdot[(2\tilde\rho\boldsymbol{u})\times\boldsymbol{e_z}]=\frac{2}{r^2}\left(-\frac{L_H}{r}\frac{\partial W}{\partial\phi}-\frac{\partial^2 W}{\partial\phi\partial r}+\vartheta_3\,Z\right). \tag{4.36}$$

The contributions of nonlinear advection and Lorentz forces that enter the equation for pressure are written this way:

$$\frac{1}{r\sin\theta}\left[\frac{\partial(\sin\theta\mathcal{A}_\theta)}{\partial\theta}+\frac{\partial\mathcal{A}_\phi}{\partial\phi}\right].$$

To make use of the recurrence relations (4.12)-(4.15), we then follow the same three steps as for the advection term entering the equation for Z.

$$\vartheta_1\,\mathcal{A}t_\ell^m+\frac{\partial\mathcal{A}p_\ell^m}{\partial\phi}. \tag{4.37}$$

Using (4.36) and (4.37), one thus finally gets

$$\boxed{\begin{aligned}\mathcal{N}_{\ell m}^P=&\frac{2}{r^2}\left[-im\,\frac{\ell(\ell+1)}{r}\,W_\ell^m-im\,\frac{\partial W_\ell^m}{\partial r}+(\ell-1)(\ell+1)\,c_\ell^m\,Z_{\ell-1}^m+\ell(\ell+2)\,c_{\ell+1}^m\,Z_{\ell+1}^m\right]\\&+(\ell+1)\,c_\ell^m\,\mathcal{A}t_{\ell-1}^m-\ell\,c_{\ell+1}^m\,\mathcal{A}t_{\ell+1}^m+im\,\mathcal{A}p_\ell^m.\end{aligned}} \tag{4.38}$$

See also:

The final calculations of (4.38) are done in the subroutine `get_td`.

4.6.4 Nonlinear terms entering the equation for s

The nonlinear terms that enter the equation for entropy/temperature (4.24) are twofold: (i) the advection term, (ii) the viscous and Ohmic heating terms (that vanish in the Boussinesq limit of the Navier Stokes equations).

Viscous and Ohmic heating are directly calculated in the physical space by the subroutine `get_nl` in the module `grid_space_arrays_mod`. Let's introduce $\mathcal{H}$, the sum of the viscous and Ohmic heating terms.

$$\mathcal{H}=\frac{Pr\,Di}{Ra}\frac{1}{\tilde{\rho}\tilde{T}}\left(\Phi_\nu+\frac{\lambda}{Pm^2\,E}\,j^2\right).$$

Expanding this term leads to:

$$\begin{aligned}\mathcal{H}=&\frac{Pr\,Di}{Ra}\frac{1}{\tilde{\rho}\tilde{T}}\left[\tilde{\rho}\nu\left\{2\left(\frac{\partial u_r}{\partial r}\right)^2+2\left(\frac{1}{r}\frac{\partial u_\theta}{\partial\theta}+\frac{u_r}{r}\right)^2+2\left(\frac{1}{r\sin\theta}\frac{\partial u_\phi}{\partial\phi}+\frac{u_r}{r}+\frac{\cos\theta}{r\sin\theta}u_\theta\right)^2\right.\\&+\left(r\frac{\partial}{\partial r}\left(\frac{u_\theta}{r}\right)+\frac{1}{r}\frac{\partial u_r}{\partial\theta}\right)^2+\left(r\frac{\partial}{\partial r}\left(\frac{u_\phi}{r}\right)+\frac{1}{r\sin\theta}\frac{\partial u_r}{\partial\phi}\right)^2\\&\left.+\left(\frac{\sin\theta}{r}\frac{\partial}{\partial\theta}\left(\frac{u_\phi}{\sin\theta}\right)+\frac{1}{r\sin\theta}\frac{\partial u_\theta}{\partial\phi}\right)^2-\frac{2}{3}\left(\frac{d\ln\tilde{\rho}}{dr}\,u_r\right)^2\right\}\\&+\frac{\lambda}{Pm^2\,E}\left\{j_r^2+j_\theta^2+j_\phi^2\right\}\Bigg].\end{aligned} \tag{4.39}$$

This term is then transformed to the spectral space with a Legendre and a Fourier transform to produce $\mathcal{H}_\ell^m$.

The treatment of the advection term $-\boldsymbol{u}\cdot\boldsymbol{\nabla}s$ is a bit different. It is in a first step rearranged as follows

$$-\boldsymbol{u}\cdot\boldsymbol{\nabla}s=-\frac{1}{\tilde{\rho}}\left[\boldsymbol{\nabla}\cdot(\tilde{\rho}s\boldsymbol{u})-s\underbrace{\boldsymbol{\nabla}\cdot(\tilde{\rho}\boldsymbol{u})}_{=0}\right].$$

The quantities that are calculated in the physical space are thus simply the product of entropy/temperature s by the velocity components. This defines three variables defined in the grid space that are computed in the subroutine `get_nl`:

$$\mathcal{US}_r=\tilde{\rho}su_r,\quad\mathcal{US}_\theta=\tilde{\rho}su_\theta,\quad\mathcal{US}_\phi=\tilde{\rho}su_\phi,$$

To get the actual advection term, one must then apply the divergence operator to get:

$$-\boldsymbol{u}\cdot\boldsymbol{\nabla}s=-\frac{1}{\tilde{\rho}}\left[\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\mathcal{US}_r\right)+\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\mathcal{US}_\theta\right)+\frac{1}{r\sin\theta}\frac{\partial\mathcal{US}_\phi}{\partial\phi}\right].$$

To make use of the recurrence relations (4.12)-(4.15), the actual strategy is then to follow the following steps:

1. Compute the quantities $r^2\mathcal{US}_r$, $\mathcal{US}_\phi/r\sin\theta$ and $\mathcal{US}_\theta/r\sin\theta$ in the physical space. In the code, this step is computed in the subroutine `get_nl` in the module `grid_space_arrays_mod`.
2. Transform $r^2\mathcal{US}_r$, $\mathcal{US}_\phi/r\sin\theta$ and $\mathcal{US}_\theta/r\sin\theta$ to the spectral space (thanks to a Legendre and a Fourier transform). In MagIC, this step is computed in the modules `legendre_grid_to_spec` and `fft`. After this step $\mathcal{US}r_\ell^m$, $\mathcal{US}t_\ell^m$ and $\mathcal{US}p_\ell^m$ are defined.
3. Calculate the colatitude and theta derivatives using the recurrence relations:

$$-\frac{1}{\tilde{\rho}}\left[\frac{1}{r^2}\frac{\partial\mathcal{US}r_\ell^m}{\partial r}+\vartheta_1\mathcal{US}t_\ell^m+\frac{\partial\mathcal{US}p_\ell^m}{\partial\phi}\right]. \tag{4.40}$$

Using (4.39) and (4.40), one thus finally gets

$$\boxed{\mathcal{N}_{\ell m}^S=-\frac{1}{\tilde{\rho}}\left[\frac{1}{r^2}\frac{\partial\mathcal{US}r_\ell^m}{\partial r}+(\ell+1)\,c_\ell^m\,\mathcal{US}t_{\ell-1}^m-\ell\,c_{\ell+1}^m\,\mathcal{US}t_{\ell+1}^m+im\mathcal{US}p_\ell^m\right]+\mathcal{H}_\ell^m.} \tag{4.41}$$

See also:

The θ and ϕ derivatives that enter (4.41) are done in the subroutine `get_td`. The radial derivative is computed afterwards at the very beginning of `updateS`.

4.6.5 Nonlinear terms entering the equation for ξ

The nonlinear term that enters the equation for chemical composition (4.25) is the advection term. This term is treated the same way as the advection term that enters the entropy equation. It is in a first step rearranged as follows

$$-\boldsymbol{u}\cdot\boldsymbol{\nabla}\xi=-\frac{1}{\tilde{\rho}}\left[\boldsymbol{\nabla}\cdot(\tilde{\rho}\xi\boldsymbol{u})-\xi\underbrace{\boldsymbol{\nabla}\cdot(\tilde{\rho}\boldsymbol{u})}_{=0}\right].$$

The quantities that are calculated in the physical space are thus simply the product of composition ξ by the velocity components. This defines three variables defined in the grid space that are computed in the subroutine `get_nl`:

$$\mathcal{UX}_r=\tilde{\rho}\xi u_r,\quad \mathcal{US}_\theta=\tilde{\rho}\xi u_\theta,\quad \mathcal{UX}_\phi=\tilde{\rho}\xi u_\phi,$$

To get the actual advection term, one must then apply the divergence operator to get:

$$-\boldsymbol{u}\cdot\boldsymbol{\nabla}\xi=-\frac{1}{\tilde{\rho}}\left[\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\mathcal{UX}_r\right)+\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\mathcal{UX}_\theta\right)+\frac{1}{r\sin\theta}\frac{\partial\mathcal{UX}_\phi}{\partial\phi}\right].$$

To make use of the recurrence relations (4.12)-(4.15), the actual strategy is then to follow the following steps:

1. Compute the quantities $r^2\mathcal{UX}_r$, $\mathcal{UX}_\phi/r\sin\theta$ and $\mathcal{UX}_\theta/r\sin\theta$ in the physical space. In the code, this step is computed in the subroutine `get_nl` in the module `grid_space_arrays_mod`.
2. Transform $r^2\mathcal{UX}_r$, $\mathcal{UX}_\phi/r\sin\theta$ and $\mathcal{UX}_\theta/r\sin\theta$ to the spectral space (thanks to a Legendre and a Fourier transform). In MagIC, this step is computed in the modules `legendre_grid_to_spec` and `fft`. After this step $\mathcal{UX}r_\ell^m$, $\mathcal{UX}t_\ell^m$ and $\mathcal{UX}p_\ell^m$ are defined.

3. Calculate the colatitude and theta derivatives using the recurrence relations:

$$-\frac{1}{\tilde{\rho}}\left[\frac{1}{r^2}\frac{\partial \mathcal{UX}r_\ell^m}{\partial r}+\vartheta_1 \mathcal{UX}t_\ell^m+\frac{\partial \mathcal{UX}p_\ell^m}{\partial \phi}\right].$$

One thus finally gets

$$\boxed{\mathcal{N}_{\ell m}^{\xi}=-\frac{1}{\tilde{\rho}}\left[\frac{1}{r^2}\frac{\partial \mathcal{UX}r_\ell^m}{\partial r}+(\ell+1)\,c_\ell^m\,\mathcal{UX}t_{\ell-1}^m-\ell\,c_{\ell+1}^m\,\mathcal{UX}t_{\ell+1}^m+im\,\mathcal{UX}p_\ell^m\right].} \tag{4.42}$$

See also:

The θ and ϕ derivatives that enter (4.42) are done in the subroutine `get_td`. The radial derivative is computed afterwards at the very beginning of `updateXi`.

4.6.6 Nonlinear terms entering the equation for g

The nonlinear term that enters the equation for the poloidal potential of the magnetic field (4.26) is the radial component of the induction term (4.28). In the following we introduce the electromotive force $\mathcal{F}=\boldsymbol{u}\times\boldsymbol{B}$ with its three components

$$\mathcal{F}_r=u_\theta B_\phi-u_\phi B_\theta,\quad \mathcal{F}_\theta=u_\phi B_r-u_r B_\phi,\quad \mathcal{F}_\phi=u_r B_\theta-u_\theta B_r\,.$$

The radial component of the induction term then reads:

$$\mathcal{N}^g=\boldsymbol{e_r}\cdot[\boldsymbol{\nabla}\times(\boldsymbol{u}\times\boldsymbol{B})]=\frac{1}{r\sin\theta}\left[\frac{\partial\left(\sin\theta\mathcal{F}_\phi\right)}{\partial\theta}-\frac{\partial\mathcal{F}_\theta}{\partial\phi}\right].$$

To make use of the recurrence relations (4.12)-(4.15), we then follow the usual following steps:

1. Compute the quantities $r^2\,\mathcal{F}_r$, $\mathcal{F}_\phi/r\sin\theta$ and $\mathcal{F}_\theta/r\sin\theta$ in the physical space. In the code, this step is computed in the subroutine `get_nl` in the module `grid_space_arrays_mod`.
2. Transform $r^2\,\mathcal{F}_r$, $\mathcal{F}_\phi/r\sin\theta$ and $\mathcal{F}_\theta/r\sin\theta$ to the spectral space (thanks to a Legendre and a Fourier transform). In MagIC, this step is computed in the modules `legendre_grid_to_spec` and `fft`. After this step ${\mathcal{F}_r}_\ell^m$, ${\mathcal{F}_\theta}_\ell^m$ and ${\mathcal{F}_\phi}_\ell^m$ are defined.
3. Calculate the colatitude and theta derivatives using the recurrence relations:

$$\vartheta_1\,{\mathcal{F}_\phi}_\ell^m-\frac{\partial\,{\mathcal{F}_\theta}_\ell^m}{\partial\phi}\,.$$

We thus finally get

$$\boxed{\mathcal{N}_{\ell m}^{g}=(\ell+1)\,c_\ell^m\,{\mathcal{F}_\phi}_{\ell-1}^m-\ell\,c_{\ell+1}^m\,{\mathcal{F}_\phi}_{\ell+1}^m-im\,{\mathcal{F}_\theta}_\ell^m\,.} \tag{4.43}$$

See also:

The final calculations of (4.43) are done in the subroutine `get_td`.

4.6.7 Nonlinear terms entering the equation for h

The nonlinear term that enters the equation for the toroidal potential of the magnetic field (4.27) is the radial component of the curl of the induction term (4.28):

$$\begin{aligned}\mathcal{N}^h = \boldsymbol{e_r}\cdot\left[\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times(\boldsymbol{u}\times\boldsymbol{B})\right] &= \boldsymbol{e_r}\cdot\left[\boldsymbol{\nabla}\left(\boldsymbol{\nabla}\cdot\boldsymbol{\mathcal{F}}\right)-\Delta\boldsymbol{\mathcal{F}}\right],\\ &= \frac{\partial}{\partial r}\left[\frac{1}{r^2}\frac{\partial(r^2\mathcal{F}_r)}{\partial r}+\frac{1}{r\sin\theta}\frac{\partial(\sin\theta\,\mathcal{F}_\theta)}{\partial\theta}+\frac{1}{r\sin\theta}\frac{\partial\mathcal{F}_\phi}{\partial\phi}\right]\\ &\quad -\Delta\mathcal{F}_r+\frac{2}{r^2}\left[\mathcal{F}_r+\frac{1}{\sin\theta}\frac{\partial(\sin\theta\,\mathcal{F}_\theta)}{\partial\theta}+\frac{1}{\sin\theta}\frac{\partial\mathcal{F}_\phi}{\partial\phi}\right],\\ &= \frac{1}{r^2}\frac{\partial}{\partial r}\left[\frac{r}{\sin\theta}\left(\frac{\partial(\sin\theta\,\mathcal{F}_\theta)}{\partial\theta}+\frac{\partial\mathcal{F}_\phi}{\partial\phi}\right)\right]-\Delta_H\,\mathcal{F}_r\,.\end{aligned}$$

To make use of the recurrence relations (4.12)-(4.15), we then follow the same steps than for the nonlinear terms that enter the equation for poloidal potential of the magnetic field g:

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left[r^2\left(\vartheta_1\,\mathcal{F}t^m_\ell+\frac{\partial\,\mathcal{F}p^m_\ell}{\partial\phi}\right)\right]+L_H\,\mathcal{F}r^m_\ell\,.$$

We thus finally get

$$\boxed{\mathcal{N}^h_{\ell m}=\ell(\ell+1)\,\mathcal{F}r^m_\ell+\frac{1}{r^2}\frac{\partial}{\partial r}\left[r^2\left\{(\ell+1)\,c^m_\ell\,\mathcal{F}t^m_{\ell-1}-\ell\,c^m_{\ell+1}\,\mathcal{F}t^m_{\ell+1}+im\,\mathcal{F}p^m_\ell\right\}\right]\,.} \tag{4.44}$$

See also:

The θ and ϕ derivatives that enter (4.44) are computed in the subroutine `get_td`. The remaining radial derivative is computed afterwards at the very beginning of `updateB`.

4.7 Boundary conditions and inner core

4.7.1 Mechanical boundary conditions

Since the system of equations is formulated on a radial grid, boundary conditions can simply be satisfied by replacing the collocation equation at grid points r_i and r_o with appropriate expressions. The condition of zero radial flow on the boundaries implies that the poloidal potential has to vanish, i.e. $W(r_o)=0$ and $W(r_i)=0$. In Chebychev representation this implies

$$\mathcal{C}_n(r)W_{\ell mn}=0\ \text{ at }\ r=r_i,r_o\ \ . \tag{4.45}$$

Note that the summation convention with respect to radial modes n is used again. **The no-slip** condition further requires that the horizontal flow components also have to vanish, provided the two boundaries are at rest. This condition is fulfilled for

$$\frac{\partial W}{\partial r}=0\ \text{ and }\ Z=0,$$

at the respective boundary. In spectral space these conditions read

$$\mathcal{C}'_n(r)W_{\ell mn}=0\ \text{ at }\ r=r_i,r_o\,, \tag{4.46}$$

and

$$\mathcal{C}_n(r)Z_{\ell mn}=0\ \text{ at }\ r=r_i,r_o\,, \tag{4.47}$$

for all spherical harmonic modes (ℓ, m). The conditions (4.45)-(4.47) replace the poloidal flow potential equations (4.20) and the pressure equation (4.22), respectively, at the collocation points r_i and r_o.

If the inner-core and/or the mantle are allowed to react to torques, a condition based on the conservation of angular momentum replaces condition (4.47) for the mode $(\ell = 1, m = 0)$:

$$\mathsf{I}\frac{\partial \omega}{\partial t} = \Gamma_L + \Gamma_\nu \, .$$

The tensor I denotes the moment of inertia of inner core or mantle, respectively, ω is the mantle or inner-core rotation rate relative to that of the reference frame, and $\Gamma_{L,\nu}$ are the respective torques associated with Lorentz or viscous forces. The torques are expressed by

$$\Gamma_L = \frac{1}{E\,Pm} \oint B_r B_\phi \, r \sin\theta \, \mathrm{d}S \, ,$$

and

$$\Gamma_\nu = \oint \tilde{\rho}\tilde{\nu} r \frac{\partial}{\partial r}\left(\frac{u_\phi}{r}\right) r \sin\theta \, \mathrm{d}S \, ,$$

where $\mathrm{d}S = r^2 \sin\theta \mathrm{d}\theta \mathrm{d}\phi$ and $r \in [r_i, r_o]$ in the above expressions. Using the following equality

$$\oint \tilde{\rho} r \sin\theta u_\phi \, \mathrm{d}S = 4\sqrt{\frac{\pi}{3}} Z_{10} r^2,$$

the viscous torques can be expressed by

$$\Gamma_\nu = \pm 4\sqrt{\frac{\pi}{3}} \tilde{\nu} r^2 \left[\frac{\partial Z_{10}}{\partial r} - \left(\frac{2}{r} + \beta\right) Z_{10}\right] \, ,$$

where the sign in front depends whether $r = r_o$ or $r = r_i$.

Free-slip boundary conditions require that the viscous stress vanishes, which in turn implies that the non-diagonal components $\mathsf{S}r_{r\phi}$ and $\mathsf{S}_{r\theta}$ of the rate-of-strain tensor vanish. Translated to the spectral representation this requires

$$\left[\mathcal{C}_n''(r) - \left(\frac{2}{r} + \frac{d\ln\tilde{\rho}}{dr}\right)\mathcal{C}_n'(r)\right] W_{\ell mn} = 0 \quad \text{and} \quad \left[\mathcal{C}_n'(r) - \left(\frac{2}{r} + \frac{d\ln\tilde{\rho}}{dr}\right)\mathcal{C}_n(r)\right] Z_{\ell mn} = 0 \, .$$

We show the derivation for the somewhat simpler Boussinesq approximation which yields the condition

$$\frac{\partial}{\partial r}\frac{\boldsymbol{u}_H}{r} = 0$$

where the index H denotes the horizonal flow components. In terms of poloidal and toroidal components this implies

$$\frac{\partial}{\partial r}\frac{1}{r}\left(\boldsymbol{\nabla}_H \frac{\partial W}{\partial r}\right) = \boldsymbol{\nabla}_H \frac{1}{r}\left(\frac{\partial^2}{\partial r^2} - \frac{2}{r}\frac{\partial}{\partial r}\right) W = 0$$

and

$$\frac{\partial}{\partial r}\frac{1}{r}\nabla \times \boldsymbol{e}_r Z = \nabla \times \boldsymbol{e}_r \frac{1}{r}\left(\frac{\partial}{\partial r} - \frac{2}{r}\right) Z = 0$$

which can be fulfilled with

$$\left(\frac{\partial^2}{\partial r^2} - \frac{2}{r}\frac{\partial}{\partial r}\right) W = 0$$

and

$$\left(\frac{\partial}{\partial r} - \frac{2}{r}\right) Z = 0 \, .$$

In spectral representation this then reads

$$\left(\mathcal{C}_n'' - \frac{2}{r}\mathcal{C}_n'\right) W_{\ell mn} = 0 \quad \text{and} \quad \left(\mathcal{C}_n' - \frac{2}{r}\mathcal{C}_n\right) Z_{\ell mn} = 0 \, .$$

4.7.2 Thermal boundary conditions

For Entropy or temperature in the Boussinesq approximation either fixed value of fixed flux conditions are used. The former implies

$$s = \text{const.} \quad \text{or} \quad T = \text{const.}$$

at r_i and/or r_o, while the latter means

$$\frac{\partial}{\partial r} s = \text{const.} \quad \text{or} \quad \frac{\partial}{\partial r} T = \text{const.}$$

In spectral representation for example the respective entropy condition read

$$\mathcal{C}_n s_{\ell mn} = \text{const.} \quad \text{or} \quad \mathcal{C'}_n s_{\ell mn} = \text{const.}$$

Appropriate constant values need to be chosen and are instrumental in driving the dynamo when flux conditions are imposed.

4.7.3 Boundary conditions for chemical composition

For the chemical composition, either the value or the flux is imposed at the boundaries. The former implies:

$$\xi = \text{const.}$$

at r_i and/or r_o, while the latter means

$$\frac{\partial}{\partial r} \xi = \text{const.}$$

In spectral representation, this then reads

$$\mathcal{C}_n \xi_{\ell mn} = \text{const.} \quad \text{or} \quad \mathcal{C'}_n \xi_{\ell mn} = \text{const.}$$

4.7.4 Magnetic boundary conditions and inner core

Three different magnetic boundary conditions are implemented in MagIC. The most simple one is the conceptual condition at the boundary to an infinite conductor. Surface current in this conductor will prevent the internally produced magnetic field from penetrating so that the field has to vanish at the boundary. The condition are thus the same as for a rigid flow (with boundaries at rest). We only provide the spectral representation here:

$$\mathcal{C}_n(r) W_{\ell mn} = 0 \quad \text{at} \quad r = r_i, r_o \quad . \tag{4.48}$$

Note that the summation convention with respect to radial modes n is used again. **The no-slip** condition further requires that the horizontal flow components also have to vanish, provided the two boundaries are at rest. This condition is fulfilled for

$$\mathcal{C}_n(r) g_{\ell mn} = 0 \quad , \quad \mathcal{C}'_n(r) g_{\ell mn} = 0 \quad \text{and} \quad \mathcal{C}_n(r) h_{\ell mn} = 0. \tag{4.49}$$

More complex are the conditions to an electrical insulator. Here we actually use matching condition to a potential field condition that are formulated like boundary conditions. Since the electrical currents have to vanish in the insulator we have $\nabla \times \boldsymbol{B}$, which means that the magnetic field is a potential field $\boldsymbol{B}^I = -\boldsymbol{\nabla} V$ with $\Delta V = 0$. This Laplace equation implies a coupling between radial and horizontal derivatives which is best solved in spectral space. Two potential contributions have to be considered depending whether the field is produced above the interface radius r_{BC}

or below. We distinguish these contributions with upper indices I for internal or below and E for external or above. The total potential then has the form:

$$V_{\ell m}(r) = r_{BC} V^I_{\ell m} \left(\frac{r_{BC}}{r}\right)^{\ell+1} + r_{BC} V^E_{\ell m} \left(\frac{r}{r_{BC}}\right)^{\ell} .$$

with the two spectral potential representations $V^I_{\ell m}$ and $V^E_{\ell m}$. This provides well defined radial derivative for both field contributions. For boundary r_o we have to use the first contribution and match the respective field as well as its radial derivative to the dynamo solution. The toroidal field cannot penetrate the insulator and thus simply vanishes which yields $h = 0$ or

$$\mathcal{C}_n h_{\ell mn} = 0$$

in spectral space. The poloidal field then has to match the potential field which implies

$$\nabla_H \frac{\partial}{\partial r} g = -\nabla_H V^I$$

for the horizontal components and

$$\frac{\nabla^2_H}{r^2} g = \frac{\partial}{\partial r} V^I$$

for the radial. In spectral space these condition can be reduce to

$$\mathcal{C'}_n(r) g_{\ell mn} = V^I_{lm} \quad \text{and} \quad \frac{\ell(\ell+1)}{r^2} \mathcal{C}_n g_{\ell mn} = -\frac{\ell+1}{r} V^I_{lm}.$$

Combining both allows to eliminate the potential and finally leads to the spectral condition used in MagIC:

$$\left(\mathcal{C'}_n(r_o) + \frac{\ell}{r_o}\mathcal{C}_n(r_o)\right) g_{\ell mn} = 0$$

Analogous consideration lead to the respective condition at the interface to an insulating inner core:

$$\left(\mathcal{C'}_n(r_i) - \frac{\ell+1}{r_i}\mathcal{C}_n(r_i)\right) g_{\ell mn} = 0.$$

If the inner core is modelled as an electrical conductor, a simplified dynamo equation has to be solved in which the fluid flow is replaced by the solid-body rotation of the inner core. The latter is described by a single toroidal flow mode $(\ell = 1, m = 0)$. The resulting nonlinear terms can be expressed by a simple spherical harmonic expansion, where the superscript I denotes values in the inner core and ω_I its differential rotation rate:

$$\int d\Omega \, Y_\ell^{m\star} \, \boldsymbol{e_r} \cdot \left[\boldsymbol{\nabla} \times \left(\boldsymbol{u^I} \times \boldsymbol{B^I}\right)\right] = -i\,\omega_I\, m\, \frac{\ell(\ell+1)}{r^2}\, g^I_{\ell m}(r) \, , \tag{4.50}$$

$$\int d\Omega \, Y_\ell^{m\star} \, \boldsymbol{e_r} \cdot \left[\boldsymbol{\nabla} \times \boldsymbol{\nabla} \times \left(\boldsymbol{u^I} \times \boldsymbol{B^I}\right)\right] = -i\,\omega_I\, m\, \frac{\ell(\ell+1)}{r^2}\, h^I_{\ell m}(r) \, . \tag{4.51}$$

The expensive back and forth transformations between spherical harmonic and grid representations are therefore not required for advancing the inner-core magnetic field in time.

In the inner core the magnetic potentials are again conveniently expanded into Chebyshev polynomials. The Chebyshev variable x spans the whole diameter of the inner core, so that grid points are dense near the inner-core boundary but sparse in the center. The mapping is given by:

$$x(r) = \frac{r}{r_i} \quad , \quad -r_i \leq r \leq r_i \quad . \tag{4.52}$$

Each point in the inner core is thus represented twice, by grid points (r, θ, ϕ) and $(-r, \pi - \theta, \phi + \pi)$. Since both representations must be identical, this imposes a symmetry constraint that can be fulfilled when the radial expansion comprises only polynomials of even order:

$$g^I_{\ell m}(r) = \left(\frac{r}{r_i}\right)^{\ell+1} \sum_{i=0}^{M-1} g^I_{\ell m\, 2i}\, \mathcal{C}_{2i}(r) \quad . \tag{4.53}$$

An equivalent expression holds for the toroidal potential in the inner core. FFTs can again by employed efficiently for the radial transformation, using the M extrema of $\mathcal{C}_{2M-1}(r)$ with $x > 0$ as grid points.

The sets of spectral magnetic field equations for the inner and the outer core are coupled via continuity equations for the magnetic field and the horizontal electric field. Continuity of the magnetic field is assured by (**i**) continuity of the toroidal potential, (**ii**) continuity of the poloidal potential, and (**iii**) continuity of the radial derivative of the latter. Continuity of the horizontal electric field demands (**iv**) that the radial derivative of the toroidal potential is continuous, provided that the horizontal flow and the electrical conductivity are continuous at the interface. These four conditions replace the spectral equations (4.26), (4.27) on the outer-core side and equations (4.50), (4.51) on the inner-core side. Employing free-slip conditions or allowing for electrical conductivity differences between inner and outer core leads to more complicated and even non-linear matching conditions.

CHAPTER

FIVE

CONTRIBUTING TO THE CODE

MagIC is an open-source code, we thus value any possible contribution! There are several ways to directly contribute to the code:

Contribute

- **Do you want to contribute to the code?** Just clone the code and start modyfing it. Make sure that your modifications *don't alter the code*, try to *document your changes* as much as you can and follow the recommended *Fortran coding style*.
- **Do you want to improve the documentation?** Feel free to document some missing features. The documentation is stored in the directory `$MAGIC_HOME/doc/sphinx` and relies on the documenting tool Sphinx. Some recommendations regarding documentation can be found *below*.
- **Did you find a bug?** Issues and feature requests should be raised in the github tracker.

5.1 Checking the consistency of the code

It is frequently required to check the consistency of the code, especially **after the implementation of new features**. For this reason, we developed the python test suite `magic_wizard.py`, located in the directory `$MAGIC_HOME/samples/`, which tests the compilation of the code and it's results against a set of standard solutions in sample directories to check if the code produces the correct output.

You can run it as follows:

```
./magic_wizard.py <options>
```

It supports the following options:

```
-h,  --help              Show usage overview
     --level LEV               Run only tests from level LEV
     --use-debug-flags         Compile MagIC with the debug flags
     --use-mpi                 Use MPI
     --use-openmp              Use the hybrid version of MagIC
     --use-mkl                 Use the MKL for FFTs and Lapack calls
     --use-shtns               Use SHTns for Legendre transforms
     --use-precond USE_PRECOND Use matrix preconditioning
     --nranks NRANKS           Specify the number of MPI ranks
     --nthreads NTHREADS       Specify the number of threads (hybrid version)
     --mpicmd MPICMD           Specify the mpi executable (mpiexec, mpirun, srun)
```

Note: Make sure that your environment variables `FC` and `CC` are correctly defined otherwise the script will use the default system compilers.

The `--level LEV` option defines the priority level of check and validation of the code. It has the following levels of checking:

Level	Cases to check (subdirectories)
0	• Boussinesq dynamo benchmark (BM1) (Christensen et al., 2001) - start from zero (`dynamo_benchmark`) • Variable transport properties (viscosity, thermal diffusivity and electrical diffusivity) in an anelastic convective model (`varProps`) • Test of a case that uses finite differences - restart from a case that used Chebyshev polynomials (`finite_differences`) • Boussinesq dynamo benchmark (BM2) (Christensen et al., 2001) - start from a saturated state (`boussBenchSat`) • Double-diffusive convection benchmark (Breuer et al., 2010) - start from a saturated state (`doubleDiffusion`) • Axisymmetric spherical Couette flow - this auto-test checks the axisymmetric version of MagIC (`couetteAxi`) • Test Precession (`precession`) • Whole sphere benchmark (Marti et al., 2014) - start from a saturated state (`full_sphere`)
1	• Test reading and writing of restart files (`testRestart`) • Test different grid truncations (`testTruncations`) • Test mapping on to a new grid (`testMapping`) • Test different outputs produced (`testOutputs`) • Test different radial outputs - `*R.TAG` (`testRadialOutputs`)
2	• Hydrodynamic anelastic benchmark (Jones et al., 2011) (`hydro_bench_anel`)
3	• Heat flux perturbation (`fluxPerturbation`) • Isothermal model with $N_\rho = 3$ (`isothermal_nrho3`) • Boussinesq Dynamo benchmark for conducting and rotating inner core (`dynamo_benchmark_condICrotIC`) • Anelastic dynamo with variable conductivity (`varCond`)
4	• Test the writing of CMB and coeff files (`testCoeffOutputs`) • Test the writing of RMS force balance (`testRMSOutputs`) • Test the writing of Graphic and Movie files (`testGraphMovieOutputs`) • Test the writing of TO and Geos outputs (`testTOGeosOutputs`)

5.2 Advices when contributing to the code

- Before commiting your modifications **always** make sure that the auto-tests pass correctly.
- Try to follow the same coding style rules as in the rest of the code:
  1. **Never** use TABS but always SPACES instead
  2. Use 3 spaces for indentation

  Note: These two rules can be easily set in your $HOME/.vimrc file if you use vim:

```
au FileType fortran set shiftwidth=3
au FileType fortran set tabstop=3
au FileType fortran set expandtab
```

  3. Never use capital letters for variable declaration or Fortran keywords
  4. Never use `dimension(len)` for declaring array but rather `real(cp) :: data(len)`
  5. Always use the default precisions when introducing new variables `(cp)`

These rules try to follow the general recommendations on modern fortran programming that can be found on www.fortran90.org or in the book Modern Fortran - style and usage by N. S. Clerman and W. Spector.

5.3 Building the documentation and contributing to it

The documentation is generated using Sphinx. To build it you'll thus need to install this python module on your machine. This is in general directly available on most of the Linux distributions under the name `python-sphinx`. Once installed, just go to the documentation directory

```
$ cd $MAGIC_HOME/doc/sphinx
```

and build the html documentation

```
$ make html
```

The complete documentation will then be built in a local directory named `$MAGIC_HOME/doc/sphinx/.build/html`.

If LaTeX is installed on your work station, it is also possible to build the corresponding manual of the documentation in the pdf format:

```
$ make latexpdf
```

The resulting pdf is then generated in a local directory named `$MAGIC_HOME/doc/sphinx/.build/latex`.

It is pretty straightforward to contribute to the documentation by simply adding some contents to the different `rst` files. Informations about reStructuredText syntax can be found on www.sphinx-doc.org, while helpful CheatSheet are accessible here or there.

CHAPTER

SIX

INPUT PARAMETERS

True runtime input parameters are read from STDIN as namelists, a Fortran feature. A namelist is identified by its unique name *&name*. The name-statement is followed by the parameters that are part of the namelist in the format *parameter=value,*. The namelist is closed by a backslash. The subroutine *defaultNamelists* (in the module *Namelist.f90*) defines a default value for each parameter. Only the parameters whose value should differ from its default have to be stated in the namelist.

An example for the short namelist defining inner core parameters is

```
&inner_core
  sigma_ratio = 1.0,
  nRotIc      = 1
```

Comas can be used to seperate namelist entries since they are not interpreted by the code.

Magic uses the following **eight namelists** :

Namelists

1. *&grid* for resolution
2. *&control* for control parameters and numerical parameters.
3. *&phys_param* for the physical parameters.
4. *&B_external* for setting up an external field contribution
5. *&start_field* to define the starting fields.
6. *&output_control* for defining the output.
7. *&mantle* for setting mantle parameters.
8. *&inner_core* for setting inner core parameters.

The number of possible input parameters has grown to more than 100/150. **Don't be confused by all the possible options though, since all parameters are internally set to a useful default value!**

Practically, in a production run, the number of parameters you may want to adjust is thus much smaller. As an example, the following namelist shows you how to initiate and quickly run one of the anelastic benchmarks by (Jones et al., 2011):

```
&grid
 n_r_max     =97,            ! 97 radial grid points
 n_cheb_max  =95,
 n_phi_tot   =288,           ! 288 points in the azimuthal direction
```

(continues on next page)

(continued from previous page)

```
 n_r_ic_max  =17,
n_cheb_ic_max=15,
 minc        =1,             ! No azimuthal symmetry
/
&control
 mode        =1,             ! This is a non-magnetic case
 tag         ="test",        ! Trailing name of the outputs produced by the code
 n_time_steps=50000,         ! Number of time steps
 courfac     =2.5D0,         ! Courant factor (flow)
 alffac      =1.0D0,         ! Courant factor (magnetic field)
 dtmax       =1.0D-4,        ! Maximum allowed time-step
 alpha       =0.6D0,
 runHours    =23,            ! Run time (hours)
 runMinutes  =30,            ! Run time (minutes)
 time_scheme ='CNAB2',       ! Name of the time stepper
/
&phys_param
 ra          =1.48638035D5, ! Rayleigh number
 ek          =1.0D-3,        ! Ekman number
 pr          =1.0D0,         ! Prandtl number
 strat       =5.D0,          ! Density contrast
 polind      =2.0D0,         ! Polytropic index
 radratio    =0.35D0,        ! Aspect ratio of the spherical shell
 g0          =0.D0,          ! Gravity profile
 g1          =0.D0,
 g2          =1.D0,
 ktops       =1,             ! Entropy boundary condition
 kbots       =1,
 ktopv       =1,             ! Mechanical boundary condition
 kbotv       =1,
/
&start_field
 l_start_file=.false.,
 start_file  ="checkpoint_end.CJ3",
 init_s1     =1919,          ! Initial entropy perturbation pattern
 amp_s1      =0.01,          ! Amplitude of the initial perturbation
/
&output_control
 n_log_step  =50,            ! Store time series every 50 time steps
 n_graphs    =1,             ! 1 G_#.TAG file produced at the end of the run
 n_specs     =5,             ! 5 spectra produced during the run
 n_rsts      =1,             ! 1 checkpoint_end.TAG file produced at the end of the run
 runid       ="C.Jones bench",
/
&mantle
 nRotMa      =0              ! Non-rotating mantle
/
&inner_core
 sigma_ratio =0.d0,          ! Non-conducting inner core
 nRotIC      =0,             ! Non-rotating inner core
/
```

This example might then be easily adapted to your desired configuration.

6.1 Grid namelist

This namelist defines the resolution of the computations. Keep in mind that **MagIC** is a 3D pseudo-spectral spherical shell code using Chebyshev polynomial expansions in the radial and spherical harmonic expansions in the angular directions.

6.1.1 Outer Core

- **n_r_max** (default `n_r_max=33`) is an integer which gives the number of grid points in the radial direction in the outer core ($[r_i, r_o]$). If Chebyshev polynomials are used for the radial integration scheme, there are some constraints on the value of *n_r_max*: first of all it must be of the form `4*n+1`, where `n` is an integer; and second the prime decomposition of *n_r_max-1* should only contain multiple of 3, 4 and 5 (this second condition comes from the limitations of the built-in discrete cosine transforms). This constraint is released when finite differences are used.

Note: If Chebyshev polynomials are used, the possible values for `n_r_max` below 220 are hence: 17, 21, 25, 33, 37, 41, 49, 61, 65, 73, 81, 97, 101, 109, 121, 129, 145, 161, 181, 193, 201, 217,...

- **n_cheb_max** (default `n_cheb_max=31`) is an integer which is the number of terms in the Chebyshev polynomial expansion to be used in the radial direction - the highest degree of Chebyshev polynomial used being `n_cheb_max-1`. Note that `n_cheb_max <= n_r_max`. This quantity will not be used if finite differences are used.
- **n_phi_tot** (default `n_phi_tot=192`) is an integer which gives the number of longitudinal/azimuthal grid points. It has the following constraints:
  - `n_phi_tot` must be a multiple of `minc` (see below)
  - `n_phi_tot/minc` must be a multiple of 4
  - `n_phi_tot` must be a multiple of 16

Note: The possible values for `n_phi_max` are thus: 16, 32, 48, 64, 96, 128, 192, 256, 288? 320, 384, 400, 512, 576, 640, 768, 864, 1024, 1280, 1536, 1792, 2048, ...

- **l_axi** (default `l_axi=.false.`) is a logical. When set to true, one considers only the axisymmetric mode (i.e. MagIC becomes a 2-D axisymmetric code).
- **n_theta_axi** (default `n_theta_axi=0`) is an integer which gives the number of latitudinal grid points when MagIC computes only the axisymmetric mode.
- **fd_order** (default `fd_order=2`) is an integer. This is the order of the finite difference scheme for the bulk points (possible values are 2, 4, 6).
- **fd_order_bound** (default `fd_order_bound=2`) is an integer. This is the order of the finite difference scheme for the boundary points (possible values are 1,2,3,4,5,6, ...). This has to be smaller than the order of the scheme used for the bulk points.
- **fd_stretch** (default `fd_stretch=0.3`) is a real. It controls the ratio between the number of points in the boundary layers and in the bulk.
- **fd_ratio** (default `fd_ratio=0.1`) is a real. It controls the ratio between the smallest grid spacing and the largest grid spacing.

Note: When *fd_ratio* is set to 1, the radial grid is regularly-spaced.

- **l_var_l** (default `l_var_l=.false.`) is a logical. The spherical harmonic degree is a function of radius, when set to true. This practically reduces the number of spherical harmonic transforms in parts of the fluid domain but it comes at the price of an MPI inbalance. This feature is useful when computing full sphere geometry to avoid a too severe time step limitation close to the center. Right now the form of the radial dependence follows:

$$\ell(r) = 1 + (\ell_{\max} - 1)\sqrt{\frac{r}{r_o}}$$

6.1.2 Inner Core

- **n_r_ic_max** (default `n_r_ic_max=17`) is an integer which gives the number of grid points in the radial direction in the inner core ($[0, r_i]$). It too, must be of the form `4*n+1`, where `n` is an integer.
- **n_cheb_ic_max** (default `n_cheb_ic_max=15`) is the number of terms in the Chebyshev polynomial expansion in the radial direction in the inner core. Only Chebyshev polynomials of even degrees are used in the expansion giving the highest degree used to be `2*n_cheb_ic_max-2`. Note that here too, `n_cheb_ic_max <= n_r_max`.

6.1.3 Symmetry and aliasing

- **minc** (default `minc=1`) is an integer which gives the longitudinal symmetry. e.g: `minc=n` would give an n-fold rotational symmetry in the azimuthal direction. One can use this to reduce computational costs when the symmetry of the solution is known. The orders of the spherical harmonic expansion (`m`) are multiples of `minc`.
- **nalias** (default `nalias=20`) is an integer which determines antialiasing used in the spherical harmonic representation. Note that `20 <= nalias <= 30`.

The number of grid points in latitude `n_theta_max = n_phi_tot/2`. The maximum degree (`l_max`) and maximum order (`m_max`) of the spherical harmonic expansion are determined by `nalias`:

```
l_max = (nalias * n_theta_max)/30
```

6.2 Control namelist

This namelist defines the numerical parameters of the problem plus the variables that control and organize the run.

- **mode** (default `mode=0`) is an integer which controls the type of calculation performed.

mode=0	Self-consistent dynamo
mode=1	Convection
mode=2	Kinematic dynamo
mode=3	Magnetic decay modes
mode=4	Magneto convection
mode=5	Linear onset of convection
mode=6	Self-consistent dynamo, but with no Lorentz force
mode=7	Super-rotating inner core or mantle, no convection and no magnetic field
mode=8	Super-rotating inner core or mantle, no convection
mode=9	Super-rotating inner core or mantle, no convection and no Lorentz force
mode=10	Super-rotating inner core or mantle, no convection, no magnetic field, no Lorentz force and no advection

- **tag** (default `tag="default"`) is a character string, used as an extension for all output files.
- **n_time_steps** (default `n_time_steps=100`) is an integer, the number of time steps to be performed.
- **tEND** (default `tEND=0.0`) is a real, which can be used to force the code to stop when :math:`t=tEND`. This is only used when `t/=tEND`.
- **alpha** (default `alpha=0.5`) is a real. This is the weight used for current time step in implicit time step.

6.2.1 Default scales

- **n_tScale** (default `n_tScale=0`) is an integer, which determines the time scaling

n_tScale=0	Use viscous time scale.	d^2/ν
n_tScale=1	Use magnetic time scale.	d^2/η
n_tScale=2	Use thermal time scale.	d^2/κ
n_tScale=3	Use rotational time scale.	Ω^{-1}

- **n_lScale** (default `n_lScale=0`) is an integer which determines the reference length scale.

n_lScale=0	Use outer core.
n_lScale=1	Use total core.

- **enscale** (default `enscale=1.0`) is a real. This is the scaling for energies.

6.2.2 Update control

- **l_update_v** (default `l_update_v=.true.`) is a logical that specifies whether the velocity field should be time-stepped or not.
- **l_update_b** (default `l_update_b=.true.`) is a logical that specifies whether the magnetic field should be time-stepped or not.
- **l_update_s** (default `l_update_s=.true.`) is a logical that specifies whether the entropy/temperature should be time-stepped or not.
- **l_update_xi** (default `l_update_xi=.true.`) is a logical that specifies whether the chemical composition should be time-stepped or not.

6.2.3 Time step control

A modified Courant criterion including a modified Alfven-velocity is used to account for the magnetic field. The relative and absolute importance of flow and Alfven-velocity can be controled by **courfac** and **alffac** respectively. The parameter **l_cour_alf_damp** allows to choose whether the actual Alven speed is used to estimate the Courant condition or if damping is included. Practically, the timestep size is controlled as follows

$$\delta t < \min_V \left(c_I \, E, \, \frac{\delta r}{|u_r|}, \, \frac{\delta h}{u_h} \right)$$

where $u_h = (u_\theta^2 + u_\phi^2)^{1/2}$, $\delta h = \dfrac{r}{\sqrt{\ell(\ell+1)}}$, and δr is the radial grid interval. The first term in the left hand side accounts for the explicit treatment of the Coriolis term.

$$|u_r| = c_F |u_{F,r}| + c_A \frac{u_{A,r}^2}{\left[u_{A,r}^2 + \left(\frac{1+Pm^{-1}}{2\delta r} \right)^2 \right]^{1/2}},$$

where $u_{F,r}$ is the radial component of the fluid velocity and $u_{A,r} = Br/\sqrt{E\,Pm}$ is the radial Alven velocity. The denominator of the rightmost term accounts for the damping of the Alven waves.

- **dtMax** (default `dtMax=1e-4`) is a real. This is the maximum allowed time step δt. If $\delta t >$ dtmax, the time step is decreased to at least dtMax (See routine *dt_courant*). Run is stopped if $\delta t <$ dtmin and dtmin $= 10^{-6}$ dtmax.
- **courfac** (default `courfac=2.5`) is a real used to scale velocity in Courant criteria. This parameter corresponds to c_F in the above equation.
- **alffac** (default `alffac=1.0`) is a real, used to scale Alfven-velocity in Courant criteria. This parameter corresponds to c_A in the above equation.
- **intfac** (default `intfac=0.15`) is a real, used to scale Coriolis factor in Courant criteria. This parameter corresponds to c_I in the above equation.
- **l_cour_alf_damp** (default `l_cour_alf_damp=.true.`) is a logical. This is used to decide whether the damping of the Alven waves is taken into account when estimating the Courant condition (see Christensen et al., GJI, 1999). At low Ekman numbers, this criterion might actually lead to spurious oscillations/instabilities of the code. When turn to False, $|u_r| = c_F |u_{F,r}| + c_A |u_{A,r}|$.
- **time_scheme** (default `time_scheme='CNAB2'`) is a character string. This is used to choose the time step integrator used in the code among the following implicit-explicit time schemes:

time_scheme='CNAB2'	Crank-Nicolson and 2nd order Adams-Bashforth scheme
time_scheme='CNLF'	Crank-Nicolson and Leap-Frog scheme
time_scheme='MODCNAB'	Modified CN/AB2
time_scheme='SBDF2'	Semi-implicit backward difference scheme of 2nd order
time_scheme='SBDF3'	Semi-implicit backward difference scheme of 3rd order
time_scheme='SBDF4'	Semi-implicit backward difference scheme of 4th order
time_scheme='ARS222'	Semi-implicit S-DIRK of 2nd order
time_scheme='ARS232'	Semi-implicit S-DIRK of 2nd order
time_scheme='CK232'	Semi-implicit S-DIRK of 2nd order
time_scheme='LZ232'	Semi-implicit S-DIRK of 2nd order
time_scheme='CB3'	Semi-implicit S-DIRK of 3rd order
time_scheme='ARS343'	Semi-implicit S-DIRK of 3rd order
time_scheme='ARS443'	Semi-implicit S-DIRK of 3rd order
time_scheme='BPR353'	Semi-implicit S-DIRK of 3rd order
time_scheme='LZ453'	Semi-implicit S-DIRK of 3rd order
time_scheme='KC343'	Semi-implicit S-DIRK of 3rd order
time_scheme='KC564'	Semi-implicit S-DIRK of 4th order
time_scheme='KC674'	Semi-implicit S-DIRK of 4th order
time_scheme='KC785'	Semi-implicit S-DIRK of 5th order

6.2.4 Run time

The total desired runtime (in human units and not in CPU units) can be specified with the three variables **runHours**, **runMinutes** and **runSeconds**.

- **runHours** (default `runHours=0`) is an integer that controls the number of run hours.
- **runMinutes** (default `runMinutes=0`) is an integer that controls the .
- **runSeconds** (default `runSeconds=0`) is an integer that controls the number of run hours.

Here is an example for a run of 23h30:

```
runHours   = 23,
runMinutes = 30,
```

6.2.5 Hyperdiffusivity

Hyperdiffusion can be applied by multiplying the diffusion operators by a factor of the form

$$d(\ell) = 1 + D \left[\frac{\ell + 1 - \ell_{hd}}{\ell_{max} + 1 - \ell_{hd}} \right]^{\beta}$$

for the spherical harmonic degrees $\ell \geq \ell_{hd}$.

- **difnu** (default `difnu=0.0`) is a real. This is the amplitude D of the viscous hyperdiffusion.
- **difkappa** (default `difkappa=0.0`) is a real. This is the amplitude D of the thermal hyperdiffusion.
- **difchem** (default `difchem=0.0`) is a real. This is the amplitude D of the hyperdiffusion applied to chemical composition.
- **difeta** (default `difeta=0.0`) is a real. This is the amplitude D of the magnetic hyperdiffusion.
- **ldif** (default `ldif=1`) is an integer. This is the degree ℓ_{hd} where hyperdiffusion starts to act.

- **ldifexp** (default `ldifexp=-1`) is an integer. This is the exponent β of hyperdiffusion.

6.2.6 Angular momentum correction

In case of the use of stress-free boundary conditions at both boundaries, it is safer to ensure that the angular momentum is correctly conserved. This can be enforced through the following input variables:

- **l_correct_AMe** (default `l_correct_AMe=.false.`) is a logical. This is used to correct the equatorial angular momentum.
- **l_correct_AMz** (default `l_correct_AMz=.false.`) is a logical. This is used to correct the axial angular momentum.

6.2.7 Radial scheme and mapping of the Gauss-Lobatto grid

In MagIC, one can either use finite differences or Chebyshev polynomials for the radial integration scheme. This choice is controlled by the following input parameter:

- **radial_scheme** (default `radial_scheme='CHEB'`) is a character string.

radial_scheme='CHEB'	Use Chebyshev polynomials
radial_scheme='FD'	Use finite differences

When Chebyshev polynomials are used, it is also possible to use a non-linear mapping function to concentrate/diperse grid points around a point inside the domain.

- **l_newmap** (default `l_newmap=.false.`) is a logical. A radial mapping can be applied to the Chebyshev grid when `l_newmap` is set to `.true.`. The radial profile of the mapping function is then stored during the initialisation of the code in the file *rNM.TAG*.
- **map_function** (default `map_function='arcsin'`) is a character string. This allows to select which mapping function is used:

map_function='TAN'	Use a tangent mapping (see Bayliss and Turkel 1992)
map_function='ARCSIN'	Use finite differences (see Kosloff and Tal-Ezer 1993)

If the tangent mapping is used, the function that re-distributes the collocation points is expressed by

$$r = \frac{1}{2}\left(\alpha_2 + \frac{\tan\left[\lambda(r_{cheb} - x_0)\right]}{\alpha_1}\right) + \frac{r_i + r_o}{2} ,$$

where the Gauss-Lobatto collocation points are

$$r_{cheb} = \cos\left(\frac{\pi(k-1)}{N_r}\right) , \quad k = 1, 2, ..., n_r , \; n_r = n\_r\_max$$

and $r \in [r_i, r_o]$, $r_{cheb} \in [-1.0, 1.0]$. The parameters to calculate r are

$$\begin{aligned} \lambda &= \frac{\tan^{-1}\left(\alpha_1(1-\alpha_2)\right)}{1 - x_0} \\ x_0 &= \frac{K-1}{K+1} \\ K &= \frac{\tan^{-1}\left(\alpha_1(1+\alpha_2)\right)}{\tan^{-1}\left(\alpha_1(1-\alpha_2)\right)} . \end{aligned}$$

The coefficient α_1 determines the degree of concentration/dispersion of the grid points around $r_{cheb}=\alpha_2$. If α_1 is too high, the r function becomes nearly discontinuous. To avoid numerical problems, α_1 should remain close to unity.

If the arcsin mapping is used, the function that re-distributes the collocation points is given by

$$r = \frac{1}{2}\left[\frac{\text{arcin}\left(\alpha_1 r_{cheb}\right)}{\text{arcsin}\alpha_1}\right] + \frac{r_i + r_o}{2} ,$$

In the Kosloff and Tal-Ezer mapping, α_1 transforms the Gauss-Lobatto grid into a more regularly-spaced grid. When $\alpha_1 \to 0$ one recovers the Gauss-Lobatto grid, while $\alpha_1 \to 1$ yields a regular grid.

Warning: The Kosloff-Tal-Ezer mapping becomes singular when $\alpha_1 = 1$. Acceptable values are $0 < \alpha_1 < 1$. Note that the error increases as $\epsilon = \left(\frac{1-\sqrt{1-\alpha_1^2}}{\alpha_1}\right)^{N_r}$.

- **alph1** (default `alph1=0.8`) is a real. This is a control parameter of the mapping function.
- **alph2** (default `alph2=0.0`) is a real. This is a control parameter of the mapping function.

6.2.8 Miscellaneous

- **l_non_rot** (default `l_non_rot=.false.`) is a logical. Use it when you want to do non-rotating numerical simulations.
- **anelastic_flavour** (default `anelastic_flavour="None"`) is a character string. This allows to change the thermal diffusion operator used within the anelastic approximation. Possible values are:

anelastic_flavour='LBR'	Entropy diffusion
anelastic_flavour='ENT'	Entropy diffusion
anelastic_flavour='ALA'	Anelastic liquid approximation
anelastic_flavour='TDIFF'	Temperature diffusion
anelastic_flavour='TEMP'	Temperature diffusion

- **polo_flow_eq** (default `polo_flow_eq="WP"`) is a character string. This allows to change how the equation for the poloidal flow potential is constructed. One can either use the radial component of the Navier-Stokes equation and hence keep a coupled system that involve the poloidal potential W and the pressure p, or take the radial component of the double-curl of the Navier-Stokes equation to suppress pressure.

polo_flow_eq='WP'	Use the pressure formulation
polo_flow_eq='DC'	Use the double-curl formulation

- **mpi_transp** (default `mpi_transp="auto"`) is a character string. It allows to change the way the global MPI transposes are handled by the code. By default, the code tries to determine by itself the fastest method. One can nevertheless force the code to use local communicators (such as Isend/Irecv/waitall), make use of the native alltoallv MPI variant or choose the alltoallw variant instead.

mpi_transp='auto'	Automatic determination of the fastest transpose
mpi_transp='p2p'	Use Isend/Irecv/Waitall communicators
mpi_transp='a2av'	Use alltoallv communicators
mpi_transp='a2aw'	Use alltoallw communicators

- **mpi_packing** (default `mpi_packing="packed"`) is a character string. It allows to change the size of the global MPI transposes. One can choose between some packing of the fields into buffers (default) or a sequence of single field transposes. There is a possible automatic detection but testing unfortunately reveals frequent false detection.

mpi_packing='auto'	Automatic determination of the fastest transpose
mpi_packing='packed'	Pack some fields into buffers
mpi_packing='single'	Transpose each field individually

- **l_adv_curl** (default `l_adv_curl=.false.`) is a logical. When set to True, the advection term is treated as $\boldsymbol{u} \times \boldsymbol{\omega}$ instead of $\boldsymbol{u}\boldsymbol{\nabla}\boldsymbol{u}$. The practical consequence of that is to reduce the number of spectral/spatial Spherical Harmonic Transforms and hence to speed-up the code. Because of the treatment of the viscous heating term in the anelastic approximation, this is only an option when considering Boussinesq models.

6.3 Physical parameters namelist

This namelist contains all the appropriate relevant control physical parameters.

6.3.1 Dimensionless control parameters

- **ra** (default `ra=0.0`) is a real. This the thermal Rayleigh number expressed by

$$Ra = \frac{\alpha g_o \Delta T d^3}{\kappa \nu}$$

- **raxi** (default `raxi=0.0`) is a real. This the compositional Rayleigh number expressed by

$$Ra_\xi = \frac{\alpha g_o \Delta \xi d^3}{\kappa_\xi \nu}$$

- **ek** (default `ek=1e-3`) is a real. This is the Ekman number expressed by

$$E = \frac{\nu}{\Omega d^2}$$

- **pr** (default `pr=1.0`) is a real. This is the Prandtl number expressed by

$$Pr = \frac{\nu}{\kappa}$$

- **sc** (default `sc=10.0`) is a real. This is the Schmidt number expressed by

$$Sc = \frac{\nu}{\kappa_\xi}$$

- **prmag** (default `prmag=5.0`) is a real. This is the magnetic Prandtl number expressed by

$$Pm = \frac{\nu}{\lambda}$$

- **po** (default `po=0.0`) is a real. This is the Poincaré number expressed by

$$Po = \frac{\Omega_p}{\Omega}$$

- **prec_angle** (default `prec_angle=23.5`) is a real. This is the angle between the precession and the rotation axes expressed in degrees.
- **radratio** (default `radratio=0.35`) is a real. This is the ratio of the inner core radius r_i to the outer core radius r_o:

$$\eta = \frac{r_i}{r_o}$$

- **strat** (default `strat=0.0`) is a real. This is the number of density scale heights of the reference state:

$$N_\rho = \ln \frac{\tilde{\rho}(r_i)}{\tilde{\rho}(r_o)}$$

- **DissNb** (default `DissNb=0.0`) is a real. This is the dissipation number:

$$Di = \frac{\alpha_o g_o d}{c_p}$$

> **Warning:** This can only be provided as a **replacement** input of **strat**. I.E., when one wants to define a reference state, one has to specify **either** `strat` **or** `DissNb` in the input namelist.

- **polind** (default `polind=1.5`) is a real. This is the polytropic index, which relates the background temperature to the background density:

$$\tilde{\rho} = \tilde{T}^m$$

> **Warning:** Be careful: in its current version the code only handles **adiabatic** backgrounds, therefore changing `polind` physically means that the nature of the fluid (in particular its Grüneisen parameter) will change. For an ideal gas, it actually always follows $m + 1 = \frac{\gamma - 1}{\gamma}$

- **l_isothermal** (default `l_isothermal=.false.`) is a logical. When set to `.true.`, makes the temperature background isothermal (i.e. $\tilde{T} = cst.$). In that case, the dissipation number Di vanishes and there is no viscous and Ohmic heating left. The only difference with the Boussinesq set of equations are thus restricted to the density background $\tilde{\rho}$ and its radial derivatives that enters the viscous stress. This approximation is also called the **zero Grüneisen parameter** and was extensively explored by Denise Tortorella during her PhD.

6.3.2 Heat sources and sinks

- **epsc0** (default `epsc0=0.0`) is a real. This is the volumetric heat source ϵ_0 that enters the thermal equilibrium relation:

$$-\nabla \cdot \left(\tilde{\rho} \tilde{T} \nabla s \right) + \epsilon_0 \, f(r) = 0 \tag{6.1}$$

  The radial function $f(r)$ can be modified with the variable `nVarEps` that enters the same input namelist.

- **epscxi0** (default `epscxi0=0.0`) is a real. This is the volumetric source ϵ_ξ that enters the compositional equilibrium relation:

$$-\nabla \cdot (\tilde{\rho} \nabla \xi) + \epsilon_\xi = 0 \tag{6.2}$$

- **nVarEps** (default `nVarEps=0`) is an integer. This is used to modify the radial-dependence ofthe volumetric heat source, i.e. $f(r)$ that enters equation (6.1).

`nVarEps=0`	Constant, i.e. $f(r) = $ cst..
`nVarEps=1`	Proportional to density, i.e. $f(r) = \tilde{\rho}(r)$.
`nVarEps=2`	Proportional to density times temperature, i.e. $f(r) = \tilde{\rho}(r)\tilde{T}$.

6.3.3 Realistic interior models

- **interior_model** (default `interior_model="None"`) is a character string. This defines a polynomial fit of the density profile of the interior structure of several astrophysical objects. Possible options are `"earth"`, `"jupiter"`, `"saturn"` and `"sun"` (the naming is **not** case sensitive).

  Warning: When `interior_model` is defined the variables `strat`, `polind`, `g0`, `g1` and `g2` are not interpreted.

  The subroutine `radial` gives the exact details of the implementation.

- **r_cut_model** (default `r_cut_model=0.98`) is a real. This defines the cut-off radius of the reference model, i.e. the fluid domain is restricted to radii with $r \leq r_{cut}$.

The following input parameters will thus define a polynomial fit to the expected interior structure of Jupiter until 99% of Jupiter's radius (assumed here at the 1 bar level)

```
interior_model="JUP",
r_cut_model   =0.99e0,
```

6.3.4 Gravity

The radial dependence of the gravity profile can be adjusted following

$$g(r) = g_0 + g_1 \frac{r}{r_o} + g_2 \left(\frac{r_o}{r}\right)^2 \tag{6.3}$$

The three following parameters are used to set this profile

- **g0** (default `g0=0`) is the pre-factor of the constant part of the gravity profile, i.e. g_0 in equation (6.3).
- **g1** (default `g1=1`) is the pre-factor of the linear part of the gravity profile, i.e. g_1 in equation (6.3).
- **g2** (default `g2=0`) is the pre-factor of the $1/r^2$ part of the gravity profile, i.e. g_2 in equation (6.3).

6.3.5 Centrifugal acceleration

The centrifugal acceleration can be computed for a polytropic background

- **dilution_fac** (default `dilution_fac=0.0`) is the ratio of the centrifugal acceleration at the equator to the surface gravitational acceleration.

$$m = \frac{\Omega^2 d}{g_o} \tag{6.4}$$

6.3.6 Transport properties

- **difExp** (default `difExp=-0.5`) is a real. This is the exponent that is used when `nVarVisc=2`, `nVarDiff=2` or `nVarCond=4`.

Electrical conductivity

There are several electrical conductivity profiles implemented in the code that can be chosen with the `nVarCond` input variable. The following one corresponds to a constant electrical conductivity in the deep interior ($r < r_m$) and an exponential decay in the outer layer.

$$\begin{aligned} \sigma(r) &= 1 + (\sigma_m - 1)\left(\frac{r - r_i}{r_m - r_i}\right)^a \quad \text{for} \quad r < r_m, \\ \sigma(r) &= \sigma_m \exp\left[a\left(\frac{r - r_m}{r_m - r_i}\right)\frac{\sigma_m - 1}{\sigma_m}\right] \quad \text{for} \quad r \geq r_m. \end{aligned} \tag{6.5}$$

- **nVarCond** (default `nVarCond=0`) is an integer. This is used to modify the radial-dependence of the electrical conductivity.

`nVarCond=0`	Constant electrical conductivity, i.e. $\sigma = \text{cst}$.
`nVarCond=1`	$\sigma \propto \tanh[a(r - r_m)]$
`nVarCond=2`	See equation (6.5).
`nVarCond=3`	**Magnetic diffusivity proportional to $1/\tilde{\rho}$, i.e.** $\lambda = \frac{\tilde{\rho}_i}{\tilde{\rho}}$
`nVarCond=4`	**Radial profile of the form:** $\lambda = \left(\frac{\tilde{\rho}(r)}{\tilde{\rho}_i}\right)^\alpha$

- **con_RadRatio** (default `con_RadRatio=0.75`) is a real. This defines the transition radius r_m that enters equation (6.5).
- **con_DecRate** (default `con_DecRate=9`) is an integer. This defines the decay rate a that enters equation (6.5).
- **con_LambdaMatch** (default `con_LambdaMatch=0.6`) is a real. This is the value of the conductivity at the transition point σ_m that enters equation (6.5).
- **con_LambdaOut** (default `con_LambdaOut=0.1`) is a real. This is the value of the conduvity at the outer boundary. This parameter is only used when `nVarCond=1`.
- **con_FuncWidth** (default `con_FuncWidth=0.25`) is a real. This parameter is only used when `nVarCond=1`.
- **r_LCR** (default `r_LCR=2.0`) is a real. `r_LCR` possibly defines a low-conductivity region for $r \geq r_{LCR}$, in which the electrical conductivity vanishes, i.e. $\lambda = 0$.

Thermal diffusivity

- **nVarDiff** (default `nVarDiff=0`) is an integer. This is used to change the radial-dependence of the thermal diffusivity:

`nVarDiff=0`	Constant thermal diffusivity κ
`nVarDiff=1`	**Constant thermal conductivity, i.e.** $$\kappa = \frac{\tilde{\rho}_i}{\tilde{\rho}(r)}$$
`nVarDiff=2`	**Radial profile of the form:** $$\kappa = \left(\frac{\tilde{\rho}(r)}{\tilde{\rho}_i}\right)^{\alpha}$$
`nVarDiff=3`	polynomial-fit to an interior model of Jupiter
`nVarDiff=4`	polynomial-fit to an interior model of the Earth liquid core

Viscosity

- **nVarVisc** (default `nVarVisc=0`) is an integer. This is used to change the radial-dependence of the viscosity:

`nVarVisc=0`	Constant kinematic viscosity ν
`nVarVisc=1`	**Constant dynamic viscosity, i.e.** $$\nu = \frac{\tilde{\rho}_o}{\tilde{\rho}(r)}$$
`nVarVisc=2`	**Radial profile of the form:** $$\nu = \left(\frac{\tilde{\rho}(r)}{\tilde{\rho}_i}\right)^{\alpha}$$

where α is an exponent set by the namelist input variable `difExp`.

6.3.7 Anelastic liquid equations

Warning: This part is still work in progress. The input parameters here are likely to be changed in the future.

- **epsS** (default `epsS=0.0`) is a real. It controls the deviation to the adiabat. It can be related to the small parameter ϵ:

$$\epsilon \simeq \frac{\Delta T}{T} \simeq \frac{\Delta s}{c_p}$$

- **cmbHflux** (default `cmbHflux=0.0`) is a real. This is the CMB heat flux that enters the calculation of the reference state of the liquid core of the Earth, when the anelastic liquid approximation is employed.

- **slopeStrat** (default `slopeStrat=20.0`) is a real. This parameter controls the transition between the convective layer and the stably-stratified layer below the CMB.

6.3.8 Boundary conditions

Thermal boundary conditions

- **ktops** (default `ktops=1`) is an integer to specify the outer boundary entropy (or temperature) boundary condition:

`ktops=1`	Fixed entropy at outer boundary: $s(r_o) = s_{top}$
`ktops=2`	Fixed entropy flux at outer boundary: $\partial s(r_o)/\partial r = q_t$
`ktops=3`	Fixed temperature at outer boundary: $T(r_o) = T_{top}$
`ktops=4`	Fixed temperature flux at outer boundary: $\partial T(r_o)/\partial r = q_t$

- **kbots** (default `ktops=1`) is an integer to specify the inner boundary entropy (or temperature) boundary condition.
- **s_top** (default `s_top= 0 0 0.0 0.0`) is a real array of lateraly varying outer heat boundary conditions. Each four consecutive numbers are interpreted as follows:
  1. Spherical harmonic degree ℓ
  2. Spherical harmonic order m
  3. Real amplitude ($\cos$ contribution)
  4. Imaginary amplitude ($\sin$ contribution)

  For example, if the boundary condition should be a combination of an $(\ell = 1, m = 0)$ sherical harmonic with the amplitude 1 and an $(\ell = 2, m = 1)$ spherical harmonic with the amplitude (0.5,0.5) the respective namelist entry could read:

```
s_top = 1, 0, 1.0, 0.0, 2, 1, 0.5, 0.5, ! The comas could be left away.
```

- **s_bot** (default `s_bot=0 0 0.0 0.0`) is a real array. This is the same as `s_top` but for the bottom boundary.
- **impS** (default `impS=0`) is an integer. This is a flag to indicate if there is a localized entropy disturbance, imposed at the CMB. The number of these input boundary conditions is stored in `n_impS` (the maximum allowed is 20), and it's given by the number of `sCMB` defined in the same namelist. The default value of `impS` is zero (no entropy disturbance). If it is set in the namelist for an integer greater than zero, then `sCMB` has to be also defined in the namelist, as shown below.
- **sCMB** (default `sCMB=0.0 0.0 0.0 0.0`) is a real array of CMB heat boundary conditions (similar to the case of `s_bot` and `s_top`). Each four consecutive numbers are interpreted as follows:
  1. Highest amplitude value of the entropy boundary condition, stored in array `peakS(20)`. When `impS<0`, `peakS` is a relative amplitude in comparison to the $(\ell = 0, m = 0)$ contribution (for example, the case `s_top= 0 0 -1 0`).
  2. θ coordinate (input has to be given in degrees), stored in array `thetaS(20)`.
  3. ϕ coordinate (input has to be given in degrees), stored in array `phiS(20)`.
  4. Angular width (input has to be given in degrees), stored in array `widthS(20)`.

Boundary conditions for chemical composition

- **ktopxi** (default `ktopxi=1`) is an integer to specify the outer boundary chemical composition boundary condition:

`ktopxi=1`	Fixed composition at outer boundary: $\xi(r_o) = \xi_{top}$
`ktopxi=2`	Fixed composition flux at outer boundary: $\partial\xi(r_o)/\partial r = q_t$

- **kbotxi** (default `ktopxi=1`) is an integer to specify the inner boundary chemical composition boundary condition.
- **xi_top** (default `xi_top= 0 0 0.0 0.0`) is a real array of lateraly varying outer chemical composition boundary conditions. Each four consecutive numbers are interpreted as follows:
  1. Spherical harmonic degree ℓ
  2. Spherical harmonic order m
  3. Real amplitude ($\cos$ contribution)
  4. Imaginary amplitude ($\sin$ contribution)

  For example, if the boundary condition should be a combination of an $(\ell = 1, m = 0)$ sherical harmonic with the amplitude 1 and an $(\ell = 2, m = 1)$ spherical harmonic with the amplitude (0.5,0.5) the respective namelist entry could read:

```
xi_top = 1, 0, 1.0, 0.0, 2, 1, 0.5, 0.5, ! The comas could be left away.
```

- **xi_bot** (default `xi_bot=0 0 0.0 0.0`) is a real array. This is the same as `xi_top` but for the bottom boundary.
- **impXi** (default `impXi=0`) is an integer. This is a flag to indicate if there is a localized chemical composition disturbance, imposed at the CMB. The number of these input boundary conditions is stored in `n_impXi` (the maximum allowed is 20), and it's given by the number of `xiCMB` defined in the same namelist. The default value of `impXi` is zero (no chemical composiiton disturbance). If it is set in the namelist for an integer greater than zero, then `xiCMB` has to be also defined in the namelist, as shown below.
- **xiCMB** (default `xiCMB=0.0 0.0 0.0 0.0`) is a real array of CMB chemical composition boundary conditions (similar to the case of `xi_bot` and `xi_top`). Each four consecutive numbers are interpreted as follows:
  1. Highest amplitude value of the chemical composition boundary condition, stored in the array `peakXi(20)`. When `impXi<0`, `peakXi` is a relative amplitude in comparison to the $(\ell = 0, m = 0)$ contribution (for example, the case `xi_top= 0 0 -1 0`).
  2. θ coordinate (input has to be given in degrees), stored in array `thetaXi(20)`.
  3. ϕ coordinate (input has to be given in degrees), stored in array `phiXi(20)`.
  4. Angular width (input has to be given in degrees), stored in array `widthXi(20)`.

Mechanical boundary conditions

- **ktopv** (default `ktopv=2`) is an integer, which corresponds to the mechanical boundary condition for $r = r_o$.

`ktopv=1`	**Stress-free outer boundary for $r = r_o$:** $$W_{\ell m}(r = r_o) = 0, \quad \frac{\partial}{\partial r}\left(\frac{1}{r^2\tilde{\rho}}\frac{\partial W_{\ell m}}{\partial r}\right) = 0$$ $$\frac{\partial}{\partial r}\left(\frac{1}{r^2\tilde{\rho}} Z_{\ell m}\right) = 0$$
`ktopv=2`	**Rigid outer boundary for $r = r_o$:** $$W_{\ell m} = 0, \quad \frac{\partial W_{\ell m}}{\partial r} = 0,$$ $$Z_{\ell m} = 0$$

- **kbotv** (default `kbotv=2`) is an integer, which corresponds to the mechanical boundary condition for $r = r_i$.

Magnetic boundary conditions

- **ktopb** (default `ktopb=1`) is an integer, which corresponds to the magnetic boundary condition for $r = r_o$.

`ktopb=1`	**Insulating outer boundary:** $$\frac{\partial g_{\ell m}}{\partial r} + \frac{\ell}{r} g_{\ell m} = 0, \quad \frac{\partial h_{\ell m}}{\partial r} = 0$$
`ktopb=2`	**Perfect condutor:** $$g_{\ell m} = \frac{\partial^2 g_{\ell m}}{\partial r^2} = 0, \quad \frac{\partial h_{\ell m}}{\partial r} = 0$$
`ktopb=3`	Finitely conducting mantle
`ktopb=4`	**Pseudo-vacuum outer boundary:** $$\frac{\partial g_{\ell m}}{\partial r} = 0, \quad h_{\ell m} = 0$$

- **kbotb** (default `kbotb=1`) is an integer, which corresponds to the magnetic boundary condition for $r = r_i$.

kbotb=1	**Insulating inner boundary:** $$\frac{\partial g_{\ell m}}{\partial r} - \frac{\ell+1}{r} g_{\ell m} = 0, \quad \frac{\partial h_{\ell m}}{\partial r} = 0$$
kbotb=2	**Perfectly-conducting inner core:** $$g_{\ell m} = \frac{\partial^2 g_{\ell m}}{\partial r^2} = 0, \quad \frac{\partial h_{\ell m}}{\partial r} = 0$$
kbotb=3	Finitely conducting inner core
kbotb=4	**Pseudo-vacuum outer boundary:** $$\frac{\partial g_{\ell m}}{\partial r} = 0, \quad h_{\ell m} = 0$$

Boundary condition for spherically-symmetric pressure

- **ktopp** (default `ktopp=1`) is an integer, which corresponds to the boundary condition for the spherically-symmetric pressure at $r = r_o$.

ktopp=1	The integral of the spherically-symmetric density perturbation vanishes.
ktopp=2	The spherically-symmetric pressure fluctuation vanishes at the outer boundary.

6.4 External Magnetic Field Namelist

The namelist `&B_external` provides options for imposing an external magnetic field.

6.4.1 Externally imposed magnetic field

- **n_imp** (default `n_imp = 0`) is an integer controling the type of external field applied.

n_imp=0	No external magnetic field
n_imp=1	Follows idea of Uli Christensen of external field compensating internal field such that radial component of magnetic field vanishes at $r/r_{cmb} = rrMP$ where `rrMP` is the 'magnetopause radius' input by the user (see below)
n_imp=2	Uniform axisymmetric magnetic field of geometry given by `l_imp` (see below)
n_imp=3	Uniform axisymmetric magnetic field which changes direction according to the direction of the axial dipole of the internal magnetic field
n_imp=4	Same as `n_imp=3` but the amplitude of the external field is scaled to the amplitude of the axial dipole of the internal field
n_imp=7	External field depends on internal axial dipole through Special Heyner feedback functions

- **rrMP** (default `rrMP = 0.0`) is a real which gives the value of 'magnetopause radius'. In other words, it gives the radius (as a fraction of `r_cmb`) at which the radial component of the magnetic field vanishes due to cancelling out of external and internal magnetic field components. Used only when `n_imp = 1`.
- **amp_imp** (default `amp_imp = 0.0`) is a real which gives the amplitude of the external magnetic field.

- **expo_imp** (default `expo_imp = 0.0`) is a real which gives the exponent of dependence of external magnetic field on the axial dipole of the internal magnetic field. Used for `n_imp=7`.
- **bmax_imp** (default `bmax_imp = 0.0`) is a real which gives the location of the maximum of the ratio of the poloidal potentials g_{ext}/g_{int}.
- **l_imp** (default `l_imp = 1`) is an integer which gives the geometry (degree of spherical harmonic) of the external magnetic field. The external field is always axisymmetric, hence `m = 0` always. This option is used when `n_imp = 2,3` or `4`.

6.4.2 Current carrying loop

To simulate experiments, an external current carrying loop, concentric to the sphere and in the equatorial plane, has been implemented in the code. It's radius is fixed at a distance $a = r_{cmb}/0.8$ to match conditions of the Maryland 3 metre experiment.

- **l_curr** (default `l_curr = .false.`) is a logical that controls switching on or off of the current carrying loop.
- **amp_curr** (default `amp_curr = 0.0`) is a real that gives the amplitude of magnetic field produced by the current carring loop.

Warning: Note that an external magnetic field is incompatible with a region of low conductivity inside the spherical shell (i.e, if `r_LCR < r_cmb`). Thus, while imposing an external magnetic field, make sure `r_LCR > r_cmb` (which is the default case). For details on `r_LCR`, have a look at the section on *electrical conductivity* in the namelist for *physical parameters*.

6.5 Start field namelist

This namelist controls whether a start field from a previous solution should be used, or a specific field should be initialized.

6.5.1 Reading an input file of start fields

- **l_start_file** (default `l_start_file=.false.`) is a logical that controls whether the code should to read a file named `start_file` or not.
- **start_file** (default `start_file="no_start_file"`) is a character string. This is the name of the *restart file*.
- **inform** (default `inform=-1`) is an integer that can be used to specify the format of `start_file`. This ensures possible backward compatibility with previous versions of the code. You shouldn't change this value except to read very old *checkpoint_end.TAG* files generated by older versions of MagIC.

`inform=0`	Oldest format used by U. Christensen
`inform=1`	Newer format used by U. Christensen
`inform=2`	Inner core introduced by J. Wicht
`inform=-1`	Default format

- **scale_s** (default `scale_s=1.0`) is a real. It can be possibly used to multiply the input entropy field from `start_file` by a constant factor `scale_s`.

- **scale_xi** (default `scale_xi=1.0`) is a real. It can be possibly used to multiply the input chemical composition field from `start_file` by a constant factor `scale_xi`.
- **scale_v** (default `scale_v=1.0`) is a real. It can be possibly used to multiply the input velocity field from `start_file` by a constant factor `scale_v`.
- **scale_b** (default `scale_b=1.0`) is a real. It can be possibly used to multiply the input magnetic field from `start_file` by a constant factor `scale_b`.
- **tipdipole** (default `tipdipole=0.0`) is a real that can be used to add non-axisymmetric disturbances to a start solution if non-axisymmetric parts have been lost due to mapping to a different symmetry. A $(\ell = 1, m = 1)$ entropy term is added with:

$$s_{10}(r) = \text{tipdipole } \sin[\pi(r - r_i)]$$

  If a magnetic field without an $m = 1$ term is mapped into a field that permits this term, the code adds the respective poloidal field using the $(\ell = 1, m = 0)$ poloidal magnetic field and scaling it with `tipdipole`.
- **l_reset_t** (default `l_reset_t=.false.`) is a logical that can be set to `.true.` in case one wants to reset the time of start file to zero.

6.5.2 Defining the starting conditions

Initialisation of entropy

The heat equation with possible heat sources and sinks given by *epsc0* is solved for the spherically-symmetric term $(\ell = 0, m = 0)$ to get its radial dependence. In addition to this initial state, two other laterally varying terms can be initialized. Their radial dependence are assumed to follow:

$$s(r) = 1 - 2\,x^2 + 3\,x^4 - x^6,$$

where

$$x = 2\,r - r_o - r_i\,.$$

The initial perturbation is thus set to zero at both boundaries r_i and r_o, and reaches its maximum amplitude of `amp_s1` or `amp_s2` at the mid-shell radius $r_i + 1/2$.

- **init_s1** (default `init_s1=0`) is an integer that controls the initial entropy. The following values are possible:
  - `init_s1=0`: nothing is initialized
  - `init_s1<100`: a random-noise of amplitude `amp_s1` is initialised. The subroutine `initS` in `init_fields.f90` gives the detail of this implementation.
  - `init_s1>100`: initialisation of mode with the spherical harmonic order m given by the last two (or three) digits of `init_s1` and the spherical harmonic degree ℓ given by the first two (or three) digits. Here are two examples:

```
init_s1   = 0707,
amp_s1    = 0.05,
```

    will introduce a perturbation on the mode $(\ell = 7, m = 7)$ with an amplitude of 0.05.

```
init_s1   = 121121,
amp_s1    = 0.01,
```

    will introduce a perturbation on the mode $(\ell = 121, m = 121)$ with an amplitude of 0.01.

- **amp_s1** (default `amp_s1=0.0`) is a real used to contol the amplitude of the perturbation defined by `init_s1`.
- **init_s2** (default `init_s2=0`) is an integer that controls a second spherical harmonic degee. It follows the same specifications as `init_s1`.
- **amp_s2** (default `amp_s2=0.0`) is a real used to contol the amplitude of the perturbation defined by `init_s2`.

Initialisation of chemical composition

The chemical composition equation with possible volumetric sources and sinks given by *epscxi0* is solved for the spherically-symmetric term $(\ell = 0, m = 0)$ to get its radial dependence. In addition to this initial state, two other laterally varying terms can be initialized. Their radial dependence are assumed to follow:

$$\xi(r) = 1 - 2\,x^2 + 3\,x^4 - x^6,$$

where

$$x = 2\,r - r_o - r_i\,.$$

The initial perturbation is thus set to zero at both boundaries r_i and r_o, and reaches its maximum amplitude of `amp_xi1` or `amp_xi2` at the mid-shell radius $r_i + 1/2$.

- **init_xi1** (default `init_xi1=0`) is an integer that controls the initial chemical composition. It follows the same specifications as `init_s1`.
- **amp_xi1** (default `amp_xi1=0.0`) is a real used to contol the amplitude of the perturbation defined by `init_xi1`.
- **init_xi2** (default `init_xi2=0`) is an integer that controls a second spherical harmonic degee. It follows the same specifications as `init_s1`.
- **amp_xi2** (default `amp_xi2=0.0`) is a real used to contol the amplitude of the perturbation defined by `init_xi2`.

Initialisation of magnetic field

- **init_b1** (default `init_b1=0`) is an integer that controls the initial magnetic field. The following values are possible:
  - `init_b1<0`: random noise initialization of all (ℓ, m) modes, except for $(\ell = 0, m = 0)$. The subroutine `initB` in the file `init_fields.f90` contains the details of the implementation.
  - `init_b1=0`: nothing is initialized
  - `init_b1=1`: diffusive toroidal field initialized. Mode determined by `imagcon`.
  - `init_b1=2`: $(\ell = 1, m = 0)$ toroidal field with a maximum field strength of `amp_b1`. The radial dependence is defined, such that the field vanishes at both the inner and outer boundaries. In case of an insulating inner core: $h(r) \approx r\,\sin[\phi(r - r_o)]$. In case of a conducting inner core: $h(r) \approx r\,\sin[\pi(r/r_o)]$.
  - `init_b1=3`: $(\ell = 1, m = 0)$ poloidal field whose field strength is `amp_b1` at $r = r_i$. The radial dependence is chosen such that the current density j is independent of r:, i.e. $\partial j/\partial r = 0$. $(\ell = 2, m = 0)$ toroidal field with maximum strength `amp_b1`.
  - `init_b1=4`: $(\ell = 1, m = 0)$ poloidal field as if the core were an insulator (potential field). Field strength at $r = r_i$ is again given by `amp_b1`.
  - `init_b1=5`: $(\ell = 1, m = 0)$ poloidal field with field strength `amp_b1` at $r = r_i$. The radial dependence is again defined by $\partial j/\partial r = 0$.

  - `init_b1=6`: $(\ell = 1, m = 0)$ poloidal field independend of r.
  - `init_b1=7`: $(\ell = 1, m = 0)$ poloidal field which fulfills symmetry condition in inner core: $g(r) \approx \left(\frac{r}{r_i}\right)^2 \left[1 - \frac{3}{5}\left(\frac{r}{r_o}\right)^2\right]$. The field strength is given by `amp_b1` at $r = r_o$.
  - `init_b1=8`: same poloidal field as for `init_b1=7`. The toroidal field fulfills symmetry conditions in inner core and has a field strength of `amp_b1` at $r = r_i$: $h(r) \approx \left(\frac{r}{r_i}\right)^3 \left[1 - \left(\frac{r}{r_o}\right)^2\right]$.
  - `init_b1=9`: $(\ell = 2, m = 0)$ poloidal field, which is a potential field at the outer boundary.
  - `init_b1=10`: equatorial dipole only.
  - `init_b1=11`: axial and equatorial dipoles.
  - `init_b1=21`: toroidal field created by inner core rotation, equatorially symmetric $(\ell = 1, m = 0)$: $h(r) = \text{ampb1}\ \left(\frac{r_i}{r}\right)^6$. The field strength is given by `amp_b1` at $r = r_i$.
  - `init_b1=22`: toroidal field created by inner core rotation, equatorially antisymmetric $(\ell = 2, m = 0)$. Same radial function as for `init_b1=21`.

- **amp_b1** (default `amp_b1=0.0`) is a real used to contol the amplitude of the function defined by `init_b1`.
- **imagcon** (default `imagcon=0`) is an integer, which determines the imposed magnetic field for magnetoconvection. The magnetic field is imposed at boundaries.
  - `imagcon=0`: no magneto-convection
  - `imagcon<0`: axial poloidal dipole imposed at ICB with a maximum magnetic field strength `amp_b1`.
  - `imagcon=10`: $(\ell = 2, m = 0)$ toroidal field imposed at ICB and CMB with a maximum amplitude `amp_b1` at both boundaries.
  - `imagcon=11`: same as `imagcon=10` but the maximum amplitude is now `amp_b1` at the ICB and `-amp_b1` at the CMB.
  - `imagcon=12`: $(\ell = 1, m = 0)$ toroidal field with a maximum amplitude of `amp_b1` at the ICB and the CMB.
- **tmagcon** (`tmagcon=0.0`) is a real.

Initialisation of velocity field

- **init_v1** (default `init_v1=0`) is an integer that controls the initial velocity. The following values are possible:
  - `init_v1=0`: nothing is initialized
  - `init_v1=1`: a differential rotation profile of the form

$$\Omega = \Omega_{ma} + 0.5\Omega_{ic} \quad \text{for} \quad s \leq r_i$$
$$\Omega = \Omega_{ma} \quad \text{for} \quad s > r_i$$

    where $s = r\sin\theta$ is the cylindrical radius. This profile only makes sense when one studies spherical Couette flows.
  - `init_v1=2`: a differential rotation profile of the form $\Omega = \frac{\text{ampv1}}{\sqrt{1+s^4}}$ is introduced.
  - `init_v1>2`: a random-noise of amplitude `amp_v1` is initialised. The subroutine `initV` in `init_fields.f90` gives the detail of this implementation.
- **amp_v1** (default `amp_v1=0.0`) is a real used to contol the amplitude of the function defined by `init_v1`.

6.6 Output control namelist

This namelist contains all the parameters that can be adjusted to control the outputs and diagnostics calculated by the code.

There are four different ways to control at which time step a specific output should be written. Outputs are generally distributed over the total calculation intervall unless an output time interval is defined by a start time `t_start` and a stop time `t_stop`. If no `t_start` is provided, the start time of the calculation is used. If no `t_stop` is provided or `t_stop>t_start` the total calculation interval is assumed

1. **Prescribed number of outputs**. The outputs are distributed evenly over the total calculation interval so that the number of timesteps between two outputs is always the same, with the possible exception of the first interval. Last output is written for the last time step, and to compensate the interval before the first output may be longer. However, if `t_stop` is provided, the outputs are distributed evenly over the interval [`t_stop`, `t_start`] with equal times intervals between them.

Note: These input variables are usually named with a pattern that follows `n_outputName`, for instance, *n_graphs*, *n_rsts*, *n_specs*, *n_logs*, etc.

In case you want to make use ot a specific time interval, the input variables follow a pattern of the form `t_outputName_start`, `t_outputName_stop`. For instance, *t_graph_start*, *t_graph_stop*, *t_log_start*, *t_log_stop*, *t_spec_start*, *t_spec_stop*, etc.

---

2. **User-defined interval between two outputs, given in number of time steps**. Again the last output is performed at the end of the run and a compensation may take place at the beginning.

Note: These input variables are usually named with a pattern that follows `n_outputName_step`, for instance, *n_graph_step*, *n_rst_step*, *n_spec_step*, *n_log_step*, *n_movie_step*, etc.

---

3. **Defined time interval between two outputs**.

Note: These input variables are usually named with a pattern that follows `dt_outputName`, for instance, *dt_graph*, *dt_rst*, *dt_spec*, *dt_log*, *dt_movie*, etc.

---

4. **User-defined times for output**. By default 5000 different times can be defined for each output type. This can be increased by increasing n_time_hits in the file `output_data.f90`. While the first three possibilities can only be used alternatively, the fourth one can be employed in addition to one of the two others.

Note: These input variables are usually named with a pattern that follows `t_outputName`, for instance, *t_graph*, *t_rst*, *t_spec*, *t_log*, *t_movie*, etc.

The different possible outputs control parameters are then extensively described in the following pages:

Possible outputs

1. *Control standard/common outputs*
2. *CMB and radial coefficients*
3. *Storage of potentials in spectral space*

4. *Torsional oscillations diagnostics*
5. *Additional possible diagnostics*

6.6.1 Standard time-series outputs

The **log** outputs controls the output of all the default time series of the file: kinetic and magnetic energies (*e_kin.TAG*, *e_mag_oc.TAG* and *e_mag_ic.TAG* files), dipole information (*dipole.TAG* file), rotation (*rot.TAG*) parameters (*par.TAG*) and various additional diagnostics (*heat.TAG*):

- **n_log_step** (default `n_log_step=50`) is an integer. This is the number of timesteps between two log outputs.

  Warning: Be careful: when using too small `n_log_step`, the disk access will dramatically increases, thus decreasing the code performance.

- **n_logs** (default `n_logs=0`) is an integer. This is the number of log-information sets to be written.
- **t_log** (default `t_log=-1.0 -1.0 ...`) is real array, which contains the times when log outputs are requested.
- **dt_log** (default `dt_log=0.0`) is a real, which defines the time interval between log outputs.
- **t_log_start** (default `t_log_start=0.0`) is a real, which defines the time to start writing log outputs.
- **t_log_stop** (default `t_log_stop=0.0`) is a real, which defines the time to stop writing log outputs.

6.6.2 Restart files

The **rst** outputs controls the output of restart files (*checkpoint_t_#.TAG*) (i.e. check points in time from which the code could be restarted):

- **n_rst_step** (default `n_rst_step=0`) is an integer. This is the number of timesteps between two restart files.
- **n_rsts** (default `n_rsts=1`) is an integer. This is the number of restart files to be written.
- **t_rst** (default `t_rst=-1.0 -1.0 ...`) is real array, which contains the times when restart files are requested.
- **dt_rst** (default `dt_rst=0.0`) is a real, which defines the time interval between restart files.
- **t_rst_start** (default `t_rst_start=0.0`) is a real, which defines the time to start writing restart files.
- **t_rst_stop** (default `t_rst_stop=0.0`) is a real, which defines the time to stop writing restart files.
- **n_stores** (default `n_stores=0`) is an integer. This is another way of requesting a certain number of restart files. However, instead of creating each time a new restart file, if `n_stores > n_rsts` the restart file is overwritten, which can possibly help saving some disk space.

Warning: The `rst` files can become quite big and writting them too frequently will slow down the code. Except for very special use, the default set up should be sufficient.

6.6.3 Graphic files

The **graph** outputs controls the output of graphic files (*G_#.TAG*) which contain a snapshot the entropy, the velocity field and the magnetic fields:

- **n_graph_step** (default `n_graph_step=0`) is an integer. This is the number of timesteps between two graphic files.
- **n_graphs** (default `n_graphs=1`) is an integer. This is the number of graphic files to be written.
- **t_graph** (default `t_graph=-1.0 -1.0 ...`) is real array, which contains the times when graphic files are requested.
- **dt_graph** (default `dt_graph=0.0`) is a real, which defines the time interval between graphic files.
- **t_graph_start** (default `t_graph_start=0.0`) is a real, which defines the time to start writing graphic files.
- **t_graph_stop** (default `t_graph_stop=0.0`) is a real, which defines the time to stop writing graphic files.

6.6.4 Spectra

The **spec** outputs controls the output of spectra: kinetic energy spectra (*kin_spec_#.TAG*), magnetic energy spectra (*mag_spec_#.TAG*) and thermal spectra (*T_spec_#.TAG*):

- **n_spec_step** (default `n_spec_step=0`) is an integer. This is the number of timesteps between two spectra.
- **n_specs** (default `n_specs=0`) is an integer. This is the number of spectra to be written.
- **t_spec** (default `t_spec=-1.0 -1.0 ...`) is real array, which contains the times when spectra are requested.
- **dt_spec** (default `dt_spec=0.0`) is a real, which defines the time interval between spectra.
- **t_spec_start** (default `t_spec_start=0.0`) is a real, which defines the time to start writing spectra.
- **t_spec_stop** (default `t_spec_stop=0.0`) is a real, which defines the time to stop writing spectra.
- **l_2D_spectra** (default `l_2D_spectra=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of 2-D spectra in the (r, ℓ) and in the (r, m) parameter spaces. Those data are stored in the files named *2D_[mag|kin]_spec_#.TAG*.

6.6.5 Movie files

The **movie** outputs controls the output of movie files (**_mov.TAG*).

Specific inputs

- **l_movie** (default `l_movie=.false.`) is a logical. It needs to be turned on to get movie computed.

  Several movie-files can be produced during a run (it is now limited to 30 by the variable``n_movies_max`` in the module `movie`). The movies are defined by a keyword determining the fields to be plotted and an expression that determines the nature of movie (r-slice, θ-slice, ϕ-slice, etc.). The code searches this information in a character string provided for each movie. These strings are elements of the array *movie*:

- **movie** (default `movie=' ', ' ', ...`) is a character string array. It contains the description of the movies one wants to compute.

  For example, to invoke a movie(file) that shows (stores) the radial magnetic component of the magnetic field at the CMB, you have to provide the line

```
movie(1)="Br CMB",
```

in the *&output* namelist. Here, `Br` is the keyword for the radial component of the magnetic field and `CMB` is the expression that defines the movie surface. If, in addition, a movie of the temperature field at the meridional slice `phi=0` and a movie of the z-vorticity in the equatorial plane are desired, the following line have to be added:

```
movie(2)="Temp phi=0",
movie(3)="Vortz eq",
```

Note that the code does **not interpret spaces and ignores additional characters** that do not form a keyword or a surface definition. Thus, for example `Br` or `B r` or `Bradial` are all interpreted as the same keyword. Furthermore, the interpretation is **not case-sensitive**. The following table gives the possible keywords for movie calculations and their corresponding physical meaning:

Keyword	Fields stored in movie file
Br[radial]	Radial component of the magnetic field B_r.
Bt[heta]	Latitudinal component of the magnetic field B_θ.
Bp[hi]	Azimuthal component of the magnetic field B_ϕ.
Bh[orizontal]	The two horizontal components of the magnetic field.
Bs	Cylindrically radial component of the magnetic field B_s.
Ba[ll]	All magnetic field components.
Fieldline[s] or FL	Axisymmetric poloidal field lines in a meridional cut.
AX[ISYMMETRIC] B or AB	Axisymmetric phi component of the magnetic field for $\phi = cst$.
Vr[adial]	Radial component of the velocity field u_r.
Vt[heta]	Latitudinal component of the velocity field u_θ.
Vp[hi]	Azimuthal component of the velocity field u_ϕ.
Vh[orizontal]	Horizontal velocity field, two components depending on the surface.
Va[ll]	All velocity field components.
Streamline[s] or SL	Field lines of axisymmetric poloidal field for $\phi = cst$.
AX[ISYMMETRIC] V or AV	Axisymmetric component of the velocity field for $\phi = cst$.
Vz	Vertical component of the velocity at the equator + vertical component of the vorticity at the equator (closest point to equator).
Voz	Vertical component of the vorticity ω_z.
Vor	Radial component of the vorticity ω_r.
Vop	Azimuthal component of vorticity ω_ϕ
Tem[perature] or Entropy	Temperature/Entropy
Entropy (or Tem[perature]) AX[ISYMMETRIC] or AT	Axisymmetric temperature/entropy field for $\phi = cst$.
Heat t[ransport]	Radial advection of temperature $u_r \frac{\partial s}{\partial r}$
HEATF AX[iSYMMETRIC]	Conducting heat flux $\partial s/\partial r$
FL Pro	Axisymmetric field line stretching.
FL Adv	Axisymmetric field line advection.
FL Dif	Axisymmetric field line diffusion.
AB Pro	Toroidal axisymmetric field production.
AB Dif	Toroidal axisymmetric field diffusion.
Br Pro	Production of radial magnetic field B_r.

continues on next page

Table 1 – continued from previous page

Keyword	Fields stored in movie file
Br Adv	Advection of radial magnetic field B_r.
Br Dif	Diffusion of radial magnetic field B_r.
Jr	Radial component of the current j_r.
Jr Pro	Production of radial current + Ω-effect.
Jr Adv	Advection of the radial component of the current j_r.
Jr Dif	Diffusion of the radial component of the current j_r.
Bz Pol	Poloidal part of vertical component of the magnetic field B_z.
Bz Pol Pro	Production of the poloidal part of the vertical component of the magnetic field B_z.
Bz Pol Adv	Advection of the poloidal part of the vertical component of the magnetic field B_z.
Bz Pol Dif	Diffusion of the poloidal part of the vertical component of the magnetic field B_z.
Jz Tor	Toroidal part of the vertical component of the current (j_z).
Jz Tor Pro	Production of the toroidal part of the vertical component of the current j_z.
Jz Tor Adv	Advection of the toroidal part of the vertical component of the current j_z.
Jz Tor Dif	Diffusion of the toroidal part of the vertical component of the current j_z.
Bp Tor	Toroidal part of the azimuthal component of the magnetic field B_ϕ.
Bp Tor Pro	Production of the toroidal part of the azimuthal component of the magnetic field B_ϕ.
Bp Tor Adv	Advection of the toroidal part of the azimuthal component of the magnetic field B_ϕ.
Bp Tor Dif	Diffusion of the toroidal part of the azimuthal component of the magnetic field B_ϕ.
HEL[ICITY]	Kinetic helicity $\mathcal{H} = \boldsymbol{u} \cdot (\boldsymbol{\nabla} \times \boldsymbol{u})$
AX[ISYMMETRIC HELICITY] or AHEL	Axisymmetric component of the kinetic helicity.
Bt Tor	Toroidal component of the latitudinal component of the magnetic field B_θ.
Pot Tor	Toroidal potential.
Pol Fieldlines	Poloidal fieldlines.
Br Shear	Azimuthal shear of the radial component of the magnetic field B_r
Lorentz[force] or LF	Lorentz force (only ϕ-component).
Br Inv	Inverse field apperance at CMB.

The following table gives the possible surface expression for movie calculations and their corresponding physical meaning:

Surface expression	Definition
CMB	Core-mantle boundary
Surface	Earth surface
EQ[uatot]	Equatorial plane
r=radius	Radial cut at r=radius with radius given in units of the outer core radius.
theta=colat	Latitudinal cut at theta=colat given in degrees
phi=phiSlice	Azimuthal cut ath phi=phiSlice given in degrees.
AX[isymmetric]	Axisymmetric quantity in an azimuthal plane
3D	3D array

Here is an additional example of the possible combinations to build your desired `movie` files.

```
l_movie  = .true.,
movie(1) = "Br CMB",
movie(2) = "Vr EQ",
movie(3) = "Vortr r=0.8",
movie(4) = "Bp theta=45",
movie(5) = "Vp phi=10",
movie(6) = "entropy AX",
movie(7) = "vr 3D",
```

Standard inputs

- **n_movie_step** (default `n_movie_step=0`) is an integer. This is the number of timesteps between two movie outputs.
- **n_movies** (default `n_movies=1`) is an integer. This is the number of movie outputs to be written.
- **t_movie** (default `t_movie=-1.0 -1.0 ...`) is real array, which contains the times when movie outputs are requested.
- **dt_movie** (default `dt_movie=0.0`) is a real, which defines the time interval between movie outputs.
- **t_movie_start** (default `t_movie_start=0.0`) is a real, which defines the time to start writing movie outputs.
- **t_movie_stop** (default `t_movie_stop=0.0`) is a real, which defines the time to stop writing movie outputs.

6.6.6 Field Averages

The code can perform on-the-fly time-averaging of entropy, velocity field and magnetic field. Respective graphic output and spectra are written into the corresponding files (with *G_ave.TAG*, *kin_spec_ave.TAG*, *mag_spec_ave.TAG*). The time-averaged energies are written into the *log.TAG* file.

- **l_average** (default `l_average=.false.`) is a logical, which enables the time-averaging of fields when set to `.true.`.

> **Warning:** Time-averaging has a large memory imprint as it requires the storage of 3-D arrays. Be careful, when using large truncations.

- **l_spec_avg** (default `l_spec_avg=.false.`) is a logical, which enables the time-averaging of spectra when set to `.true.`. It is always set to `.true.`, if *l_average=.true.*.

6.6.7 Poloidal magnetic field potential at CMB

The **cmb** outputs controls the output of poloidal field potential coefficients at the CMB $b_{\ell m}(r = r_o)$: *B_coeff_cmb.TAG* up to a maximum spherical harmonic degree `l_max_cmb`.

Note: This calculation is **only** enabled when `l_cmb_field=.true.` or when `l_dt_cmb_field=.true.`.

Specific inputs

- **l_cmb_field** (default `l_cmb_field=.false.`) is a logical. It needs to be turned on to get `cmb` files computed.
- **l_dt_cmb_field** (default `l_dt_cmb_field=.false.`) is a logical. When set to `.true.`, it allows the calculation of the secular variation of the magnetic field at the CMB.
- **l_max_cmb** (default `l_max_cmb=14`) is an integer. This is the maximum spherical harmonic degree ℓ stored in *B_coeff_cmb.TAG*, i.e. only $\ell \leq \ell_{maxcmb}$ are stored. For example, the following input parameter means that the *B_coeff_cmb.TAG* file is stored up to a spherical harmonic degree of ℓ:

```
l_cmb_field = .true.,
l_max_cmb   = 20,
```

Standard inputs

- **n_cmb_step** (default `n_cmb_step=0`) is an integer. This is the number of timesteps between two `cmb` outputs.
- **n_cmbs** (default `n_cmbs=0`) is an integer. This is the number of `cmb` outputs to be written.
- **t_cmb** (default `t_cmb=-1.0 -1.0 ...`) is real array, which contains the times when `cmb` outputs are requested.
- **dt_cmb** (default `dt_cmb=0.0`) is a real, which defines the time interval between `cmb` outputs.
- **t_cmb_start** (default `t_cmb_start=0.0`) is a real, which defines the time to start writing `cmb` outputs.
- **t_cmb_stop** (default `t_cmb_stop=0.0`) is a real, which defines the time to stop writing `cmb` outputs.

6.6.8 Poloidal and toroidal potentials at several depths

The `coeff_r#` outputs controls the output of the poloidal and toroidal potential coefficients at several depths up to a maximum spherical harmonic degree `l_max_r`. The files *B_coeff_r#.TAG* and *V_coeff_r#.TAG* are written when `l_r_field=.true.`. The file *T_coeff_r#.TAG* is written when `l_r_fieldT=.true.`.

Note: This calculation is **only** enabled when `l_r_field=.true.` or when `l_r_fieldT=.true.`.

Specific inputs

- **l_r_field** (default `l_r_field=.false.`) is a logical. It needs to be turned on to get `r_field` files computed.
- **l_r_fieldT** (default `l_r_fieldT=.false.`) is a logical. When set to `.true.`, the thermal field is also stored in a file named *T_coeff_r*.TAG*.
- **l_max_r** (default `l_max_r=l_max`) is an integer. This is the maximum spherical harmonic degree ℓ stored in the `r_field` file, i.e. only $\ell \leq \ell_{maxcmb}$ are stored.

There are two ways to specify the radial grid points where you want to store the `[B|V|T]_coeff_r#.TAG` files. You can specify a stepping `n_r_step`: in that case **5** `coeff_r#.TAG` files will be stored at 5 different radial levels every `n_r_step` grid point:

```
l_r_field = .true.,
n_r_step  = 6,
l_max_r   = 30,
```

This will produces 5 files that contain the poloidal and toroidal potentials up to spherical harmonic degree $\ell = 30$:

- `[B|V|T]_coeff_r1.TAG` corresponds to the radial grid point with the index `nR=6`.
- `[B|V|T]_coeff_r2.TAG` to `nR=12`.
- `[B|V|T]_coeff_r3.TAG` to `nR=18`.
- `[B|V|T]_coeff_r4.TAG` to `nR=24`.
- `[B|V|T]_coeff_r5.TAG` to `nR=30`.

- **n_r_step** (default `n_r_step=2`) is an integer. This specifies the stepping between two consecutive `[B|V|T]_coeff_r#.TAG` files.

Alternatively, the input array `n_r_array` can be used to specify the radial grid points you exactly want to store:

```
l_r_field = .true.,
n_r_array = 8, 24, 47,
l_max_r   = 10,
```

This will produces 3 files that contain the poloidal and toroidal potentials up to spherical harmonic degree $\ell = 10$:

- `[B|V|T]_coeff_r1.TAG` corresponds to the radial grid point with the index `nR=8`.
- `[B|V|T]_coeff_r2.TAG` to `nR=24`.
- `[B|V|T]_coeff_r3.TAG` to `nR=47`.

- **n_r_array** (default `n_r_array=0 0 0 ...`) a an integer array. You can specify the radial grid points (starting from `n_r_cmb=1`) where you want to store the coefficients.

Standard inputs

- **n_r_field_step** (default `n_r_field_step=0`) is an integer. This is the number of timesteps between two `r_field` outputs.
- **n_r_fields** (default `n_r_fields=0`) is an integer. This is the number of `r_field` outputs to be written.
- **t_r_field** (default `t_r_field=-1.0 -1.0 ...`) is real array, which contains the times when `r_field` outputs are requested.
- **dt_r_field** (default `dt_r_field=0.0`) is a real, which defines the time interval between `r_field` outputs.
- **t_r_field_start** (default `t_r_field_start=0.0`) is a real, which defines the time to start writing `r_field` outputs.
- **t_r_field_stop** (default `t_r_field_stop=0.0`) is a real, which defines the time to stop writing `r_field` outputs.

6.6.9 Poloidal and toroidal potentials in spectral and radial space

The **[V|B|T]_lmr** outputs controls the output of potential files (*V_lmr_#.TAG*, *B_lmr_#.TAG* and *T_lmr_#.TAG*). These are files contain the poloidal and toroidal flow and magnetic field potentials (and entropy/temperature) written in spectral and radial spaces (for instance `w(lm_max, n_r_max)`). These files can be quite handy since they can be possibly used to reconstruct any quantity in the spectral space or in the physical space you may be interested in.

Standard inputs

- **n_pot_step** (default `n_pot_step=0`) is an integer. This is the number of timesteps between two `[V|B|T|Xi]_lmr` outputs.
- **n_pots** (default `n_pots=1`) is an integer. This is the number of `[V|B|T|Xi]_lmr` outputs to be written.
- **t_pot** (default `t_pot=-1.0 -1.0 ...`) is real array, which contains the times when `[V|B|T|Xi]_lmr` outputs are requested.
- **dt_pot** (default `dt_pot=0.0`) is a real, which defines the time interval between two `[V|B|T|Xi]_lmr` outputs.
- **t_pot_start** (default `t_pot_start=0.0`) is a real, which defines the time to start writing `[V|B|T|Xi]_lmr` outputs.
- **t_pot_stop** (default `t_pot_stop=0.0`) is a real, which defines the time to stop writing `[V|B|T|Xi]_lmr` outputs.

6.6.10 Torsional oscillations (`TO`)

Specific inputs

- **l_TO** (default `l_TO=.false.`) is a logical. It needs to be turned on to compute the torsional oscillations diagnostics (`TO`) computed.
- **l_TOmovie** (default `l_TOmovie=.false`) is a logical. It needs to be turned on to store the *TO_movie.TAG* files.
- **sDens** (default `sDens=1.0`) is a float. It gives the relative point density of the cylindrical grid (in the radial direction).
- **zDens** (default `zDens=1.0`) is a float. It gives the relative point density of the cylindrical grid (in the vertical direction).

Standard inputs

- **n_TO_step** (default `n_TO_step=0`) is an integer. This is the number of timesteps between two `TO` outputs.
- **n_TOs** (default `n_TOs=1`) is an integer. This is the number of `TO` outputs to be written.
- **t_TO** (default `t_TO=-1.0 -1.0 ...`) is real array, which contains the times when `TO` outputs are requested.
- **dt_TO** (default `dt_TO=0.0`) is a real, which defines the time interval between `TO` outputs.
- **t_TO_start** (default `t_TO_start=0.0`) is a real, which defines the time to start writing `TO` outputs.
- **t_TO_stop** (default `t_TO_stop=0.0`) is a real, which defines the time to stop writing `TO` outputs.
- **n_TOmovie_step** (default `n_TOmovie_step=0`) is an integer. This is the number of timesteps between two `TO_mov` outputs.

- **n_TOmovie_frames** (default `n_TOmovies=1`) is an integer. This is the number of `TO_mov` outputs to be written.
- **t_TOmovie** (default `t_TOmovie=-1.0 -1.0 ...`) is real array, which contains the times when `TO_mov` outputs are requested.
- **dt_TOmovie** (default `dt_TOmovie=0.0`) is a real, which defines the time interval between `TO_mov` outputs.
- **t_TOmovie_start** (default `t_TOmovie_start=0.0`) is a real, which defines the time to start writing `TO_mov` outputs.
- **t_TOmovie_stop** (default `t_TOmovie_stop=0.0`) is a real, which defines the time to stop writing `TO_mov` outputs.

6.6.11 RMS force balance

The code can compute the RMS contributions of the different forces that contribute to the Navier-Stokes equation and the the different terms that enter the induction equation.

- **l_RMS** (default `l_RMS=.false.`) is a logical, which enables the calculation of RMS force balance, when set to `.true.`. The outputs are stored in *dtVrms.TAG*, *dtBrms.TAG* and *dtVrms_spec.TAG*.
- **rCut** (default `rCut=0.0`) is a float. This is the thickness of the layer which is left out at both boundaries for the RMS calculation. `rCut=0.075` actually means that 7.5% below the CMB and above the ICB are disregarded in the force balance calculation.
- **rDea** (default `rDea=0.0`) is a float. This controls the dealiasing in RMS calculations. `rDea=0.1` means that the highest 10% of the Chebyshev modes are set to zero.
- **l_2D_RMS** (default `l_2D_RMS=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of 2-D force balance in the (r, ℓ) and parameter space. Those data are stored in the files named *2D_dtVrms_spec.TAG*.

6.6.12 Additional possible diagnostics

Geostrophy

- **l_par** (default `l_par=.false.`) is a logical. When set to `.true.`, this logical enables additional calculations (for instance the degree of geostrophy). The details of these calculations can be found in the subroutine `getEgeos` in the `Egeos.f90` file. These quantities are then stored in the columns 10-16 of the *geos.TAG* file.
- **l_corrMov** (default `l_corrMov=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of a movie file that stores North/South correlation in the `CVorz_mov.TAG` file.

Helicity

- **l_hel** (default `l_hel=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of helicity (RMS, northern and southern hemisphere, etc.). The outputs are stored in the columns 6-9 of the *helicity.TAG* file.

Power budget

- **l_power** (default `l_power.false.`) is a logical. When set to `.true.`, this logical enables the calculation of input and output power (buoyancy, viscous and ohmic dissipations, torques). The time series are stored in *power.TAG* and *dtE.TAG* and the time-averaged radial profiles in *powerR.TAG*.

Angular momentum

- **l_AM** (default `l_AM=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of angular momentum. The time series are stored in *AM.TAG*.

Earth-likeness of the CMB field

- **l_earth_likeness** (default `l_earth_likeness=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of the Earth-likeness of the CMB magnetic field following (Christensen et al., 2010). The time series of the four criteria are stored in *earth_like.TAG*.
- **l_max_comp** (default `l_max_comp=8`) is an integer. This is the maximum spherical harmonic degree used to calculate the Earth-likeness of the CMB field.

Drift rates

- **l_drift** (default `l_drift=.false.`) is a logical. When set to `.true.`, this logical enables the storage of some selected coefficients to allow the calculation of the drift rate. The time series are stored in *drift[V\B][DQ].TAG*.

Inertial modes

- **l_iner** (default `l_iner=.false.`) is a logical. When set to `.true.`, this logical enables the storage of some selected $w(\ell, m)$ at mid-shell (stored in *inerP.TAG*) and $z(\ell, m)$ at mid-shell (stored in *inerT.TAG*). Those files can be further used to identify inertial modes.

Radial spectra

- **l_rMagSpec** (default `l_rMagSpec=.false`) is a logical. When set to `.true.`, the magnetic spectra for the first 6 spherical harmonic degree ℓ for all radii are stored at times of log ouputs. This produces the unformatted fortran files *rBrSpec.TAG* and *rBpSpec.TAG*.
- **l_DTrMagSpec** (default `l_DTrMagSpec=.false`) is a logical. When set to `.true.`, the magnetic spectra of the magnetic field production terms for the first 6 spherical harmonic degree ℓ for all radii are stored at times of log ouputs. This produces the unformatted fortran files `rBrProSpec.TAG`, `rBrAdvSpec.TAG`, `rBrDifSpec.TAG`, `rBrDynSpec.TAG`, `rBpProSpec.TAG`, `rBpAdvSpec.TAG`, `rBpDifSpec.TAG` and `rBpDynSpec.TAG`. All those files have exactly the same format as the *rBrSpec.TAG*.

Heat transport

- **l_fluxProfs** (default `l_fluxProfs=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of time-averaged radial heat flux profiles (conductive flux, convective flux, kinetic flux, viscous flux, Poynting flux and resistive flux). The time-averaged radial profiles are stored in the *fluxesR.TAG* file.

Boundary layer analysis

- **l_viscBcCalc** (default `l_viscBcCalc=.false.`) is a logical. When set to `.true.`, this logical enables the calculation of time-averaged radial profiles that can be further use to determine the viscous and thermal boundary layer thicknesses: temperature, temperature variance, horizontal velocity, etc. The time-averaged radial profiles are stored in the *bLayersR.TAG* file.

Parallel/perpendicular decomposition

- **l_perpPar** (default `l_perpPar=.false.`) is a logical. When set to `.true.`, this logical enables the decomposition of kinetic energy into components parallel and perpendicular to the rotation axis. The time series are stored in *perpPar.TAG* and the time-averaged radial profiles in *perpParR.TAG*.

Pressure

- **l_PressGraph** (default `l_PressGraph=.true.`) is a logical. When set to `.true.`, this logical enables the storage of pressure in the *graphic files*.

Time evolution of the m-spectra

- **l_energy_modes** (default `l_energy_modes=.false.`) is a logical. When set to `.true.`, this logical enables the storage of the time-evolution of the kinetic and magnetic energy spectra for a given range of spherical harmonic orders: *time spectra*.
- **m_max_modes** (default `m_max_modes=13`) is an integer. This controls the maximum spherical harmonic order when `l_energy_modes=.true.`.

6.6.13 Generic options

- **l_save_out** (default `l_save_out=.false.`) is a logical. When set to `.true.`, the diagnostic files will be safely opened and closed before and after any outputs. When set to `.false.`, the diagnostic files will be opened before the first iteration timestep and close at the end of the run. This may cost some computing time, but guarantees that only minimal information is lost in case of a crash.
- **lVerbose** (default `lVerbose=.false.`) is a logical. When set to `.true.`, the code displays a lot of debugging informations.

> **Warning:** Never set `lVerbose` to `.true.` for a production run!

- **runid** (default, `runid="MAGIC default run"`) is a character string. This can be used to briefly describe your run. This information is then for instance stored in the header of the graphic files.

6.7 Mantle and Inner Core Namelists

6.7.1 Mantle Namelist

This namelist defines mantle properties

- **conductance_ma** (default `conductance_ma=0.0`) is a real that defines the conductance (dimensionless) of the mantle.
- **nRotMa** (default `nRotMa=0`) is an integer that defines the rotation of the mantle:

nRotMa=-1	Mantle rotates with prescribed rate (see `omega_ma1` and `omega_ma2` below)
nRotMa=0	Fixed, non-rotating mantle
nRotMa=1	Mantle rotates according to torques

- **rho_ratio_ma** (default `rho_ratio_ma=1`) is a real which gives the density of the mantle in terms of that of the outer core.
- **omega_ma1** (default `omega_ma1=0.0`) is a real which defines a mantle rotation rate (used when `nRotMa=-1`).
- **omegaOsz_ma1** (default `omegaOsz_ma1=0.0`) is a real which prescribes the oscillation frequency of the mantle rotation rate. In this case, `omega_ma1` is the amplitude of the oscillation.
- **tShift_ma1** (default `tShift_ma1=0.0`) is a real which defines the time shift of the mantle rotation rate `omega_ma1`.
- **omega_ma2** (default `omega_ma2=0.0`) is a real which defines a second mantle rotation rate.
- **omegaOsz_ma2** (default `omegaOsz_ma2=0.0`) is a real which defines the oscillation frequency of the second mantle rotation rate `omega_ma2`.
- **tShift_ma2** (default `tShift_ma2=0.0`) is a real which defines the time shift for `omega_ma2`.

The resultant prescribed mantle rotation rate is computed as:

```
omega_ma = omega_ma1*cos(omegaOsz_ma1*(time+tShift_ma1)) + &
           omega_ma2*cos(omegaOsz_ma2*(time+tShift_ma2))
```

The following defines the parameters when one wants to excite inertial modes in the system artificially using a method similar to Rieutord et. al 2012 .

- **amp_RiMaSym** (default `amp_RiMaSym=0.0`) is a real which defines the amplitude of forcing on the outer boundary for an equatorially symmetric mode
- **omega_RiMaSym** (default `omega_RiMaSym=0.0`) is a real which defines the frequency of forcing on the outer boundary for an equatorially symmetric mode
- **m_RiMaSym** (default `m_RiMaSym=0.0`) is an integer which defines the wavenumber of the equatorially symmetric mode one wants to excite

The following variables define the same for an equatorially anti-symmetric mode:

- **amp_RiMaAsym** (default `amp_RiMaAsym=0.0`)
- **omega_RiMaAsym** (default `omega_RiMaAsym=0.0`)
- **m_RiMaAsym** (default `m_RiMaAsym=0.0`)

6.7.2 Inner Core Namelist

This namelist defines properties of the inner core

- **sigma_ratio** (default `sigma_ratio=0.0`) is a real that defines the conductivity of the inner core with respect to the value of the outer core. `sigma_ratio=0` thus corresponds to a non-conducting inner core.
- **nRotIc** (default `nRotIc=0`) is an integer that defines the rotation of the inner core. Behaves the same way as `nRotMa` (above).
- **rho_ratio_ic** (default `rho_ratio_ic=1.0`) is a real which defines the density of the inner core in terms of that of the outer core.
- **BIC** (default `BIC=0.0`) is a real which gives the imposed dipole field strength at the Inner Core Boundary. Having `BIC > 0` implies that the inner core acts as a dipole magnet - as implemented in the DTS experiment at Grenoble, France.
- **Variables prescribing rotation rate of inner core** The following variables are used to prescribe rotation rate of the inner core. They behave in the same way as the corresponding variables for the mantle. They are usd only when `nRotIC=0`.
  - **omega_ic1** (default `omega_ic1=0.0`)
  - **omegaOsz_ic1** (default `omegaOsz_ic1=0.0`)
  - **tShift_ic1** (default `tShift_ic1=0.0`)
  - **omega_ic2** (default `omega_ic2=0.0`)
  - **omegaOsz_ic2** (default `omegaOsz_ic2=0.0`)
  - **tShift_ic2** (default `tShift_ic2=0.0`)

As with the mantle, the resultant prescribed rotation rate for the inner core is computed as:

```
omega_ic = omega_ic1*cos(omegaOsz_ic1*(time+tShift_ic1)) + &
           omega_ic2*cos(omegaOsz_ic2*(time+tShift_ic2))
```

The following, as for the mantle namelist, is for artificially exciting inertial modes in the spherical shell, but for the inner boundary.

- **amp_RiIcSym** (default `amp_RiIcSym=0.0`) is a real which defines the amplitude of forcing on the inner boundary for an equatorially symmetric mode
- **omega_RiIcSym** (default `omega_RiIcSym=0.0`) is a real which defines the frequency of forcing on the inner boundary for an equatorially symmetric mode
- **m_RiIcSym** (default `m_RiIcSym=0.0`) is an integer which defines the wavenumber of the equatorially symmetric mode one wants to excite

The following variables define the same for an equatorially anti-symmetric mode:

- **amp_RiIcAsym** (default `amp_RiIcAsym=0.0`)
- **omega_RiIcAsym** (default `omega_RiIcAsym=0.0`)
- **m_RiIcAsym** (default `m_RiIcAsym=0.0`)

CHAPTER

SEVEN

INTERACTIVE COMMUNICATION WITH THE CODE USING `SIGNAL.TAG`

It is possible to interactively communicate with the MagIC code **during a run**, using a file which is systematically created when the simulation starts, called **signal.TAG**. By default, this file contains only the word `NOT` and does nothing to the simulation. Replacing `NOT` by one of the following allowed keywords will have some influence on the outputs or possibly force the code to terminate its execution:

- **END**: Changing the word `NOT` to `END` will cause the code to finish after the current time step and write all the outputs as if it was programmed to finish at that time from the start. This will thus normally produce the *checkpoint_end.TAG* file that will possibly allow you to continue this run later at your convenience.
- **GRA**: Changing the word `NOT` to `GRA` will cause the code to produce a graphic ouptut file *G_#.TAG*. The keyword will be automatically restored to `NOT` once the graphic file has been produced.
- **RST**: Changing the word `NOT` to `RST` will cause the code to produce a restart file *checkpoint_t#.TAG*. The keyword will then be restored to `NOT` once the restart file has been written.
- **SPE**: Changing the word `NOT` to `SPE` will cause the code to produce spectra *kin_spec_#.TAG* (and possibly *mag_spec_#.TAG* and *T_spec_#.TAG* <*secTSpecFile*>) depending if the run is magnetic or not, or if it solves a temperature/entropy equation). Once the spectra files have been written, the keyword will be automatically replaced by `NOT`.
- **POT**: Changing the word `NOT` to `POT` will cause the code to produce the potential files *V_lmr_#.TAG* (and possibly *B_lmr_#.TAG*, *T_lmr_#.TAG* <*secPotFiles*> and *Xi_lmr_#.TAG* <*secPotFiles*> depending if the run is magnetic or not, or if it solves a temperature/entropy and/or chemical composition equations). Once the potential files have been written, the keyword will be automatically replaced by `NOT`.

Note: Those keywords are **case-insensitive**.

Instead of editing the file with your favorite editor to specify the requested keyword, we recommand using instead the shell command `echo` to avoid some possible crash during the code execution when writing into the `signal.TAG` file. For instance, if you want a *graphic output file*, just use the following command (adapted to your current *TAG*):

```
$ echo GRA > signal.TAG
```

CHAPTER

EIGHT

OUTPUT FILES

While some information of a run is written into `STDOUT` to monitor its progress, most outputs are printed into dedicated files identified by the chosen *TAG* extension. These files can be parsed and analysed using the *python classes*. The following pages describe the content and the structure of the different type of output files:

1. Most of the information found in `STDOUT` is also written to the **log-file** called *log.TAG*. In addition, this file contains all input parameters, truncation, information on other output files, and some results like the time averaged energies (when *l_average=.true.*).
2. There are several ascii files that contain the **time-evolution of integrated quantities** (energies, heat fluxes, rotation rate, Reynolds numbers, etc.) that are systematically produced:
   - Kinetic energies: *e_kin.TAG*,
   - Magnetic energies: *e_mag_oc.TAG* and *e_mag_ic.TAG*,
   - Rotation rates: *rot.TAG*,
   - Informations about the dipolar component of the magnetic field: *dipole.TAG*,
   - Diagnostic parameters (Reynolds, Elsasser, etc.): *par.TAG*,
3. There are **additional conditional time series** that contain the time-evolution of other physical quantities that depend on the chosen *input parameters*:
   - Angular momentum balance: *AM.TAG*,
   - Heat transport: *heat.TAG*,
   - Helicity: *helicity.TAG*,
   - Power budget: *power.TAG* and *dtE.TAG*,
   - Square velocities: *u_square.TAG*,
   - Drift rates: *drift[V\|B][D\|Q].TAG* and *iner[P\|T].TAG*,
   - Torques: *SR[IC\|MA].TAG*,
   - Geostrophy: *geos.TAG*,
   - RMS calculations of the force balances: *dtVrms.TAG* and *dtBrms.TAG*,
   - Kinetic energies perpendicular and parallel to the rotation axis: *perpPar.TAG*.
4. **Time-averaged radial profiles**:
   - Kinetic energies: *eKinR.TAG*,
   - Magnetic energies: *eMagR.TAG*,
   - Diagnostic quantities: *parR.TAG*,

- Power budget: *powerR.TAG*,
- Average temperature, entropy and pressure: *heatR.TAG*,
- Heat fluxes: *fluxesR.TAG*,
- Temperature and horizontal velocities: *bLayersR.TAG*,
- Kinetic energies perpendicular and parallel to the rotation axis: *perpParR.TAG*.

5. **Radial profiles of the transport properties** of the reference state (those files will only be produced when the appropriate input option is chosen):
   - Temperature, density and gravity: *anel.TAG*,
   - Electrical conductivity: *varCond.TAG*,
   - Thermal conductivity: *varDiff.TAG*,
   - Kinematic viscosity: *varVisc.TAG*,
   - Mapping of the Chebyshev grid: *rNM.TAG*.
6. Kinetic energy, magnetic energy and temperature/entropy **spectra**:
   - Kinetic energy: *kin_spec_#.TAG*,
   - Magnetic energy: *kin_spec_#.TAG*,
   - Velocity square: *u2_spec_#.TAG*,
   - Temperature/entropy: *T_spec_#.TAG*,
   - Time-averaged kinetic energy: *kin_spec_ave.TAG*,
   - Time-averaged magnetic energy: *mag_spec_ave.TAG*,
   - Time-averaged temperature/entropy: *T_spec_ave.TAG*,
   - 2-D ([r,ell] and [r,m]) spectra: *2D_[mag|kin]_spec_#.TAG*.
   - Time-averaged 2-D ([r,ell] and [r,m]) spectra: *2D_[mag|kin]_spec_ave.TAG*.
7. Output snapshot that contains the 3-D components of the velocity field, the magnetic field and the temperature/entropy. Those files are named **graphic files** *G_#.TAG* (or *G_ave.TAG* for its time-averaged counterpart).
8. Time evolution of some chosen fields. Those files are named **movie files**: **_mov.TAG*.
9. Checkpoints outputs that will allow the code to restart. Those files are named **restart files**: *checkpoint_end.TAG*.
10. **Time-evolution of the poloidal and toroidal coefficients** at diffent depths:
    - Time evolution of the poloidal magnetic field at the CMB: *B_coeff_cmb.TAG*,
    - Time evolution of the potentials at several depths: *[V|T|B]_coeff_r#.TAG*
11. **Additional specific outputs**:
    - Torsional oscillations (see *here*),
    - Potential files: *V_lmr_#.TAG*, *B_lmr_#.TAG* and *T_lmr_#.TAG*,
    - Magnetic spectra for various radii: *rB[r|p]Spec.TAG*.

8.1 Log file: `log.TAG`

This is a text file contains information about the run, including many of the things which are printed to `STDOUT`. It has the following information in order of appearance:

- **Code version**: the version of the code
- **Parallelization**: information about number of MPI ranks being used, blocking information of OpenMP chunks and processor load balancing
- **Namelists**: displays values of all namelist variables. The ones input by the user should have the input values while the rest of them are set to their default values.
- **Mode** The mode of the run - self-consistent/kinematic dynamo, convection, couette flow etc. See the *control namelist* for more information about *mode*.
- **Grid parameters**: information about the grid sizes and truncation being used. More information about this in the *grid namelist*. If a new grid, different from that in the restart file is used, then a comparison is shown between old and new grid parameters and the user is informed that the data is being mapped from the old to the new grid.
- **Progress**: information about the progress of the run for every 10% of the run and the mean wall time for time step.
- **Writing of graphic, movie, restart and spectra files**: displays the time step and tells the user whenever a *G_#.TAG*, *checkpoint_#.TAG* or *spectra* file or a *movie frame* is written disk.
- **Energies**: gives kinetic and magnetic energies (total, poloidal, toroidal, total density) at the end of the run.
- **Time averages**: this part gives time averaged kinetic and magnetic energies (total, poloidal, toroidal, total density) and time averaged parameters (Rm, Elsass, Rol etc.). If *l_spec_avg=.true.*, this section also provides information about average spectra being written. If *l_average=.true.*, it is additionally mentioned that time averaged graphic files are written.
- **Wall times**: this is the last part of the log file and it provides information about the mean wall time for running different parts of the code. These values can be used to judge the speed and scaling capabilities of your computer.

Most of these informations can be parsed and stored into a python class using *MagicSetup*:

```
>>> # read log.N0m2
>>> stp = MagicSetup(nml='log.N0m2')
>>> print(stp.ek, stp.prmag) # print Ekman and magnetic Prandtl numbers
>>> print(stp.l_max) # print l_max
```

8.2 Default time-series outputs

8.2.1 `e_kin.TAG`

This file contains the kinetic energy of the outer core, defined by

$$
\begin{aligned}
E_k = \frac{1}{2}\int_V \tilde{\rho} u^2 \, \mathrm{d}V = E_{pol} + E_{tor} \\
= \frac{1}{2}\sum_{\ell,m} \ell(\ell+1) \int_{r_i}^{r_o} \frac{1}{\tilde{\rho}} \left[\frac{\ell(\ell+1)}{r^2} |W_{\ell m}|^2 + \left| \frac{\mathrm{d}W_{\ell m}}{\mathrm{d}r} \right|^2 \right] \mathrm{d}r \\
+ \frac{1}{2}\sum_{\ell,m} \ell(\ell+1) \int_{r_i}^{r_o} \frac{1}{\tilde{\rho}} |Z_{\ell m}|^2 \, \mathrm{d}r
\end{aligned}
\tag{8.1}
$$

The detailed calculations are done in the subroutine `get_e_kin`. This file contains the following informations:

No. of column	Contents
1	time
2	poloidal energy
3	toroidal energy
4	axisymmetric poloidal energy
5	axisymmetric toroidal energy
6	equatorial symmetric poloidal energy
7	equatorial symmetric toroidal energy
8	equatorial symmetric and axisymmetric poloidal energy
9	equatorial symmetric and axisymmetric toroidal energy

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the e_kin.TAG files of the current directory
>>> ts = MagicTs(field='e_kin', all=True)
>>> # To only read e_kin.N0m2
>>> ts = MagicTs(field='e_kin', tag='N0m2')
```

8.2.2 e_mag_oc.TAG

This file contains the magnetic energy of the outer core, defined by

$$\begin{aligned} E_m = \frac{1}{2}\int_V B^2 \,\mathrm{d}V = E_{pol} + E_{tor} \\ = \frac{1}{2}\sum_{\ell,m}\ell(\ell+1)\int_{r_i}^{r_o}\left[\frac{\ell(\ell+1)}{r^2}|b_{\ell m}|^2 + \left|\frac{\mathrm{d}b_{\ell m}}{\mathrm{d}r}\right|^2\right]\mathrm{d}r \\ + \frac{1}{2}\sum_{\ell,m}\ell(\ell+1)\int_{r_i}^{r_o}|j_{\ell m}|^2\,\mathrm{d}r \end{aligned} \tag{8.2}$$

The detailed calculations are done in the subroutine `get_e_mag`. This file contains the following informations:

No. of column	Contents
1	time
2	outer core poloidal energy
3	outer core toroidal energy
4	outer core axisymmetric poloidal energy
5	outer core axisymmetric toroidal energy
6	outside potential field energy
7	outside axisymmetric potential field energy
8	equatorial symmetric poloidal energy
9	equatorial symmetric toroidal energy
10	equatorial symmetric and axisymmetric poloidal energy
11	equatorial symmetric and axisymmetric toroidal energy
12	outside potential field energy
13	outside potential field axisymmetric energy

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the e_mag_oc.TAG files of the current directory
>>> ts = MagicTs(field='e_mag_oc', all=True)
>>> # To only read e_mag_oc.N0m2
>>> ts = MagicTs(field='e_mag_oc', tag='N0m2')
```

8.2.3 e_mag_ic.TAG

This file contains the magnetic energy of the inner core. The detailed calculations are done in the subroutine `get_e_mag`. This file contains the following informations:

No. of column	Contents
1	time
2	inner core poloidal energy
3	inner core toroidal energy
4	inner core axisymmetric poloidal energy
5	inner core axisymmetric toroidal energy

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the e_mag_ic.TAG files of the current directory
>>> ts = MagicTs(field='e_mag_ic', all=True)
>>> # To only read e_mag_ic.N0m2
>>> ts = MagicTs(field='e_mag_ic', tag='N0m2')
```

8.2.4 rot.TAG

This files contains the rotation of the inner core and the mantle. Output concerning the rotation of inner core and mantle. This file is written by the subroutine `write_rot`.

No. of column	Contents
1	time
2	Inner core rotation rate
3	Lorentz torque on inner core
4	viscous torque on inner core
5	mantle rotation rate
6	Lorentz torque on mantle
7	viscous torque on mantle

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the rot.TAG files of the current directory
>>> ts = MagicTs(field='rot', iplot=False, all=True)
```

8.2.5 dipole.TAG

This file contains several informations about the magnetic dipole. This file is written by the subroutine `get_e_mag`.

No. of column	Contents
1	time
2	tilt angle (colatitude in degrees) of the dipole
3	longitude (in degress) of dipole-pole
4	relative energy of the axisymmetric dipole
5	relative energy of the axisymmetric dipole at the CMB
6	energy of the axisymmetric dipole at the CMB normalized with the total energy up to spherical harmonic degree and order 11
7	relative energy of the total (axisymmetric and equatorial) dipole
8	relative energy of the total (axisymmetric and equatorial) dipole in the outer core
9	relative energy of the total dipole (axisymmetric and equatorial) at the CMB
10	energy of the total (axisymmetric and equatorial) dipole at the CMB
11	energy of the axisymmetric dipole at the CMB
12	energy of the dipole
13	energy of the axisymmetric dipole
14	magnetic energy at the CMB
15	magnetic energy up to spherical harmonic degree and order 11
16	ratio between equatorial dipole energy and equatorial poloidal energy
17	difference between energy at the CMB and equatorial symmetric energy at the CMB, normalized by energy at the CMB
18	difference between energy at the CMB and axisymmetric energy at the CMB, normalized by energy at the CMB
19	difference between total energy and equatorial symmetric part of the total energy, normalized by the total energy
20	difference between total energy and axisymmetric part of the total energy, normalized by the total energy

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the dipole.TAG files of the current directory
>>> ts = MagicTs(field='dipole', all=True)
```

8.2.6 par.TAG

This files contains the outputs of several parameters that describe flow and magnetic fields (Reynolds number, Elsasser number, flow lengthscales, etc.). This file is written by the subroutine `output`.

No. of column	Contents
1	time
2	(magnetic) Reynolds number
3	Elsasser number
4	Local Rossby number Rol
5	Realtive geostrophic kinetic energy
6	Total dipolarity
7	CMB dipolarity
8	Axial flow length scale dlV
9	Flow length scale dmV
10	Flow length scale dpV
11	Flow length scale dzV
12	Dissipation length scale lvDiss
13	Dissipation length scale lbDiss
14	Magnetic length scale dlB
15	Magnetic length scale dmB
16	Elsasser number at CMB
17	Local Rol based on non-ax. flow
18	Convective flow length scale dlVc
19	Peak of the poloidal kinetic energy
20	CMB zonal flow at the equator

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the par.TAG files of the current directory
>>> ts = MagicTs(field='par', all=True)
```

8.3 Additional optional time-series outputs

8.3.1 heat.TAG

This files contains informations about the heat transfer (Nusselt number, entropy and temperature at both boundaries). This file is written by the subroutine `outHeat`.

No. of column	Contents
1	time
2	Nusselt number at the inner boundary
3	Nusselt number at the outer boundary
4	Nusselt number based on ΔT ratio
5	Temperature at the inner boundary
6	Temperature at the outer boundary
7	Entropy at the inner boundary
8	Entropy at the outer boundary
9	Heat flux at the inner boundary
10	Heat flux at the outer boundary
11	Pressure perturbation at the outer boundary
12	volume integrated mass perturbation
13	Sherwood number at the inner boundary
14	Sherwood number at the outer boundary
15	Sherwood number based on $\Delta\xi$ ratio
16	Chemical composition at the inner boundary
17	Chemical composition at the outer boundary

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the heat.TAG files of the current directory
>>> ts = MagicTs(field='heat', all=True)
```

8.3.2 AM.TAG

Note: This file is **only** written when *l_AM=.true.*

This file contains the time series of the angular momentum of the inner core, the outer core and the mantle. This file is written by the subroutine `write_rot`.

No. of column	Contents
1	time
2	angular momentum of the outer core
3	angular momentum of the inner core
4	angular momentum of the mantle
5	total angular momentum
6	relative in angular momentum, per time step
7	total kinetic angular momentum
8	relative change in kinetic energy, per time step
9	kinetic angular momentum of the inner core
10	kinetic angular momentum of the outer core
11	kinetic angular momentum of the mantle

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the AM.TAG files of the current directory
>>> ts = MagicTs(field='AM', all=True)
```

8.3.3 power.TAG

Note: This file is **only** written when *l_power=.true.*

This file contains the power budget diagnostic. This file is computed by the subroutine `get_power`.

No. of column	Contents
1	time
2	Buoyancy power: $Ra\, g(r)\, \langle u_r T' \rangle_s$
3	Chemical power: $Ra_\xi\, g(r)\, \langle u_r \xi' \rangle_s$
4	Viscous power at the inner boundary (ICB)
5	Viscous power at the outer boundary (CMB)
6	Viscous dissipation: $\langle (\nabla \times u)^2 \rangle_s$
7	Ohmic dissipation: $\langle (\nabla \times B)^2 \rangle_s$
8	Total power at the CMB (viscous + Lorentz)
9	Total power at the ICB (viscous + Lorentz)
10	Total power
11	Time variation of total power

This file can be read using *MagicTs* with the following options:

```
>>> # To stack the files that match the pattern  ``power.N0m2*``
>>> ts = MagicTs(field='power', tags='N0m2*')
```

8.3.4 dtE.TAG

Note: This file is **only** written when *l_power=.true.*

This file contains the time-derivatives of the total energy. It allows to accurately monitor how the total energy varies with time. This file is generated by the subroutine `output`.

No. of column	Contents
1	time
2	time-derivative of the total energy $\partial E/\partial t$
3	integrated time variation of the total energy
4	relative time variation of the total energy

8.3.5 earth_like.TAG

This contains informations about the Earth-likeness of the CMB radial magnetic field. This file is written by the subroutine `get_e_mag`.

Note: This file is **only** calculated when *l_earth_like=.true.*.

No. of column	Contents
1	time
2	Ratio of axial dipole to non-dipole component at the CMB
3	Equatorial symmetry of the CMB field (odd/even ratio)
4	Zonality: zonal to non-zonal ratio of the CMB field
5	Magnetic flux concentration at the CMB

The details of the calculations are given in (Christensen et al., 2010).

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the earth_like.TAG files of the current directory
>>> ts = MagicTs(field='earth_like', all=True)
```

8.3.6 geos.TAG

This file contains informations about the geostrophy of the flow. This file is written by the subroutine `getEgeos`.

Note: This file is **only** calculated when *l_par=.true.*.

No. of column	Contents
1	time
2	Relative geostrophic kinetic energy
3	Relative kinetic energy in the northern part of the TC
4	Relative kinetic energy in the southern part of the TC
5	Kinetic energy (calculated on the cylindrical grid)
6	North/South correlation of Vz, outside the TC
7	North/South correlation of vorticity outside the TC
8	North/South correlation of helicity outside the TC
9	Geostrophy of axisymmetic flow
10	Geostrophy of zonal flow
11	Geostrophy of meridional flow
12	Geostrophy of non-axisymmetric flow

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the geos.TAG files of the current directory
>>> ts = MagicTs(field='geos', all=True)
```

8.3.7 helicity.TAG

This files contains informations about the kinetic helicity in both the Northern and the Southern hemispheres. This file is written by the subroutine `outHelicity`.

Note: This file is **only** calculated when *l_hel=.true.*.

No. of column	Contents
1	time
2	Helicity (northern hemisphere)
3	Helicity (southern hemisphere)
4	RMS helicity (northern hemisphere)
5	RMS helicity (southern hemisphere)
6	Helicity (northern hemisphere, only non-axisym. flow)
6	Helicity (southern hemisphere, only non-axisym. flow)
8	RMS helicity (northern hemisphere, only non-axisym. flow)
9	RMS helicity (southern hemisphere, only non-axisym. flow)

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the helicity.TAG files of the current directory
>>> ts = MagicTs(field='helicity', all=True)
```

8.3.8 u_square.TAG

Note: This file is **only** written in anelastic models, i.e. either when *strat/=0* or when *interior_model/="None"*

This file contains the square velocity of the outer core. It is actually very similar to the *e_kin.TAG* file, except that the density background $\tilde{\rho}$ is removed:

$$\begin{aligned}\mathcal{U} = \frac{1}{2}\int_V u^2 \, \mathrm{d}V = \mathcal{U}_{pol} + \mathcal{U}_{tor} \\ = \frac{1}{2}\sum_{\ell,m}\ell(\ell+1)\int_{r_i}^{r_o}\frac{1}{\tilde{\rho}^2}\left[\frac{\ell(\ell+1)}{r^2}|W_{\ell m}|^2 + \left|\frac{\mathrm{d}W_{\ell m}}{\mathrm{d}r}\right|^2\right]\mathrm{d}r \\ + \frac{1}{2}\sum_{\ell,m}\ell(\ell+1)\int_{r_i}^{r_o}\frac{1}{\tilde{\rho}^2}|Z_{\ell m}|^2\,\mathrm{d}r\end{aligned}$$

The detailed calculations are done in the subroutine `get_u_square`. This file contains the following informations:

No. of columns	Contents
1	time
2	poloidal part $\mathcal{U}_{pol}$
3	toroidal part $\mathcal{U}_{pol}$
4	axisymmetric contribution to the poloidal part
5	axisymmetric contribution to the toroidal part
6	Rossby number: $Ro = E\sqrt{\frac{2\mathcal{U}}{V}}$
7	Magnetic Reynolds number: $Rm = Pm\sqrt{\frac{2\mathcal{U}}{V}}$
8	local Rossby number: $Ro_l = Ro\frac{d}{l}$
9	average flow length scale: l
10	local Rossby number based on the non-axisymmetric components of the flow
11	average flow length scale based on the non-axisymmetric components of the flow

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the u_square.TAG files of the current directory
>>> ts = MagicTs(field='u_square', all=True)
```

8.3.9 drift[V|B][D|Q].TAG

Note: These files are **only** written when *l_drift=.true.*

These files store spherical harmonic coefficients of the toroidal (poloidal) potential of the flow (magnetic) field, only for $\ell = m$ or $\ell = m + 1$ depending on the symmetry - `D` for **D** ipolar and `Q` for **Q** uadrupolar. The coefficients are stored at different three different radial levels - `n_r1, nr_2, n_r3` for the velocity and two different radial levels - `n_r1` and `n_r2` - for the magnetic field.

The symmetries can be summarized below:

Field	Dipolar	Quadrupolar
Velocity	$\ell = m$	$\ell = m + 1$
Magnetic	$\ell = m + 1$	$\ell = m$

$\ell+m =$ even for toroidal potential refers to an equatorially antisymmetric field (*Dipolar*), while the same for a poloidal potential is associated with an equatorially symmetric field (*Quadrupolar*). The sense is opposite when $\ell + m =$ odd. This is the reason for the choice of selecting these specific coefficients.

The columns of the files look like follows:

For the flow field:

- n_r1 = (1/3) * *n_r_max-1*
- n_r2 = (2/3) * *n_r_max-1*
- n_r3 = *n_r_max-1*

Column no.	DriftVD.TAG	DriftVQ.TAG
1	Time	Time
2	z (*minc*, minc) at n_r1	z (*minc+1*, minc) at n_r1
3	z (2*minc, 2*minc) at n_r1	z (2*minc+1, 2*minc) at n_r1
4	z (3*minc, 3*minc) at n_r1	z (3*minc+1, 3*minc) at n_r1
5	z (4*minc, 4*minc) at n_r1	z (4*minc+1, 4*minc) at n_r1
6	z (minc, minc) at n_r2	z (minc+1, minc) at n_r2
7	z (2*minc, 2*minc) at n_r2	z (2*minc+1, 2*minc) at n_r2
8	z (3*minc, 3*minc) at n_r2	z (3*minc+1, 3*minc) at n_r2
9	z (4*minc, 4*minc) at n_r2	z (4*minc+1, 4*minc) at n_r2
10	z (minc, minc) at n_r3	z (minc+1, minc) at n_r3
11	z (2*minc, 2*minc) at n_r3	z (2*minc+1, 2*minc) at n_r3
12	z (3*minc, 3*minc) at n_r3	z (3*minc+1, 3*minc) at n_r3
13	z (4*minc, 4*minc) at n_r3	z (4*minc+1, 4*minc) at n_r3

For the magnetic field:

- n_r1 = `n_r_ICB`

- n_r2 = `n_r_CMB`

Column no.	DriftBD.TAG	DriftBQ.TAG
1	Time	Time
2	b (*minc+1*, minc) at n_r1	b (*minc*, minc) at n_r1
3	b (2*minc+1, 2*minc) at n_r1	b (2*minc, 2*minc) at n_r1
4	b (3*minc+1, 3*minc) at n_r1	b (3*minc, 3*minc) at n_r1
5	b (4*minc+1, 4*minc) at n_r1	b (4*minc, 4*minc) at n_r1
6	b (minc+1, minc) at n_r2	b (minc, minc) at n_r2
7	b (2*minc+1, 2*minc) at n_r2	b (2*minc, 2*minc) at n_r2
8	b (3*minc+1, 3*minc) at n_r2	b (3*minc, 3*minc) at n_r2
9	b (4*minc+1, 4*minc) at n_r2	b (4*minc, 4*minc) at n_r2

Analysis of these files can give you information about the drift frequency of the solution and it's symmetry.

8.3.10 `iner[P|T].TAG`

Note: These files are **only** written when *l_iner=.true.* and *minc = 1*.

These files contain time series of spherical harmonic coefficients upto degree, $\ell = 6$ at a radius $r = (r_{cmb} - r_{icb})/2$. The `inerP.TAG` contains coefficients of the poloidal potential while the `inerT.TAG` contains coefficients of the toroidal potential.These files are written by the subroutine `write_rot`. The oscillations of these coefficients can be analysed to look for inertial modes. The columns of the `inerP.TAG` look like follows:

No. of column	Coefficient
1	$w(\ell = 1, m = 1)$
2	$w(\ell = 2, m = 1)$
3	$w(\ell = 2, m = 2)$
4	$w(\ell = 3, m = 1)$
...	
20	$w(\ell = 6, m = 5)$
21	$w(\ell = 6, m = 6)$

where $w(\ell, m)$ is the poloidal potential with degree ℓ and order m.

The columns of the `inerT.TAG` follow the following structure:

No. of column	Coefficient
1	$z(\ell = 1, m = 1)$
2	$z(\ell = 2, m = 1)$
3	$z(\ell = 2, m = 2)$
4	$z(\ell = 3, m = 1)$
...	
20	$z(\ell = 6, m = 5)$
21	$z(\ell = 6, m = 6)$

where $z(\ell, m)$ is the toroidal potential with degree ℓ and order m.

8.3.11 SR[IC|MA].TAG

Note: These files are **only** written for *nRotIc=-1* (for `SRIC.TAG`) or *nRotMa=-1* (for `SRMA.TAG`). In other words, these outputs are produced **only** when one of the boundaries is made to rotate at a prescribed rotation rate.

These files contain information about power due to torque from viscous and Lorentz forces at the inner core boundary (`SRIC.TAG`) or core mantle boundary (`SRMA.TAG`).The columns look like follows:

No. of column	Contents
1	Time
2	$\Omega_{IC}\|\Omega_{MA}$
3	Total power = Lorentz + Viscous
4	Viscous power
5	Lorentz force power

8.3.12 dtVrms.TAG

Note: This file is **only** written when *l_RMS=.true.*

This files contains the RMS force balance of the Navier Stokes equation. This file is written by the subroutine `dtVrms`.

No. of column	Contents
1	Time
2	Total inertia: dU/dt and advection
3	Coriolis force
4	Lorentz force
5	Advection term
6	Diffusion term
7	Thermal buoyancy term
8	Chemical buoyancy term
9	Pressure gradient term
10	Sum of force terms: geostrophic balance
11	Sum of force terms: pressure, Coriolis and Lorentz
12	Sum of force terms: pressure, buoyancy and Coriolis
13	Sum of force terms: pressure, buoyancy, Coriolis and Lorentz
14	Sum of force terms: Lorentz/Coriolis
15	Sum of force terms: Pressure/Lorentz
16	Sum of force terms: Coriolis/Inertia/Archimedean

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the dtVrms.TAG files of the current directory
>>> ts = MagicTs(field='dtVrms', all=True)
```

8.3.13 dtBrms.TAG

Note: This file is **only** written when *l_RMS=.true.*

This files contains the RMS terms that enter the induction equation. This file is written by the subroutine `dtBrms`.

No. of column	Contents
1	time
2	Changes in magnetic field (poloidal)
3	Changes in magnetic field (toroidal)
4	Poloidal induction term
5	Toroidal induction term
8	Poloidal diffusion term
9	Toroidal diffusion term
10	Omega effect / toroidal induction term
11	Omega effect
12	Production of the dipole field
13	Production of the axisymmetric dipole field

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the dtBrms.TAG files of the current directory
>>> ts = MagicTs(field='dtBrms', all=True)
```

8.3.14 perpPar.TAG

Note: This file is **only** written when *l_perpPar=.true.*

This file contains several time series that decompose the kinetic energy into components parallel and perpendicular to the rotation axis. This file is calculated by the subroutine `outPerpPar`.

No. of column	Contents
1	time
2	Total kinetic energy perpendicular to the rotation axis: $\frac{1}{2}\langle u_s^2 + u_\phi^2 \rangle_V$
3	Total kinetic energy parallel to the rotation axis: $\frac{1}{2}\langle u_z^2 \rangle_V$
4	Axisymmetric kinetic energy perpendicular to the rotation axis
5	Axisymmetric kinetic energy parallel to the rotation axis

This file can be read using *MagicTs* with the following options:

```
>>> # To stack all the perpPar.TAG files of the current directory
>>> ts = MagicTs(field='perpPar', all=True)
```

8.4 Time-averaged radial profiles

8.4.1 eKinR.TAG

This file contains the time and horizontally averaged outer core kinetic energy along the radius. This file is calculated by the subroutine `get_e_kin`.

No. of column	Contents
1	radial level
2	time and horizontally averaged poloidal energy
3	time and horizontally averaged axisymmetric poloidal energy
4	time and horizontally averaged toroidal energy
5	time and horizontally averaged axisymmetric toroidal energy
6	time and horizontally averaged poloidal energy, normalized by surface area at this radial level
7	time and horizontally averaged axisymmetric poloidal energy, normalized by surface area at this radial level
8	time and horizontally averaged toroidal energy, normalized by surface area at this radial level
9	time and horizontally averaged axisymmetric toroidal energy, normalized by surface area at this radial level

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='eKinR')
```

8.4.2 eMagR.TAG

This file contains the time and horizontally averaged outer core magnetic energy along the radius. This file is calculated by the subroutine `get_e_mag`.

No. of column	Contents
1	radial level
2	time and horizontally averaged poloidal energy
3	time and horizontally averaged axisymmetric poloidal energy
4	time and horizontally averaged toroidal energy
5	time and horizontally averaged axisymmetric toroidal energy
6	time and horizontally averaged poloidal energy, normalized by surface area at this radial level
7	time and horizontally averaged axisymmetric poloidal energy, normalized by surface area at this radial level
8	time and horizontally averaged toroidal energy, normalized by surface area at this radial level
9	time and horizontally averaged axisymmetric toroidal energy, normalized by surface area at this radial level
10	ratio between time-averaged dipole energy and time-averaged total energy

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='eMagR')
```

8.4.3 parR.TAG

This file contains several time and horizontally averaged flow properties (magnetic Reynolds number, Rossby number, etc.). This file is calculated by the subroutine `outPar`.

No. of column	Contents
1	radial level
2	Magnetic Reynolds number
3	Local Rossby number (based on the mass-weighted velocity)
4	Local Rossby number (based on the RMS velocity)
5	Local flow length-scale
6	Local flow length-scale based on the non-axisymmetric flow components
7	Local flow length-scale based on the peak of the poloidal kinetic energy
8	Standard deviation of magnetic Reynolds number
9	Standard deviation of local Rossby number (mass-weighted)
10	Standard deviation of local Rossby number (RMS velocity)
11	Standard deviation of convective lengthscale
12	Standard deviation of convective lengthscale (non-axi)
13	Standard deviation of convective lengthscale (pol. peak)

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='parR')
```

8.4.4 heatR.TAG

Note: This file is **only** written when an equation for the heat transport (temperature or entropy) is solved.

This file contains several time and horizontally averaged thermodynamic properties (temperature, pressure, entropy, etc.) and their variance. This file is calculated by the subroutine `outHeat`.

No. of column	Contents
1	Radial level
2	Entropy (spherically-symetric contribution)
3	Temperature (spherically-symetric contribution)
4	Pressure (spherically-symetric contribution)
5	Density (spherically-symetric contribution)
6	Chemical composition (spherically-symetric contribution)
7	Standard deviation of entropy
8	Standard deviation of temperature
9	Standard deviation of pressure
10	Standard deviation of density
11	Standard deviation of chemical composition

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='heatR')
```

8.4.5 powerR.TAG

Note: This file is **only** written when *l_power=.true.*

This file contains the time and horizontally averaged power input (Buoyancy power) and outputs (viscous and Ohmic heating). This file is calculated by the subroutine `get_power`.

No. of column	Contents
1	radial level
2	Buoyancy power: $Ra\, g(r)\, \langle u_r T' \rangle_s$
3	Chemical power: $Ra_\xi\, g(r)\, \langle u_r \xi' \rangle_s$
4	Viscous dissipation: $\langle (\sigma)^2 \rangle_s$
5	Ohmic dissipation: $\langle (\nabla \times B)^2 \rangle_s$
6	Standard deviation of buoyancy power
7	Standard deviation of chemical power
8	Standard deviation of viscous dissipation
9	Standard deviation of ohmic dissipation

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='powerR')
```

8.4.6 fluxesR.TAG

Note: This file is **only** written when *l_fluxProfs=.true.*

This file contains the time and horizontally averaged heat flux carried out by several physical processes: conductive flux, convective flux, kinetic flux, viscous flux, Poynting flux and resistive flux. This file is calculated by the subroutine `outPar`.

No. of column	Contents
1	radial level
2	**conductive flux:** $$\mathcal{F}_{cond} = -\frac{1}{Pr}\kappa\tilde{\rho}\tilde{T}\frac{\partial\langle s\rangle_s}{\partial r}$$
3	**convective flux:** $$\mathcal{F}_{conv} = \tilde{\rho}\tilde{T}\langle s\, u_r\rangle_s + \frac{Pr\, Di}{E\, Ra}\langle p\, u_r\rangle_s$$
4	**kinetic flux:** $$\mathcal{F}_{kin} = \frac{1}{2}\frac{Pr\, Di}{Ra}\langle u_r(\tilde{\rho}u^2)\rangle_s$$
5	**viscous flux:** $$\mathcal{F}_{visc} = -\frac{Pr\, Di}{Ra}\langle \boldsymbol{u}\cdot S\rangle_s$$
6	**Poynting flux:** $$\mathcal{F}_{poyn} = -\frac{Pr\, Di}{Ra\, E\, Pm}\langle(\boldsymbol{u}\times\boldsymbol{B})\times\boldsymbol{B}\rangle_s$$
7	**resistive flux:** $$\mathcal{F}_{poyn} = \frac{Pr\, Di}{Ra\, E\, Pm^2}\langle(\boldsymbol{\nabla}\times\boldsymbol{B})\times\boldsymbol{B}\rangle_s$$
8	Standard deviation of conductive flux
9	Standard deviation of convective flux
10	Standard deviation of kinetic flux
11	Standard deviation of viscous flux
12	Standard deviation of Poynting flux
13	Standard deviation of resistive flux

This file can be read using `MagicRadial` with the following options:

```
>>> rad = MagicRadial(field='fluxesR')
```

8.4.7 bLayersR.TAG

Note: This file is **only** written when *l_viscBcCalc=.true.*

This file contains several time and horizontally averaged profiles that can be further used to determine thermal and viscous boundary layers: entropy (or temperature), entropy variance, horizontal velocity, radial derivative of the horizontal velocity, thermal dissipation rate. This file is calculated by the subroutine `outPar`.

No. of column	Contents
1	radial level
2	entropy: $\langle s \rangle_s$
3	**horizontal velocity:** $$u_h = \left\langle \sqrt{u_\theta^2 + u_\phi^2} \right\rangle_s$$
4	**radial derivative of the horizontal velocity:** $$\partial u_h / \partial r$$
5	**thermal dissipation rate:** $$\epsilon_T = \langle (\nabla T)^2 \rangle_s$$
6	Standard deviation of entropy
7	Standard deviation of horizontal velocity u_h
8	Standard deviation of the radial derivative of u_h
9	Standard deviation of the thermal dissipation rate

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='bLayersR')
```

Additional analyses of the boundary layers can then be carried out using *BLayers*:

```
>>> bl = BLayers(iplot=True)
```

8.4.8 perpParR.TAG

Note: This file is **only** written when *l_perpPar=.true.*

This file contains several time and horizontally averaged profiles that decompose the kinetic energy into components parallel and perpendicular to the rotation axis. This file is calculated by the subroutine `outPerpPar`.

No. of column	Contents
1	radial level
2	**Total kinetic energy perpendicular to the rotation axis:** $$\frac{1}{2}\langle u_s^2 + u_\phi^2 \rangle_s$$
3	**Total kinetic energy parallel to the rotation axis:** $$\frac{1}{2}\langle u_z^2 \rangle_s$$
4	Axisymmetric kinetic energy perpendicular to the rotation axis
5	Axisymmetric kinetic energy parallel to the rotation axis
6	Standard deviation of energy perpendicular to the rotation axis
7	Standard deviation of energy parallel to the rotation axis
8	Standard deviation of axisymmetric energy perpendicular to the rotation axis
9	Standard deviation of axisymmetric energy parallel to the rotation axis

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='perpParR')
```

8.5 Transport properties of the reference state

These files define the radial transport properties of the reference state. These arrays are calculated in the subroutines `radial` and `transportProperties`. The output files are written in the subroutine `preCalc`.

8.5.1 anel.TAG

Note: This output is only calculated when an anelastic model is run, that is when `l_anel=.true.` or `l_anelastic_liquid=.true.`.

This file contains the radial profiles of the reference state (density, temperature, gravity, etc.).

No. of column	Contents
1	radial level: r
2	temperature: $\tilde{T}(r)$
3	density: $\tilde{\rho}(r)$
4	radial derivative of the log of the density: $\beta = \mathrm{d}\ln\tilde{\rho}/\mathrm{d}r$
5	radial derivative of β: $\mathrm{d}\beta/\mathrm{d}r$
6	gravity: $g(r)$
7	entropy gradient: $\mathrm{d}s_0/\mathrm{d}r$
8	thermal diffusion operator: $\nabla\cdot(K(r)\tilde{T}(r)\nabla s_0)$
9	inverse of the Gruneisen parameter :math` 1/Gamma`: $(\partial\ln\tilde{\rho}/\partial\ln\tilde{T})_S$
10	radial derivative of the log of temperature: $\beta = \mathrm{d}\ln\tilde{T}/\mathrm{d}r$

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='anel')
>>> # print radius and density
>>> print(rad.radius, rad.rho0)
```

8.5.2 varCond.TAG

Note: This output is only calculated when the electrical conductivity varies with radius, i.e. when *nVarCond /= 0*

This file contains the radial profiles of the electrical conductivity, the electrical diffusivity and its radial derivative.

No. of column	Contents
1	radial level: r
2	electrical conductivity: $\sigma(r)$
3	electrical diffusivity: $\lambda(r) = 1/\sigma(r)$
4	radial derivative of the electrical diffusivity: $\mathrm{d}\ln\lambda/\mathrm{d}r$

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='varCond')
>>> print(rad.conduc) # Electrical conductivity
```

8.5.3 varDiff.TAG

Note: This output is only calculated when the thermal diffusivity varies with radius, i.e. when *nVarDiff /= 0*

This file contains the radial profiles of the thermal conductivity, the thermal diffusivity and its radial derivative.

No. of column	Contents
1	radial level: r
2	thermal conductivity: $K(r)$
3	thermal diffusivity: $\kappa(r) = K(r)/\tilde{\rho}(r)$
4	radial derivative of the electrical diffusivity: $\mathrm{d}\ln\kappa/\mathrm{d}r$
5	Prandtl number: $Pr(r) = \nu(r)/\kappa(r)$

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='varDiff')
>>> print(rad.kappa) # Thermal diffusivity
```

8.5.4 varVisc.TAG

Note: This output is only calculated when the kinematic viscosity varies with radius, i.e. when *nVarVisc /= 0*

This file contains the radial profiles of the dynamic viscosity, the kinematic viscosity and its radial derivative.

No. of column	Contents
1	radial level: r
2	dynamic viscosity: $\mu(r)$
3	kinetmatic viscosity: $\nu(r) = \mu(r)/\tilde{\rho}(r)$
4	radial derivative of the kinematic viscosity: $\mathrm{d}\ln\nu/\mathrm{d}r$
5	Prandtl number: $Pr(r) = \nu(r)/\kappa(r)$
6	magnetic Prandtl number $Pm(r) = \nu(r)/\lambda(r)$

This file can be read using *MagicRadial* with the following options:

```
>>> rad = MagicRadial(field='varVisc')
>>> # print kinematic viscosity and Ekman
>>> print(rad.kinVisc, rad.ekman)
```

8.6 Nonlinear mapping of the Chebyshev grid

8.6.1 rNM.TAG

Note: This file is only written when *l_newmap=.true.*.

This file contains the profile of the radial mapping and its derivatives:

No. of column	Contents
1	Grid point index
2	Radius of a grid point
3	First derivative of the mapping at a grid point
4	Second derivative of the mapping at a grid point
5	Third derivative of the mapping at a grid point

8.7 Spectra

8.7.1 kin_spec_#.TAG

This file contains the kinetic energy spectra. This file is written by the subroutine `spectrum`.

No. of column	Contents
1	degree / order
2	Poloidal kinetic energy versus degree
3	Poloidal kinetic energy versus order
4	Toroidal kinetic energy versus degree
5	Toroidal kinetic energy versus order

This file can be read using *MagicSpectrum* with the following options:

```
>>> sp = MagicSpectrum(field='ekin')
```

8.7.2 mag_spec_#.TAG

This file contains the magnetic energy spectra. This file is written by the subroutine `spectrum`.

No. of column	Contents
1	degree / order
2	Poloidal magnetic energy in the outer core versus degree
3	Poloidal magnetic energy in the outer core versus order
4	Toroidal magnetic energy in the outer core versus degree
5	Toroidal magnetic energy in the outer core versus order
6	Poloidal magnetic energy in the inner core versus degree
7	Poloidal magnetic energy in the inner core versus order
8	Toroidal magnetic energy in the inner core versus degree
9	Toroidal magnetic energy in the inner core versus order
10	Poloidal magnetic energy at the CMB versus degree
11	Poloidal magnetic energy at the CMB versus order
12	Poloidal magnetic energy at the CMB

This file can be read using *MagicSpectrum* with the following options:

```
>>> sp = MagicSpectrum(field='emag')
```

8.7.3 u2_spec_#.TAG

Note: This file is **only** written in anelastic models, i.e. either when *strat/=0* or when *interior_model/="None"*

This file contains the spectra of the square velocity. This file is written by the subroutine `spectrum`.

No. of column	Contents
1	degree / order
2	Poloidal contribution per degree in the outer core
3	Poloidal contribution per order in the outer core
4	Toroidal contribution per degree in the outer core
5	Toroidal contribution per order in the outer core

This file can be read using *MagicSpectrum* with the following options:

```
>>> # To read the file ``u2_spec_1.test``:
>>> sp = MagicSpectrum(field='u2', ispec=1, tag='test')
```

8.7.4 T_spec_#.TAG

This file contains the temperature/entropy spectra. It is written by the subroutine `spectrum_temp`.

No. of column	Contents
1	degree / order
2	RMS temperature/entropy versus degree
3	RMS temperature/entropy versus order
4	RMS temperature/entropy at the ICB versus degree
5	RMS temperature/entropy at the ICB versus order
6	RMS radial derivative of temperature/entropy at the ICB versus degree
7	RMS radial derivative of temperature/entropy at the ICB versus order

8.7.5 2D spectra 2D_[kin|mag]_spec_#.TAG and 2D_[kin|mag]_spec_ave.TAG

Note: Those files are **only** written when *l_2D_spectra=.true.*. The time-averaged files also require that *l_spec_avg=.true.*.

Those files contain 2-D spectra in the (r, ℓ) and in the (r, m) planes. In other words, the poloidal and toroidal energies versus degree ℓ or versus order m are computed for all radii. There are two kinds of those files that correspond to the aforementioned spectra, namely **2D_kin_spec_#.TAG**, **2D_mag_spec_#.TAG**. In case time-averages are requested, **2D_kin_spec_ave.TAG** and **2D_mag_spec_ave.TAG** will also be stored. The calculations are done in the subroutine `spectrum`. The structure of the output files are same for these three outputs. They are stored as fortran unformatted files.

Unformatted files are not directly human readable, and are used to store binary data and move it around without changing the internal representation. In fortran, the open, read and write operations for these files are performed as follows:

```
open(unit=4, file='test', form='unformatted')
read(unit=4) readVar
write(unit=n_out, iostat=ios) writeVar !Unformatted write
```

The structure of the 2D spectra files are as follows:

```
!-------------
! Line 1
!-------------

time, n_r_max, l_max, minc ! Time, resolution, max(\ell), azimuthal symmetry

!-------------
! Line 2
!-------------

r(1), r(2), r(3), ..., r(n_r_max)                    ! Radius
```

(continues on next page)

(continued from previous page)

```
!-------------
! Line 3
!-------------

e_p_l(l=1,r=1), e_p_l(l=1,r=2), ..., e_p_l(l=1,r=n_r_max),       ! Poloidal␣
↪energy
...                                                               ! versus␣
↪degree
e_p_l(l=l_max,r=1), e_p_l(l=l_max,r=2), ..., e_p_l(l=l_max,r=n_r_max),

!-------------
! Line 4
!-------------

e_p_m(m=0,r=1), e_p_l(m=0,r=2), ..., e_p_l(m=1,r=n_r_max),       ! Poloidal␣
↪energy
...                                                               ! versus␣
↪order
e_p_l(m=l_max,r=1), e_p_l(m=l_max,r=2), ..., e_p_l(m=l_max,r=n_r_max),

!-------------
! Line 3
!-------------

e_t_l(l=1,r=1), e_t_l(l=1,r=2), ..., e_t_l(l=1,r=n_r_max),       ! Toroidal␣
↪energy
...                                                               ! versus␣
↪degree
e_t_l(l=l_max,r=1), e_t_l(l=l_max,r=2), ..., e_t_l(l=l_max,r=n_r_max),

!-------------
! Line 4
!-------------

e_t_m(m=0,r=1), e_t_l(m=0,r=2), ..., e_t_l(m=1,r=n_r_max),       ! Toroidal␣
↪energy
...                                                               ! versus␣
↪order
e_t_l(m=l_max,r=1), e_t_l(m=l_max,r=2), ..., e_t_l(m=l_max,r=n_r_max),
```

Those files can be read using the python class *MagicSpectrum2D* with the following options:

```
>>> # Read the file 2D_mag_spec_3.ext
>>> sp = MagicSpectrum2D(tag='ext', field='e_mag', ispec=3)
>>> # Print e_pol_l and e_tor_m
>>> print(sp.e_pol_l, sp.e_tor_m)
```

8.7.6 kin_spec_ave.TAG

Note: This file is **only** written when *l_spec_avg=.true.*

This file contains the time-average kinetic energy spectra as well as squared quantities to allow a possible further reconstruction of the standard deviation. This file is written by the subroutine `spectrum`.

No. of column	Contents
1	degree / order
2	Time-averaged poloidal kinetic energy versus degree
3	Time-averaged poloidal kinetic energy versus order
4	Time-averaged toroidal kinetic energy versus degree
5	Time-averaged toroidal kinetic energy versus order
6	Standard deviation of poloidal kinetic energy versus degree
7	Standard deviation of poloidal kinetic energy versus order
8	Standard deviation of toroidal kinetic energy versus degree
9	Standard deviation of toroidal kinetic energy versus order

This file can be read using *MagicSpectrum* with the following options:

```
>>> # To read the file ``kin_spec_ave.test``:
>>> sp = MagicSpectrum(field='kin', ave=True, tag='test')
```

8.7.7 mag_spec_ave.TAG

Note: This file is **only** written when *l_spec_avg=.true.* and the run is magnetic

This file contains the time-average magnetic energy spectra. This file is written by the subroutine `spectrum`.

No. of column	Contents
1	degree / order
2	Time-averaged poloidal magnetic energy in the outer core versus degree
3	Time-averaged poloidal magnetic energy in the outer core versus order
4	Time-averaged toroidal magnetic energy in the outer core versus degree
5	Time-averaged toroidal magnetic energy in the outer core versus order
6	Time-averaged poloidal magnetic energy at the CMB versus degree
7	Time-averaged poloidal magnetic energy at the CMB versus order
8	Standard deviation of the poloidal magnetic energy in the outer core versus degree
9	Standard deviation of the poloidal magnetic energy in the outer core versus order
10	Standard deviation of the toroidal magnetic energy in the outer core versus degree
11	Standard deviation of the toroidal magnetic energy in the outer core versus order
12	Standard deviation of the magnetic energy at the CMB versus degree
13	Standard deviation of the magnetic energy at the CMB versus order

This file can be read using *MagicSpectrum* with the following options:

```
>>> # To read the file ``mag_spec_ave.test``:
>>> sp = MagicSpectrum(field='mag', ave=True, tag='test')
```

8.7.8 T_spec_ave.TAG

Note: This file is **only** written when *l_spec_avg=.true.*

This file contains the time-averaged temperature/entropy spectra and their standard deviation. It is written by the subroutine `spectrum_temp_average`.

No. of column	Contents
1	Spherical harmonic degree/order
2	Time-averaged RMS temperature/entropy versus degree
3	Time-averaged RMS temperature/entropy versus order
4	Time-averaged RMS temperature/entropy at the ICB versus degree
5	Time-averaged RMS temperature/entropy at the ICB versus order
6	Time-averaged temperature/entropy gradient at the ICB versus degree
7	Time-averaged temperature/entropy gradient at the ICB versus order
8	Standard deviation of the temperature/entropy versus degree
9	Standard deviation of the temperature/entropy versus order
10	Standard deviation of the temperature/entropy at the ICB versus degree
11	Standard deviation of the temperature/entropy at the ICB versus order
12	Standard deviation of the temperature/entropy gradient at the ICB versus degree
13	Standard deviation of the temperature/entropy gradient at the ICB versus order

8.7.9 dtVrms_spec.TAG

Note: This file is **only** written when *l_RMS=.true.*

This file contains the time-averaged force balance spectra as well as their standard deviation. The calculations are done in the subroutine `dtVrms`.

No. of column	Contents
1	degree + 1
2	Time-averaged Inertia versus degree
3	Time-averaged Coriolis force versus degree
4	Time-averaged Lorentz force versus degree
5	Time-averaged Advection term versus degree
6	Time-averaged Viscous force versus degree
7	Time-averaged thermal Buoyancy versus degree
8	Time-averaged chemical Buoyancy versus degree
9	Time-averaged Pressure gradient versus degree
10 | Time-averaged Pressure/Coriolis balance versus degree	
11	Time-averaged Pressure/Coriolis/Lorentz balance versus degree
12	Time-averaged Pressure/Coriolis/Buoyancy balance versus degree
13	Time-averaged Pressure/Coriolis/Lorentz/Buoyancy balance versus degree
14	Time-averaged Coriolis/Lorentz balance versus degree
15	Time-averaged Pressure/Lorentz balance versus degree
16	Time-averaged Coriolis/Inertia/Buoyancy balance versus degree
17	Standard deviation of Inertia versus degree

continues on next page

Table 1 – continued from previous page

No. of column	Contents
18	Standard deviation of Coriolis force versus degree
19	Standard deviation of Lorentz force versus degree
20	Standard deviation of Advection term versus degree
21	Standard deviation of Viscous force versus degree
22	Standard deviation of thermal Buoyancy versus degree
23	Standard deviation of chemical Buoyancy versus degree
24	Standard deviation of Pressure gradient versus degree
25	Standard deviation of Pressure/Coriolis balance versus degree
26	Standard deviation of Pressure/Coriolis/Lorentz balance versus degree
27	Standard deviation of Pressure/Coriolis/Buoyancy balance versus degree
28	Standard deviation of Pressure/Coriolis/Lorentz/Buoyancy balance versus degree
29	Standard deviation of Coriolis/Lorentz balance versus degree
30	Standard deviation of Pressure/Lorentz balance versus degree
31	Standard deviation of Coriolis/Inertia/Buoyancy balance versus degree

This file can be read using *MagicSpectrum* with the following options:

```
>>> # To read the file ``dtVrms_spec.test``:
>>> sp = MagicSpectrum(field='dtVrms', tag='test')
```

8.7.10 2D force balance spectra `2D_dtVrms_spec.TAG`

Note: Those files are **only** written when *l_RMS=.true.* and *l_2D_RMS=.true.*.

Those files contain 2-D force balance spectra in the (r, ℓ) plane. The calculations are done in the subroutine `dtVrms`. The output file is stored as a Fortran unformatted file.

The structure of the 2D force balance spectra files are as follows:

```
!------------
! Line 1
!------------

version

!-------------
! Line 2
!-------------

n_r_max, l_max ! radial resolution, max(\ell)

!-------------
! Line 3
!-------------

r(1), r(2), r(3), ..., r(n_r_max)                    ! Radius

!-------------
! Line 4
```

(continues on next page)

(continued from previous page)

```
!-------------

Cor_l(l=1,r=1), Cor_l(l=1,r=2), ..., Cor_l(l=1,r=n_r_max),       ! Coriolis␣
↪force
...                                                              ! versus␣
↪degree
Cor_l(l=l_max,r=1), Cor_l(l=l_max,r=2), ..., Cor_l(l=l_max,r=n_r_max),

!-------------
! Line 5
!-------------

Adv_l ! Advection

!-------------
! Line 6
!-------------

LF_l ! Lorentz force

!-------------
! Line 7
!-------------

Buo_temp_l ! Thermal buoyancy

!------------
! Line 8
!------------

Buo_xi_l ! Chemical buoyancy

!-------------
! Line 9
!-------------

Pre_l ! Pressure

!-------------
! Line 10
!-------------

Dif_l ! Viscosity

!-------------
! Line 11
!-------------

Iner_l ! Inertia

!-------------
! Line 12
!-------------

Geo_l ! Sum of force terms: geostrophic balance

!-------------
```

(continues on next page)

(continued from previous page)

```
! Line 13
!-------------

Mag_l ! Sum of force terms: pressure, Coriolis and Lorentz

!-------------
! Line 14
!-------------

Arc_l ! Sum of force terms: pressure, buoyancy and Coriolis

!-------------
! Line 15
!-------------

ArcMag_l ! Sum of force terms: pressure, buoyancy, Coriolis and Lorentz

!-------------
! Line 16
!-------------

CIA_l ! Sum of force terms Coriolis/Inertia/Archimedean

!-------------
! Line 17
!-------------

CLF_l ! Sum of force terms Coriolis/Lorentz

!-------------
! Line 18
!-------------

PLF_l ! Sum of force terms Pression/Lorentz
```

Those files can be read using the python class *MagicSpectrum2D* with the following options:

```
>>> # Read the file 2D_dtVrms_spec.ext
>>> sp = MagicSpectrum2D(tag='ext', field='dtVrms')
>>> # Print Cor_l
>>> print(sp.Cor_l)
```

8.7.11 2D spectra *am_[kin|mag]_[pol|tor].TAG*

Those files contain the time evolution of the poloidal and toroidal kinetic and magnetic spectra for a given range of spherical harmonic orders m. There are four kinds of those files that correspond to the aforementioned spectra, namely **am_kin_pol.TAG**, **am_kin_tor.TAG**, **am_mag_pol.TAG** and **am_mag_tor.TAG**. The calculations are done in the subroutine `get_amplitude`. The structure of the output files is the same for the four outputs (fortran unformatted files):

```
!-------------
! Line 1
!-------------

time(t=0), e_p_m(m=0,t=0), e_p_m(m=1,t=0), ..., e_p_m(m=m_max_modes,t=0)
```

(continues on next page)

(continued from previous page)

```
...

!-------------
! Line N
!-------------

time(t=N), e_p_m(m=0,t=N), e_p_m(m=1,t=N), ..., e_p_m(m=m_max_modes,t=N)

...
```

Those files can be read using the python class *MagicTs* with the following options:

```
>>> # Read the file am_mag_pol.ext
>>> ts = MagicTs(field='am_mag_pol', tag='ext')
>>> # Print the time
>>> print(ts.time)
>>> # Print the energy content in m=11 for all times
>>> print(ts.coeffs[:, 11])
```

8.8 Graphic files G_#.TAG and G_ave.TAG

These are fortran unformatted files containing 3D data (in the form vector_array(phi, theta, r)) which can be used to visualize the solution. They are written after a fixed number of time steps as specified by the user in the *Output Control namelist* using the parameters listed in the section on *output of graphic files*. In case *l_average* is set to `.true.`, then an average graphic file, named `G_ave.TAG`, containing time averaged values of 3D data, is also written at the end of the simulation.

These files are written in chunks of latitude for one radial level at a time by the subroutine `graphOut` or by `graphOut_mpi` depending on whether `USE_MPI` is set to `Yes` or `No` in the Makefile. The structure of the file looks like below:

```
!-------------
! Line 1
!-------------

version    !Graphout_version_9 (using MPI without comp. without pressure)
           !Graphout_version_10 (using MPI, without comp. with pressure)
           !Graphout_version_11 (using MPI, with comp. without pressure)
           !Graphout_version_12 (using MPI, with comp. with pressure)
           !Graphout_version_5 (without MPI, with pressure and comp.)
           !Graphout_version_6 (without MPI, with comp. without pressure)
           !Graphout_version_7 (without MPI, without comp. without pressure)
           !Graphout_version_8 (without MPI, without comp. with pressure)
!----------
! Line 2
!----------

runid

!----------
! Line 3
!----------
```

(continues on next page)

(continued from previous page)

```
time, n_r_max, n_theta_max, n_phi_tot,          !time = Time of writing
n_r_ic_max-1, minc, nThetasBs,                  !(Simulation time),
ra, ek, pr, prmag,                              !nThetasBs = no. of
radratio, sigma_ratio                           !theta blocks

!----------
! Line 4
!----------

theta(1:n_theta_max)

!------------------------------------------------------------------------

!-----------------------------
!Graphout_version_[9/10/11/12]
!-----------------------------

! These versions are written when the code uses MPI (USE_MPI=yes). Parallel
! chunks of fields are written for different radial levels. Chunks in theta
! are written in parallel using OpenMP

!---------------
! Data
!---------------

!-------------
! Block N
!-------------

!------------
! Line 4 + N
!------------


n_r-1, r(n_r)/r(1), n_theta_start, n_theta_stop   !Radial index, radius in␣
↪terms
                                                  !of r_cmb, start and stop␣
↪of
                                                  !the theta block

!---------------
! Line 4 + (N+1)
!---------------

sr(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !Entropy

!---------------
! Line 4 + (N+2)
!---------------

vr(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !Radial velocity

!---------------
! Line 4 + (N+3)
!---------------

vt(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !Theta component of␣
↪velocity
```

(continues on next page)

(continued from previous page)

```
!---------------
! Line 4 + (N+4)
!---------------

vp(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !Zonal (phi component) of
                                                  !velocity
if (l_chemical_conv):                             !If composition is stored

  !---------------
  ! Line 4 + (N+5)
  !---------------

  xir(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !composition



if (l_PressGraph):                                !If pressure is stored

  !-----------------
  ! Line 4 + (N+6/7)
  !-----------------

  pr(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !pressure


if (l_mag):                                       !For a magnetic run

  !-------------------
  ! Line 4 + (N+5/6/7)
  !-------------------

  br(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !Radial magnetic field

  !-------------------
  ! Line 4 + (N+6/7/8)
  !-------------------

  bt(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !Theta component of
                                                    !magnetic field

  !-------------------
  ! Line 4 + (N+7/8/9)
  !-------------------

  bp(1:n_phi_tot, n_theta_start:n_theta_stop, n_r)  !Zonal (phi component)
                                                    !of magnetic field


!--------------------------
!Graphout_version_[5/6/7/8]
!--------------------------

!This version is written when the code does not use MPI (USE_MPI=no).
!Chunks in theta are written in parallel with OpenMP.

!---------------
```

(continues on next page)

(continued from previous page)

```
! Data
!---------------

!-------------
! Block N
!-------------

!---------------
! Line 4 + (N+1)
!---------------

n_r-1, r(n_r)/r(1), n_theta_start, n_theta_stop

!------------------------------------------------------------
! Each of the following data point is written in a new line
!------------------------------------------------------------

!-----------------
! Entropy
!-----------------

sr(1,n_theta_start,n_r)            !n_phi = 1, n_theta = n_theta_start, n_r
sr(2,n_theta_start,n_r)            !n_phi = 2, n_theta = n_theta_start, n_r
...
sr(n_phi_tot,n_theta_start,n_r)    !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
sr(1,n_theta_start+1,n_r)          !n_phi = 1, n_theta = n_theta_start+1, n_r
...
sr(n_phi_tot,n_theta_start+1,n_r)
...
sr(1,n_theta_stop,n_r)             !n_phi = 1, n_theta = n_theta_stop, n_r
sr(2,n_theta_stop,n_r)             !n_phi = 2, n_theta = n_theta_stop, n_r
...
sr(n_phi_tot,n_theta_stop,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r

!-----------------
! Radial velocity
!-----------------

vr(1,n_theta_start,n_r)            !n_phi = 1, n_theta = n_theta_start, n_r
vr(2,n_theta_start,n_r)            !n_phi = 2, n_theta = n_theta_start, n_r
...
vr(n_phi_tot,n_theta_start,n_r)    !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
vr(1,n_theta_start+1,n_r)          !n_phi = 1, n_theta = n_theta_start+1, n_r
...
vr(n_phi_tot,n_theta_start+1,n_r)
...
vr(1,n_theta_stop,n_r)             !n_phi = 1, n_theta = n_theta_stop, n_r
vr(2,n_theta_stop,n_r)             !n_phi = 2, n_theta = n_theta_stop, n_r
...
vr(n_phi_tot,n_theta_stop,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r

!-----------------------------
! Theta component of velocity
```

(continues on next page)

(continued from previous page)

```
!----------------------------

vt(1,n_theta_start,n_r)            !n_phi = 1, n_theta = n_theta_start, n_r
vt(2,n_theta_start,n_r)            !n_phi = 2, n_theta = n_theta_start, n_r
...
vt(n_phi_tot,n_theta_start,n_r)    !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
vt(1,n_theta_start+1,n_r)          !n_phi = 1, n_theta = n_theta_start+1, n_r
...
vt(n_phi_tot,n_theta_start+1,n_r)
...
vt(1,n_theta_stop,n_r)             !n_phi = 1, n_theta = n_theta_stop, n_r
vt(2,n_theta_stop,n_r)             !n_phi = 2, n_theta = n_theta_stop, n_r
...
vt(n_phi_tot,n_theta_stop,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r


!----------------------------------
! Zonal (phi component) of velocity
!----------------------------------

vp(1,n_theta_start,n_r)            !n_phi = 1, n_theta = n_theta_start, n_r
vp(2,n_theta_start,n_r)            !n_phi = 2, n_theta = n_theta_start, n_r
...
vp(n_phi_tot,n_theta_start,n_r)    !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
vp(1,n_theta_start+1,n_r)          !n_phi = 1, n_theta = n_theta_start+1, n_r
...
vp(n_phi_tot,n_theta_start+1,n_r)
...
vp(1,n_theta_stop,n_r)             !n_phi = 1, n_theta = n_theta_stop, n_r
vp(2,n_theta_stop,n_r)             !n_phi = 2, n_theta = n_theta_stop, n_r
...
vp(n_phi_tot,n_theta_stop,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r

if (l_chemical_conv):              !If chemical composition is stored

!-------------
! Composition
!-------------

xi(1,n_theta_start,n_r)            !n_phi = 1, n_theta = n_theta_start, n_r
xi(2,n_theta_start,n_r)            !n_phi = 2, n_theta = n_theta_start, n_r
...
xi(n_phi_tot,n_theta_start,n_r)    !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
xi(1,n_theta_start+1,n_r)          !n_phi = 1, n_theta = n_theta_start+1, n_r
...
xi(n_phi_tot,n_theta_start+1,n_r)
...
xi(1,n_theta_stop,n_r)             !n_phi = 1, n_theta = n_theta_stop, n_r
xi(2,n_theta_stop,n_r)             !n_phi = 2, n_theta = n_theta_stop, n_r
...
xi(n_phi_tot,n_theta_stop,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r
```

(continues on next page)

(continued from previous page)

```
if (l_PressGraph):                  !If pressure is stored

!-----------
! Pressure
!-----------

pr(1,n_theta_start,n_r)             !n_phi = 1, n_theta = n_theta_start, n_r
pr(2,n_theta_start,n_r)             !n_phi = 2, n_theta = n_theta_start, n_r
...
pr(n_phi_tot,n_theta_start,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
pr(1,n_theta_start+1,n_r)           !n_phi = 1, n_theta = n_theta_start+1, n_r
...
pr(n_phi_tot,n_theta_start+1,n_r)
...
pr(1,n_theta_stop,n_r)              !n_phi = 1, n_theta = n_theta_stop, n_r
pr(2,n_theta_stop,n_r)              !n_phi = 2, n_theta = n_theta_stop, n_r
...
pr(n_phi_tot,n_theta_stop,n_r)      !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r


if (l_mag):                         !Only if it is a magnetic case

!----------------------
! Radial magnetic field
!----------------------

br(1,n_theta_start,n_r)             !n_phi = 1, n_theta = n_theta_start, n_r
br(2,n_theta_start,n_r)             !n_phi = 2, n_theta = n_theta_start, n_r
...
br(n_phi_tot,n_theta_start,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
br(1,n_theta_start+1,n_r)           !n_phi = 1, n_theta = n_theta_start+1, n_r
...
br(n_phi_tot,n_theta_start+1,n_r)
...
br(1,n_theta_stop,n_r)              !n_phi = 1, n_theta = n_theta_stop, n_r
br(2,n_theta_stop,n_r)              !n_phi = 2, n_theta = n_theta_stop, n_r
...
br(n_phi_tot,n_theta_stop,n_r)      !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r


!----------------------------------
! Theta component of magnetic field
!----------------------------------

bt(1,n_theta_start,n_r)             !n_phi = 1, n_theta = n_theta_start, n_r
bt(2,n_theta_start,n_r)             !n_phi = 2, n_theta = n_theta_start, n_r
...
bt(n_phi_tot,n_theta_start,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
bt(1,n_theta_start+1,n_r)           !n_phi = 1, n_theta = n_theta_start+1, n_r
...
```

(continues on next page)

(continued from previous page)

```
bt(n_phi_tot,n_theta_start+1,n_r)
...
bt(1,n_theta_stop,n_r)              !n_phi = 1, n_theta = n_theta_stop, n_r
bt(2,n_theta_stop,n_r)              !n_phi = 2, n_theta = n_theta_stop, n_r
...
bt(n_phi_tot,n_theta_stop,n_r)      !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r


!----------------------------------------
! Zonal (phi component) of magnetic field
!----------------------------------------

bp(1,n_theta_start,n_r)             !n_phi = 1, n_theta = n_theta_start, n_r
bp(2,n_theta_start,n_r)             !n_phi = 2, n_theta = n_theta_start, n_r
...
bp(n_phi_tot,n_theta_start,n_r)     !n_phi = n_phi_tot, n_theta = n_theta_
↪start, n_r
bp(1,n_theta_start+1,n_r)           !n_phi = 1, n_theta = n_theta_start+1, n_r
...
bp(n_phi_tot,n_theta_start+1,n_r)
...
bp(1,n_theta_stop,n_r)              !n_phi = 1, n_theta = n_theta_stop, n_r
bp(2,n_theta_stop,n_r)              !n_phi = 2, n_theta = n_theta_stop, n_r
...
bp(n_phi_tot,n_theta_stop,n_r)      !n_phi = n_phi_tot, n_theta = n_theta_stop,
↪ n_r

!-----------------
!Subsequent blocks
!-----------------

!Block N+1 in both cases have data at the same radial level but the next
!theta chunk (n_theta_start + nThetaB, n_theta_stop + n_thetaB)

!After data for all the theta blocks have been written for one radial
!level, everything above is repeated for the next radial level
```

The graphic files can be read using the python class *MagicGraph*.

```
>>> gr = MagicGraph(ivar = 1, tag='TAG')
>>> # print radial velocity
>>> print(gr.vr)
```

They can be visualized using the *Surf* class:

```
>>> s = Surf(tag='TAG')
>>> # Surface map of radial velocity:
>>> s.surf(field = 'vr', r = 0.5, cm = 'jet', levels = 50)
>>> s.slice(field = 'br', lon_0 = [0]) # Longitudinal Slice of radial magnetic field
>>> s.equat(field = 'entropy')         # Equatorial slice of entropy
```

8.9 Movie files `*_mov.TAG`

Note: These files are written **only** when *l_movie = .true.* or when a finite number of movie frames are asked for using the input parameters described in the *standard inputs section* of the *output control namelist*.

These are unformatted fortran files containing time evolution of fields on different surfaces - constant radius, colatitude or azimuth or on the full 3D grid. The fields can be of various types like radial magnetic field or velocity, entropy, helicity etc. The type of field and the type of surface can be specified using a string that begins with the field name, followed by the surface type (or 'full 3D', when a 3D movie is desired). One such example is as follows:

```
l_movie = .true.,
n_movie_frames = 1000,
movie(1) = "B r r=0.5",
movie(2) = "V all 3D",
movie(3) = "Hel Eq"
```

The code does not interpret any whitespaces and is not case-sensitive so there's no difference between, say, `B r cmb` and `brcmb`. For further details and a list of keywords for different fields and surfaces, please refer to the *movie* in the *output control namelist*.

These files are written by the subroutine `write_movie_frame`.

The movie files are suitably named to reflect the type of field and surface. Their names begin with the keyword for the type of movie asked for, followed by the type of surface, followed by the word 'mov'. Thus, a generic movie name looks like:

```
Keyword_SurType_mov.TAG
```

E.g: if one asks for the radial component of magnetic field on surface of CMB, the movie would be named as `Br_CMB_mov.TAG`.

When asks multiple movies for same surface types but different surface levels, the surfaces are numbered with integers. Thus, for the following namelist input,

```
l_movie = .true.,
n_movie_frames = 1000,
movie(1) = "B r r=0.5",
movie(2) = "V p r=0.5",
movie(3) = "V r r=0.8",
```

one would get the following movie files as output:

```
Br_R=C1_mov.TAG
Vp_R=C1_mov.TAG
Vr_R=C2_mov.TAG
```

The structure of a generic movie file is as follows:

```
!-------
! Line 1
!-------

version                         !Movie version: 'JW_Movie_Version_2'

!-------
! Line 2
```

(continues on next page)

(continued from previous page)

```
!-------

n_type, n_surface,             !Type of movie,
const, n_fields                !Type of surface (r,theta,phi,CMB,Eq etc.)

!-------
! Line 3
!-------

n_movie_field_type(1:n_fields, n_movie) !Type of fields (velocity,
                                        !mag field, vorticity etc.)
!-------
! Line 4
!-------

runid

!-------
! Line 5
!-------

n_r_mov_tot, n_r_max,          !Total number of
n_theta_max, n_phi_max,        !radial grid points (including IC),
minc, ra, ek, pr, prmag,       !grid data, physical parameters
radratio, tScale

!-------
! Line 6
!-------

r_mov_tot(1:n_r_mov_tot)/r_cmb !All radii in terms of r_CMB

!-------
! Line 7
!-------

theta(1:n_theta_max)           !All theta points

!-------
! Line 8
!-------

phi(1:n_phi_max)               !All phi points

!----------------------------------------------------------------------

!---------
! Frame N
!---------

!-----------
! Line 8 + N
!-----------

n_frame, t_movie(N), omega_ic, omega_ma, dipLat, dipLon, dipStr, dipStrGeo

!---------------
```

(continues on next page)

(continued from previous page)

```
! Line 8 + (N+1)
!---------------

frame_data(1:n_fields,n_start:n_stop)  !Desired field data on a
                                       !surface or 3D volume
                                       !n_start = start index of a field
                                       !n_stop  = last index of a field

!-----------
! Frame N+1
!-----------

!---------------
! Line 8 + (N+2)
!---------------

n_frame, t_movie(N+1), omega_ic, omega_ma, dipLat, dipLon, dipStr, dipStrGeo

!---------------
! Line 8 + (N+3)
!---------------

frame_data(1:n_fields,n_start:n_stop)  !Desired field data on a
                                       !surface or 3D volume
                                       !n_start = start index of a field
                                       !n_stop  = last index of a field

...

!-----------
! Frame N+M                           !M is the desired number of movie␣
↪frames
!-----------

!---------------
! Line 8 + (N+M)
!---------------

n_frame, t_movie(N+M), omega_ic, omega_ma, dipLat, dipLon, dipStr, dipStrGeo

!---------------
! Line 8 + (N+M)
!---------------

frame_data(1:n_fields,n_start:n_stop)  !Desired field data on a
                                       !surface or 3D volume
                                       !n_start = start index of a field
                                       !n_stop  = last index of a field
```

The 2D movie files can be read and displayed using the python class *Movie* as follows:

```
>>> Movie()   #Lists out available movie files to choose from
>>> M = Movie(file = 'Vr_R=C1_mov.TAG')
```

The 3D movie files can be read using the python class *Movie3D*:

```
>>> M = Movie3D(file = 'V_3D_mov.TAG')
```

8.10 Restart files checkpoint_*.TAG

Note: These frequency of writing these files are determined by the standard inputs mentioned in the section on *restart files* in the *output control namelist*. If nothing is specified then, by default one restart file is written at the end of the run.

Note: A restart file is read **only** when *l_start = .true.*

These are unformatted fortran files containing a snapshot of information about spectral coefficients and physical and grid parameters. As the name suggests, these files are used to 'restart' a run from a specific time. One such file is read by the code at the beginning and are used as initial conditions for the run. These are very useful for continuing a simulation for a long time on computing clusters where the time for a single run is limited.

The file to be read at the beginning is specified by the input parameter *start_file* which takes in a string providing path to the file.

These files are written by the subroutine `store`.

The following notations will be used for the coefficients of potentials (note that scalar fields like temperature and pressure do not have a poloidal/toroidal decomposition):

Field	Poloidal	Toroidal
Magnetic	`b`	`aj`
Velocity	`w`	`z`
Temperature	`s`	
Pressure	`p`	

Time derivatives are denoted with a self-explanatory notation. e.g, `dbdt` is the first derivative of `b`.

The word `Last` appended to a variable name denotes that the value is of the time-step previous to the one during which the file is being written. They are needed for the time-stepping schemes.

`_ic` with a variable name says that it belongs to the Inner Core.

```
!--------
! Line 1
!--------

time*tScale, dt*tScale, ra, pr, prmag, ek, radratio, inform, n_r_max,
n_theta_max, n_phi_tot, minc, nalias, n_r_ic_max, sigma_ratio

if (l_heat):                    !Run involving heat transport
                                !(Convection)
!---------
! Line 2
!---------

   w,z,p,s
```

(continues on next page)

(continued from previous page)

```
!---------
! Line 3
!---------


   dsdtLast,dwdtLast,dzdtLast,dpdtLast

else:

!---------
! Line 2
!---------


   w,z,p

!---------
! Line 3
!---------

  dwdtLast,dzdtLast,dpdtLast

if (l_mag):                                    !If magnetic run

!---------
! Line 4
!---------


  b, aj, dbdtLast, djdtLast

 if(l_mag .and. l_cond_ic):                    !If magnetic run
                                               !and conducting inner core
!---------
! Line 5
!---------

  b_ic, aj_ic, dbdt_icLast, djdt_icLast

!-------------------------------------------------
! Line 4 or 5 or 6 depending on l_mag and l_cond_ic
!-------------------------------------------------

lorentz_torque_icLast, lorentz_torque_maLast, !Information about torques,
omega_ic1, omegaOsz_ic1, tOmega_ic1,          !prescribed rotation and
omega_ic2, omegaOsz_ic2, tOmega_ic2,          !oscillation rates,
omega_ma1, omegaOsz_ma1, tOmega_ma1,          !and the time step-size
omega_ma2, omegaOsz_ma2, tOmega_ma2,
dtNew
```

The checkpoint files can be read using the python class `MagicCheckpoint`.

```
>>> chk = MagicCheckpoint(filename='checkpoint_end.test')
>>> # print size of poloidal and l_max
>>> print(chk.wpol.shape, chk.l_max)
>>> # convert from cheb to FD using 96 grid points
>>> chk.cheb2fd(96)
```

(continues on next page)

(continued from previous page)

```
>>> write new file
>>> chk.write('checkpoint_fd.test')
```

8.11 Poloidal and toroidal potentials at given depths

These are fortran unformatted files which store time series of poloidal and toroidal coefficients of different fields (magnetic field, velocity and temeperature) at specific depths.

In the following, `time(j)` is the time during the j^{th} time step, `time(N)` being the last step. `real` and `imag` denote real and imaginary parts, respectively, of spherical harmonic coefficients. Also, the following notations will be used for the coefficients of potentials (note that scalar fields like temperature do not have a poloidal/toroidal decomposition):

Field	Poloidal	Toroidal
Magnetic	`b`	`aj`
Velocity	`w`	`z`
Temperature	`s`	

First and second derivatives are denoted with a differential notation. e.g: `dw` is the first derivative of `w`, while `ddb` is the second derivative of `b`.

8.11.1 `B_coeff_cmb.TAG`

Note: This file is **only** written when *l_cmb_field=.true.*

This file contains time series of spherical harmonic coefficients for the poloidal potential of the magnetic field at the outer boundary (CMB) up to a spherical harmonic degree given by *l_max_cmb*. The detailed calculations are done in the subroutine `write_Bcmb`. The contents of the file look as follows:

- **Header** The file header consists of the information: *l_max_cmb*, *minc* and the number of data points `n_data`.
- **Data** Each chunk of data after the header has the same pattern of `time` followed by a list of real and imaginary values of coefficients.

Thus, on a whole, the structure of the file looks like follows:

```
!------------
! Line 1
!------------

l_max_cmb, minc, n_data

!------------------------------
...

!------------
! Line j + 1
!------------

time(j),
real(b(l=1,m=0)), imag(b(l=1,m=0)),
```

(continues on next page)

(continued from previous page)

```
real(b(l=2,m=0)), imag(b(l=2,m=0)),
...
real(b(l=l_max_cmb,m=l_max_cmb)), imag(b(l=l_max_cmb,m=l_max_cmb)),

...

!-------------
! Line N + 1
!-------------

time(N),
real(b(l=1,m=0)), imag(b(l=1,m=0)),
real(b(l=2,m=0)), imag(b(l=2,m=0)),
...
real(b(l=l_max_cmb,m=l_max_cmb)), imag(b(l=l_max_cmb,m=l_max_cmb))
```

This file can be read using *MagicCoeffCmb* with the following options:

```
>>> # To stack the files B_cmb_coeff.testc to B_cmb_coeff.testf
>>> cmb = MagicCoeffCmb(tag='test[c-f]')
>>> # print Gauss coefficient for (\ell=10, m=3)
>>> print(cmb.glm[:, 10, 3])
```

8.11.2 Coefficients at desired radii

The following files **[B|V|T]_coeff_r#.TAG** save coefficients at specified depths and are written by the subroutine `write_coeff_r`. See the section on *CMB and radial coefficients* in the *ouput control namelist* for details of specifying depth, using *n_r_step* or *n_r_array* and desired maximum degree of output *l_max_r*. A separate file for each desired radius is written, numbered suitably as `[B|V|T]_coeff_r1.TAG`, `[B|V|T]_coeff_r2.TAG` etc.

8.11.3 `B_coeff_r#.TAG`

Note: This file is **only** written when *l_r_field=.true.*.

This file contains output of time series of the spherical harmonic coefficients of the poloidal and toroidal magnetic field potentials and the first and second derivatives of the poloidal potential coefficients in the order `b`, `db`, `aj` and `ddb`. The output is for a specific radius, r up to degree *l_max_r*.

- **Header** The file header consists of the information: *l_max_r*, *minc*, the number of data points `n_data` and the radius, `r`.
- **Data** Each chunk of data after the header contains the `time` at which the coefficients are stored, followed by the real and imaginary parts of: the poloidal coefficient `b`, it's first derivative `db`, the toroidal coefficient `aj` and the second derivative of the poloidal coefficient `ddb`.

The complete structure of the file looks like follows:

```
!------------
! Line 1
!------------

l_max_r, minc, n_data, r
```

(continues on next page)

(continued from previous page)

```
!-------------------------------------------
...

!------------
! Line j + 1
!------------

time(j),
real(b(l=1,m=0)), imag(b(l=1,m=0)),
real(b(l=2,m=0)), imag(b(l=2,m=0)),
...
real(b(l=l_max_cmb,m=l_max_cmb)), imag(b(l=l_max_cmb,m=l_max_cmb)),
real(db(l=1,m=0)), imag(db(l=1,m=0)),
real(db(l=2,m=0)), imag(db(l=2,m=0)),
...
real(db(l=l_max_cmb,m=l_max_cmb)), imag(db(l=l_max_cmb,m=l_max_cmb)),
real(aj(l=1,m=0)), imag(aj(l=1,m=0)),
real(aj(l=2,m=0)), imag(aj(l=2,m=0)),
...
real(aj(l=l_max_cmb,m=l_max_cmb)), imag(aj(l=l_max_cmb,m=l_max_cmb)),
real(ddb(l=1,m=0)), imag(ddb(l=1,m=0)),
real(ddb(l=1,m=0)), imag(ddb(l=1,m=0)),
...
real(ddb(l=l_max_cmb,m=l_max_cmb)), imag(ddb(l=l_max_cmb,m=l_max_cmb)),

...

!------------
! Line N + 1
!------------

time(N),
real(b(l=1,m=0)), imag(b(l=1,m=0)),
real(b(l=2,m=0)), imag(b(l=2,m=0)),
...
real(b(l=l_max_cmb,m=l_max_cmb)), imag(b(l=l_max_cmb,m=l_max_cmb)),
real(db(l=1,m=0)), imag(db(l=1,m=0)),
real(db(l=2,m=0)), imag(db(l=2,m=0)),
...
real(db(l=l_max_cmb,m=l_max_cmb)), imag(db(l=l_max_cmb,m=l_max_cmb)),
real(aj(l=1,m=0)), imag(aj(l=1,m=0)),
real(aj(l=2,m=0)), imag(aj(l=2,m=0)),
...
real(aj(l=l_max_cmb,m=l_max_cmb)), imag(aj(l=l_max_cmb,m=l_max_cmb)),
real(ddb(l=0,m=0)), imag(ddb(l=0,m=0)),
real(ddb(l=1,m=0)), imag(ddb(l=1,m=0)),
...
real(ddb(l=l_max_cmb,m=l_max_cmb)), imag(ddb(l=l_max_cmb,m=l_max_cmb))
```

This file can be read using *MagicCoeffR* with the following options:

```
>>> # To stack the files B_coeff_r3.test* from the working directory
>>> cr = MagicCoeffR(tag='test*', field='B', r=3)
>>> # print the time and the poloidal potential for (\ell=3, m=3)
>>> print(cr.time, cr.wlm[:, 3, 3])
```

8.11.4 V_coeff_r#.TAG

Note: This file is **only** written when *l_r_field=.true.*

This file contains output of time series of the spherical harmonic coefficients of the poloidal and toroidal velocity field potentials and the first derivatives of the poloidal potential coefficients in the order `w`, `dw`, and `z`. The output is for a specific radius, r up to degree *l_max_r*.

- **Header** The file header consists of the information: *l_max_r*, *minc*, the number of data points `n_data` and the radius, `r`.
- **Data** Each chunk of data after the header contains the `time` at which the coefficients are stored, followed by the real and imaginary parts of: the poloidal coefficient `w`, it's first derivative `dw` and the toroidal coefficient `z`.

The complete structure of the file looks like follows:

```
!------------
! Line 1
!------------

l_max_r, minc, n_data, r

!----------------------------------
...

!------------
! Line j + 1
!------------

time(j),
real(w(l=1,m=0)), imag(w(l=1,m=0)),
real(w(l=2,m=0)), imag(w(l=2,m=0)),
...
real(w(l=l_max_cmb,m=l_max_cmb)), imag(w(l=l_max_cmb,m=l_max_cmb)),
real(dw(l=1,m=0)), imag(dw(l=1,m=0)),
real(dw(l=2,m=0)), imag(dw(l=2,m=0)),
...
real(dw(l=l_max_cmb,m=l_max_cmb)), imag(dw(l=l_max_cmb,m=l_max_cmb)),
real(z(l=1,m=0)), imag(z(l=1,m=0)),
real(z(l=2,m=0)), imag(z(l=2,m=0)),
...
real(z(l=l_max_cmb,m=l_max_cmb)), imag(z(l=l_max_cmb,m=l_max_cmb)),

...

!--------------
! Line N + 1
!--------------

time(N),
real(w(l=1,m=0)), imag(w(l=1,m=0)),
real(w(l=2,m=0)), imag(w(l=2,m=0)),
...
real(w(l=l_max_cmb,m=l_max_cmb)), imag(w(l=l_max_cmb,m=l_max_cmb)),
real(dw(l=1,m=0)), imag(dw(l=1,m=0)),
real(dw(l=2,m=0)), imag(dw(l=2,m=0)),
...
```

(continues on next page)

(continued from previous page)

```
real(dw(l=l_max_cmb,m=l_max_cmb)), imag(dw(l=l_max_cmb,m=l_max_cmb)),
real(z(l=1,m=0)), imag(z(l=1,m=0)),
real(z(l=2,m=0)), imag(z(l=2,m=0)),
...
real(z(l=l_max_cmb,m=l_max_cmb)), imag(z(l=l_max_cmb,m=l_max_cmb))
```

This file can be read using *MagicCoeffR* with the following options:

```
>>> # To stack the files V_coeff_r3.test* from the working directory
>>> cr = MagicCoeffR(tag='test*', field='V', r=3)
>>> # print the poloidal and toroidal potentials for (\ell=6, m=0)
>>> print(cr.wlm[:, 6, 0], cr.zlm[:, 6, 0])
```

8.11.5 T_coeff_r#.TAG

Note: This file is **only** written when *l_r_fieldT=.true.*

This file contains output of time series of the spherical harmonic coefficients of the temperature (or entropy) field. The output is for a specific radius, r up to degree *l_max_r*.

- **Header** The file header consists of the information: *l_max_r*, *minc*, the number of data points `n_data` and the radius, `r`.
- **Data** Each chunk of data after the header contains the `time` at which the coefficients are stored, followed by the real and imaginary parts of the coefficient `s`.

The complete structure of the file looks like follows:

```
!------------
! Line 1
!------------

l_max_r, minc, n_data, r

!----------------------------------

...

!------------
! Line j + 1
!------------

time(j),
real(s(l=0,m=0)), imag(s(l=0,m=0)),
real(s(l=1,m=0)), imag(s(l=1,m=0)),
real(s(l=2,m=0)), imag(s(l=2,m=0)),
...
real(s(l=l_max_cmb,m=l_max_cmb)), imag(s(l=l_max_cmb,m=l_max_cmb)),

!------------
! Line N + 1
!------------

time(N),
```

(continues on next page)

(continued from previous page)

```
real(s(l=0,m=0)), imag(s(l=0,m=0)),
real(s(l=1,m=0)), imag(s(l=1,m=0)),
real(s(l=2,m=0)), imag(s(l=2,m=0)),
...
real(s(l=l_max_cmb,m=l_max_cmb)), imag(s(l=l_max_cmb,m=l_max_cmb)),
```

8.12 TO outputs

Note: These output files are **only** written when *l_TO=.true.*

8.12.1 Tay.TAG

This file contains the time series of the Taylorisation as well as some measures of the relative geostrophic energy.

This file can be read using *MagicTs* with the following options:

```
>>> # To load the most recent 'Tay.TAG' file in a directory
>>> ts = MagicTs(tag='Tay')
```

8.12.2 TOnhs.TAG and TOshs.TAG

Those files correspond to the z-averaging of the axisymmetric phi component of the Navier-Stokes equations. It contains the different cylindrical profiles of the forces involved the zonal equation as well as some additional measures of the Taylorization of the solution. shs corresponds to Southern Hemisphere (inside the tangent cylinder), while nhs corresponds to Nothern Hemisphere).

Those files can be read using *MagicTOHemi* with the following options:

```
>>> # To load 'TOshs.test' and plot the time-averaged forces:
>>> tos = MagicTOHemi(tag='test', hemi='s', iplot=True)
```

8.12.3 TO_mov.TAG files

Note: This file is **only** written when *l_TOmovie=.true.*

This file contains the time evolution of the different forces that enter the phi-average of the azimuthal component of the Navier-Stokes equation. This is a special kind of *movie file* that contains seven different azimuthally-averaged fields in a (r, θ) plane : the axisymmetric zonal flow component , the azimuthal component of the Reynolds stresses, the azimuthal component of advection, the azimuthal component of viscosity, the azimuthal component of Lorentz force, the azimuthal component of Coriolis force and the azimuthal component of the time-derivative. The structure of the file is similar to a *movie file*, i.e. an unformatted fortran binary file with a header that describes the type of the movie file. The detailed calculations can be found in the subroutine `outTO`.

On a whole, the structure of the file looks like follows:

```
!----------
! Line 1
!----------

version

!----------
! Line 2
!----------

n_type, n_surface, const, n_fields

!----------
! Line 3
!----------

runid

!----------
! Line 4
!----------

n_r_movie_max, n_r_max, n_theta_max, n_phi_tot, minc, ra, ek, pr,
prmag, radratio, tScale

!----------
! Line 5
!----------

r(1), r(2), ..., r(n_r_movie_max)

!----------
! Line 6
!----------

theta(1), theta(2), ..., theta(n_theta_max)

!----------
! Line 7
!----------

phi(1), phi(2), ..., phi(n_theta_max)

...

!----------
! Line 7+N
!----------

n_frame, t_movie(N), omega_ic, omega_ma, dipLat, dipLon, dipStr, dipStrGeo

!----------
! Line 7+(N+1)
!----------

vphi(t=t_movie(N),phi=1,theta=1),
vphi(t=t_movie(N),phi=1,theta=2),
```

(continues on next page)

(continued from previous page)

```
...,
vphi(t=t_movie(N),phi=n_phi_max,theta=n_theta_max)

!----------
! Line 7+(N+2)
!----------

rey(t=t_movie(N),phi=1,theta=1),
rey(t=t_movie(N),phi=1,theta=2),
...,
rey(t=t_movie(N),phi=n_phi_max,theta=n_theta_max)

...

!----------
! Line 7+(N+7)
!----------

dtVphi(t=t_movie(N),phi=1,theta=1),
dtVphi(t=t_movie(N),phi=1,theta=2),
...,
dtVphi(t=t_movie(N),phi=n_phi_max,theta=n_theta_max)
```

This file can be read using *TOMovie* with the following options:

```
>>> # To load 'TO_mov.test' and time-average it:
>>> to = TOMOvie(file='TO_mov.test', avg=True, levels=65, cm='seismic')
```

8.13 Radial spectra `rB[r|p]Spec.TAG`

Note: This files are **only** written when *l_rMagSpec=.true.*

Those files contain the time-evolution of the poloidal (**rBrSpec.TAG**) and the toroidal (**rBpSpec.TAG**) magnetic energies for all radii including the inner core and for spherical harmonic degrees from $\ell = 1$ to $\ell = 6$. The calculations are done in the subroutines `rBrSpec` and `rBpSpec`, respectively. The outputs are stored as a fortran unformatted file which follows the following structure for `rBrSpec.TAG`:

```
!-------------
! Line N
!-------------

time[N],
(real(e_p(l=1,n_r),kind=outp),n_r=1,n_r_tot-1), ! Poloidal energy for \ell=0
(real(e_p(l=2,n_r),kind=outp),n_r=1,n_r_tot-1),
...
(real(e_p(l=6,n_r),kind=outp),n_r=1,n_r_tot-1)  ! Poloidal energy for \ell=6

!-------------
! Line N+1
!-------------

time[N],
```

(continues on next page)

(continued from previous page)

```
(real(e_p_ax(l=1,n_r),kind=outp),n_r=1,n_r_tot-1), ! Pol. energy for \ell=0,␣
↪m=0
(real(e_p_ax(l=2,n_r),kind=outp),n_r=1,n_r_tot-1),
...
(real(e_p_ax(l=6,n_r),kind=outp),n_r=1,n_r_tot-1)

!-------------
! Line N+2
!-------------

time[N+1]
...
```

The `rBpSpec.TAG` files have exactly the same structure (just replacing the poloidal energy by its toroidal counterpart).

Warning: Be careful that in this file, `n_r_tot` is the **total** number of grid points (thus including the inner core).

Those files can be read using the python class *MagicRSpec* with the following options:

```
>>> # Read the files BrSpec.testa, BrSpec.testb and BrSpec.testc and stack them
>>> rsp = MagicRSpec(tag='test[a-c]', field='Br')
>>> # Print time and the time evolution of e_pol(\ell=4) at the 10th radial grid point
>>> print(rsp.time, rsp.e_pol[:, 10, 3])
```

8.14 Potential files `[V|B|T|Xi]_lmr_#.TAG`

Those files contain a snapshot of either poloidal/toroidal potentials `V_lmr_#.TAG` and `B_lmr_#.TAG` or a scalar like temperature/entropy or chemical composition (`T_lmr_#.TAG` or `Xi_lmr_#.TAG`) in the radial space for all spherical harmonic degrees and orders. The detailed calculations are done in the subroutine `write_Pot`. The outputs are stored as a fortran unformatted file with a stream access. It has the following structure

```
!-------------
! Header
!-------------
version
time, ra, pr, raxi, sc, prmag, ek, radratio, sigma_ration ! Parameters
n_r_max, n_r_ic_max, l_max, minc, lm_max                  ! Truncation
omega_ic, omega_ma                                        ! Rotation rates
r(1), r(2), ..., r(n_r_max)                               ! Radius
rho0(1), rho0(2), ..., rho0(n_r_max)                      ! Background␣
↪density

!-------------
! Poloidal potential or scalar
!-------------
w(lm=1,n_r=1), w(lm=2, n_r=1), ..., w(lm=lm_max, n_r=1),
...
w(lm=1,n_r=n_r_max), ..., w(lm=lm_max,n_r=n_r_max)

!-------------
! If stored: toroidal potential
```

(continues on next page)

(continued from previous page)

```
!-------------
z(lm=1,n_r=1), z(lm=2, n_r=1), ..., z(lm=lm_max, n_r=1),
...
z(lm=1,n_r=n_r_max), ..., z(lm=lm_max,n_r=n_r_max)

!******************************************************************************!
! This last part is optional and are is written when there is                  !
! an electrically-conducting inner-core                                        !
!******************************************************************************!
b_ic(lm=1,n_r=1), b_ic(lm=2, n_r=1), ..., b_ic(lm=lm_max, n_r=1),
...
b_ic(lm=1,n_r=n_r_max), ..., b_ic(lm=lm_max,n_r=n_r_max)
aj_ic(lm=1,n_r=1), aj_ic(lm=2, n_r=1), ..., aj_ic(lm=lm_max, n_r=1),
...
aj_ic(lm=1,n_r=n_r_max), ..., aj_ic(lm=lm_max,n_r=n_r_max)
```

The potential files can be read and transformed to the physical space using the python class *MagicPotential*.

```
>>> p = MagicPotential(field='V')
>>> print(p.pol[p.idex[3,2], 32]) # print w(l=3,m=2,n_r=32)
```

Once transformed to the physical space using a Fourier and a Legendre transform, they can be displayed:

```
>>> p.equat(field='vr', cm='jet', levels=50) # Equatorial cut of vr
>>> p.avg(field='vp') # Azimuthal average of vphi
>>> p.surf(field='vt', r=0.8) # Radial cut of vtheta at r=0.8r_o
```

CHAPTER

NINE

DATA VISUALISATION AND POST-PROCESSING

Most of the *output files* written during a run of MagIC can be treated with the python post-processing classes and functions present in the `$MAGIC_HOME/python/magic` directory. These classes depend on several python libraries that can be usually found in most of the Linux distributions.

9.1 Requirements

9.1.1 Hard dependencies

- python 2.7/3.3 or higher.
- matplotlib 1.0 or higher.
- scipy 0.10 or higher.

9.1.2 Optional dependencies

- Although entirely optional, the installation of ipython makes the interactive use of the post-processing python functions much more pleasant. Installing it is therefore recommanded for a smoother interactive usage of the python functions.
- The installation of the basemap toolkit is optional. If installed, additional projections for the *`magic.Surf`* (Aitoff, orthographic, Mollweide, etc.) class will be provided for 2-D surface plotting. Otherwise, the usage of *`magic.Surf`* is limited to the standalone Hammer projection.

9.2 Configuration: `magic.cfg` file

A file named **magic.cfg** located in `$MAGIC_HOME/python/magic/magic.cfg` should have been created when you used the `source path/sourceme.sh` command for the first time on your machine. At that stage, it tried to **automatically fill the best options** that correspond to your setup. Although tested on several various machine configurations, the auto-configuration script might however fail on your setup. The paragraph below details the possible options that you may want to adjust in the `magic.cfg` file.

9.2.1 Detailed options

In case, the file `magic.cfg` doesn't exist in the directory `$MAGIC_HOME/python/magic`, you can easily copy it from the default configuration `magic.cfg.default` and then adjust the options manually:

```
$ cp $MAGIC_HOME/python/magic/magic.cfg.default $MAGIC_HOME/python/magic/magic.cfg
```

In that file, you can set up the default matplotlib rendering backend (among the possible options: `TkAgg`, `GTKAgg`, `Qt5Agg`, `Qt4Agg`, ...). The default configuration is

```
backend = TkAgg
```

Note: This is usually the default configuration which is the most likely to work on supercomputing clusters.

If LaTeX is installed on your work station, you might also want to make use of the better looking LaTeX fonts for all your displayed matplotlib figures (labels, caption, ticks, etc.). Be careful though that most of the time LaTeX is **not installed** on supercomputers. The default configuration is thus:

```
labTex = False
```

You can change the default colormap that will be used in the plotting routines.

```
defaultCm = seismic
```

You cant change the default number of contours that will be used in the plotting routines.

```
defaultLevels = 65
```

If you want to enable all the features of the python functions (faster reading the *G_#.TAG*, conversion to the VTK/VTS file format, potential extrapolation of the field lines, etc.), some fortran libraries present in the *$MAGIC_HOME/python/magic/fortranLib* directory need to be built using the f2py, which should be available on your Linux workstation if all the required python libraries have been correctly installed. The boolean `buildLib` can control whether you want to try building the fortran libraries with f2py. The following configuration will try to build the libraries:

```
buildLib = True
```

The exact name of the executable f2py however varies from one Linux distribution to the other. Among possible options, one frequently finds: `f2py`, `f2py2`, `f2py3`. This can be set to your proper configuration using the `f2pyexec` option of the `magic.cfg` file. The default configuration is:

```
f2pyexec = f2py2
```

You can also choose the fortran compiler you want to use on your machine. A list of the installed compilers can be obtained by using (where `f2py` has to be replaced by your own executable):

```
$ f2py -c --help-fcompiler
```

The most frequent options are:

- `gnu95` for the GNU gfortran compiler.
- `intelem` for the Intel ifort compiler.
- `pg` for the Portlang group pgf compiler.

Once you've decided the ideal configuration for your machine, set it up via the option `fcompiler`:

```
fcompiler = intelem
```

Finally, he same configuration procedure can be applied to the C compiler using the variable named `ccompiler`. The possible options are:

- `unix` for the GNU gcc compiler.
- `intelem` for the Intel icc compiler.

In most of the configurations, the default configuration should do a good job:

```
ccompiler = unix
```

If you encounter any problem during the building stage, you can try playing with this parameter though.

9.2.2 Ready?!

Once you think you set up your `magic.cfg` file correctly, you can test your configuration. If you decided to build the fortran libraries (i.e. `buildLib=True`), you can easily test it with any python shell by typing the following command:

```
>>> from magic import *
```

If the build was successful, it should display:

```
Please wait: building greader_single...
Please wait: building greader_double...
Please wait: building lmrreader_single...
Please wait: building Legendre transforms...
Please wait: building vtklib...
Please wait: building cylavg...
```

Once the libraries have been successfully installed, this message won't be displayed again, except if you remove the `*.so` files that are now present in the `$MAGIC_HOME/python/magic/` directory.

9.3 Python functions and classes

Once the python environment is correctly configured you can use the available functions and classes to analyse and post-process your data. The following pages will give you the detailed API of the available classes, as well as some practical examples:

Python classes

1. To read the **log.TAG** files , see *here*.
2. To read and analyse the time series, see *here*.
3. To read and analyse the radial profiles, see *here*.
4. To read and analyse spectra **_spec_#.TAG**, see *here*.
5. To read and analyse the **G_#.TAG** files, see *here*.
6. To read and analyse the **checkpoint_#.TAG** files, see *here*.

7. To read and analyse movie files **_mov.TAG**, see *here*.
8. To read and analyse coeff files **[V|B|T]_coeff.TAG**, see *here*.
9. To read and analyse potential files **[V|B|T]_lmr_#.TAG**, see *here*.
10. To read and analyse radial spectra **B[r|p]Spec.TAG**, see *here*.
11. To read and analyse TO outputs, see *here*.
12. To compare several runs simultaneously, see *here*.
13. To transform the graphic files **G_#.TAG** to a file format readable by paraview, VisIt or mayavi and do some fancy 3-D visualisation, see *here*.
14. For additional diagnostics (boundary layer, heat transport, interpolation on cylindrical grids, etc.), see *here*.
15. To take a look at the additional useful functions available (derivation, integration, interpolation, etc.), see *here*.

9.3.1 Support for the `log.TAG` files

class magic.**MagicSetup**(*datadir='.'*, *nml='input.nml'*, *quiet=False*)

This class allows to read the input namelist or the log file of a current job and creates an object that contains all the parameters found in the namelist/log file.

```
>>> stp = MagicSetup(nml='log.test', quiet=True)
>>> print(stp.ra) # print the Rayleigh number
>>> print(stp.n_r_max) # print n_r_max
```

__init__(*datadir='.'*, *nml='input.nml'*, *quiet=False*)

Parameters

- **datadir** (*str*) – the working directory
- **nml** (*str*) – name of the input namelist/ log file
- **quiet** (*bool*) – when set to True, makes the output silent (default False)

__weakref__

list of weak references to the object (if defined)

9.3.2 Support for the time series

class magic.**MagicTs**(*datadir='.'*, *field='e_kin'*, *iplot=True*, *all=False*, *tag=None*)

This python class is used to read and plot the different time series written by the code:

- Kinetic energy: *e_kin.TAG*
- Magnetic energy of the outer core: *e_mag_oc.TAG*
- Magnetic energy of the inner core: *e_mag_ic.TAG*
- Dipole information: *dipole.TAG*
- Rotation: *rot.TAG*
- Diagnostic parameters: *par.TAG*
- Geostrophy: *geos.TAG*

- Taylorization measures: *Tay.TAG*
- Heat transfer: *heat.TAG*
- Helicity: *helicity.TAG*
- Velocity square: *u_square.TAG*
- Angular momentum: *AM.TAG*
- Power budget: *power.TAG*
- Earth-likeness of the CMB field: *earth_like.TAG*
- Parallel and perpendicular decomposition: *perpPar.TAG*
- RMS force balance: *dtVrms.TAG*
- RMS induction terms: *dtBrms.TAG*
- Time-evolution of m-spectra: *am_[kin\|mag]_[pol\|tor].TAG*

Here are a couple of examples of how to use this function.

```
>>> # plot the most recent e_kin.TAG file found in the directoy
>>> MagicTs(field='e_kin')
>>>
>>> # stack **all** the power.TAG file found in the directory
>>> ts = MagicTs(field='power', all=True)
>>> print(ts.time, ts.buoPower) # print time and buoyancy power
>>>
>>> # If you only want to read the file ``heat.N0m2z``
>>> ts = MagicTs(field='heat', tag='N0m2z', iplot=False)
```

__init__(*datadir='.'*, *field='e_kin'*, *iplot=True*, *all=False*, *tag=None*)

Parameters

- **datadir** (*str*) – working directory
- **field** (*str*) – the file you want to plot
- **iplot** (*bool*) – when set to True, display the plots (default True)
- **all** (*bool*) – when set to True, the complete time series is reconstructed by stacking all the corresponding files from the working directory (default False)
- **tag** (*str*) – read the time series that exactly corresponds to the specified tag

plot()

Plotting subroutines. Only called if 'iplot=True'

9.3.3 Averaging the time series

class magic.**AvgField**(*tstart=None*, *tag=None*, *dipExtra=False*, *perpPar=False*, *std=False*)

This class calculates the time-average properties from time series. It will store the input starting time in a small file named `tInitAvg`, such that the next time you use it you don't need to give `tstart` again.

```
>>> # Average from t=2.11 and also store the additional dipole.TAG informations
>>> a = AvgField(tstart=2.11, dipExtra=True)
>>> # Average only the files that match the pattern N0m2[a-c]
>>> a = AvgField(tstart=2.11, tag='N0m2[a-c]')
>>> # Average only the files that match the pattern N0m2Z*
```

(continues on next page)

(continued from previous page)

```
>>> a = AvgField(tstart=2.11, tag='N0m2Z*')
>>> print(a) # print the formatted output
```

__init__(*tstart=None*, *tag=None*, *dipExtra=False*, *perpPar=False*, *std=False*)

Parameters

- **tstart** (*float*) – the starting time for averaging
- **tag** (*str*) – if you specify an input tag (generic regExp pattern), the averaging process will only happen on the time series that match this input pattern
- **dipExtra** (*bool*) – if this parameter is set to `True`, then additional values extracted from *dipole.TAG* are also computed
- **perpPar** (*bool*) – additional values extracted from *perpPar.TAG* are also computed

__str__()
Formatted output

__weakref__
list of weak references to the object (if defined)

9.3.4 Some resolution/convergence checks

magic.checker.**MagicCheck**(*tstart=None*)
This function is used to compute several sanity checks that can be evaluated if the power.TAG and some spectra have been produced in the current directory. If in addition the tInitAvg file is also there in the directory it averages only from this starting time.

```
>>> MagicCheck(tstart=10.)
```

9.3.5 Support for time-averaged radial profiles

class magic.**MagicRadial**(*datadir='.'*, *field='eKin'*, *iplot=True*, *tag=None*, *tags=None*, *normalize_radius=False*, *quiet=False*)
This class can be used to read and display the time and horizontally averaged files:

- Kinetic energy: *eKinR.TAG*
- Magnetic energy: *eMagR.TAG*
- Anelastic reference state: *anel.TAG*
- Variable electrical conductivity: *varCond.TAG*
- Variable thermal diffusivity: *varDiff.TAG*
- Variable kinematic viscosity: *varVisc.TAG*
- Diagnostic parameters: *parR.TAG*
- Power budget: *powerR.TAG*
- Heat fluxes: *fluxesR.TAG*
- Mean entropy, temperature and pressure: *heatR.TAG*
- Radial profiles used for boundary layers: *bLayersR.TAG*
- Parallel/perpendicular decomposition: *perpParR.TAG*

```
>>> rad = MagicRadial(field='eKinR') # display the content of eKinR.tag
>>> print(rad.radius, rad.ekin_pol_axi) # print radius and poloidal energy
```

__init__(*datadir='.'*, *field='eKin'*, *iplot=True*, *tag=None*, *tags=None*, *normalize_radius=False*, *quiet=False*)

Parameters

- **datadir** (*str*) – working directory
- **field** (*str*) – the field you want to plot
- **iplot** (*bool*) – to plot the output, default is True
- **tag** (*str*) – a specific tag, default is None
- **tags** (*list*) – a list that contains multiple tags: useful to sum several radial files
- **quiet** (*bool*) – when set to True, makes the output silent (default False)

plot()

Display the result when `iplot=True`

9.3.6 Support for the spectra files `(kin|mag|u2)_spec_#.TAG`

class magic.**MagicSpectrum**(*datadir='.'*, *field='e_kin'*, *iplot=True*, *ispec=None*, *ave=False*, *normalize=False*, *tag=None*, *quiet=False*)

This class can be used to read and display the spectra:

- Kinetic energy spectra: *kin_spec_#.TAG*
- Magnetic energy spectra: *mag_spec_#.TAG*
- Spectra of the velocity square: *u2_spec_#.TAG*

```
>>> # display the content of kin_spec_1.tag
>>> # where tag is the most recent file in the current directory
>>> sp = MagicSpectrum(field='e_kin', ispec=1)
>>> # display the content of mag_spec_ave.test on one single figure
>>> sp = MagicSpectrum(field='e_mag', tag='test', ave=True)
```

__init__(*datadir='.'*, *field='e_kin'*, *iplot=True*, *ispec=None*, *ave=False*, *normalize=False*, *tag=None*, *quiet=False*)

Parameters

- **field** (*str*) – the spectrum you want to plot, 'e_kin' for kinetic energy, 'e_mag' for magnetic
- **iplot** (*bool*) – display the output plot when set to True (default is True)
- **ispec** (*int*) – the number of the spectrum you want to plot
- **tag** (*str*) – file suffix (tag), if not specified the most recent one in the current directory is chosen
- **ave** (*bool*) – plot a time-averaged spectrum when set to True
- **datadir** (*str*) – current working directory
- **quiet** (*bool*) – when set to True, makes the output silent (default False)

plot()

Plotting function

9.3.7 Support for the 2-D spectra files

class magic.**MagicSpectrum2D**(*datadir='.'*, *field='e_mag'*, *iplot=False*, *ispec=None*, *tag=None*, *cm='jet'*, *levels=33*, *precision=<class 'numpy.float64'>*, *ave=False*)

This class can be used to read and display 2-D spectra in the (r, ℓ) and in the (r, m) planes

- Kinetic energy spectra: *2D_kin_spec_#.TAG*
- Magnetic energy spectra: *2D_mag_spec_#.TAG*

```
>>> # display the content of 2D_kin_spec_1.tag
>>> # where tag is the most recent file in the current directory
>>> sp = MagicSpectrum2D(field='e_kin', ispec=1, levels=17, cm='seismic')
>>> # display the content of 2D_mag_spec_3.test
>>> sp = MagicSpectrum2D(field='e_mag', tag='test', ispec=3)
```

__init__(*datadir='.'*, *field='e_mag'*, *iplot=False*, *ispec=None*, *tag=None*, *cm='jet'*, *levels=33*, *precision=<class 'numpy.float64'>*, *ave=False*)

Parameters

- **field** (*str*) – the spectrum you want to plot, 'e_kin' for kinetic energy, 'e_mag' for magnetic
- **iplot** (*bool*) – display the output when set to True (default is True)
- **ispec** (*int*) – the number of the spectrum you want to plot
- **tag** (*str*) – file suffix (tag=, if not specified the most recent one in the current directory is chosen
- **cm** (*str*) – name of the colormap (default='jet')
- **levels** (*int*) – number of contour levels (default 33)
- **precision** (*str*) – single or double precision
- **datadir** (*str*) – current working directory
- **ave** (*bool*) – plot a time-averaged spectrum when set to True

plot(*levels*, *cm*, *cut=1.0*)

Plotting function

Parameters

- **levels** (*int*) – number of contour levels
- **cm** – name of the colormap
- **cut** (*float*) – adjust the contour maximum to max(abs(data))*cut

9.3.8 Support for G_#.TAG files

class magic.**MagicGraph**(*ivar=None*, *datadir='.'*, *quiet=True*, *ave=False*, *tag=None*, *precision=<class 'numpy.float32'>*)

This class allows to read the 3-D graphic outputs of the MagIC code (*G_#.TAG* and G_ave.TAG) files. Those are binary unformatted outputs, there are therefore two ways to load them:

- If buildLib=True in magic.cfg and the fortran libraries were correctly built, then the reader uses a fortran program that is expected to be much faster than the pure python routine.
- If buildLib=False, then a pure python program is used to read the G files.

```
>>> # Regular G files
>>> gr = MagicGraph(ivar=1, tag='N0m2a')
>>> print(gr.vr.shape) # shape of vr
>>> print(gr.ek) # print ekman number
>>> print(gr.minc) # azimuthal symmetry
>>> # Averaged G file with double precision
>>> gr = MagicGraph(ave=True, tag='N0m2', precision=np.float64)
```

__init__(*ivar=None*, *datadir='.'*, *quiet=True*, *ave=False*, *tag=None*, *precision=<class 'numpy.float32'>*)

Parameters

- **ave** (*bool*) – when set to True, it tries to find an average G file (G_ave.TAG)
- **ivar** (*int*) – the number of the G file
- **tag** (*str*) – extension TAG of the G file. If not specified, the most recent G_#.TAG file found in the directory will be selected.
- **quiet** (*bool*) – when set to True, makes the output silent
- **datadir** (*str*) – directory of the G file (default is .)
- **precision** (*str*) – single or double precision (default np.float32)

read_record_marker(*filename*, *endian*)

This function is used to read a Graphic file that contains record markers.

Parameters

- **filename** (*str*) – name of the graphic file
- **endian** (*str*) – endianness of the file

read_stream(*filename*, *endian*)

This function is used to read a Graphic file that has no record marker.

Parameters

- **filename** (*str*) – name of the graphic file
- **endian** (*str*) – endianness of the file

rearangeLat(*field*)

This function is used to unfold the colatitudes

Parameters **field** (*numpy.ndarray*) – input array with MagIC ordering of colatitudes (i.e. successively Northern Hemisphere and Southern Hemisphere)

Returns an array with the regular ordering of the colatitudes

Return type numpy.ndarray

class magic.**Surf**(*ivar=None*, *datadir='.'*, *vort=False*, *ave=False*, *tag=None*, *precision=<class 'numpy.float32'>*)

This class allows to display the content of a graphic file (*G_#.TAG* or G_ave.TAG). It allows to plot radial, azimuthal and equatorial cuts as well as phi-averages.

```
>>> # To read G_1.test
>>> s = Surf(ivar=1, ave=False, tag='test')
>>> # To read the latest G file in the working directory (double precision)
>>> s = Surf(precision=np.float64)
```

```
>>> # Possible plots
>>> s.equat(field='vr')
>>> s.avg(field='vp')
>>> s.surf(field='entropy', r=0.8)
>>> s.slice(field='Br', lon_0=[0, 30])
```

__init__(*ivar=None*, *datadir='.'*, *vort=False*, *ave=False*, *tag=None*, *precision=<class 'numpy.float32'>*)

Parameters

- **ivar** (*int*) – index of the graphic file
- **ave** (*bool*) – when set to True, it tries to read a time-averaged graphic file
- **tag** (*str*) – TAG suffix extension of the graphic file
- **vort** (*bool*) – a boolean to specify whether one wants to compute the 3-D vorticiy components (take care of the memory imprint)
- **datadir** (*str*) – the working directory
- **precision** (*str*) – the storage precision of the graphic file (single or double precision). Default is np.float32 (single)

__weakref__
list of weak references to the object (if defined)

avg(*field='vphi'*, *levels=65*, *cm='seismic'*, *normed=True*, *vmax=None*, *vmin=None*, *cbar=True*, *tit=True*, *pol=False*, *tor=False*, *mer=False*, *merLevels=16*, *polLevels=16*, *ic=False*)
Plot the azimutal average of a given field.

```
>>> s = Surf()
>>> # Axisymmetric zonal flows, 65 contour levels
>>> s.avg(field='vp', levels=65, cm='seismic')
```

```
>>> # Minimal plot (no cbar, not title)
>>> s.avg(field='Br', tit=False, cbar=False)
```

```
>>> # Axisymmetric Bphi + poloidal field lines
>>> s.avg(field='Bp', pol=True, polLevels=8)
```

```
>>> # Omega-effect, contours truncated from -1e3 to 1e3
>>> s.avg(field='omeffect', vmax=1e3, vmin=-1e3)
```

Parameters

- **field** (*str*) – the field you want to display
- **levels** (*int*) – the number of levels in the contourf plot
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels

- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.
- **pol** (*bool*) – diplay the poloidal field lines contours when set to True
- **tor** (*bool*) – diplay the toroidal axisymmetric field contours when set to True
- **mer** (*bool*) – display the meridional circulation contours when set to True
- **merLevels** (*int*) – number of contour levels to display meridional circulation
- **polLevels** (*int*) – number of contour levels to display poloidal field lines
- **ic** (*bool*) – when set to True, also display the contour levels in the inner core

equat (*field='vr'*, *levels=65*, *cm='seismic'*, *normed=True*, *vmax=None*, *vmin=None*, *cbar=True*, *tit=True*, *avg=False*, *normRad=False*, *ic=False*)
Plot the equatorial cut of a given field

```
>>> s = Surf()
>>> # Equatorial cut of the z-vorticity, 65 contour levels
>>> s.equat(field='vortz', levels=65, cm='seismic')
```

```
>>> # Minimal plot (no cbar, not title)
>>> s.equat(field='bphi', tit=False, cbar=False)
```

```
>>> # Control the limit of the colormap from -1e3 to 1e3
>>> s.equat(field='vr', vmin=-1e3, vmax=1e3, levels=33)
```

```
>>> # Normalise the contour levels radius by radius
>>> s.equat(field='jphi', normRad=True)
```

Parameters

- **field** (*str*) – the name of the input physical quantity you want to display
- **avg** (*bool*) – when set to True, an additional figure which shows the radial profile of the input physical quantity (azimuthal average) is also displayed
- **normRad** (*bool*) – when set to True, the contour levels are normalised radius by radius (default is False)
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.
- **ic** (*bool*) – when set to True, also display the contour levels in the inner core

slice (*field='Bphi'*, *lon_0=0.0*, *levels=65*, *cm='seismic'*, *normed=True*, *vmin=None*, *vmax=None*, *cbar=True*, *tit=True*, *grid=False*, *nGridLevs=16*, *normRad=False*, *ic=False*)
Plot an azimuthal slice of a given field.

```
>>> s = Surf()
>>> # vphi at 0, 30, 60 degrees in longitude
>>> s.slice(field='vp', lon_0=[0, 30, 60], levels=65, cm='seismic')
```

```
>>> # Minimal plot (no cbar, not title)
>>> s.avg(field='vp', lon_0=32, tit=False, cbar=False)
```

```
>>> # Axisymmetric Bphi + poloidal field lines
>>> s.avg(field='Bp', pol=True, polLevels=8)
```

```
>>> # Omega-effect, contours truncated from -1e3 to 1e3
>>> s.avg(field='omeffect', vmax=1e3, vmin=-1e3)
```

Parameters

- **field** (*str*) – the field you want to display
- **lon_0** (*float or list*) – the longitude of the slice in degrees, or a list of longitudes
- **levels** (*int*) – the number of levels in the contourf plot
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **grid** (*bool*) – display or hide the grid
- **nGridLevs** (*int*) – number of grid levels
- **normRad** (*bool*) – when set to True, the contour levels are normalised radius by radius (default is False)
- **ic** (*bool*) – when set to True, also display the contour levels in the inner core

surf (*field='Bphi'*, *proj='hammer'*, *lon_0=0.0*, *r=0.85*, *vmax=None*, *vmin=None*, *lat_0=30.0*, *levels=65*, *cm='seismic'*, *ic=False*, *lon_shift=0*, *normed=True*, *cbar=True*, *tit=True*, *lines=False*)

Plot the surface distribution of an input field at a given input radius (normalised by the outer boundary radius).

```
>>> s = Surf()
>>> # Radial flow component at ``r=0.95 r_o``, 65 contour levels
>>> s.surf(field='vr', r=0.95, levels=65, cm='seismic')
```

```
>>> # Minimal plot (no cbar, not title)
>>> s.surf(field='entropyfluct', r=0.6, tit=False, cbar=False)
```

```
>>> # Control the limit of the colormap from -1e3 to 1e3
>>> s.surf(field='vp', r=1., vmin=-1e3, vmax=1e3, levels=33)
```

```
>>> # If basemap is installed, additional projections are available
>>> s.surf(field='Br', r=0.95, proj='ortho', lat_0=45, lon_0=45)
```

Parameters

- **field** (*str*) – the name of the field you want to display
- **proj** (*str*) – the type of projection. Default is Hammer, in case you want to use 'ortho' or 'moll', then Basemap is required.
- **r** (*float*) – the radius at which you want to display the input data (in normalised units with the radius of the outer boundary)
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **lon_shift** (*int*) – translate map in azimuth (in degrees)
- **lon_0** (*float*) – central azimuth (only used with Basemap)
- **lat_0** (*float*) – central latitude (only used with Basemap)
- **tit** (*bool*) – display the title of the figure when set to True
- **tit** – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **lines** (*bool*) – when set to True, over-plot solid lines to highlight the limits between two adjacent contour levels
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.

9.3.9 Support for checkpoint_#.TAG files

magic.checkpoint.**Graph2Rst**(*gr*, *filename='checkpoint_ave'*)

This function allows to transform an input Graphic file into a checkpoint file format that can be read by MagIC to restart a simulation.

```
>>> # Load a Graphic File
>>> gr = MagicGraph()
>>> # Produce the file checkpoint_ave.from_G
>>> Graph2Rst(gr, filename='checkpoint_ave.from_G')
```

Parameters

- **gr** (magic.MagicGraph) – the input graphic file one wants to convert into a restart file
- **filename** (*str*) – name of the checkpoint file

class magic.checkpoint.**MagicCheckpoint**(*l_read=True*, *filename=None*)

This class allows to manipulate checkpoint files produced by MagIC. It can read it as

```
>>> chk = MagicCheckpoint(filename='checkpoint_end.test')
>>> print(chk.wpol.shape, chk.l_max)
```

This class can also be used to intepolate from FD to Cheb or the opposite >>> chk.cheb2fd(96) >>> chk.write('checkpoint_fd.test')

One can also transform a Graphic file into a checkpoint >>> gr = MagicGraph() >>> chk = MagicCheckpoint(l_read=False) >>> chk.graph2rst(gr)

Finally one can convert checkpoints from XSHELLS >>> chk = MagicCheckpoint(l_read=False) >>> chk.xshells2magic('st0', 161, rscheme='cheb', cond_state='deltaT')

__init__(*l_read=True*, *filename=None*)

Parameters

- **l_read** (*bool*) – a boolean to decide whether one reads a checkpoint or not
- **filename** (*str*) – name of the checkpoint file to be read

__weakref__
list of weak references to the object (if defined)

cheb2fd(*n_r_max*, *fd_stretch=0.3*, *fd_ratio=0.1*)
This routine is used to convert a checkpoint that has a Gauss-Lobatto grid into a finite-difference grid.

Parameters

- **n_r_max** (*int*) – number of radial grid points of the finite difference grid
- **fd_stretch** (*float*) – stretching of the radial grid
- **fd_ratio** (*float*) – ratio of smallest to largest grid spacing

fd2cheb(*n_r_max*)
This routine is used to convert a checkpoint that has finite differences in radius into a Gauss-Lobatto grid.

Parameters **n_r_max** (*int*) – number of radial grid points of the Gauss-Lobatto grid

graph2rst(*gr*, *filename='checkpoint_ave.from_chk'*)

Parameters

- **gr** (magic.MagicGraph) – the input graphic file one wants to convert into a restart file
- **filename** (*str*) – name of the checkpoint file

read(*filename*)
This routine is used to read a checkpoint file.

Parameters **filename** (*str*) – name of the checkpoint file

write(*filename*)
This routine is used to store a checkpoint file. It only stores the state vector not the past quantities required to restart a multistep scheme.

Parameters **filename** (*str*) – name of the checkpoint file

xshells2magic(*xsh_trailing*, *n_r_max*, *rscheme='cheb'*, *cond_state='deltaT'*, *scale_b=1.0*, *filename='checkpoint_end.from_xhells'*)
This routine is used to convert XSHELLS field[U,B,T].xsh_trailing files into a MagIC checkpoint file.

```
>>> chk = MagicCheckPoint()
>>> # Convert field[U,T,B].st1ns_hr2 into a MagIC checkpoint file
>>> chk.xshells2magic('st1ns_hr2', 512, rscheme='fd', cond_state='mixed',
                      scale_b=4.472136e-4)
```

Parameters

- **xsh_trailing** (*str*) – trailing of the field[U,B,T].xsh_trailing files
- **n_r_max** (*int*) – number of radial grid points to be used
- **rscheme** (*str*) – the type of radial scheme ('cheb' or 'fd')

- **cond_state** (*str*) – the type of conducting state: - 'deltaT': fixed temperature contrast - 'mixed': hybrid forcing (STEP1-2 like)
- **scale_b** (*float*) – a rescaling factor for the magnetic field

magic.checkpoint.**get_map**(*lm_max*, *lmax*, *mmax*, *minc*)
This routine determines the look-up tables to convert the indices (l, m) to the single index lm.

Parameters

- **lm_max** (*int*) – total number of lm combinations.
- **lmax** (*int*) – maximum spherical harmonic degree
- **mmax** (*int*) – maximum spherical harmonic order
- **minc** (*int*) – azimuthal symmetry

Returns returns a list of three look-up tables: idx, lm2l, lm2m

Return type list

magic.checkpoint.**get_truncation**(*n_theta_max*, *nalias*, *minc*)
This routine determines l_max, m_max and lm_max from the values of n_theta_max, minc and nalias.

Parameters

- **n_theta_max** (*int*) – number of points along the colatitude
- **nalias** (*int*) – dealiasing paramete (20 is fully dealiased)
- **minc** (*int*) – azimuthal symmetry

Returns returns a list of three integers: l_max, m_max and lm_max

Return type list

magic.checkpoint.**interp_one_field**(*field*, *rold*, *rnew*, *rfac=None*)
This routine interpolates a complex input field from an old radial grid to a new one.

Parameters

- **field** (*numpy.ndarray*) – the field to be interpolated
- **rold** (*numpy.ndarray*) – the old radial grid points
- **rnew** (*numpy.ndarray*) – the new radial grid points
- **rfac** (*numpy.ndarray*) – a rescaling function that depends on the radius

Returns the field interpolated on the new radial grid

Return type numpy.ndarray

9.3.10 Support for movie files *_mov.TAG

class magic.**Movie**(*file=None*, *iplot=True*, *step=1*, *png=False*, *lastvar=None*, *nvar='all'*, *levels=12*, *cm='RdYlBu_r'*, *cut=0.5*, *bgcolor=None*, *fluct=False*, *normed=False*, *avg=False*, *std=False*, *dpi=80*, *normRad=False*, *precision=<class 'numpy.float32'>*, *deminc=True*, *ifield=0*)
This class allows to read the *movie files* generated by the MagIC code.

```
>>> m = Movie()
>>> # This returns a list of the available movies in the directory
>>> # and lets you decide which one you want to read
```

```
>>> # Reads and display AV_mov.test
>>> m = Movie(filed='AV_mov.test')
>>> print(m.data) # access to the data
```

```
>>> # Read three movie files (no display)
>>> m1 = Movie(file='AV_mov.testa', iplot=False)
>>> m2 = Movie(file='AV_mov.testb', iplot=False)
>>> m3 = Movie(file='AV_mov.testc', iplot=False)
>>> # Stack them together
>>> m = m1+m2+m3
>>> # Display
>>> m.plot(levels=33, cm='seismic', cut=0.5)
```

```
>>> # Store the outputs in movie/img_#.png
>>> # Only from the timesteps 280 to 380
>>> m = Movie(file='AB_mov.test', png=True, nvar=100, lastvar=380)
```

__add__(*new*)
Built-in function to sum two movies

Note: So far this function only works for two movies with the same grid sizes. At some point, we might introduce grid extrapolation to allow any summation.

__init__(*file=None*, *iplot=True*, *step=1*, *png=False*, *lastvar=None*, *nvar='all'*, *levels=12*, *cm='RdYlBu_r'*, *cut=0.5*, *bgcolor=None*, *fluct=False*, *normed=False*, *avg=False*, *std=False*, *dpi=80*, *normRad=False*, *precision=<class 'numpy.float32'>*, *deminc=True*, *ifield=0*)

Parameters

- **nvar** (*int*) – the number of timesteps of the movie file we want to plot starting from the last line
- **png** (*bool*) – if png=True, write the png files instead of display
- **iplot** (*bool*) – if iplot=True, display otherwise just read
- **lastvar** (*int*) – the number of the last timesteps to be read
- **step** (*int*) – the stepping between two timesteps
- **levels** (*int*) – the number of contour levels
- **cm** (*str*) – the name of the color map
- **fluct** (*bool*) – if fluct=True, substract the axisymmetric part
- **normed** (*bool*) – the colormap is rescaled every timestep when set to True, otherwise it is calculated from the global extrema
- **avg** (*bool*) – if avg=True, time-average is displayed
- **std** (*bool*) – if std=True, standard deviation is displayed
- **dpi** (*int*) – dot per inch when saving PNGs
- **normRad** (*bool*) – if normRad=True, then we normalise for each radial level
- **precision** (*str*) – precision of the input file, np.float32 for single precision, np.float64 for double precision

- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut
- **bgcolor** (*str*) – background color of the figure
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry
- **ifield** (*int*) – in case of a multiple-field movie file, you can change the default field displayed using the parameter ifield

__weakref__
list of weak references to the object (if defined)

avgStd(*ifield=0*, *std=False*, *cut=0.5*, *levels=12*, *cmap='RdYlBu_r'*, *ic=False*)
Plot time-average or standard deviation

Parameters

- **ifield** (*int*) – in case of a multiple-field movie file, you can change the default field displayed using the parameter ifield
- **std** (*bool*) – the standard deviation is computed instead the average when std is True
- **levels** (*int*) – number of contour levels
- **cmap** (*str*) – name of the colormap
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut

plot(*ifield=0*, *cut=0.5*, *levels=12*, *cmap='RdYlBu_r'*, *png=False*, *step=1*, *normed=False*, *dpi=80*, *bgcolor=None*, *deminc=True*, *ic=False*)
Plotting function (it can also write the png files)

Parameters

- **ifield** (*int*) – in case of a multiple-field movie file, you can change the default field displayed using the parameter ifield
- **levels** (*int*) – number of contour levels
- **cmap** (*str*) – name of the colormap
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut
- **png** (*bool*) – save the movie as a series of png files when set to True
- **dpi** (*int*) – dot per inch when saving PNGs
- **bgcolor** (*str*) – background color of the figure
- **normed** (*bool*) – the colormap is rescaled every timestep when set to True, otherwise it is calculated from the global extrema
- **step** (*int*) – the stepping between two timesteps
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry

timeLongitude(*ifield=0*, *removeMean=True*, *lat0=0.0*, *levels=12*, *cm='RdYlBu_r'*, *deminc=True*)
Plot the time-longitude diagram (input latitude can be chosen)

Parameters

- **ifield** (*int*) – in case of a multiple-field movie file, you can change the default field displayed using the parameter ifield
- **lat0** (*float*) – value of the latitude

- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the colormap
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry
- **removeMean** (*bool*) – remove the time-averaged part when set to True

class magic.**Movie3D**(*file=None*, *step=1*, *lastvar=None*, *nvar='all'*, *nrout=48*, *ratio_out=2.0*, *potExtra=False*, *precision=<class 'numpy.float32'>*)

This class allows to read the 3D movie files *(B|V)_3D_.TAG* and transform them into a series of VTS files `./vtsFiles/B3D_#.TAG` that can be further read using paraview.

```
>>> Movie3D(file='B_3D.TAG')
```

__init__(*file=None*, *step=1*, *lastvar=None*, *nvar='all'*, *nrout=48*, *ratio_out=2.0*, *potExtra=False*, *precision=<class 'numpy.float32'>*)

Parameters

- **file** (*str*) – file name
- **nvar** (*int*) – the number of timesteps of the movie file we want to plot starting from the last line
- **lastvar** (*int*) – the number of the last timestep to be read
- **step** (*int*) – the stepping between two timesteps
- **precision** (*str*) – precision of the input file, np.float32 for single precision, np.float64 for double precision
- **potExtra** (*bool*) – when set to True, potential extrapolation of the magnetic field outside the fluid domain is also computed
- **ratio_out** (*float*) – ratio of desired external radius to the CMB radius. This is is only used when potExtra=True
- **nrout** (*int*) – number of additional radial grid points to compute the potential extrapolation. This is only used when potExtra=True

__weakref__

list of weak references to the object (if defined)

9.3.11 Support for `B_cmb_coeff.TAG` and `(V|B)_coeff_r#.TAG` files

class magic.coeff.**MagicCoeffCmb**(*tag=None*, *datadir='.'*, *ratio_cmb_surface=1*, *scale_b=1*, *iplot=True*, *lCut=None*, *precision=<class 'numpy.float64'>*, *ave=False*, *sv=False*, *quiet=False*)

This class allows to read the *B_coeff_cmb.TAG* files. It first read the poloidal potential at the CMB and then transform it to the Gauss coefficients $g_{\ell m}$ and $h_{\ell m}$ using the getGauss function.

```
>>> # Reads the files B_coeff_cmb.testa, B_coeff_cmb.testb
>>> # and B_coeff_cmb.testc and stack them in one single time series
>>> cmb = MagicCoeffCmb(tag='test[a-c]')
>>> print(cmb.ell, cmb.glm) # print \ell and g_{\ell m}
>>> print(cmb.glm[:, cmb.idx[1, 0]]) # time-series of the axisymmetric dipole
>>> plot(cmb.time, cmb.dglmdt[:, cmb.idx[2, 0]]) # Secular variation of the␣
↪quadrupole
>>> # Display the time-evolution of the CMB field
```

(continues on next page)

(continued from previous page)

```
>>> cmb.movieCmb(levels=12, cm='seismic')
>>> # Save the time-evolution of the CMB field
>>> cmb.movieCmb(levels=12, cm='seismic', png=True)
```

__add__(*new*)
Built-in function to sum two cmb files

Note: So far this function only works for two cmb files with the same grid sizes. At some point, we might introduce grid extrapolation to allow any summation/

__init__(*tag=None*, *datadir='.'*, *ratio_cmb_surface=1*, *scale_b=1*, *iplot=True*, *lCut=None*, *precision=<class 'numpy.float64'>*, *ave=False*, *sv=False*, *quiet=False*)
A class to read the B_coeff_cmb files

Parameters

- **tag** (*str*) – if you specify a pattern, it tries to read the corresponding files
- **ratio_cmb_surface** (*float*) – ratio of surface ratio to CMB radius (default is 1)
- **scale_b** (*float*) – magnetic field unit (default is 1)
- **iplot** (*int*) – a logical to toggle the plot (default is True)
- **precision** (*char*) – single or double precision
- **ave** (*bool*) – load a time-averaged CMB file when set to True
- **sv** (*bool*) – load a dt_b CMB file when set to True
- **quiet** (*bool*) – verbose when toggled to True (default is True)
- **lCut** (*int*) – reduce the spherical harmonic truncation to l <= lCut
- **datadir** (*str*) – working directory

movieCmb(*cut=0.5*, *levels=12*, *cm='RdYlBu_r'*, *png=False*, *step=1*, *normed=False*, *dpi=80*, *bgcolor=None*, *deminc=True*, *removeMean=False*, *precision=<class 'numpy.float64'>*, *contour=False*, *mer=False*)
Plotting function (it can also write the png files)

Parameters

- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the colormap
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut
- **png** (*bool*) – save the movie as a series of png files when set to True
- **dpi** (*int*) – dot per inch when saving PNGs
- **bgcolor** (*str*) – background color of the figure
- **normed** (*bool*) – the colormap is rescaled every timestep when set to True, otherwise it is calculated from the global extrema
- **step** (*int*) – the stepping between two timesteps
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry
- **precision** (*char*) – single or double precision

- **contour** (*bool*) – also display the solid contour levels when set to True
- **mer** (*bool*) – display meridians and circles when set to True
- **removeMean** (*bool*) – remove the time-averaged part when set to True

plot()
Display some results when iplot is set to True

timeLongitude(*removeMean=True*, *lat0=0.0*, *levels=12*, *cm='RdYlBu_r'*, *deminc=True*, *shtns_lib='shtns'*)
Plot the time-longitude diagram of Br (input latitude can be chosen)

> **Warning:** the python bindings of SHTns are mandatory to use this plotting function!

Parameters

- **lat0** (*float*) – value of the latitude
- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the colormap
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry
- **shtns_lib** (*char*) – version of shtns library used: can be either 'shtns' or 'shtns-magic'
- **removeMean** (*bool*) – remove the time-averaged part when set to True

truncate(*lCut*)

Parameters **lCut** (*int*) – truncate to spherical harmonic degree lCut

class magic.coeff.**MagicCoeffR**(*tag*, *ratio_cmb_surface=1*, *scale_b=1*, *iplot=True*, *field='B'*, *r=1*, *precision=<class 'numpy.float64'>*, *lCut=None*, *quiet=False*)
This class allows to read the *B_coeff_r#.TAG* and *V_coeff_r#.TAG* files. It reads the poloidal and toroidal potentials and reconstruct the time series (or the energy) contained in any given mode.

```
>>> # Reads the files V_coeff_r2.test*
>>> cr = MagicCoeffR(tag='test*', field='V', r=2)
>>> print(cr.ell, cr.wlm) # print \ell and w_{\ell m}
>>> # Time-evolution of the poloidal energy in the (\ell=10, m=10) mode
>>> plot(cr.time, cr.epolLM[:, cr.idx[10, 10]])
```

__init__(*tag*, *ratio_cmb_surface=1*, *scale_b=1*, *iplot=True*, *field='B'*, *r=1*, *precision=<class 'numpy.float64'>*, *lCut=None*, *quiet=False*)

Parameters

- **tag** (*str*) – if you specify a pattern, it tries to read the corresponding files
- **ratio_cmb_surface** (*float*) – ratio of surface ratio to CMB radius (default is 1)
- **scale_b** (*float*) – magnetic field unit (default is 1)
- **iplot** (*bool*) – a logical to toggle the plot (default is True)
- **field** (*str*) – 'B', 'V' or 'T' (magnetic field, velocity field or temperature)
- **r** (*int*) – an integer to characterise which file we want to plot
- **precision** (*str*) – single or double precision

- **lCut** (*int*) – reduce the spherical harmonic truncation to l <= lCut
- **quiet** (*bool*) – verbose when toggled to True (default is True)

fft()
Fourier transform of the poloidal energy

movieRad(*cut=0.5*, *levels=12*, *cm='RdYlBu_r'*, *png=False*, *step=1*, *normed=False*, *dpi=80*, *bgcolor=None*, *deminc=True*, *removeMean=False*, *precision=<class 'numpy.float64'>*, *contour=False*, *mer=False*)
Plotting function (it can also write the png files)

> **Warning:** the python bindings of SHTns are mandatory to use this plotting function!

Parameters

- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the colormap
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut
- **png** (*bool*) – save the movie as a series of png files when set to True
- **dpi** (*int*) – dot per inch when saving PNGs
- **bgcolor** (*str*) – background color of the figure
- **normed** (*bool*) – the colormap is rescaled every timestep when set to True, otherwise it is calculated from the global extrema
- **step** (*int*) – the stepping between two timesteps
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry
- **precision** (*char*) – single or double precision
- **contour** (*bool*) – also display the solid contour levels when set to True
- **mer** (*bool*) – display meridians and circles when set to True
- **removeMean** (*bool*) – remove the time-averaged part when set to True

truncate(*lCut*, *field='B'*)

Parameters

- **lCut** (*int*) – truncate to spherical harmonic degree lCut
- **field** (*char*) – name of the field ('V', 'B' or 'T')

magic.coeff.**deriv**(*x*, *y*, *axis=0*)
This function is a simple second order derivative

Parameters

- **x** (*numpy.ndarray*) – input x-axis
- **y** (*numpy.ndarray*) – input array

Returns an array that contains the derivatives

Return type numpy.ndarray

magic.coeff.**getGauss**(*alm*, *blm*, *ell*, *m*, *scale_b*, *ratio_cmb_surface*, *rcmb*)
Get the Gauss coefficients from the real and imaginary parts of the poloidal potential

Parameters

- **alm** (*numpy.ndarray*) – real part of the poloidal potential
- **blm** (*numpy.ndarray*) – imaginary part of the poloidal potential
- **ell** (*numpy.ndarray*) – spherical harmonic degree ell
- **scale_b** (*float*) – magnetic field unit (default is 1)
- **ratio_cmb_surface** (*float*) – ratio of surface ratio to CMB radius (default is 1)
- **rcmb** (*float*) – radius of the outer boundary

magic.coeff.**rearrangeLat**(*field*)
This function is used to unfold the colatitudes

Parameters **field** (*numpy.ndarray*) – input array with MagIC ordering of colatitudes (i.e. successively Northern Hemisphere and Southern Hemisphere)

Returns an array with the regular ordering of the colatitudes

Return type numpy.ndarray

9.3.12 Support for B[rp]Spec.TAG

class magic.**MagicRSpec**(*tag*, *field='Br'*, *precision=<class 'numpy.float32'>*, *avg=False*)
This class allows to read the *rB[r|p]Spec.TAG files*. Those files contain the time-evolution of the poloidal/toroidal magnetic energy for all radii and for spherical harmonic degrees from 1 to 6. This is an unformatted fortran file.

```
>>> # Read all the `BrSpec.test*` files in the current working directory and
>>> # stack them.
>>> rsp = MagicRSpec(tag='test*', field='Br')
```

__init__(*tag*, *field='Br'*, *precision=<class 'numpy.float32'>*, *avg=False*)

Parameters

- **tag** (*str*) – if you specify a pattern, it tries to read the corresponding files and stack them.
- **field** (*str*) – nature of the radial spectra. Possible choices are 'Bt' or 'Bp'
- **precision** (*str*) – single or double precision (default single, i.e. np.float32)
- **avg** (*bool*) – when set to True, display time averaged quantities

plotAvg()
Plotting function for time-averaged profiles

9.3.13 Support for `[V|B|T]_lmr_#.TAG`

class magic.**MagicPotential**(*field='V'*, *datadir='.'*, *tag=None*, *ave=False*, *ipot=None*, *precision=<class 'numpy.float32'>*, *verbose=True*, *ic=False*)

This class allows to load and display the content of the potential files: *V_lmr.TAG*, *B_lmr.TAG* and *T_lmr.TAG*. This class allows to transform the poloidal/toroidal potential in spectral space to the physical quantities in the physical space. It allows to plot radial and equatorial cuts as well as phi-averages.

```
>>> # To read T_lmr.test
>>> p = MagicPotential(field='T', ipot=1, tag='test')
>>> # To read the latest V_lmr file in the working directory
>>> p = MagicPotential(field='V')
>>> # Get the poloidal potential (lm, nR)
>>> wlm = p.pol
>>> # Obtain the value of w(l=12, m=12, nR=33)
>>> print( p.pol[p.idx[12,12], 32] )
```

```
>>> # Possible plots
>>> p.equat(field='vr')
>>> p.avg(field='vp')
>>> p.surf(field='vt', r=0.8)
```

__init__(*field='V'*, *datadir='.'*, *tag=None*, *ave=False*, *ipot=None*, *precision=<class 'numpy.float32'>*, *verbose=True*, *ic=False*)

Parameters

- **field** (*str*) – 'B', 'V', 'T' or 'Xi' (magnetic field, velocity field, temperature or chemical composition)
- **datadir** (*str*) – the working directory
- **tag** (*str*) – if you specify a pattern, it tries to read the corresponding files
- **ave** (*bool*) – plot a time-averaged spectrum when set to True
- **ipot** (*int*) – the number of the lmr file you want to plot
- **precision** (*str*) – single or double precision
- **verbose** (*bool*) – some info about the SHT layout
- **ic** (*bool*) – read or don't read the inner core

avg(*field='vphi'*, *levels=65*, *cm='seismic'*, *normed=True*, *vmax=None*, *vmin=None*, *cbar=True*, *tit=True*)

Plot the azimutal average of a given field.

```
>>> p = MagicPotential(field='V')
>>> # Axisymmetric zonal flows, 65 contour levels
>>> p.avg(field='vp', levels=65, cm='seismic')
```

```
>>> # Minimal plot (no cbar, not title)
>>> p.avg(field='vr', tit=False, cbar=False)
```

Parameters

- **field** (*str*) – the field you want to display
- **levels** (*int*) – the number of levels in the contourf plot
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)

- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.

equat (*field='vr'*, *levels=65*, *cm='seismic'*, *normed=True*, *vmax=None*, *vmin=None*, *cbar=True*, *tit=True*, *normRad=False*)
Plot the equatorial cut of a given field

```
>>> p = MagicPotential(field='B')
>>> # Equatorial cut of the Br
>>> p.equat(field='Br')
```

```
>>> # Normalise the contour levels radius by radius
>>> p.equat(field='Bphi', normRad=True)
```

Parameters

- **field** (*str*) – the name of the input physical quantity you want to display
- **normRad** (*bool*) – when set to True, the contour levels are normalised radius by radius (default is False)
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.

read (*filename*, *field*, *endian*, *record_marker*, *ic=False*, *precision=<class 'numpy.float32'>*)
This routine defines a reader for the various versions of the lmr files.

Parameters

- **filename** (*str*) – name of the input lmr file
- **field** (*str*) – 'B', 'V', 'T' or 'Xi' (magnetic field, velocity field, temperature or chemical composition)
- **endian** (*str*) – a character string that specifies the endianness of the input file ('B' for big endian or 'l' for little endian)
- **record_marker** (*bool*) – a boolean to specify whether the file contains record marker
- **ic** (*bool*) – read or don't read the inner core
- **precision** (*str*) – single or double precision

surf (*field='vr'*, *proj='hammer'*, *lon_0=0.0*, *r=0.85*, *vmax=None*, *vmin=None*, *lat_0=30.0*, *levels=65*, *cm='seismic'*, *lon_shift=0*, *normed=True*, *cbar=True*, *tit=True*, *lines=False*)

Plot the surface distribution of an input field at a given input radius (normalised by the outer boundary radius).

```
>>> p = MagicPotential(field='V')
>>> # Radial flow component at ``r=0.95 r_o``, 65 contour levels
>>> p.surf(field='vr', r=0.95, levels=65, cm='seismic')
```

```
>>> # Control the limit of the colormap from -1e3 to 1e3
>>> p.surf(field='vp', r=1., vmin=-1e3, vmax=1e3, levels=33)
```

Parameters

- **field** (*str*) – the name of the field you want to display
- **proj** (*str*) – the type of projection. Default is Hammer, in case you want to use 'ortho' or 'moll', then Basemap is required.
- **lon_0** (*float*) – central azimuth (only used with Basemap)
- **lat_0** (*float*) – central latitude (only used with Basemap)
- **r** (*float*) – the radius at which you want to display the input data (in normalised units with the radius of the outer boundary)
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **lines** (*bool*) – when set to True, over-plot solid lines to highlight the limits between two adjacent contour levels
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero.

9.3.14 Support for TO outputs

class magic.**TOMovie**(*file=None*, *iplot=True*, *cm='seismic'*, *cut=0.5*, *levels=33*, *avg=True*, *precision=<class 'numpy.float32'>*)

This class allows to read and display the *TO_mov.TAG* generated when *l_TOmovie=.true.* is True.

```
>>> # This will allow you to pick up one TO_mov files among the existing ones
>>> t = TOMovie()
```

```
>>> # Read TO_mov.N0m2, time-averaged it and display it with 65 contour levels
>>> t = TOMovie(file='TO_mov.N0m2', avg=True, levels=65, cm='seismic')
```

__add__(*new*)

Built-in function to sum two TO movies

Note: So far this function only works for two TO movies with the same grid sizes. At some point, we might introduce grid extrapolation to allow any summation.

__init__(*file=None*, *iplot=True*, *cm='seismic'*, *cut=0.5*, *levels=33*, *avg=True*, *precision=<class 'numpy.float32'>*)

Parameters

- **file** (*str*) – the filename of the TO_mov file
- **cm** (*str*) – the name of the color map
- **levels** (*int*) – the number of contour levels
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut
- **iplot** (*bool*) – a boolean to specify if one wants to plot or not the results
- **avg** (*bool*) – time average of the different forces
- **precision** (*str*) – precision of the input file, np.float32 for single precision, np.float64 for double precision

__weakref__
list of weak references to the object (if defined)

plot(*cut=0.8*, *levs=16*, *avg=True*, *cmap='RdYlBu_r'*)
Plotting function

Parameters

- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut
- **levs** (*int*) – number of contour levels
- **avg** (*bool*) – when set to True, quantities are time-averaged
- **cmap** (*str*) – name of the colormap

class magic.**MagicTOHemi**(*datadir='.'*, *hemi='n'*, *tag=None*, *precision=<class 'numpy.float32'>*, *iplot=False*)
This class can be used to read and display z-integrated quantities produced by the TO outputs. Those are basically the TO[s|n]hn.TAG files

```
>>> to = MagicTOHemi(hemi='n', iplot=True) # For the Northern hemisphere
```

__init__(*datadir='.'*, *hemi='n'*, *tag=None*, *precision=<class 'numpy.float32'>*, *iplot=False*)

Parameters

- **datadir** (*str*) – current working directory
- **hemi** (*str*) – Northern or Southern hemisphere ('n' or 's')
- **tag** (*str*) – file suffix (tag), if not specified the most recent one in the current directory is chosen
- **precision** (*str*) – single or double precision
- **iplot** (*bool*) – display the output plot when set to True (default is True)

plot()
Plotting function

9.3.15 Run comparison

class magic.**CompSims**(*file='liste'*, *field='ts'*, *ncol=4*, *cm='RdYlBu_r'*, *dpi=96*, *normed=True*, *levels=16*, *type=None*, *r=0.9*, *bw=False*, *ave=False*, *cut=1*)

This class allows to compare an analyse several DNS simultaneously. It is possible to compare time-series or *graphic files*. To set it up, you first need to create a file that contains the list of directories you want to analyse:

```
$ cat inputList
E3e4Eps5e3Q05
E3e4Eps2e3Q07
E3e4Eps2e3Q08
E3e4Eps2e3Q09
```

This list thus contains four directories (one run per directory) that can be further analysed:

```
>>> # Display the time-series of kinetic energy on 2 columns
>>> CompSims(file='inputList', field='ts', ncol=2)
>>> # Display the equatorial cuts of v_r
>>> CompSims(file='inputList', field='vr', type='equat', levels=65, cm='seismic')
>>> # Display the radial cuts of B_r at r=0.8 r_o
>>> CompSims(file='inputList', field='br', type='surf', r=0.8)
>>> # Display the average zonal flow
>>> CompSims(file='inputList', field='vp', type='avg')
```

__init__(*file='liste'*, *field='ts'*, *ncol=4*, *cm='RdYlBu_r'*, *dpi=96*, *normed=True*, *levels=16*, *type=None*, *r=0.9*, *bw=False*, *ave=False*, *cut=1*)

Parameters

- **file** (*str*) – the input file that contains the list of directories that one wants to analyse
- **field** (*str*) – name of the input field. Possible options are: 'ts': displaye the time-series of kinetic energy; 'e_mag': display the time-series of magnetic energy; 'flux': display the time-series of the Nusselt numbers; 'zonal': display the surface zonal flow; 'Anything else': try to interpret the field
- **type** (*str*) – nature of the plot. Possible values are: 'avg' or 'slice': phi-average or phi-slice; 'equat': equatorial cut; 'surf': radial cut; 'ts*: time series
- **ncol** (*int*) – number of columns of the figure
- **ave** (*bool*) – when set to True, it tries to read a time-averaged graphic file
- **r** (*float*) – the radius at which you want to display the input data (in normalised units with the radius of the outer boundary)
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.
- **dpi** (*int*) – dot per inch when saving PNGs
- **bw** (*bool*) – when set to True, display grey-scaled contour levels
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut

__weakref__

list of weak references to the object (if defined)

plotAvg()
Plot azimutal averages in (theta, r) planes.

plotEmag()
Plot time-series of the magnetic energy

plotEquat()
Plot equatorial cuts in (phi, r) planes.

plotFlux()
Plot time-series of the top and bottom Nusselt numbers

plotSurf()
Plot radial cuts in (phi, theta) planes using the Hammer projection.

plotTs()
Plot time-series of the kinetic energy

plotZonal()
Plot surface zonal flow profiles.

9.3.16 Conversion of G_#.TAG files to vts/vti files

class magic.graph2vtk.**Graph2Vtk**(*gr*, *filename='out'*, *scals=['vr', 'emag', 'tfluct']*, *vecs=['u', 'B']*, *potExtra=False*, *ratio_out=2*, *nrout=32*, *deminc=True*, *outType='vts'*, *nFiles=1*, *nx=96*, *ny=96*, *nz=96*, *labFrame=False*)

This class allows to transform an input graphic file to a file format readable by paraview/visit or mayavi. It also allows to compute a possible potential extrapolation of the field lines in an arbitrary outer spherical shell domain

```
>>> # Load a graphic file
>>> gr = MagicGraph(ivar=1)
>>> # store myOut.vts
>>> Graph2Vtk(gr, 'myOut', outType='vts')
>>> # store u' and B for the vector fields and vortz and T for the scalars
>>> Graph2Vtk(gr, scals=['temp', 'vortz'], vecs=['ufluct', 'B'])
>>> # store only T'
>>> Graph2Vtk(gr, scals=['tempfluct'], vecs=[])
>>> # store only B with its potential extrapolation up to 3*r_cmb
>>> Graph2Vtk(gr, scals=[], vecs=['B'], potExtra=True, ratio_out=3)
>>> # Extrapolate on a cartesian grid of size 128^3
>>> Graph2Vtk(gr, outType='vti', nx=128, ny=128, nz=128)
```

__init__(*gr*, *filename='out'*, *scals=['vr', 'emag', 'tfluct']*, *vecs=['u', 'B']*, *potExtra=False*, *ratio_out=2*, *nrout=32*, *deminc=True*, *outType='vts'*, *nFiles=1*, *nx=96*, *ny=96*, *nz=96*, *labFrame=False*)

Parameters

- **filename** (*str*) – the file name of the output (without extension)
- **gr** (magic.MagicGraph) – the input graphic file one wants to transform to vts/vti
- **scals** (*list(str)*) – a list that contains the possible input scalars: ‘entropy’, ‘vr’, ‘vp’, ‘tfluct’, ‘vortz’, ‘vortzfluct’, ‘ekin’, ‘emag’, ‘vortr’, ‘colat’
- **vecs** (*list(str)*) – a list that contains the possible input vectors: ‘u’, ‘b’, ‘ufluct’, ‘bfluct’
- **potExtra** (*bool*) – when set to True, calculates the potential extrapolation of the magnetic field up to ratio_out*r_cmb

- **ratio_out** (*float*) – in case of potential extrapolation, this is the ratio of the external outer radius to r_cmb (rout/rcmb)
- **nrout** (*integer*) – in case of potential extrapolation, this input allows to specify thenumber of radial grid points in the outer spherical envelope
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry
- **outType** (*str*) – nature of the VTK file produced. This can be either 'vts' for the spherical grid or 'vti' for an extrapolation on a cartesian grid
- **nFiles** (*int*) – number of output chunks in case of parallel vts file format (pvts)
- **nx** (*int*) – number of grid points in the x direction
- **ny** (*int*) – number of grid points in the x direction
- **nz** (*int*) – number of grid points in the x direction
- **labFrame** (*bool*) – when set to True, transform the velocity to the lab frame

__weakref__
list of weak references to the object (if defined)

writeVTI(*filename*, *nx=96*, *ny=96*, *nz=96*)
In this case, the output is extrapolated on a cartesian grid and then written in a vti file.

Parameters

- **filename** (*str*) – the file name of the output (without extension)
- **nx** (*int*) – number of grid points in the x direction
- **ny** (*int*) – number of grid points in the x direction
- **nz** (*int*) – number of grid points in the x direction

writeVTS(*filename*, *nFiles*)
This function stores the output on a structured-grid vts file.

Parameters

- **filename** (*str*) – the file name of the output (without extension)
- **nFiles** (*int*) – number of outpute files (in case of pvts)

magic.graph2vtk.**sph2cart_scal**(*scals*, *radius*, *nx=96*, *ny=96*, *nz=96*, *minc=1*)
This function interpolates a series of scalar fields from the spherical coordinates to the cartesian coordinates.

Parameters

- **scals** (*numpy.ndarray[nscals,nphi,ntheta,nr]*) – an array that contains the different scalar quantities
- **radius** (*numpy.ndarray*) – the input radius
- **nx** (*int*) – number of grid points in the x direction
- **ny** (*int*) – number of grid points in the x direction
- **nz** (*int*) – number of grid points in the x direction
- **minc** (*int*) – azimuthal symmetry

Returns a tuple that contains the scalars, the max of the grid and the grid spacing

Return type (numpy.ndarray[nscals,nz,ny,nx],float,float)

magic.graph2vtk.**sph2cart_vec**(*vecr*, *vect*, *vecp*, *radius*, *nx=96*, *ny=96*, *nz=96*, *minc=1*)

This function interpolates a series of vector fields from the spherical coordinates to the cartesian coordinates.

Parameters

- **vecr** (*numpy.ndarray[nvecs,nphi,ntheta,nr]*) – the radial components of the different vector fields
- **vect** (*numpy.ndarray[nvecs,nphi,ntheta,nr]*) – the latitudinal components of the different vector fields
- **vecp** (*numpy.ndarray[nvecs,nphi,ntheta,nr]*) – the azimuthal components of the different vector fields
- **radius** (*numpy.ndarray*) – the input radius
- **nx** (*int*) – number of grid points in the x direction
- **ny** (*int*) – number of grid points in the x direction
- **nz** (*int*) – number of grid points in the x direction
- **minc** (*int*) – azimuthal symmetry

Returns a tuple that contains the three vectors components

Return type (numpy.ndarray[nvecs,nz,ny,nx],..)

9.3.17 Potential extrapolation

class magic.potExtra.**ExtraPot**(*rcmb*, *brcmb*, *minc*, *ratio_out=2.0*, *nrout=32*, *cutCMB=False*, *deminc=True*)

This class is used to compute the potential field extrapolation of the magnetic field in an arbitrary outer spherical shell domain. It takes as an input the magnetic field at the CMB.

__init__(*rcmb*, *brcmb*, *minc*, *ratio_out=2.0*, *nrout=32*, *cutCMB=False*, *deminc=True*)

Parameters

- **bcmb** (*numpy.ndarary*) – the surface radial field, array of dimension [np, nt]
- **rcmb** (*float*) – the value of the radius at the surface
- **minc** (*int*) – azimuthal symmetry
- **ratio_out** (*float*) – the ratio of the outer sphere radius to the surface radius
- **nrout** (*int*) – the number of radial point (linearly spaced) of the extrapolated field in the outer spherical domain
- **cutCMB** (*bool*) – a logical if one wants to remove the first grid point (useful if one then wants to merge the graphic file with the extrapolation)
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry

__weakref__

list of weak references to the object (if defined)

avg(*field='br'*, *levels=12*, *cm='RdYlBu_r'*, *normed=True*, *vmax=None*, *vmin=None*)

A small routine to plot the azimuthal averages of the extrapolated fields.

Parameters

- **field** (*str*) – the quantity you want to plot: 'br' or 'bp'

- **levels** (*int*) – the number of contour levels
- **cm** (*str*) – the name of the colormap
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.

9.3.18 Additional possible analyses

class magic.bLayers.**BLayers**(*iplot=False*, *quiet=False*)

This class allows to determine the viscous and thermal boundary layers using several classical methods (slope method, peak values, dissipation rates, etc.). It uses the following files:

- Kinetic energy: *eKinR.TAG*
- Power budget: *powerR.TAG*
- Radial profiles used for boundary layers: *bLayersR.TAG*

This function can thus **only** be used when both *powerR.TAG* and *bLayersR.TAG* exist in the working directory.

> **Warning:** This function works well as long as rigid boundaries and fixed temperature boundary conditions are employed. Other combination of boundary conditions (fixed fluxes and/or stress-free) might give wrong results, since boundary layers become awkward to define in that case.

Since this function is supposed to use time-averaged quantities, the usual procedure is first to define the initial averaging time using *AvgField*: (this needs to be done only once)

```
>>> a = AvgField(tstart=2.58)
```

Once the tInitAvg file exists, the boundary layer calculation can be done:

```
>>> bl = BLayers(iplot=True)
>>> # print the formatted output
>>> print(bl)
```

__init__(*iplot=False*, *quiet=False*)

Parameters

- **iplot** (*bool*) – display the result when set to True (default False)
- **quiet** (*bool*) – less verbose when set to True (default is False)

__str__()

Formatted output

plot()

Plotting function

magic.bLayers.**getAccuratePeaks**(*rad*, *uh*, *uhTop*, *uhBot*, *ri*, *ro*)

This functions performs a spline extrapolation around the maxima of the input array uh to define a more accurate location of the boundary layer.

Parameters

- **rad** (*numpy.ndarray*) – radius
- **uh** (*numpy.ndarray*) – the horizontal velocity profile
- **uhTop** (*float*) – first peak value of uh close to the outer boundary
- **uhBot** (*float*) – first peak value of uh close to the inner boundary
- **ri** (*float*) – the inner core radius
- **ro** (*float*) – the outer core radius

Returns four floats: thickness of the bottom boundary layer, thickness of the top boundary layer, extrapolated value of uh at the bottom boundary layer, extrapolated value of uh at the top boundary layer

Return type list

magic.bLayers.**getMaxima**(*field*)

This function determines the local maxima of the input array field

Parameters **field** (*numpy.ndarray*) – the input array

Returns a list containing the indices of the local maxima

Return type list

magic.bLayers.**integBotTop**(*rad*, *field*, *ri*, *ro*, *lambdai*, *lambdao*, *normed=False*)

This function evaluates the radial integral of the input array field in the bottom and top boundary layers separately.

Parameters

- **rad** (*numpy.ndarray*) – radius
- **field** (*numpy.ndarray*) – the input radial profile
- **ri** (*float*) – the inner core radius
- **ro** (*float*) – the outer core radius
- **lambdai** (*float*) – thickness of the inner boundary layer
- **lambdao** (*float*) – thickness of the outer boundary layer
- **normed** (*bool*) – when set to True, the outputs are normalised by the volumes of the boundary layers. In that case, the outputs are volume-averaged quantities.

Returns two floats that contains the bottom and top boundary layers integrations (integBot, integTop)

Return type list

magic.bLayers.**integBulkBc**(*rad*, *field*, *ri*, *ro*, *lambdai*, *lambdao*, *normed=False*)

This function evaluates the radial integral of the input array field in the boundary layer and in the bulk separately.

Parameters

- **rad** (*numpy.ndarray*) – radius
- **field** (*numpy.ndarray*) – the input radial profile
- **ri** (*float*) – the inner core radius
- **ro** (*float*) – the outer core radius
- **lambdai** (*float*) – thickness of the inner boundary layer
- **lambdao** (*float*) – thickness of the outer boundary layer

- **normed** (*bool*) – when set to True, the outputs are normalised by the volumes of the boundary layers and the fluid bulk, respectively. In that case, the outputs are volume-averaged quantities.

Returns two floats that contains the boundary layer and the bulk integrations (integBc, integBulk)

Return type list

class magic.**ThetaHeat**(*iplot=False*, *angle=10*, *pickleName='thHeat.pickle'*)

This class allows to conduct some analysis of the latitudinal variation of the heat transfer. It relies on the movie files *ATmov.TAG* and *AHF_mov.TAG*. As it's a bit time-consuming, the calculations are stored in a python.pickle file to quicken future usage of the data.

This function can **only** be used when *bLayersR.TAG* exist in the working directory.

Since this function is supposed to use time-averaged quantities, the usual procedure is first to define the initial averaging time using *AvgField*: (this needs to be done only once)

```
>>> a = AvgField(tstart=2.58)
```

Once the tInitAvg file exists, the latitudinal heat transfer analysis can be done using:

```
>>> # For chunk-averages over 10^\degree in the polar and equatorial regions.
>>> th = ThetaHeat(angle=10)
>>> # Formatted output
>>> print(th)
```

__init__(*iplot=False*, *angle=10*, *pickleName='thHeat.pickle'*)

Parameters

- **iplot** (*bool*) – a boolean to toggle the plots on/off
- **angle** (*float*) – the integration angle in degrees

PickleName calculations a

__str__()

Formatted outputs

```
>>> th = ThetaHeat()
>>> print(th)
```

plot()

Plotting function

class magic.cyl.**Cyl**(*ivar=1*, *datadir='.'*, *ns=None*)

This class allows to extrapolate a given *graphic file* on a cylindrical grid. Once done, the extrapolated file is stored in a python.pickle file. It is then possible to display 2-D cuts of the extrapolated arrays (radial cuts, phi-averages, equatorial cuts, z-averages and phi-slices)

Warning: This process is actually **very demanding** and it might take a lot of time to extrapolate the G_#.TAG file. Be careful when choosing the input value of ns!

```
>>> # Extrapolate the G file to the cylindrical grid (ns=128, nz=2*ns)
>>> c = Cyl(ivar=1, ns=128)
>>> # Radial cut of v_r
>>> c.surf(field='vr', r=0.8)
>>> # Vertical average of B_\phi
```

(continues on next page)

(continued from previous page)

```
>>> c.avgz(field='Bphi', cm='seismic', levels=33)
>>> # Azimuthal average of v_\phi
>>> c.avg(field='Bphi')
>>> # Equatorial cut of of v_theta
>>> c.equat(field='vtheta')
```

__init__ (*ivar=1*, *datadir='.'*, *ns=None*)

Parameters

- **ivar** (*int*) – the number of the Graphic file
- **datadir** (*str*) – working directory
- **ns** (*int*) – number of grid points in the radial direction

avg (*field='Bphi'*, *levels=16*, *cm='RdYlBu_r'*, *normed=True*, *vmax=None*, *vmin=None*)
Plot the azimutal average of a given field.

```
>>> c = Cyl(ns=65)
>>> # Azimuthal average of B_r
>>> c.avg(field='Br', cm='seismic', levels=33)
```

Parameters

- **field** (*str*) – name of the input field
- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the color map
- **normed** (*bool*) – when set to True, the contours are normalised fro -max(field), max(field)
- **vmin** (*float*) – truncate the contour levels to values > vmin
- **vmax** (*float*) – truncate the contour levels to values < vmax

avgz (*field='vs'*, *levels=16*, *cm='RdYlBu_r'*, *normed=True*, *vmin=None*, *vmax=None*, *avg=False*)
Plot the vertical average of a given field.

```
>>> c = Cyl(ns=65)
>>> # Vertical average of v_s
>>> c.avg(field='vs', cm='seismic', levels=33)
```

Parameters

- **field** (*str*) – name of the input field
- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the color map
- **normed** (*bool*) – when set to True, the contours are normalised fro -max(field), max(field)
- **vmin** (*float*) – truncate the contour levels to values > vmin
- **vmax** (*float*) – truncate the contour levels to values < vmax
- **avg** (*bool*) – when set to True, an additional figure with the phi-average profile is also displayed

equat (*field='vs'*, *levels=16*, *cm='RdYlBu_r'*, *normed=True*, *vmax=None*, *vmin=None*)
Plot an input field in the equatorial plane.

```
>>> c = Cyl(ns=65)
>>> # Equatorial cut of v_\phi
>>> c.equat(field='vphi', cm='seismic', levels=33)
```

Parameters

- **field** (*str*) – name of the input field
- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the color map
- **normed** (*bool*) – when set to True, the contours are normalised fro -max(field), max(field)
- **vmin** (*float*) – truncate the contour levels to values > vmin
- **vmax** (*float*) – truncate the contour levels to values < vmax

slice (*field='Bphi'*, *lon_0=0.0*, *levels=16*, *cm='RdYlBu_r'*, *normed=True*)
Plot an azimuthal slice of a given field.

```
>>> c = Cyl(ns=65)
>>> # Slices of v_r at 30 and 60 degrees
>>> c.slice(field='vr', lon_0=[30, 60])
```

Parameters

- **field** (*str*) – name of the input field
- **lon_0** (*float or list*) – the longitude of the slice in degrees, or a list of longitudes
- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the color map
- **normed** (*bool*) – when set to True, the contours are normalised fro -max(field), max(field)

surf (*field='Bphi'*, *r=0.85*, *vmin=None*, *vmax=None*, *levels=16*, *cm='RdYlBu_r'*, *normed=True*, *figsize=None*)
Plot the surface distribution of an input field at a given input radius (normalised by the outer boundary radius).

```
>>> c = Cyl(ns=65)
>>> # Surface plot of B_\phi from -10 to 10
>>> c.surf(field='Bphi', r=0.6, vmin=-10, vmax=10, levels=65)
```

Parameters

- **field** (*str*) – name of the input field
- **r** (*float*) – radial level (normalised to the outer boundary radius)
- **levels** (*int*) – number of contour levels
- **cm** (*str*) – name of the color map

- **normed** (*bool*) – when set to True, the contours are normalised fro -max(field), max(field)
- **vmin** (*float*) – truncate the contour levels to values > vmin
- **vmax** (*float*) – truncate the contour levels to values < vmax

magic.cyl.**sph2cyl**(*g*, *ns=None*, *nz=None*)

This function interpolates the three flow (or magnetic field) component of a *G_#.TAG* file on a cylindrical grid of size (ns, nz).

> **Warning:** This might be really slow!

Parameters

- **g** (*magic.MagicGraph*) – input graphic output file
- **ns** (*int*) – number of grid points in the radial direction
- **nz** (*int*) – number of grid points in the vertical direction

Returns a python tuple of five numpy.ndarray (S,Z,vs,vp_cyl,vz). S[nz,ns] is a meshgrid that contains the radial coordinate. Z[nz,ns] is a meshgrid that contains the vertical coordinate. vs[nz,ns] is the radial component of the velocity (or magnetic field), vp_cyl[nz,ns] the azimuthal component and vz[nz,ns] the vertical component.

Return type tuple

magic.cyl.**sph2cyl_plane**(*data*, *rad*, *ns*, *nz*)

This function extrapolates a phi-slice of a spherical shell on a cylindrical grid

```
>>> # Read G_1.test
>>> gr = MagicGraph(ivar=1, tag='test')
>>> # phi-average v_\phi and s
>>> vpm = gr.vphi.mean(axis=0)
>>> sm = gr.entropy.mean(axis=0)
>>> # Interpolate on a cylindrical grid
>>> Z, S, outputs = sph2cyl_plane([vpm, sm], gr.radius, 512, 1024)
>>> vpm_cyl, sm_cyl = outputs
```

Parameters

- **data** (*list(numpy.ndarray)*) – a list of 2-D arrays [(ntheta, nr), (ntheta, nr), ...]
- **rad** (*numpy.ndarray*) – radius
- **ns** (*int*) – number of grid points in s direction
- **nz** (*int*) – number of grid points in z direction

Returns a python tuple that contains two numpy.ndarray and a list (S,Z,output). S[nz,ns] is a meshgrid that contains the radial coordinate. Z[nz,ns] is a meshgrid that contains the vertical coordinate. output=[arr1[nz,ns], ..., arrN[nz,ns]] is a list of the interpolated array on the cylindrical grid.

Return type tuple

magic.cyl.**zavg**(*input*, *radius*, *ns*, *minc*, *save=True*, *filename='vp.pickle'*, *normed=True*)

This function computes a z-integration of a list of input arrays (on the spherical grid). This works well for 2-D (phi-slice) arrays. In case of 3-D arrays, only one element is allowed (too demanding otherwise).

Parameters

- **input** (*list(numpy.ndarray)*) – a list of 2-D or 3-D arrays
- **radius** (*numpy.ndarray*) – spherical radius
- **ns** (*int*) – radial resolution of the cylindrical grid (nz=2*ns)
- **minc** (*int*) – azimuthal symmetry
- **save** (*bool*) – a boolean to specify if one wants to save the outputs into a pickle (default is True)
- **filename** (*str*) – name of the output pickle when save=True
- **normed** (*bool*) – a boolean to specify if ones wants to simply integrate over z or compute a z-average (default is True: average)

Returns a python tuple that contains two numpy.ndarray and a list (height,cylRad,output) height[ns] is the height of the spherical shell for all radii. cylRad[ns] is the cylindrical radius. output=[arr1[ns], ..., arrN[ns]] contains the z-integrated output arrays.

Return type tuple

class magic.**Butterfly**(*file=None*, *step=1*, *iplot=True*, *rad=0.8*, *lastvar=None*, *nvar='all'*, *levels=20*, *cm='RdYlBu_r'*, *precision=<class 'numpy.float32'>*, *cut=0.8*)

This class can be used to display the time evolution of the magnetic field for various latitudes (i.e. the well-known butterfly diagrams). These diagrams are usually constructed using MagIC's *movie files*: either radial cuts (like Br_CMB_mov.TAG) or azimuthal-average (like AB_mov.TAG).

```
>>> # Read Br_CMB_mov.ccondAne1N3MagRa2e7Pm2ggg
>>> t1 = Butterfly(file='Br_CMB_mov.ccondAne1N3MagRa2e7Pm2ggg', step=1,
                   iplot=False)
>>> # Plot it
>>> t1.plot(levels=33, cm='seismic', cut=0.6)
```

__add__(*new*)

Overload of the addition operator

```
>>> # Read 2 files
>>> b1 = Butterfly(file='AB_mov.test1', iplot=False)
>>> b2 = Butterfly(file='AB_mov.test2', iplot=False)
>>> # Stack them and display the whole thing
>>> b = b1+b2
>>> b.plot(levels=33, contour=True, cut=0.8, cm='seismic')
```

__init__(*file=None*, *step=1*, *iplot=True*, *rad=0.8*, *lastvar=None*, *nvar='all'*, *levels=20*, *cm='RdYlBu_r'*, *precision=<class 'numpy.float32'>*, *cut=0.8*)

Parameters

- **file** (*str*) – when specified, the constructor reads this file, otherwise a list with the possible options is displayed
- **rad** (*float*) – radial level (normalised to the outer boundary radius)
- **iplot** (*bool*) – display/hide the plots (default is True)
- **nvar** (*int*) – the number of time steps (lines) of the movie file one wants to plot starting from the last line

- **lastvar** (*int*) – the number of the last time step to be read
- **step** (*int*) – the stepping between two lines
- **levels** (*int*) – the number of contour levels
- **cm** (*str*) – the name of the color map
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut
- **precision** (*bool*) – precision of the input file, np.float32 for single precision, np.float64 for double precision

__weakref__
list of weak references to the object (if defined)

fourier2D(*renorm=False*)
This function allows to conduct some basic Fourier analysis on the data. It displays two figures: the first one is a contour levels in the (Frequency, Latitude) plane, the second one is integrated over latitudes (thus a simple, power vs Frequency plot)

```
>>> # Load the data without plotting
>>> b1 = Butterfly(file='AB_mov.test1', iplot=False)
>>> # Fourier analysis
>>> b1.fourier2D()
```

Parameters **renorm** (*bool*) – when set to True, it rebins the time series in case of irregularly spaced data

plot(*levels=12*, *contour=False*, *renorm=False*, *cut=0.5*, *mesh=3*, *cm='RdYlBu_R'*)
Plotting function

Parameters

- **cm** (*str*) – name of the colormap
- **levels** (*int*) – the number of contour levels (only used when iplot=True and contour=True)
- **contour** (*bool*) – when set to True, display contour levels (pylab.contourf), when set to False, display an image (pylab.imshow)
- **renorm** (*bool*) – when set to True, it re-bins the time series in case of irregularly time-spaced data
- **mesh** (*int*) – when renorm=True, factor of regriding: NewTime = mesh*OldTime
- **cut** (*float*) – adjust the contour extrema to max(abs(data))*cut

9.3.19 Spectral tansforms

class magic.spectralTransforms.**SpectralTransforms**(*l_max=32*, *minc=1*, *lm_max=561*, *n_theta_max=64*, *verbose=True*)
This python class is used to compute Legendre and Fourier transforms from spectral to physical space. It works in two steps: one first needs to initialize the transform

```
>>> sh = SpectralTransforms( l_max=256, lm_max=33153, n_theta_max=384)
>>> print(Tlm[:, 10].shape) # lm_max (Temperature at ir=10)
>>> T = sh.spec_spat(Tlm) # T[n_phi_max, n_theta_max]
```

spat_spec(**args*)

This subroutine computes a transfrom from spatial representation (n_phi,n_theta) to spectral representation (lm_max). It returns one complex 1-D array (dimension(n_phi_max))

```
>>> gr = MagicGraph()
>>> sh = SpectralTransforms(gr.l_max, gr.minc, gr.lm_max, gr.n_theta_max)
>>> vr = gr.vr[:,:,30] # Radius ir=30
>>> vrlm = sh.spat_spec(vr) # vrlm is a complex array (lm_max)
>>> # Caculation of the poloidal potential from vr:
>>> wlm = np.zeros_like(vrlm)
>>> wlm[1:] = vrlm[1:]/(sh.ell[1:]*(sh.ell[1:]+1))*gr.radius[30]**2
>>> # Spheroidal/Toroidal transform
>>> vtlm, vplm = spec_spat(gr.vtheta, gr.vphi)
```

Parameters **input** (*numpy.ndarray*) – input array in the physical space (n_phi,n_theta)

Returns output array in the spectral space (lm_max)

Return type numpy.ndarray

spec_spat(**args*, ***kwargs*)

This subroutine computes a transfrom from spectral to spatial for all latitudes. It returns either one or two 2-D arrays (dimension(n_phi_max,n_theta_max)) depending if only the poloidal or both the poloidal and the toroidal potentials are given as input quantities.

```
>>> print(wlmr.shape) # lm_max
>>> vr = spec_spat_equat(wlmr)
>>> print(vr.shape) # n_phi, n_theta
>>> vt, vp = spec_spat_equat(dwdrlmr, zlmr)
```

spec_spat_dphi(*polo*)

This routine computes the phi-derivative and the transform from spectral to spatrial spaces. It returns a 2-D array of dimension (n_phi,n_theta)

```
>>> p = MagicPotential('V')
>>> vrlm = p.pol*p.ell*(p.ell+1)/p.radius[ir]**2/p.rho0[ir] # vr at r=ir
>>> dvrdp = p.sh.spec_spat_dphi(vrlm) # phi-derivative of vr
```

Parameters **polo** (*numpy.ndarray*) – the input array(lm_max) in spectral space

Returns the phi derivative in the physical space (n_phi, n_theta)

Return type numpy.ndarray

spec_spat_dtheta(*polo*, *l_axi=False*)

This routine computes the theta-derivative and the transform from spectral to spatrial spaces. It returns a 2-D array of dimension (n_phi,n_theta)

```
>>> p = MagicPotential('V')
>>> vrlm = p.pol*p.ell*(p.ell+1)/p.radius[ir]**2/p.rho0[ir] # vr at r=ir
>>> dvrdt = p.sh.spec_spat_dtheta(vrlm) # theta-derivative of vr
```

Parameters

- **polo** (*numpy.ndarray*) – the input array(lm_max) in spectral space
- **l_axi** (*bool*) – switch to True, if only the axisymmetric field is needed

Returns the theta derivative in the physical space (n_phi, n_theta)

Return type numpy.ndarray

spec_spat_equat(**args*)

This subroutine computes a transfrom from spectral to spatial at the equator. It returns either one or two 1-D arrays (dimension(n_phi_max)) depending if only the poloidal or both the poloidal and the toroidal potentials are given as input quantities.

```
>>> print(wlmr.shape) # lm_max
>>> vr = spec_spat_equat(wlmr)
>>> print(vr.shape) # n_phi
>>> vt, vp = spec_spat_equat(dwdrlmr, zlmr)
```

9.3.20 Plotting functions

magic.plotlib.**cut**(*dat*, *vmax=None*, *vmin=None*)

This functions truncates the values of an input array that are beyond vmax or below vmin and replace them by vmax and vmin, respectively.

```
>>> # Keep only values between -1e3 and 1e3
>>> datNew = cut(dat, vmin=-1e3, vmax=1e3)
```

Parameters

- **dat** (*numpy.ndarray*) – an input array
- **vmax** (*float*) – maximum upper bound
- **vmin** (*float*) – minimum lower bound

Returns an array where the values >=vmax have been replaced by vmax and the values <=vmin have been replaced by vmin

Return type numpy.ndarray

magic.plotlib.**equatContour**(*data*, *radius*, *minc=1*, *label=None*, *levels=65*, *cm='seismic'*, *normed=True*, *vmax=None*, *vmin=None*, *cbar=True*, *tit=True*, *normRad=False*, *deminc=True*, *bounds=True*)

Plot the equatorial cut of a given field

Parameters

- **data** (*numpy.ndarray*) – the input data (an array of size (nphi,nr))
- **radius** (*numpy.ndarray*) – the input radius
- **minc** (*int*) – azimuthal symmetry
- **label** (*str*) – the name of the input physical quantity you want to display
- **normRad** (*bool*) – when set to True, the contour levels are normalised radius by radius (default is False)
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True

- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.
- **deminc** (*bool*) – a logical to indicate if one wants do get rid of the possible azimuthal symmetry
- **bounds** (*bool*) – a boolean to determine if one wants to plot the limits of the domain (True by default)

magic.plotlib.**hammer2cart**(*ttheta*, *pphi*, *colat=False*)

This function is used to define the Hammer projection used when plotting surface contours in *magic.Surf*

```
>>> # Load Graphic file
>>> gr = MagicGraph()
>>> # Meshgrid
>>> pphi, ttheta = mgrid[-np.pi:np.pi:gr.nphi*1j, np.pi/2.:-np.pi/2.:gr.ntheta*1j]
>>> x,y = hammer2cart(ttheta, pphi)
>>> # Contour plots
>>> contourf(x, y, gr.vphi)
```

Parameters

- **ttheta** (*numpy.ndarray*) – meshgrid [nphi, ntheta] for the latitudinal direction
- **pphi** – meshgrid [nphi, ntheta] for the azimuthal direction
- **colat** (*numpy.ndarray*) – colatitudes (when not specified a regular grid is assumed)

Returns a tuple that contains two [nphi, ntheta] arrays: the x, y meshgrid used in contour plots

Return type (numpy.ndarray, numpy.ndarray)

magic.plotlib.**merContour**(*data*, *radius*, *label=None*, *levels=65*, *cm='seismic'*, *normed=True*, *vmax=None*, *vmin=None*, *cbar=True*, *tit=True*, *fig=None*, *ax=None*, *bounds=True*)

Plot a meridional cut of a given field

Parameters

- **data** (*numpy.ndarray*) – the input data (an array of size (ntheta,nr))
- **radius** (*numpy.ndarray*) – the input radius
- **label** (*str*) – the name of the input physical quantity you want to display
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.

- **bounds** (*bool*) – a boolean to determine if one wants to plot the limits of the domain (True by default)
- **fig** (*matplotlib.figure.Figure*) – a pre-existing figure (if needed)
- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – a pre-existing axis

magic.plotlib.**radialContour**(*data*, *rad=0.85*, *label=None*, *proj='hammer'*, *lon_0=0.0*, *vmax=None*, *vmin=None*, *lat_0=30.0*, *levels=65*, *cm='seismic'*, *normed=True*, *cbar=True*, *tit=True*, *lines=False*, *fig=None*, *ax=None*)

Plot the radial cut of a given field

Parameters

- **data** (*numpy.ndarray*) – the input data (an array of size (nphi,ntheta))
- **rad** (*float*) – the value of the selected radius
- **label** (*str*) – the name of the input physical quantity you want to display
- **proj** (*str*) – the type of projection. Default is Hammer, in case you want to use 'ortho' or 'moll', then Basemap is required.
- **levels** (*int*) – the number of levels in the contour
- **cm** (*str*) – name of the colormap ('jet', 'seismic', 'RdYlBu_r', etc.)
- **tit** (*bool*) – display the title of the figure when set to True
- **cbar** (*bool*) – display the colorbar when set to True
- **lines** (*bool*) – when set to True, over-plot solid lines to highlight the limits between two adjacent contour levels
- **vmax** (*float*) – maximum value of the contour levels
- **vmin** (*float*) – minimum value of the contour levels
- **normed** (*bool*) – when set to True, the colormap is centered around zero. Default is True, except for entropy/temperature plots.
- **fig** (*matplotlib.figure.Figure*) – a pre-existing figure (if needed)
- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – a pre-existing axis

9.3.21 Various useful functions

magic.libmagic.**ReadBinaryTimeseries**(*infile*, *ncols*, *datatype='f8'*, *endianness='>'*)

This function reads binary timeseries. It is then faster than the fast_read function.

Parameters

- **infile** (*string*) – the file to read
- **ncols** (*int*) – number of columns of the file
- **datatype** (*string*) – 'f8' = 64-bit floating-point number 'f4' = 32-bit floating-point number
- **endianness** (*string*) – '>' = big-endian ; '<' = small-endian

Returns an array[nlines, ncols] that contains the data of the binary file

Return type numpy.ndarray

magic.libmagic.**anelprof**(*radius*, *strat*, *polind*, *g0=0.0*, *g1=0.0*, *g2=1.0*)

This functions calculates the reference temperature and density profiles of an anelastic model.

```
>>> rad = chebgrid(65, 1.5, 2.5)
>>> temp, rho, beta = anelprof(rad, strat=5., polind=2.)
```

Parameters

- **radius** (*numpy.ndarray*) – the radial gridpoints
- **polind** (*float*) – the polytropic index
- **strat** (*float*) – the number of the density scale heights between the inner and the outer boundary
- **g0** (*float*) – gravity profile: g=g0
- **g1** (*float*) – gravity profile: g=g1*r/r_o
- **g2** (*float*) – gravity profile: g=g2*(r_o/r)**2

Returns a tuple that contains the temperature profile, the density profile and the log-derivative of the density profile versus radius

Return type (numpy.ndarray, numpy.ndarray, numpy.ndarray)

magic.libmagic.**avgField**(*time*, *field*, *tstart=None*, *std=False*)

This subroutine computes the time-average (and the std) of a time series

```
>>> ts = MagicTs(field='misc', iplot=False, all=True)
>>> nuavg = avgField(ts.time, ts.topnuss, 0.35)
>>> print(nuavg)
```

Parameters

- **time** (*numpy.ndarray*) – time
- **field** (*numpy.ndarray*) – the time series of a given field
- **tstart** (*float*) – the starting time of the averaging
- **std** (*bool*) – when set to True, the standard deviation is also calculated

Returns the time-averaged quantity

Return type float

magic.libmagic.**chebgrid**(*nr*, *a*, *b*)

This function defines a Gauss-Lobatto grid from a to b.

```
>>> r_icb = 0.5 ; r_cmb = 1.5; n_r_max=65
>>> rr = chebgrid(n_r_max, r_icb, r_cmb)
```

Parameters

- **nr** (*int*) – number of radial grid points plus one (Nr+1)
- **a** (*float*) – lower limit of the Gauss-Lobatto grid
- **b** (*float*) – upper limit of the Gauss-Lobatto grid

Returns the Gauss-Lobatto grid

Return type numpy.ndarray

magic.libmagic.**cylSder**(*radius*, *data*, *order=4*)
This function computes the s derivative of an input array defined on a regularly-spaced cylindrical grid.

```
>>> s = linspace(0., 1., 129 ; dat = cos(s)
>>> ddatds = cylSder(s, dat)
```

Parameters

- **radius** (*numpy.ndarray*) – cylindrical radius
- **data** (*numpy.ndarray*) – input data
- **order** (*int*) – order of the finite-difference scheme (possible values are 2 or 4)

Returns s derivative

Return type numpy.ndarray

magic.libmagic.**cylZder**(*z*, *data*)
This function computes the z derivative of an input array defined on a regularly-spaced cylindrical grid.

```
>>> z = linspace(-1., 1., 129 ; dat = cos(z)
>>> ddatdz = cylZder(z, dat)
```

Parameters

- **z** (*numpy.ndarray*) – height of the cylinder
- **data** (*numpy.ndarray*) – input data

Returns z derivative

Return type numpy.ndarray

magic.libmagic.**fast_read**(*file*, *skiplines=0*, *binary=False*, *precision=<class 'numpy.float64'>*)
This function reads an input ascii table (can read both formatted or unformatted fortran)

```
>>> # Read 'e_kin.test', skip the first 10 lines
>>> data = fast_read('e_kin.test', skiplines=10)
```

Parameters

- **file** (*str*) – name of the input file
- **skiplines** (*int*) – number of header lines to be skept during reading
- **binary** (*bool*) – when set to True, try to read an unformatted binray Fortran file (default is False)
- **precision** (*str*) – single (np.float32) or double precision (np.float64)

Returns an array[nlines, ncols] that contains the data of the ascii file

Return type numpy.ndarray

magic.libmagic.**fd_grid**(*nr*, *a*, *b*, *fd_stretching=0.3*, *fd_ratio=0.1*)
This function defines a stretched grid between a and b

```
>>> r_icb = 0.5 ; r_cmb = 1.5; n_r_max=64
>>> rr = fd_grid(n_r_max, r_cmb, r_icb)
```

Parameters

- **nr** (*int*) – number of radial grid points
- **a** (*float*) – upper boundary of the grid
- **b** (*float*) – lower boundary of the grid
- **fd_stretching** (*float*) – fraction of points in the bulk
- **fd_ratio** (*float*) – ratio of minimum to maximum spacing

Returns the radial grid

Returns the radial grid

Return type numpy.ndarray

magic.libmagic.**getCpuTime**(*file*)

This function calculates the CPU time from one given log file

Parameters **file** (*file*) – the log file you want to analyze

Returns the total CPU time

Return type float

magic.libmagic.**getTotalRunTime**()

This function calculates the total CPU time of one run directory

Returns the total RUN time

Return type float

magic.libmagic.**intcheb**(*f*, *nr*, *z1*, *z2*)

This function integrates an input function f defined on the Gauss-Lobatto grid.

```
>>> print(intcheb(f, 65, 0.5, 1.5))
```

Parameters

- **f** – an input array
- **nr** (*int*) – number of radial grid points
- **z1** (*float*) – lower limit of the Gauss-Lobatto grid
- **z2** (*float*) – upper limit of the Gauss-Lobatto grid

Type numpy.ndarray

Returns the integrated quantity

Return type float

magic.libmagic.**matder**(*nr*, *z1*, *z2*)

This function calculates the derivative in Chebyshev space.

```
>>> r_icb = 0.5 ; r_cmb = 1.5; n_r_max=65
>>> d1 = matder(n_r_max, r_icb, r_cmb)
>>> # Chebyshev grid and data
>>> rr = chebgrid(n_r_max, r_icb, r_cmb)
>>> f = sin(rr)
>>> # Radial derivative
>>> df = dot(d1, f)
```

Parameters

- **nr** (*int*) – number of radial grid points
- **z1** (*float*) – lower limit of the Gauss-Lobatto grid
- **z2** (*float*) – upper limit of the Gauss-Lobatto grid

Returns a matrix of dimension (nr,nr) to calculate the derivatives

Return type numpy.ndarray

magic.libmagic.**phideravg**(*data*, *minc=1*, *order=4*)

phi-derivative of an input array

```
>>> gr = MagicGraph()
>>> dvphidp = phideravg(gr.vphi, minc=gr.minc)
```

Parameters

- **data** (*numpy.ndarray*) – input array
- **minc** (*int*) – azimuthal symmetry
- **order** (*int*) – order of the finite-difference scheme (possible values are 2 or 4)

Returns the phi-derivative of the input array

Return type numpy.ndarray

magic.libmagic.**prime_factors**(*n*)

This function returns all prime factors of a number

Returns all prime factors

Return type list

magic.libmagic.**progressbar**(*it*, *prefix=''*, *size=60*)

Fancy progress-bar for loops

```
for i in progressbar(range(1000000)):
    x = i
```

Parameters

- **prefix** (*str*) – prefix string before progress bar
- **size** (*int*) – width of the progress bar (in points of xterm width)

magic.libmagic.**rderavg**(*data*, *eta=0.35*, *spectral=True*, *exclude=False*)

Radial derivative of an input array

```
>>> gr = MagiGraph()
>>> dvrdr = rderavg(gr.vr, eta=gr.radratio)
```

Parameters

- **data** (*numpy.ndarray*) – input array
- **eta** (*float*) – aspect ratio of the spherical shell
- **spectral** (*bool*) – when set to True use Chebyshev derivatives, otherwise use finite differences (default is True)
- **exclude** (*bool*) – when set to True, exclude the first and last radial grid points and replace them by a spline extrapolation (default is False)

Returns the radial derivative of the input array

Return type numpy.ndarray

magic.libmagic.**scanDir**(*pattern*, *tfix=None*)

This function sorts the files which match a given input pattern from the oldest to the most recent one (in the current working directory)

```
>>> dat = scanDir('log.*')
>>> print(log)
```

Parameters

- **pattern** (*str*) – a classical regexp pattern
- **tfix** (*float*) – in case you want to add only the files that are more recent than a certain date, use tfix (computer 1970 format!!)

Returns a list of files that match the input pattern

Return type list

magic.libmagic.**sderavg**(*data*, *eta=0.35*, *spectral=True*, *colat=None*, *exclude=False*)

s derivative of an input array

```
>>> gr = MagiGraph()
>>> dvpds = sderavg(gr.vphi, eta=gr.radratio, colat=gr.colatitude)
```

Parameters

- **data** (*numpy.ndarray*) – input array
- **eta** (*float*) – aspect ratio of the spherical shell
- **spectral** (*bool*) – when set to True use Chebyshev derivatives, otherwise use finite differences (default is True)
- **exclude** (*bool*) – when set to True, exclude the first and last radial grid points and replace them by a spline extrapolation (default is False)
- **colat** (*numpy.ndarray*) – colatitudes (when not specified a regular grid is assumed)

Returns the s derivative of the input array

Return type numpy.ndarray

magic.libmagic.**secondtimeder**(*time*, *y*)
second time derivative of an input array

computed with central differences (numpy.gradient)

```
>>> ts = MagicTs(field='e_kin')
>>> dt_ekinpol = secondtimeder(ts,field='ekin_pol')
```

magic.libmagic.**selectField**(*obj*, *field*, *labTex=True*, *ic=False*)
This function selects for you which field you want to display. It actually allows to avoid possible variables miss-spelling: i.e. 'Bphi'='bp'='Bp'='bphi'

Parameters

- **obj** (*magic.MagicGraph*) – a graphic output file
- **field** (*str*) – the name of the field one wants to select
- **labTex** (*bool*) – when set to True, format the labels using LaTeX fonts

Returns a tuple that contains the selected physical field and its label

Return type (numpy.ndarray, str)

magic.libmagic.**symmetrize**(*data*, *ms*, *reversed=False*)
Symmetrise an array which is defined only with an azimuthal symmetry minc=ms

Parameters

- **data** (*numpy.ndarray*) – the input array
- **ms** (*int*) – the azimuthal symmetry
- **reversed** (*bool*) – set to True, in case the array is reversed (i.e. n_phi is the last column)

Returns an output array of dimension (data.shape[0]*ms+1)

Return type numpy.ndarray

magic.libmagic.**thetaderavg**(*data*, *order=4*)
Theta-derivative of an input array (finite differences)

```
>>> gr = MagiGraph()
>>> dvtdt = thetaderavg(gr.vtheta)
```

Parameters

- **data** (*numpy.ndarray*) – input array
- **order** (*int*) – order of the finite-difference scheme (possible values are 2 or 4)

Returns the theta-derivative of the input array

Return type numpy.ndarray

magic.libmagic.**timeder**(*time*, *y*)
time derivative of an input array

computed with central differences (numpy.gradient)

```
>>> ts = MagicTs(field='e_kin')
>>> dt_ekinpol = timeder(ts,field='ekin_pol')
```

magic.libmagic.**writeVpEq**(*par*, *tstart*)
This function computes the time-averaged surface zonal flow (and Rolc) and format the output

```
>>> # Reads all the par.* files from the current directory
>>> par = MagicTs(field='par', iplot=False, all=True)
>>> # Time-average
>>> st = writeVpEq(par, tstart=2.1)
>>> print(st)
```

Parameters

- **par** (*magic.MagicTs*) – a *MagicTs* object containing the par file
- **tstart** (*float*) – the starting time of the averaging

Returns a formatted string

Return type str

magic.libmagic.**zderavg**(*data*, *eta=0.35*, *spectral=True*, *colat=None*, *exclude=False*)

z derivative of an input array

```
>>> gr = MagiGraph()
>>> dvrdz = zderavg(gr.vr, eta=gr.radratio, colat=gr.colatitude)
```

Parameters

- **data** (*numpy.ndarray*) – input array
- **eta** (*float*) – aspect ratio of the spherical shell
- **spectral** (*bool*) – when set to True use Chebyshev derivatives, otherwise use finite differences (default is True)
- **exclude** (*bool*) – when set to True, exclude the first and last radial grid points and replace them by a spline extrapolation (default is False)
- **colat** (*numpy.ndarray*) – colatitudes (when not specified a regular grid is assumed)

Returns the z derivative of the input array

Return type numpy.ndarray

CHAPTER

TEN

DESCRIPTION OF THE FORTRAN MODULES

The following pages contain an exhaustive description of the different variables, subroutines and modules used in MagIC. This documentation is automatically generated from the source code docstrings using the Sphinx extention for the Fortran domain.

Fortran modules

1. For the main program file `magic.f90`, see *here*.
2. For the core modules that contain most of the global variables, see *here*.
3. For the MPI related modules, see *here*.
4. For the code initialization and the pre-calculations done in the initial stage of the computation (before the time-stepping loop), see *here* and *there*.
5. For the time-stepping loop, see *here*.
6. For the calculation of the non-linear terms (in the physical space) and their time-advance, see *here*.
7. For the calculation of the linear terms (in spectral space) and their time-advance, see *here*.
8. For the Chebyshev, Fourier and Legendre transforms, see *here*.
9. For the computation of the radial derivatives (Chebyshev) and the integration, see *here*.
10. For the definition of the blocking, see *here*.
11. For the calculation of the standard outputs (time-series, spectra and radial files), see *here*.
12. For the calculation of binary outputs (graphic files, movie files, potential and coeff files), see *here*.
13. For the additional calculations of specific outputs (torsional oscillations, RMS force balance, etc.), see *here*.
14. For reading and writing the check points (restart files), see *here*.
15. For additional useful functions (string manipulation, etc.), see *here*.

10.1 Main program `magic.f90`

10.2 Base modules

10.2.1 `precision.f90`

10.2.2 `truncation.f90`

10.2.3 `num_param.f90`

10.2.4 `phys_param.f90`

10.2.5 `logic.f90`

10.2.6 `fields.f90`

10.2.7 `dt_fieldsLast.f90`

10.2.8 `output_data.f90`

10.2.9 `constants.f90`

10.2.10 `special.f90`

10.3 MPI related modules

10.3.1 `parallel.f90`

10.3.2 `radial_data.f90`

10.3.3 `communications.f90`

10.3.4 `mpi_transpose.f90`

10.3.5 `parallel_solvers.f90`

10.4 Code initialization

10.4.1 `Namelists.f90`

10.4.2 `startFiels.f90`

10.4.3 `init_fields.f90`

10.5 Pre-calculations

10.5.1 `preCalc.f90`

10.5.2 `radial.f90`

10.5.3 `horizontal.f90`

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

M

P