The Pencil Code:

A High-Order MPI code for MHD Turbulence

User's and Reference Manual

Date: 2014-03-17 23:16:54 +0100 (Mon, 17 Mar 2014) , Revision: 21616

http://www.nordita.org/software/pencil-code/

http://pencil-code.googlecode.com/

The PENCIL CODE: multi-purpose and multi-user maintained
http://www.nordita.org/~brandenb/talks/misc/PencilCode04.htm

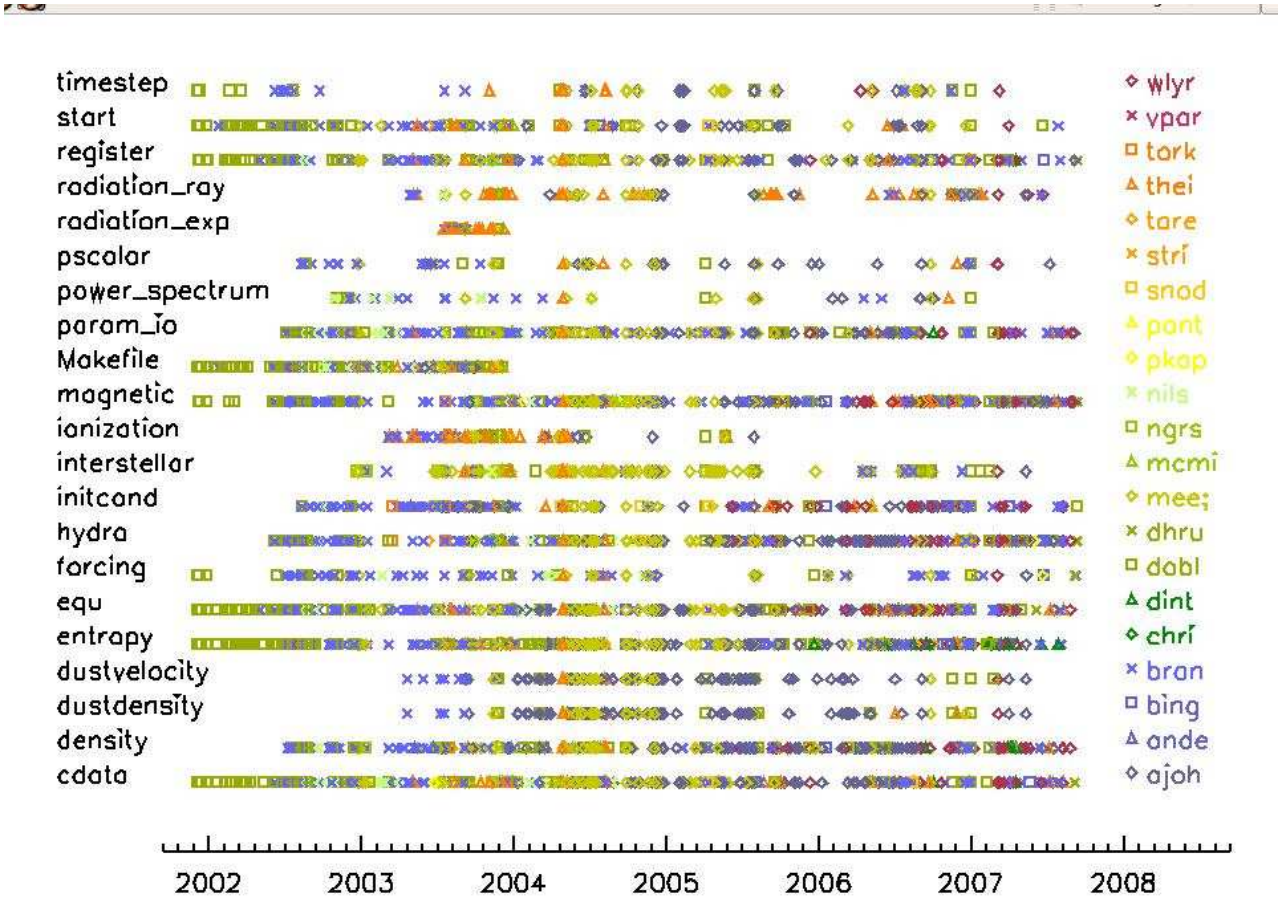

Figure 1: Check-in patterns as a function of time for different subroutines. The different users are marked by different symbols and different colors.

Contributors to the code
(in inverse alphabetical order according to their user name)

An up to date list of Pencil Code contributers can be found at Google Code.

wladimir.lyra	Wladimir Lyra	Caltech-JPL
weezy	S. Louise Wilkin	University of Newcastle
wdobler	Wolfgang Dobler	Potsdam
vpariev	Vladimir Pariev	University of Rochester
torkel	Ulf Torkelsson	Chalmers University
tavo.buk	Gustavo Guerrero	Stanford University
thomas.gastine	Thomas Gastine	MPI for Solar System Research
theine	Tobias (Tobi) Heinemann	IAS Princeton
tarek	Tarek A. Yousef	University of Trondheim
sven.bingert	Sven Bingert	MPI for Solar System Research
steveb	Steve Berukoff	UCLA
snod	Andrew Snodin	University of Newcastle
pkapyla	Petri Käpylä	University of Helsinki
nils.e.haugen	Nils Erland L. Haugen	SINTEF
ngrs	Graeme R. Sarson	University of Newcastle
NBabkovskaia	Natalia Babkovskaia	University of Helsinki
mreinhardt	Matthias Rheinhardt	University of Helsinki
mkorpi	Maarit J. Mantere (née Korpi)	University of Helsinki
miikkavaisala	Miikka Väisälä	University of Helsinki
mee	Antony (tOnY) Mee	University of Newcastle
mcmillan	David McMillan	York University, Toronto
mattias	Mattias Christensson	formerly at Nordita
koenkemel	Koen Kemel	Nordita, Stockholm
karlsson	Torgny Karlsson	Nordita
joishi	Jeff S. Oishi	Kavli Institute for Particle Astrophysics &
joern.warnecke	Jörn Warnecke	MPI for Solar System Research, Lindau
Iomsn1	Simon Candelaresi	University of Dundee, Dundee
fadiesis	Fabio Del Sordo	Nordita, Stockholm
dorch	Bertil Dorch	University of Copenhagen
boris.dintrans	Boris Dintrans	Observatoire Midi-Pyrénées, Toulouse
dhruba.mitra	Dhrubaditya Mitra	Nordita, Stockholm
ccyang	Chia-Chun Yang	Lick Observatory
christer	Christer Sandin	University of Uppsala
Bourdin.KIS	Philippe Bourdin	MPI for Solar System Research
AxelBrandenburg	Axel Brandenburg	Nordita
apichat	Apichat Neamvonk	University of Newcastle
amjed	Amjed Mohammed	University of Oldenburg
alex.i.hubbard	Alex Hubbard	Am. Museum Nat. History
michiel.lambrechts	Michiel Lambrechts	Lund Observatory, Lund University
anders	Anders Johansen	Lund Observatory, Lund University

License agreement and giving credit

The content of all files under `:pserver:$USER@svn.nordita.org:/var/cvs/brandenb` are under the GNU General Public License (`http://www.gnu.org/licenses/gpl.html`).

We, the PENCIL CODE community, ask that in publications and presentations the use of the code (or parts of it) be acknowledged with reference to the web site `http://www.nordita.org/software/pencil-code/` or (equivalently) to. `http://pencil-code.googlecode.com/`. As a courtesy to the people involved in developing particularly important parts of the program (use `svn annotate src/*.f90` to find out who did what!) we suggest to give appropriate reference to one or several of the following (or other appropriate) papers (listed here in temporal order):

Dobler, W., Haugen, N. E. L., Yousef, T. A., & Brandenburg, A.: 2003, "Bottleneck effect in three-dimensional turbulence simulations," *Phys. Rev.* **E 68**, 026304, 1-8 (astro-ph/0303324)

Haugen, N. E. L., Brandenburg, A., & Dobler, W.: 2003, "Is nonhelical hydromagnetic turbulence peaked at small scales?" *Astrophys. J. Lett.* **597**, L141-L144 (astro-ph/0303372)

Brandenburg, A., Käpylä, P., & Mohammed, A.: 2004, "Non-Fickian diffusion and tau-approximation from numerical turbulence," *Phys. Fluids* **16**, 1020-1027 (astro-ph/0306521)

Johansen, A., Andersen, A. C., & Brandenburg, A.: 2004, "Simulations of dust-trapping vortices in protoplanetary discs," *Astron. Astrophys.* **417**, 361-371 (astro-ph/0310059)

Haugen, N. E. L., Brandenburg, A., & Mee, A. J.: 2004, "Mach number dependence of the onset of dynamo action," *Monthly Notices Roy. Astron. Soc.* **353**, 947-952 (astro-ph/0405453)

Brandenburg, A., & Multamäki, T.: 2004, "How long can left and right handed life forms coexist?" *Int. J. Astrobiol.* **3**, 209-219 (q-bio/0407008)

McMillan, D. G., & Sarson, G. R.: 2005, "Dynamo simulations in a spherical shell of ideal gas using a high-order Cartesian magnetohydrodynamics code," *Phys. Earth Planet. Int.***153**, 124-135

Heinemann, T., Dobler, W., Nordlund, Å., & Brandenburg, A.: 2006, "Radiative transfer in decomposed domains," *Astron. Astrophys.* **448**, 731-737 (astro-ph/0503510)

Dobler, W., Stix, M., & Brandenburg, A.: 2006, "Convection and magnetic field generation in fully convective spheres," *Astrophys. J.* **638**, 336-347 (astro-ph/0410645)

Snodin, A. P., Brandenburg, A., Mee, A. J., & Shukurov, A.: 2006, "Simulating field-aligned diffusion of a cosmic ray gas," *Monthly Notices Roy. Astron. Soc.* **373**, 643-652 (astro-ph/0507176)

Johansen, A., Klahr, H., & Henning, T.: 2006, "Dust sedimentation and self-sustained Kelvin-Helmholtz turbulence in protoplanetary disc mid-planes," *Astrophys. J.* **636**, 1121-1134 (astro-ph/0512272)

de Val-Borro, M. and 22 coauthors (incl. Lyra, W.): 2006, "A comparative study of disc-planet interaction," *Monthly Notices Roy. Astron. Soc.* **370**, 529-558 (astro-ph/0605237)

Johansen, A., Oishi, J. S., Mac Low, M. M., Klahr, H., Henning, T., & Youdin, A.: 2007, "Rapid planetesimal formation in turbulent circumstellar disks," *Nature* **448**, 1022–1025 (arXiv/0708.3890)

Lyra, W., Johansen, A., Klahr, H., & Piskunov, N.: 2008, "Global magnetohydrodynamical models of turbulence in protoplanetary disks I. A cylindrical potential

on a Cartesian grid and transport of solids," *Astron. Astrophys.* **479**, 883-901 (arXiv/0705.4090)

Brandenburg, A., Rädler, K.-H., Rheinhardt, M., & Käpylä, P. J.: 2008, "Magnetic diffusivity tensor and dynamo effects in rotating and shearing turbulence," *Astrophys. J.* **676**, 740-751 (arXiv/0710.4059)

Lyra, W., Johansen, A., Klahr, H., & Piskunov, N.: 2008, "Embryos grown in the dead zone. Assembling the first protoplanetary cores in low-mass selfgravitating circumstellar disks of gas and solids," *Astron. Astrophys.* **491**, L41-L44

Lyra, W., Johansen, A., Klahr, H., & Piskunov, N.: 2009, "Standing on the shoulders of giants. Trojan Earths and vortex trapping in low-mass selfgravitating protoplanetary disks of gas and solids," *Astron. Astrophys.* **493**, 1125-1139

Lyra, W., Johansen, A., Zsom, A., Klahr, H., & Piskunov, N.: 2009, "Planet formation bursts at the borders of the dead zone in 2D numerical simulations of circumstellar disks," *Astron. Astrophys.* **497**, 869-888 (arXiv/0901.1638)

Mitra, D., Tavakol, R., Brandenburg, A., & Moss, D.: 2009, "Turbulent dynamos in spherical shell segments of varying geometrical extent," *Astrophys. J.* **697**, 923-933 (arXiv/0812.3106)

Haugen, N. E. L., & Kragset, S.: 2010, "Particle impaction on a cylinder in a crossflow as function of Stokes and Reynolds numbers," *J. Fluid Mech.* **661**, 239-261

Rheinhardt, M., & Brandenburg, A.: 2010, "Test-field method for mean-field coefficients with MHD background," *Astron. Astrophys.* **520**, A28 (arXiv/1004.0689)

Babkovskaia, N., Haugen, N. E. L., Brandenburg, A.: 2011, "A high-order public domain code for direct numerical simulations of turbulent combustion," *J. Comp. Phys.* **230**, 1-12 (arXiv/1005.5301)

Johansen, A., Youdin, A. N., & Lithwick, Y.: 2012, "Adding particle collisions to the formation of asteroids and Kuiper belt objects via streaming instabilities," *Astron. Astrophys.* **537**, A125

Lyra, W. & Kuchner, W. : 2013, "Formation of sharp eccentric rings in debris disks with gas but without planets," *Nature* **499**, 184–187

This list is not always up-to-date. We therefore ask the developers to check in new relevant papers, avoiding however redundancies.

Foreword

This code was originally developed at the Turbulence Summer School of the Helmholtz Institute in Potsdam (2001). While some SPH and PPM codes for hydrodynamics and magnetohydrodynamics are publicly available, this does not generally seem to be the case for higher order finite-difference or spectral codes. Having been approached by people interested in using our code, we decided to make it as flexible as possible and as user-friendly as seems reasonable, and to put it onto a public *CVS* repository. Since 21 September 2008 it is distributed via `http://code.google.com/p/pencil-code/`. The code can certainly not be treated as a black box (no code can), and in order to solve a new problem in an optimal way, users will need to find their own optimal set of parameters. In particular, you need to be careful in choosing the right values of viscosity, magnetic diffusivity, and radiative conductivity.

The PENCIL CODE is primarily designed to deal with weakly compressible turbulent flows, which is why we use high-order first and second derivatives. To achieve good parallelization, we use explicit (as opposed to compact) finite differences. Typical scientific targets include driven MHD turbulence in a periodic box, convection in a slab with nonperiodic upper and lower boundaries, a convective star embedded in a fully nonperiodic box, accretion disc turbulence in the shearing sheet approximation, etc. Furthermore, nonlocal radiation transport, inertial particles, dust coagulation, self-gravity, chemical reaction networks, and several other physical components are installed, but this number increases steadily. In addition to Cartesian coordinates, the code can also deal with spherical and cylindrical polar coordinates.

Magnetic fields are implemented in terms of the magnetic vector potential to ensure that the field remains solenoidal (divergence-free). At the same time, having the magnetic vector potential readily available is a big advantage if one wants to monitor the magnetic helicity, for example. The code is therefore particularly well suited for all kinds of dynamo problems.

The code is normally non-conservative; thus, conserved quantities should only be conserved up to the discretization error of the scheme (not to machine accuracy). There is no guarantee that a conservative code is more accurate with respect to quantities that are not explicitly conserved, such as entropy. Another important quantity that is (to our knowledge) not strictly conserved by ordinary flux conserving schemes is *magnetic helicity*.

There are currently no plans to implement adaptive mesh refinement into the code, which would cause major technical complications. Given that turbulence is generically space-filling, local refinement to smaller scales would often not be very useful anyway. On the other hand, in some geometries turbulence may well be confined to certain regions in space, so one could indeed gain by solving the outer regions with fewer points.

In order to be cache-efficient, we solve the equations along *pencils* in the x direction. One very convenient side-effect is that auxiliary and derived variables use very little memory, as they are only ever defined on one pencil. The domain can be tiled in the y and z directions. On multiprocessor computers, the code can use *MPI* (Message Passing Interface) calls to communicate between processors. An easy switching mechanism allows the user to run the code on a machine without MPI libraries (e.g. a notebook computer). Ghost zones are used to implement boundary conditions on physical and processor boundaries.

A high level of flexibility is achieved by encapsulating individual physical processes and variables in individual *modules*, which can be switched on or off in the file 'Makefile.local' in the local 'src' directory. This approach avoids the use of difficult-to-read preprocessor directives, at the price of requiring one dummy module for each physics module. For nonmagnetic hydrodynamics, for example, one will use the module 'nomagnetic.f90' and specifies

```
MAGNETIC=nomagnetic
```

in 'Makefile.local', while for MHD simulations, 'magnetic.f90' will be used:

```
MAGNETIC=magnetic
```

Note that the term *module* as used here is only loosely related to Fortran modules: both 'magnetic.f90' and 'nomagnetic.f90' define an F90 module named *Magnetic* — this is the basis of the switching mechanism we are using.

Input parameters (which are set in the files 'start.in', 'run.in') can be changed without recompilation. Furthermore, one can change the list of variables for monitoring (diagnostic) output on the fly, and there are mechanisms for making the code reload new parameters or exit gracefully at runtime.

The requirements for using the Pencil-MPI code are modest: you can use it on any Linux or Unix system with a F90/F95 compiler. Although the PENCIL CODE is mainly designed to run on supercomputers, more than 50% of the users run their code also on Macs, and the other half uses either directly Linux on their laptops or they use VirtualBox on their Windows machine on which they install Ubuntu Linux. If you have *IDL* as well, you will be able to visualize the results (a number of sample procedures are provided), but other tools such as *Python*, *DX* (OpenDX, data explorer) can also be used and some relevant tools and routines come with the PENCIL CODE.

If you want to make creative use of the code, this manual will contain far too little information. Its major aim is to give you an idea of the way the code is organized, so you can more efficiently *read the source code*, which contains a reasonable amount of comments. You might want to read through the various sample directories that are checked in. Choose one that is closest to your application and start modifying. For further enhancements that you may want to add to the code, you can take as an example the lines in the code that deal with related variables, functions, diagnostics, equations etc., which have already been implemented. Just remember: grep is one of your best friends when you want to understand how certain variables or functions are used in the code.

We will be happy to include user-supplied changes and updates to the code in future releases and welcome any feedback.

wdobler@gmail.com Potsdam
AxelBrandenburg@gmail.com Stockholm

Acknowledgments

Many people have contributed in different ways to the development of this code. We thank first of all Åke Nordlund (Copenhagen Observatory) and Bob Stein (University of Michigan) who introduced us to the idea of using high-order schemes in compressible flows and who taught us a lot about simulations in general.

The calculation of the power spectra, structure functions, the remeshing procedures, routines for changing the number of processors, as well as the shearing sheet approximation and the flux-limited diffusion approximation for radiative transfer were implemented by Nils Erland L. Haugen (University of Trondheim). Tobi Heinemann added the long characteristics method for radiative transfer as well as hydrogen ionization. He also added and/or improved shock diffusion for other variables and improved the resulting timestep control. Anders Johansen, Wladimir (Wlad) Lyra, and Jeff Oishi contributed to the implementation of the dust equations (which now comprises an array of different components). Antony (Tony) Mee (University of Newcastle) implemented shock viscosity and added the interstellar module together with Graeme R. Sarson (also University of Newcastle), who also implemented the geodynamo set-up together with David McMillan (currently also at the University of Newcastle). Tony also included a method for outputting auxiliary variables and enhanced the overall functionality of the code and related idl and dx procedures. He also added, together with Andrew Snodin, the evolution equations for the cosmic ray energy density. Vladimir Pariev (University of Rochester) contributed to the development and testing of the potential field boundary condition at an early stage. The implementation of spherical and cylindrical coordinates is due to Dhrubaditya (Dhruba) Mitra and Wladimir Lyra. Wlad also implemented the global set-up for protoplanetary disks (as opposed to the local shearing sheet formalism). He also added a N-body code (based on the particle module coded by Anders Johansen and Tony), and implemented the coupled evolution equations of neutrals and ions for two-fluid models of ambipolar diffusion. Boris Dintrans is in charge of implementing the anelastic and Boussinesq modules. Philippe-A. Bourdin implemented HDF5 support and wrote the optional IO-modules for high-performance computing featuring various communication strategies. He also contributed to the solar-corona module and worked on the IDL GUI, including the IDL routines for reading and working with large amounts of data. Again, this list contains other recent items that are not yet fully documented and acknowledged.

Use of the PPARC supported supercomputers in St Andrews (Mhd) and Leicester (Ukaff) is acknowledged. We also acknowledge the Danish Center for Scientific Computing for granting time on Horseshoe, which is a 512+140 processor Beowulf cluster in Odense (Horseshoe).

Contents

Part I

Using the PENCIL CODE

1 System requirements

To use the code, you will need the following:

1. Absolutely needed:
   - *F95* compiler
   - *C* compiler
2. Used heavily (if you don't have one of these, you will need to adjust many things manually):
   - a *Unix*/*Linux*-type system with *make* and *csh*
   - *Perl* (remember: if it doesn't run Perl, it's not a computer)
3. The following are dispensable, but enhance functionality in one way or the other:
   - an *MPI* implementation (for parallelization on multiprocessor systems)
   - *DX* alias *OpenDX* or *data explorer* (for 3-D visualization of results)
   - *IDL* (for visualization of results; the 7-minute demo license will do for many applications)

2 Obtaining the code

The code is now distributed via `http://code.google.com/p/pencil-code/`, where you can either download a tarball, or, preferably, download it via *svn*. In Iran and some other countries, Google Code is not currently available. To alleviate this problem, we have made a recent copy available on `http://www.nordita.org/software/pencil-code/`. If you want us to update this tarball, please contact us.

To ensure at least some level of stability of the *svn* versions, a set of test problems (listed in '`$PENCIL_HOME/bin/auto-test`') are routinely tested. This includes all problems in '`$PENCIL_HOME/samples`'. See Sect. 10 for details.

2.1 Obtaining the code via svn

1. Many machines have *svn* installed (try `svn -v` or `which svn`). On Ubuntu, for example, *svn* comes under the package name subversion.
2. Unless you are a privileged users with write access, you can download the code with the command

   ```
   svn checkout http://pencil-code.googlecode.com/svn/trunk/ pencil-code
   ```

   Privileged users with write access to the original repository should use something like

   ```
   svn checkout https://pencil-code.googlecode.com/svn/trunk/ pencil-code --username
   ```

   where *USERNAME* is to be replaced by *your* googlecode user name. Note that the svn password is not the usual google password, but it is the more complicated character string generated by google when clicking on → profile → settings. There it says "Your googlecode.com password: ...".

   Be sure to run `auto-test` before you check anything back in again. It can be very annoying for someone else to figure out what's wrong, especially if you are just up to something else. At the very least, you should do

   ```
   pc_auto-test --level=0 --no-pencil-check -C
   ```

   This allows you to run just 2 of the most essential tests starting with all the no-modules and then most-modules.

2.2 Updating via svn

Independent of how you installed the code in the first place (from tarball or via *svn*), you can update your version using *svn*. If you have done nontrivial alterations to your version of the code, you ought to be careful about upgrading: although *svn* is an excellent tool for distributed programming, conflicts are quite possible, since many of us are going to touch many parts of the code while we develop it further. Thus, despite the fact that the code is under *svn*, you should probably back up your important changes before upgrading.

Here is the upgrading procedure:

1. Perform a `svn` update of the tree:

   ```
   unix>  pc_svnup
   ```

2. Fix any conflicts you encounter and make sure the examples in the directory '`samples/`' are still working.

If you have made useful changes, please contact one of the (currently) 10 "Owners" (listed under `https://code.google.com/p/pencil-code/people/list`) who can give you check-in permission. Be sure to have sufficient comments in the code and please follow our standard coding conventions explained in Section 9.1.

2.3 Getting the last validated version

The script `pc_svnup` accepts arguments -val or -validated, which means that the current changes on a user's machine will be merged into the last working version. This way every user can be sure that any problems with the code must be due to the current changes done by this user since the last check-in.

Examples:

```
unix>  pc_svnup -src -s -validated
```

brings all files in '`$PENCIL_HOME/src`' to the last validated status, and merges all your changes into this version. This allows you to work with this, but in order to check in your changes you have to update everything to the most recent status first, i.e.

```
unix>  pc_svnup -src
```

Your own changes will be merged into this latest version as before.

NOTE: The functionality of the head of the trunk should be preserved at all times. However, accidents do happen. For the benefit of all other developers, any errors should be corrected within 1-2 hours. This is the reason why the code comes with a file '`pencil-code/license/developers.txt`', which should contain contact details of all developers. The `pc_svnup -val` option allows all other people to stay away from any trouble.

2.4 Getting older versions

You may find that the latest *svn* version of the code produces errors. If you have reasons to believe that this is due to changes introduced on 27 November 2008 (to give an example), you can check out the version prior to this by specifying a revision number with `svn update -r #####`. One reason why one cannot always reproduce exactly the same situation too far back in time is connected with the fact that processor architecture and the compiler were different, resulting e.g. in different rounding errors.

3 Getting started

To get yourself started, you should run one or several examples which are provided in one of the 'samples/' subdirectories. Note that you will only be able to fully assess the numerical solutions if you visualize them with *IDL*, *DX* or other tools (see Sect. 5.19).

3.1 Setup

3.1.1 Environment settings

The functionality of helper scripts and IDL routines relies on a few environment variables being set correctly. The simplest way to achieve this is to go to the top directory of the code and source one of the two scripts 'sourceme.csh' or 'sourceme.sh' (depending on the type of shell you are using):

```
csh>  cd pencil-code
csh>  source ./sourceme.csh
```

for *tcsh* or *csh* users; or

```
sh>  cd pencil-code
sh>  . ./sourceme.sh
```

for users of *bash*, *Bourne shell*, or similar shells. You should get output similar to

```
PENCIL_HOME = </home/dobler/f90/pencil-code>
Adding /home/dobler/f90/pencil-code/bin to PATH
```

Apart from the PATH variable, the environment variable IDL_PATH is set to something like ./idl:../idl:+$PENCIL_HOME/idl:./data:<IDL_DEFAULT> .

Note 1 The <IDL_DEFAULT> mechanism does not work for IDL versions 5.2 or older. In this case, you will have to edit the path manually, or adapt the 'sourceme' scripts.

Note 2 If you don't want to rely on the 'sourceme' scripts' (quite heuristic) ability to correctly identify the code's main directory, you can set the environment variable PENCIL_HOME explicitly before you run the source command.

Note 3 Do not just source the 'sourceme' script from your shell startup file ('~/.cshrc' or '~/.bashrc', because it outputs a few lines of diagnostics for each sub-shell, which will break many applications. To suppress all output, follow the instructions given in the header documentation of 'sourceme.csh' and 'sourceme.sh'.

Note 4 The second time you source 'sourceme', it will not add anything to your PATH variable. This is on purpose to avoid cluttering of your environment: you can source the file as often as you like (in your shell startup script, then manually and in addition in some script you have written), without thinking twice. If, however, at the first sourcing, the setting of PENCIL_HOME was wrong, this mechanism would keep you from ever adding the right directory to your PATH. In this case, you need to first undefine the environment variable PENCIL_HOME:

```
csh>  unsetenv PENCIL_HOME
csh>  source ./sourceme.csh
or
sh>  unset PENCIL_HOME
sh>  . ./sourceme.sh
```

3.1.2 Linking scripts and source files

With your environment set up correctly, you can now go to the directory you want to work in and set up subdirectories and links. This is accomplished by the script '`pc_setupsrc`', which is located in '`$PENCIL_HOME/bin`' and is thus now in your executable path.

For concreteness, let us assume you want to use '`samples/conv-slab`' as your *run directory*, i.e. you want to run a three-layer slab model of solar convection. You then do the following:

```
unix>  cd samples/conv-slab
unix>  pc_setupsrc
src already exists
2 files already exist in src
```

The script has linked a number of scripts from '`$PENCIL_HOME/bin`', generated a directory '`src`' for the source code and linked the Fortran source files (plus a few more files) from '`$PENCIL_HOME/src`' to that directory:

```
unix>  ls -F
reference.out  src/
start.csh@     run.csh@  getconf.csh@
start.in       run.in    print.in
```

3.1.3 Adapting '`Makefile.src`'

This step requires some input from you, but you only have to do this once for each machine you want to run the code on. See Sect. 5.2 for a description of the steps you need to take here.

Note: If you are lucky and use compilers similar to the ones we have, you may be able to skip this step; but blame yourself if things don't compile, then. If not, you can run `make` with explicit flags, see Sect. 5.2 and in particular Table 1.

3.1.4 Running `make`

Next, you run `make` in the '`src`' subdirectory of your run directory. Since you are using one of the predefined test problems, the settings in '`src/Makefile.local`' and '`src/cparam.local`' are all reasonable, and you just do

```
unix>  make
```

If you have set up the compiler flags correctly, compilation should complete successfully.

3.1.5 Choosing a data directory

The code will by default write data like snapshot files to the subdirectory '`data`' of the run directory. Since this will involve a large volume of IO-operations (at least for large grid sizes), one will normally try to avoid writing the data via NFS. The recommended way to set up a '`data`' data directory is to generate a corresponding directory on the local disc of the computer you are running on and (soft-)link it to '`./data`'. Even if the link is part of an NFS directory, all the IO operations will be local. For example, if you have a local disc '`/scratch`', you can do the following:

```
unix>  mkdir -p /scratch/$USER/pencil-data/samples/conv-slab
unix>  ln -s /scratch/$USER/pencil-data/samples/conv-slab ./data
```

This is done automatically by the `pc_mkdatadir` command which, in turn, is invoked when making a new run directory with the `pc_newrun` command, for example.

Even if you don't have an NFS-mounted directory (say, on your notebook computer), it is probably still a good idea to have code and data well separated by a scheme like the one described above.

An alternative to symbolic links, is to provide a file called '`datadir.in`' in the root of the run directory. This file should contain one line of text specifying the absolute or relative data directory path to use. This facility is useful if one wishes to switch one run directory between different data directories. It is suggested that in such cases symbolic links are again made in the run directory to the various locations, then the '`datadir.in`' need contain only a short relative path.

3.1.6 Running the code

You are now ready to start the code:

```
unix>  start.csh
Linux cincinnatus 2.4.18-4GB #1 Wed Mar 27 13:57:05 UTC 2002 i686 unknown
Non-MPI version
datadir = data
Fri Aug  8 21:36:43 CEST 2003
   src/start.x
CVS: io_dist.f90        v. 1.61         (brandenb  ) 2003/08/03 09:26:55
[...]
CVS: start.in           v. 1.4          (dobler    ) 2002/10/02 20:11:14
 nxgrid,nygrid,nzgrid=          32          32          32
 thermodynamics: assume cp=1

 uu: up-down
 piecewise polytropic vertical stratification (lnrho)
 init_lnrho: cs2bot,cs2top=   1.450000      0.3333330
 e.g. for ionization runs: cs2bot,cs2top not yet set
 piecewise polytropic vertical stratification (ss)

 start.x has completed successfully

0.070u 0.020s 0:00.14 64.2%     0+0k 0+0io 180pf+0w
```

```
Fri Aug  8 21:36:43 CEST 2003
```

This runs 'src/start.x' to construct an initial condition based on the parameters set in 'start.in'. This initial condition is stored in 'data/proc0/var.dat' (and in 'data/proc1/var.dat', etc. if you run the multiprocessor version). It is fair to say that this is now a rather primitive routine; see 'pencil-code/idl/read' for various reading routines. You can then visualize the data using standard idl language.

If you visualize the profiles using *IDL* (see below), the result should bear some resemblance to Fig. 2, but with different values in the ghost zones (the correct values are set at run-time only) and a simpler velocity profile.

Now we run the code:

```
unix>  run.csh
```

This runs 'src/run.x' and carries out *nt* time steps, where *nt* and other run time parameters are specified in 'run.in'. On a decent PC (1.7 GHz), 50 time steps take about 10 seconds.

The relevant part of the code's output looks like

```
--it----t-------dt-------urms----umax----rhom------ssm-----dtc----dtu---dtnu---dtchi-
   0   0.34  6.792E-03  0.0060  0.0452  14.4708  -0.4478  0.978  0.013  0.207  0.346
  10   0.41  6.787E-03  0.0062  0.0440  14.4707  -0.4480  0.978  0.013  0.207  0.345
  20   0.48  6.781E-03  0.0064  0.0429  14.4705  -0.4481  0.977  0.012  0.207  0.345
  30   0.54  6.777E-03  0.0067  0.0408  14.4703  -0.4482  0.977  0.012  0.207  0.345
  40   0.61  6.776E-03  0.0069  0.0381  14.4702  -0.4482  0.977  0.011  0.207  0.346
```

and lists

1. the number *it* of the current time step;
2. the time, t;
3. the time step, *dt*;
4. the rms velocity, $urms = \sqrt{\langle \boldsymbol{u}^2 \rangle}$;
5. the maximum velocity, $umax = \max |\boldsymbol{u}|$;
6. the mean density, $rhom = \langle \rho \rangle$;
7. the mean entropy, $ssm = \langle s \rangle / c_p$;
8. the time step in units of the acoustic Courant step, $dtc = \delta t / (c_{\rm s0}\, \delta x_{\min})$;
9. the time step in units of the advective time step, $dtu = \delta t / (c_{\delta t}\, \delta x / \max |\mathbf{u}|)$;
10. the time step in units of viscous time step, $dtnu = \delta t / (c_{\delta t,{\rm v}}\, \delta x^2 / \nu_{\max})$;
11. the time step in units of the conductive time step, $dtchi = \delta t / (c_{\delta t,{\rm v}}\, \delta x^2 / \chi_{\max})$.

The entries in this list can be added, removed or reformatted in the file 'print.in', see Sects 5.5 and J.3. The output is also saved in 'data/time_series.dat' and should be identical to the content of 'reference.out'.

If you have *IDL*, you can visualize the stratification with (see Sect. 5.19.4 for details)

```
unix >  idl
IDL >   .r start
```

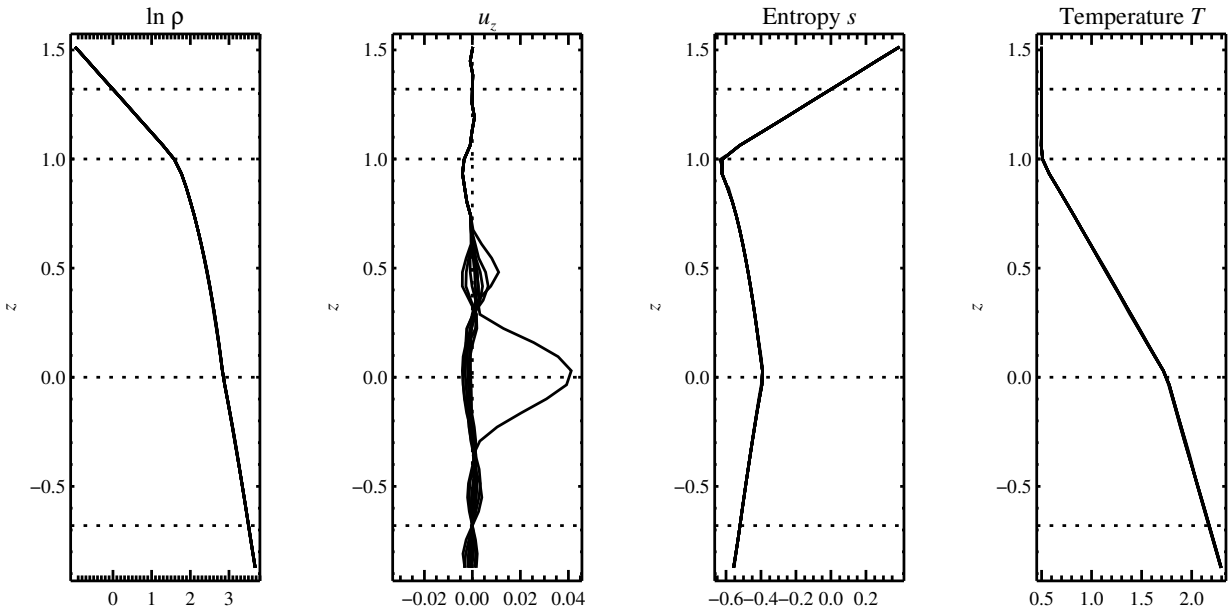

Figure 2: Stratification of the three-layer convection model in 'samples/conv-slab' after 50 timesteps ($t = 0.428$). Shown are (from left to right) density ρ, vertical velocity u_z, entropy s/c_p and temperature T as functions of the vertical coordinate z for about ten different vertical lines in the computational box. The dashed lines denote domain boundaries: $z < -0.68$ is the lower ghost zone (points have no physical significance); $-0.68 < z < 0$ is a stably stratified layer ($ds/dz > 0$); $0 < z < 1$ is the unstable layer ($ds/dz < 0$); $1 < z < 1.32$ is the isothermal top layer; $z > 1.32$ is the upper ghost zone (points have no physical significance).

```
IDL >    .r r
IDL >    .r thermo
IDL >    .r pvert
```

The result should look like Fig. 2.

Note: If you want to run the code with *MPI*, you will probably need to adapt 'getconf.csh', which defines the commands and flags used to run MPI jobs (and which is sourced by the scripts 'start.csh' and 'run.csh'). Try

```
csh -v getconf.csh
 or
csh -x getconf.csh
```

to see how 'getconf.csh' makes its decisions. You would add a section for the host name of your machine with the particular settings. Since 'getconf.csh' is linked from the central directory 'pencil-code/bin', your changes will be useful for all your other runs too.

3.2 Further tests

There is a number of other tests in the 'samples/' directory. You can use the script 'bin/auto-test' to automatically run these tests and have the output compared to reference results.

4 Code structure

4.1 Directory tree

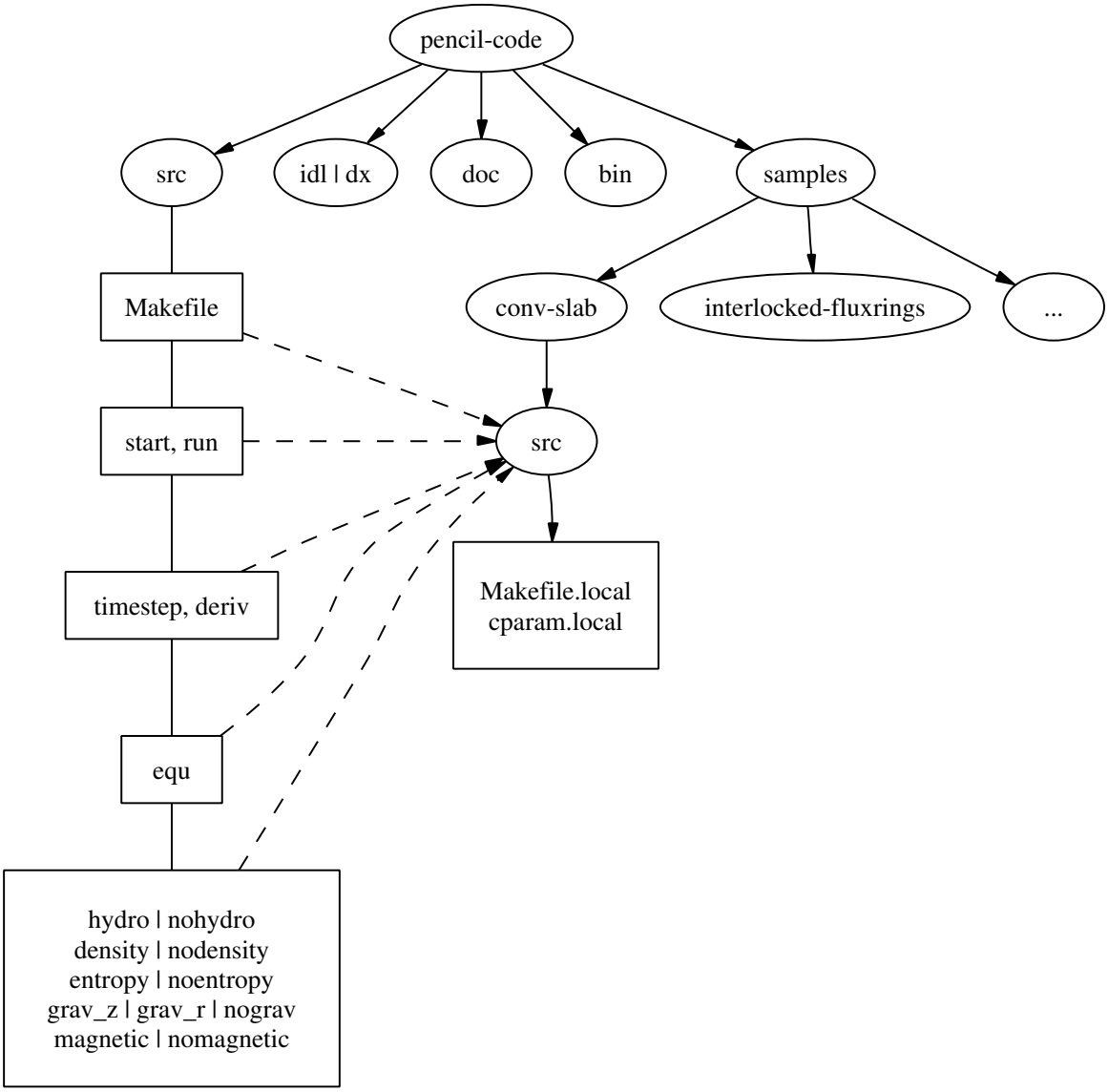

Figure 3: The basic structure of the code

The overall directory structure of the code is shown in Fig. 3. Under '`pencil-code`', there are currently the following files and directories:

```
bin/   config/  doc/  idl/  license/  perl/   samples/      sourceme.sh  utils/
bugs/  dx/   lib/  misc/     README  sourceme.csh  src/          www/
```

Almost all of the source code is contained in the directory '`src/`', but in order to encapsulate individual applications, the code is compiled separately for each run in a local directory '`src`' below the individual run directory, like e. g. '`samples/conv-slab/src`'.

It may be a good idea to keep your own runs also under *svn* or *cvs* (which is older than but similar to *svn*), but this would normally be a different repository. On the machine where you are running the code, you may want to check them out into a subdirectory of

'pencil-code/'. For example, we have our own runs in a repository called 'pencil-runs', so we do

```
unix>  cd $PENCIL_HOME
unix>  svn co runs pencil-runs
```

In this case, 'runs' contains individual run directories, grouped in classes (like 'spher' for spherical calculations, or 'kinematic' for kinematic dynamo simulations). The current list of classes in our own 'pencil-runs' repository is

```
1d-tests/   disc/          kinematic/  rings/
2d-tests/   discont/       Misc/       slab_conv/
3d-tests/   discussion/    OLD/        test/
buoy_tube/  forced/        pass_only/
convstar/   interstellar/  radiation/
```

The directory 'forced/' contains some forced turbulence runs (both magnetic and non-magnetic); 'gravz/' contains runs with vertical gravity; 'rings/' contains decaying MHD problems (interlocked flux rings as initial condition, for example); and 'kinematic/' contains kinematic dynamo problems where the hydrodynamics is turned off entirely. The file 'samples/README' should contain an up-to-date list and short description of the individual classes.[1]

The subdirectory 'src' of each run directory contains a few local configuration files (currently these are 'Makefile.local' and 'cparam.local') and possibly 'ctimeavg.local'. To compile the samples, links the files '.f90', '.c' and 'Makefile.src' need to be linked from the top file[src/]src directory to the local directory './src'. These links are set up by the script pc_setupsrc) when used in the root of a run directory.

General-purpose visualization routines for *IDL* or *DX* are in the directories 'idl' and 'dx', respectively. There are additional and more specialized *IDL* directories in the different branches under 'pencil-runs'.

The directory 'doc' contains this manual; 'bin' contains a number of utility scripts (mostly written in *csh* and *Perl*), and in particular the 'start.csh', 'run.csh', and 'getconf.csh' scripts. The '.svn' directory is used (you guessed it) by *.svn*, and is not normally directly accessed by the user; 'bugs', finally is used by us for internal purposes.

The files 'sourceme.csh' and 'sourceme.sh' will set up some environment variables — in particular PATH — and aliases/shell functions for your convenience. If you do not want to source one of these files, you need to make sure your *IDL* path is set appropriately (provided you want to use *IDL*) and you will need to address the scripts from 'bin' with their explicit path name, or adjust your PATH manually.

4.2 Basic concepts

4.2.1 Data access in pencils

Unlike the CRAY computers that dominated supercomputing in the 80s and early 90s, all modern computers have a cache that constitutes a significant bottleneck for many

[1] Our 'pencil-runs' directory also contains runs that were done some time ago. Occasionally, we try to update these, especially if we have changed names or other input conventions.

codes. This is the case if large three-dimensional arrays are constantly used within each time step, which has the obvious advantage of working on long arrays and allows vectorization of elementary machine operations. This approach also implies conceptual simplicity of the code and allows extensive use of the intuitive F90 array syntax. However, a more cache-efficient way of coding is to calculate an entire time step (or substep of a multi-stage time-stepping scheme) only along a one-dimensional pencil of data within the numerical grid. This technique is more efficient for modern RISC processors: on Linux PCs and SGI workstations, for example, we have found a speed-up by about 60% in some cases. An additional advantage is a drastic reduction in temporary storage for auxiliary variables within each time step.

4.2.2 Modularity

Each run directory has a file '`src/Makefile.local`' in which you choose certain *modules*[2], which tell the code whether or not entropy, magnetic fields, hydrodynamics, forcing, etc. should be invoked. For example, the settings for forced turbulent MHD simulations are

```
HYDRO     =   hydro
DENSITY   =   density
ENTROPY   = noentropy
MAGNETIC  =   magnetic
GRAVITY   = nogravity
FORCING   =   forcing

MPICOMM   = nompicomm
GLOBAL    = noglobal
IO        =   io_dist
FOURIER   = nofourier
```

This file will be processed by *make* and the settings are thus assignments of *make* variables. Apart from the physics modules (equation of motion: yes, density [pressure]: yes, entropy equation: no, magnetic fields: yes, gravity: no, forcing: yes), a few technical modules can also be used or deactivated; in the example above, these are *MPI* (switched off), additional global variables (none), input/output (distributed), and *FFT* (not used). The table in Sect. I in the Appendix lists all currently available modules.

Note that most of these *make* variables *must* be set, but they will normally obtain reasonable default values in '`Makefile`' (so you only need to set the non-standard ones in '`Makefile.local`'). It is by using this switching mechanism through `make` that we achieve high flexibility without resorting to excessive amounts of cryptic preprocessor directives or other switches within the code.

Many possible combinations of modules have already been tested and examples are part of the distribution, but you may be interested in a combination which was never tried before and which may not work yet, since the modules are not fully orthogonal. In such cases, we depend on user feedback for fixing problems and documenting the changes for others.

[2]We stress once more that we are not talking about F90 modules here, although there is some connection, as most of our modules define F90 modules: For example each of the modules *gravity_simple*, *grav_r* and *nogravity* defines a Fortran module *Gravity*.

4.3 Files in the run directories

4.3.1 'start.in', 'run.in', 'print.in'

These files specify the startup and runtime parameters (see Sects. 5.12 and J.2), and the list of diagnostic variables to print (see 5.5). They specify the setup of a given simulation and are kept under *svn* in the individual 'samples' directories.

4.3.2 'datadir.in'

If this file exists, it must contain the name of an existing directory, which will be used as *data directory*, i. e. the directory where all results are written. If 'datadir.in' does not exist, the data directory is 'data/'.

4.3.3 'reference.out'

If present, 'reference.out' contains the output you should obtain in the given run directory, provided you have not changed any parameters. To see whether the results of your run are OK, compare 'time_series.dat' to 'reference.out':

```
unix>  diff data/time_series.dat reference.out
```

4.3.4 'start.csh', 'run.csh', 'getconf.csh' [obsolete; see Sect. 5.1]

These are links to '$PENCIL_HOME/bin'. You will be constantly using the scripts 'start.csh' and 'run.csh' to initialize the code. Things that are needed by both (like the name of the mpirun executable, *MPI* options, or the number of processors) are located in 'getconf.csh', which is never directly invoked.

4.3.5 'src/'

The 'src' directory contains two local files, 'src/Makefile.local' and 'src/cparam.local', which allow the user to choose individual modules (see 4.2.2) and to set parameters like the grid size and the number of processors for each direction. These two files are part of the setup of a given simulation and are kept under *svn* in the individual 'samples' directories.

The file 'src/cparam.inc' is automatically generated by the script 'mkcparam' and contains the number of fundamental variables for a given setup.

All other files in 'src/' are either links to source files (and 'Makefile.src') in the '$PENCIL_HOME/src' directory, or object and module files generated by the compiler.

4.3.6 'data/'

This directory (the name of which will actually be overwritten by the first line of 'datadir.in', if that file is present; see §4.3.2) contains the output from the code:

'`data/dim.dat`' The global array dimensions.

'`data/legend.dat`' The header line specifying the names of the diagnostic variables in '`time_series.dat`'.

'`data/time_series.dat`' Time series of diagnostic variables (also printed to stdout). You can use this file directly for plotting with *Gnuplot*, *IDL*, *Xmgrace* or similar tools (see also §5.19).

'`data/tsnap.dat`', '`data/tvid.dat`' Time when the next snapshot '`VAR`N' or animation slice should be taken.

'`data/params.log`' Keeps a log of all your parameters: '`start.x`' writes the startup parameters to this file, '`run.x`' appends the runtime parameters and appends them anew, each time you use the '`RELOAD`' mechanism (see §5.10).

'`data/param.nml`' Complete set of startup parameters, printed as Fortran namelist. This file is read in by '`run.x`' (this is how values of startup parameters are propagated to '`run.x`') and by *IDL* (if you use it).

'`data/param2.nml`' Complete set of runtime parameters, printed as Fortran namelist. This file is read by *IDL* (if you use it).

'`data/index.pro`' Can be used as include file in *IDL* and contains the column in which certain variables appear in the diagnostics file ('`time_series.dat`'). It also contains the positions of variables in the '`VAR`N' files. These positions depend on whether *entropy* or *noentropy*, etc, are invoked. This is a temporary solution and the file may disappear in future releases.

'`data/interstellar.dat`' Unformatted file containing the time at which the next supernova event will occur, under certain supernova schemes. (Only needed by the *interstellar* module.)

'`data/proc0`', '`data/proc1`', ... These are the directories containing data from the individual processors. So after running an *MPI* job on two processors, you will have the two directories '`data/proc0`' and '`data/proc1`'. Each of the directories can contain the following files:

'`var.dat`' binary file containing the latest snapshot;

'`VAR`N' binary file containing individual snapshot number N;

'`dim.dat`' ASCII file containing the array dimensions as seen by the given processor;

'`time.dat`' ASCII file containing the time corresponding to '`var.dat`' (not actually *used* by the code, unless you use the *io_mpiodist.f90* module);

'`grid.dat`' binary file containing the part of the grid seen by the given processor;

'`seed.dat`' the random seed for the next time step (saved for reasons of reproducibility). For multi-processor runs with velocity forcing, the files '`proc`N`/seed.dat`' must all contain the same numbers, because globally coherent waves of given wavenumber are used;

'X`.xy`', 'X`.xz`', 'X`.yz`' two-dimensional sections of variable X, where X stands for the corresponding variable. The current list includes

```
bx.xy  bx.xz  by.xy  by.xz  bz.xy  bz.xz  divu.xy  lnrho.xz
ss.xz  ux.xy  ux.xz  uz.xy  uz.xz
```

Each processor writes its own slice, so these need to be reassembled if one wants to plot a full slice.

5 Using the code

5.1 Configuring the code to compile and run on your computer

Note: The procedure described here is currently (September 2009) relatively new. For some time, you may alternatively configure the code as described in Sects. 5.2 and 4.3.4. You are however advised to use the new procedure, as the old approach will eventually not be supported (and some of the scripts it uses will be removed).

Quick instructions:

1. Run `pc_build --debug-config` to find out which configuration files are tried and pick one.
2. Choose one of the existing files from '`$PENCIL_HOME/hosts/`' as blueprint, copy it to the file name from item 1 and adapt it to your system.

If you don't know what this was all about, read on.

In essence, configuration, compiling and running the code work like this:

1. Create a configuration file for your computer's *host ID*.
2. Compile the code using `pc_build`.
3. Run the code using `pc_run`

In the following, we will discuss the essentials of this scheme. Exhaustive documentation is available with the commands `perldoc Pencil::ConfigFinder` and `perldoc PENCIL::ConfigParser`.

5.1.1 Locating the configuration file

Commands like `pc_build` and `pc_run` use the Perl module '`Pencil::ConfigFinder`' to locate an appropriate configuration file and '`Pencil::ConfigParser`' to read and interpret it. When you use '`ConfigFinder`' on a given computer, it constructs a *host ID* for the system it is running on, and looks for a file '`host_ID.conf`' in any subdirectory of '`$PENCIL_HOME/config/hosts`'.

E.g., if the host ID is "workhorse.pencil.org", '`ConfigFinder`' would consider the file '`$PENCIL_HOME/config/hosts/pencil.org/workhorse.pencil.org.conf`'.

Note 1: The location in the tree under '`hosts/`' is irrelevant, which allows you to group related hosts by institution, owner, hardware, etc.

Note2: '`ConfigFinder`' actually uses the following search path:

1. '`./config`'
2. '`$PENCIL_HOME/config-local`'
3. '`$HOME/.pencil/config-local`'

4. '$PENCIL_HOME/config'

This allows you to override part of the 'config/' tree globally on the given file system, or locally for a particular run directory, or for one given copy of the PENCIL CODE. This search path is used both, for locating the configuration file for your host ID, and for locating included files (see below).

The host ID is constructed based on information that is easily available for your system. The algorithm is as follows:

1. Most commands using 'ConfigFinder' have a '--host-id' (sometimes abbreviated as '-H') option to explicitly set the host ID.
2. If the environment variable *PENCIL_HOST_ID* is set, its value is used.
3. If any of the files './host-ID', '$PENCIL_HOME/host-ID', '$HOME/.pencil/host-ID', exists, its first line is used.
4. If 'ConfigFinder' can get hold of a *fully qualified host name*, that is used as host ID.
5. Else, a combination of host name, operating system name and possibly some other information characterizing the system is used.
6. If no config file for the host ID is found, the current operating system name is tried as fallback host ID.

To see which host IDs are tried (up to the first one for which a configuration file is found), run

```
unix>  pc_build --debug-config
```

5.1.2 Structure of configuration files

It is strongly recommended to include in a users configuration file one of the preset compiler suite configuration files. Then, only minor options need to be set by a user, e.g. the optimization flags. One of those user configuration files looks rather simple:

```
# Simple config file. Most files don't need more.
%include compilers/GNU-GCC
```

or if you prefer a different compiler:

```
# Simple Intel compiler suite config file.
%include compilers/Intel
```

A more complex file (here using MPI) could include additional options:

```
# More complex config file.
%include compilers/GNU-GCC_MPI

%section Makefile
  MAKE_VAR1 = -j4   # joined compilation with four threads
  FFLAGS += -O3 -Wall -fbacktrace   # don't redefine, but append with '+='
%endsection Makefile
```

```
%section runtime
  mpiexec = mpirun   # some MPI backends need a redefinition of mpiexec
%endsection runtime

%section environment
  SCRATCH_DIR=/var/tmp/$USER
%endsection environment
```

Adding "_MPI" to a compiler suites name is usually sufficient to use MPI. Also the user is strongly discouraged to mix Fortran- and C-compilers across compiler suites by manually including separate compiler configuration files...

Note 3: We strongly advice to use at maximum the optimization level '-O2' for the Intel compiler and '-O3' for all other compilers. Higher optimization levels implicate an inadequate loss of precision.

The .conf file consists of the following elements:

Comments: A # sign and any text following it on the same line are ignored.

Sections: There are three sections:

Makefile for setting `make` parameters

runtime for adding compiler flags used by `pc_run`

environment shell environment variables for compilation and running

Include statements: An `%include ...` statement is recursively replaced by the contents of the files it points to.[3]

The included path gets a `.conf` suffix appended. Included paths are typically "absolute", e.g.

```
%include os/Unix
```

will include the file '`os/Unix.conf`' in the search path listed above (typically from '`$PENCIL_HOME/config`'). However, if the included path starts with a dot, it is a relative path, so

```
%include ./Unix
```

will only search in the directory where the including file is located.

Assignments: Statements like `FFLAGS += -O3` or `mpiexec=mpirun` are assignments and will set parameters that are used by `pc_build/make` or `pc_run`.

Lines ending with a backslash '\' are continuation lines.

If possible, one should always use *incremental assignments*, indicated by using a += sign instead of =, instead of redefining certain flags.

Thus,

```
CFLAGS += -O3
CFLAGS += -I../include -Wall
```

[3]However, if the include statement is inside a section, only the file's contents inside that section are inserted.

is the same as

```
CFLAGS = $(CFLAGS) -O3 -I../include -Wall
```

5.1.3 Compiling the code

Use the `pc_build` command to compile the code:

```
unix>  pc_build                                # use default compiler suite
unix>  pc_build -f Intel_MPI                   # specify a compiler suite
unix>  pc_build -f os/GNU_Linux,mpi/open-mpi   # explicitly specify config files
unix>  pc_build VAR=something                  # set variables for the makefile
unix>  pc_build --cleanall                     # remove generated files
```

The third example circumvents the whole host ID mechanism by explicitly instructing `pc_build` which configuration files to use. The fourth example shows how to define extra variables (`VAR=something`) for the execution of the `Makefile`.

See `pc_build --help` for a complete list of options.

5.1.4 Running the code

Use the `pc_run` command to run the code:

```
unix>  pc_run                    # start if necessary, then run
unix>  pc_run start
unix>  pc_run run

unix>  pc_run start run^3        # start, then run 3 times
unix>  pc_run start run run run  # start, then run 3 times
unix>  pc_run ^3                 # start if necessary, then run 3 times
```

See `pc_run --help` for a complete list of options.

5.1.5 Testing the code

The `pc_auto-test` command uses `pc_build` for compiling and `pc_run` for running the tests. Here are a few useful options:

```
unix>  pc_auto-test
unix>  pc_auto-test --no-pencil-check   # suppress pencil consistency check
unix>  pc_auto-test --max-level=1       # run only tests in level 0 and 1
unix>  pc_auto-test --time-limit=2m     # kill each test after 2 minutes
```

See `pc_auto-test --help` for a complete list of options.

The `pencil-test` script will use `pc_auto-test` if given the '`--use-pc_auto-test`' or '`-b`' option:

```
unix>  pencil-test --use-pc_auto-test
unix>  pencil-test -b                   # ditto
unix>  pencil-test -b               -Wa,--max-level=1,--no-pencil-check  # quick pencil
```

See `pencil-test --help` for a complete list of options.

5.2 Adapting 'Makefile.src' [obsolete; see Sect. 5.1]

By default, one should use the above described configuration mechanism for compilation. If for whatever reason one needs to work with a modified 'Makefile', there is a mechanism for picking the right set of compiler flags based on the hostname.

To give you an idea of how to add your own machines, let us assume you have several Linux boxes running a compiler f95 that needs the options '-O2 -u', while one of them, *Janus*, runs a compiler f90 which needs the flags '-O3' and requires the additional options '-lmpi -llam' for *MPI*.

The 'Makefile.src' you need will have the following section:

```
### Begin machine dependent

## IRIX64:
[...]  (leave as it is in the original Makefile)
## OSF1:
[...]  (leave as it is in the original Makefile)

## Linux:
[...]  (leave everything from original Makefile and add:)
#FC=f95
#FFLAGS=-O2 -u
#FC=f90              #(Janus)
#FFLAGS=-O3          #(Janus)
#LDMPI=-lmpi -llam   #(Janus)

## SunOS:
[...]  (leave as it is in the original Makefile)
## UNICOS/mk:
[...]  (leave as it is in the original Makefile)
## HI-UX/MPP:
[...]  (leave as it is in the original Makefile)
## AIX:
[...]  (leave as it is in the original Makefile)

### End machine dependent
```

Note 1 There is a script for adapting the Makefile: 'adapt-mkfile'. In the example above, #(Janus) is *not* a comment, but marks this line to be activated (uncommented) by adapt-mkfile if your hostname (`uname -n`) matches 'Janus' or 'janus' (capitalization is irrelevant). You can combine machine names with a vertical bar: a line containing #(onsager|Janus) will be activated on both, *Janus* and *Onsager*.

Note 2 If you want to experiment with compiler flags, or if you want to get things running without setting up the machine-dependent section of the 'Makefile', you can set *make* variables at the command line in the usual manner:

```
src>  make FC=f90 FFLAGS='-fast -u'
```

Table 1: Compiler flags for common compilers. Note that some combinations of OS and compiler require much more elaborate settings; also, if you use MPI, you will have to set `LDMPI`.

Compiler	FC	FFLAGS	CC	CFLAGS
Unix/POSIX:				
GNU	`gfortran`	`-O3`	`gcc`	`-O3 -DFUNDERSC=1`
Intel	`ifort`	`-O2`	`icc`	`-O3 -DFUNDERSC=1`
PGI	`pgf95`	`-O3`	`pgcc`	`-O3 -DFUNDERSC=1`
G95	`g95`	`-O3 -fno-second-underscore`	`gcc`	`-O3 -DFUNDERSC=1`
Absoft	`f95`	`-O3 -N15`	`gcc`	`-O3 -DFUNDERSC=1`
IBM XL	`xlf95`	`-qsuffix=f=f90 -O3`	`xlc`	`-O3 -DFUNDERSC=1`
outdated:				
IRIX Mips	`f90`	`-64 -O3 -mips4`	`cc`	`-O3 -64 -DFUNDERSC=1`
Compaq	`f90`	`-fast -O3`	`cc`	`-O3 -DFUNDERSC=1`

will use the compiler `f90` and the flags '`-fast -u`' for both compilation and linking. Table 1 summarizes flags we use for common compilers.

5.3 Changing the resolution

It is advisable to produce a new run directory each time you run a new case. (This does not include restarts from an old run, of course.) If you have a 32^3 run in some directory '`runA_32a`', then go to its parent directory, i.e.

```
runA_32a>  cd ..
forced>  pc_newrun runA_32a runA_64a
forced>  cd runA_64a/src
forced>  vi cparam.local
```

and edit the '`cparam.local`' for the new resolution.

If you have ever wondered why we don't do dynamic allocation of the main variable (f) array, the main reason it that with static allocation the compiler can check whether we are out of bounds.

5.4 Using a non-equidistant grid

We introduce a non-equidistant grid z_i (by now, this is also implemented for the other directions) as a function $= z(\zeta)$ of an equidistant grid ζ_i with grid spacing $\Delta\zeta = 1$.

The way the parameters are handled, the box size and position are *not* changed when you switch to a non-equidistant grid, i.e. they are still determined by *xyz0* and *Lxyz*.

The first and second derivatives can be calculated by

$$\frac{df}{dz} = \frac{df}{d\zeta}\frac{d\zeta}{dz} = \frac{1}{z'}f'(\zeta)\ , \qquad \frac{d^2f}{dz^2} = \frac{1}{z'^2}f''(\zeta) - \frac{z''}{z'^3}f'(\zeta)\ , \tag{1}$$

which can be written somewhat more compactly using the inverse function $\zeta(z)$:

$$\frac{df}{dz} = \zeta'(z)\,f'(\zeta)\ , \qquad \frac{d^2f}{dz^2} = \zeta'^2(z)\,f''(\zeta) + \zeta''(z)\zeta'(z)f'(\zeta)\ . \tag{2}$$

Internally, the code uses the quantities *dz_1* $\equiv 1/z' = \zeta'(z)$ and *dz_tilde* $\equiv -z''/z'^2 = \zeta''/\zeta'$, and stores them in '`data/proc`N`/grid.dat`'.

The parameters *lequidist* (a 3-element logical array), *grid_func*, *coeff_grid* (a $\geq$ 2-element real array) are used to choose a non-equidistant grid and define the function $z(\zeta)$. So far, one can choose between `grid_function='sinh'`, `grid_function='linear'` (which produces an equidistant grid for testing purposes), and `grid_function='step-linear'`.

The `sinh` profile: For `grid_function='sinh'`, the function $z(\zeta)$ is given by

$$z(\zeta) = z_0 + L_z \frac{\sinh a(\zeta - \zeta_*) + \sinh a(\zeta_* - \zeta_1)}{\sinh a(\zeta_2 - \zeta_*) + \sinh a(\zeta_* - \zeta_1)} \ , \tag{3}$$

where z_0 and $z_0 + L_z$ are the lowest and uppermost levels, ζ_1, ζ_2 are the ζ values representing those levels (normally $\zeta_1 = 0, \zeta_2 = N_z - 1$ for a grid of N_z vertical layers [excluding ghost layers]), and ζ_* is the ζ value of the inflection point of the $\sinh$ function. The z coordinate and ζ value of the inflection point are related via

$$z_* = z_0 + L_z \frac{\sinh a(\zeta_* - \zeta_1)}{\sinh a(\zeta_2 - \zeta_*) + \sinh a(\zeta_* - \zeta_1)} \ , \tag{4}$$

which can be inverted ("after some algebra") to

$$\zeta_* = \frac{\zeta_1 + \zeta_2}{2} + \frac{1}{a} \operatorname{artanh} \left[\left(2 \frac{z_* - z_0}{L_z} - 1 \right) \tanh a \frac{\zeta_2 - \zeta_1}{2} \right] \ . \tag{5}$$

General profile: For a general monotonic function $\psi()$ instead of $\sinh$ we get,

$$z(\zeta) = z_0 + L_z \frac{\psi[a(\zeta - \zeta_*)] + \psi[a(\zeta_* - \zeta_1)]}{\psi[a(\zeta_2 - \zeta_*)] + \psi[a(\zeta_* - \zeta_1)]} \ , \tag{6}$$

and the reference point ζ_* is found by inverting

$$z_* = z_0 + L_z \frac{\psi[a(\zeta_* - \zeta_1)]}{\psi[a(\zeta_2 - \zeta_*)] + \psi[a(\zeta_* - \zeta_1)]} \ , \tag{7}$$

numerically.

Duct flow: The profile function `grid_function='duct'` generates a grid profile for turbulent flow in ducts. For a duct flow, most gradients are steepest close to the walls, and hence very fine resolution is required near the walls. Here we follow the method of [18] and use a Chebyshev-type grid with a cosine distribution of the grid points such that in the y direction they are located at

$$y_j = h \cos \theta_j \ , \tag{8}$$

where

$$\theta_j = \frac{(N_y - j)\,\pi}{N_y - 1}, \quad j = 1, 2, \ldots, N_y \tag{9}$$

and $h = L_y/2$ is the duct half-width.

Currently this method is only adapted for ducts where x is the stream-wise direction, z is in the span-wise direction and the walls are at $y = y_0$ and $y = y_0 + L_y$.

In order to have fine enough resolution, the first grid point should be a distance $\delta = 0.05\, l_w$ from the wall, where

$$l_w = \frac{\nu}{u_\tau}\,, \qquad u_\tau = \sqrt{\frac{\tau_w}{\rho}}\,, \tag{10}$$

and τ_w is the shear wall stress. This is accomplished by using at least

$$N_y \;\geq\; N_y^* \;=\; \frac{\pi}{\arccos\!\left(1-\dfrac{\delta}{h}\right)} + 1 \tag{11}$$

$$\;=\; \pi\sqrt{\frac{h}{2\,\delta}} + 1 - \frac{\pi}{24}\sqrt{\frac{2\,\delta}{h}} + O\left[\left(\frac{\delta}{h}\right)^{3/2}\right] \tag{12}$$

grid points in the y-direction. After rounding up to the next integer value, we find that the truncated condition

$$N_y \geq \left\lceil \pi\sqrt{\frac{h}{2\,\delta}} \right\rceil + 1 \tag{13}$$

(where $\lceil x \rceil$ denotes the *ceiling function*, i.e. the smallest integer equal to, or larger than, x) gives practically identical results.

Example: To apply the `sinh` profile, you can set the following in '`start.in`' (this example is from '`samples/sound-spherical-noequi`'):

```
&init_pars
  [...]
  xyz0  = -2., -2., -2.       ! first corner of box
  Lxyz  =  4.,  4.,  4.       ! box size
  lperi =  F ,  F ,  F        ! periodic direction?
  lequidist = F, F, T         ! non-equidistant grid in z
  xyz_star   = , , -2.        ! position of inflection point
  grid_func  = , , 'sinh'     ! sinh function: linear for small, but
                              ! exponential for large z
  coeff_grid = , , 0.5
/
```

The parameter array *coeff_grid* represents z_* and a; the bottom height z_0 and the total box height L_z are taken from *xyz0* and *Lxyz* as in the equidistant case. The inflection point of the $\sinh$ profile (the part where it is linear) is not in the middle of the box, because we have set *xyz_star(3)* (i. e. z_*) to -2.

5.5 Diagnostic output

Every *it1* time steps (*it1* is a runtime parameter, see Sect. J.2), the code writes monitoring output to *stdout* and, parallel to this, to the file '`data/time_series.dat`'. The variables that appear in this listing and the output format are defined in the file '`print.in`' and can be changed without touching the code (even while the code is running). A simple example of '`print.in`' may look like this:

```
t(F10.3)
urms(E13.4)
rhom(F10.5)
oum
```

which means that the output table will contain time t in the first column formatted as `(F10.3)`, followed by the mean squared velocity, *urms* (i.e. $\langle \boldsymbol{u}^2 \rangle^{1/2}$) in the second column with format `(E13.4)`, the average density *rhom* ($\langle \rho \rangle$, which allows to monitor mass conservation) formatted `(F10.5)` and the kinetic helicity *oum* (that is $\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle$) in the last column with the default format `(E10.2)`.[4] The corresponding diagnostic output will look like

```
----t---------urms--------rhom------oum----
    0.000   4.9643E-03  14.42457 -8.62E-06
    0.032   3.9423E-03  14.42446 -5.25E-06
    0.063   6.8399E-03  14.42449 -3.50E-06
    0.095   1.1437E-02  14.42455 -2.58E-06
    0.126   1.6980E-02  14.42457 -1.93E-06
```

5.6 Data files

5.6.1 Snapshot files

Snapshot files contain the values of all evolving variables and are sufficient to restart a run. In the case of an MHD simulation with entropy equation, for example, the snapshot files will contain the values of velocity, logarithmic density, entropy and the magnetic vector potential.

There are two kinds of snapshot files: the current snapshot and permanent snapshots, both of which reside in the directory '`data/proc`N`/`'. The parameter *isav* determines the frequency at which the *current snapshot* '`data/proc`N`/var.dat`' is written. If you keep this frequency too high, the code will spend a lot of time on I/O, in particular for large jobs; too low a frequency makes it difficult to follow the evolution interactively during test runs. There is also the *ialive* parameter. Setting this to 1 or 10 gives an updated timestep in the files '`data/proc*/alive.info`'. You can put *ialive=0* to turn this off to limit the I/O on your machine.

The *permanent snapshots* '`data/proc*/VAR`N' are written every *dsnap* time units. These files are numbered consecutively from $N = 1$ upward and for long runs they can occupy quite some disk space. On the other hand, if after a run you realize that some additional quantity q would have been important to print out, these files are the only way to reconstruct the time evolution of q without re-running the code.

File structure Snapshot files consist of the following *Fortran records*[5]:

1. variable vector `f` [*mx*×*my*×*mz*×*nvar*]

[4] The format specifiers are like in Fortran, apart from the fact that the `E` format will use standard scientific format, corresponding to the Fortran `1pE` syntax. Seasoned Fortran IV programmers may use formats like `(0pE13.4)` to enjoy nostalgic feelings, or `(1pF10.5)` if they depend on getting wrong numbers.

[5] A *Fortran record* is marked by the 4-byte integer byte count of the data in the record at the beginning and the end, i.e. has the form $\langle N_{\rm bytes}, \texttt{raw\_data}, N_{\rm bytes} \rangle$

2. time t [1], coordinate vectors x [*mx*], y [*my*], z [*mz*], grid spacings δx [1], δy [1], δz [1], shearing-box shift Δy [1]

All numbers (apart from the record markers) are single precision (4-byte) floating point numbers, unless you use double precision (see §5.21, in which case all numbers are 8-byte floating point numbers, while the record markers remain 4-byte integers.

The script `pc_tsnap` allows you to determine the time t of a snapshot file:

```
unix>  pc_tsnap data/proc0/var.dat
data/proc0/var.dat:        t = 8.32456
unix>  pc_tsnap data/proc0/VAR2
data/proc0/VAR2:           t = 2.00603
```

5.7 Video files and slices

We use the terms *video files* and *slice files* interchangeably. These files contain a time series of values of one variable in a given plane. The output frequency of these video snapshots is set by the parameter *dvid* (in code time units).

When output to video files is activated by some settings in '`run.in`' (see example below) and the existence of '`video.in`', slices are written for four planes:

1. x-z plane (y index *iy*; file suffix `.xz`)
2. y-z plane (y index *ix*; suffix `.yz`)
3. x-y plane (y index *iz*; suffix `.xy`)
4. another slice parallel to the x-y plane (y index *iz2*; suffix `.xy2`)

You can specify the position of the slice planes by setting the parameters *ix*, *iy*, *iz* and *iz2* in the namelist *run_pars* in '`run.in`'. Alternatively, you can set the input parameter *slice_position* to one of `'p'` (periphery of box) or `'m'` (middle of box).

In the file '`video.in`' of your run directory, you can choose for which variables you want to get video files; valid choices are listed in § J.4.

The *slice files* are written in each processor directory '`data/proc*/`' and have a file name indicating the individual variable (e. g. '`slice_uu1.yz`' for a slice of u_x in the y-z plane). Before visualizing slices one normally wants to combine the sub-slices written by each processor into one global slice (for each plane and variable). This is done by running '`src/read_videofiles.x`', which will prompt for the variable name, read the individual sub-slices and write global slices to '`data/`' Once all global slices have been assembled you may want to remove the local slices '`data/proc*/slice*`'.

To read all sub-slices demanded in '`video.in`' at once use '`src/read_all_videofiles.x`'. This program doesn't expect any user input and can thus be submitted in computing queues.

For visualization of slices, you can use the *IDL* routines '`rvid_box.pro`', '`rvid_plane.pro`', or '`rvid_line.pro`' which allows the flag '`/png`' for writing *PNG* images that can then be combined into an *MPEG* movie using *mpeg_encode*. Based on '`rvid_box`', you can write your own video routines in *IDL*.

An example Suppose you have set up a run using *entropy.f90* and *magnetic.f90* (most probably together with *hydro.f90* and other modules). In order to animate slices of entropy s and the z-component B_z of the magnetic field, in planes passing through the center of the box, do the following:

1. Write the following lines to '`video.in`' in your run directory:

```
ss
bb
divu
```

2. Edit the namelist *run_pars* in the file '`run.in`'. Request slices by setting *write_slices* and set *dvid* and *slice_position* to reasonable values, say

```
lwrite_slices=T
dvid=0.05
slice_position='m'
```

3. Run the PENCIL CODE:

```
unix>  start.csh
unix>  run.csh
```

4. Run '`src/read_videofiles.x`' to assemble global slice files from those scattered across '`data/proc*/`':

```
unix>  src/read_videofiles.x
  enter name of variable (lnrho, uu1, ..., bb3):  ss
unix>  src/read_videofiles.x
  enter name of variable (lnrho, uu1, ..., bb3):  bb3
```

5. Start *IDL* and run '`rvid_box`':

```
unix>  idl
IDL>  rvid_box,'bb3'
IDL>  rvid_box,'ss',min=-0.3,max=2.
```

   etc.

File structure *Slice files* consist of one Fortran record (see footnote on page 23) for each slice, which contains the data of the variable (without ghost zones), the time t of the snapshot and the position of the sliced variable (e. g. the x position for a y-z slice):

1. data$_1$ [$nx \times ny \times nz$], time t_1 [1], position$_1$ [1]
2. data$_2$ [$nx \times ny \times nz$], time t_2 [1], position$_2$ [1]
3. data$_3$ [$nx \times ny \times nz$], time t_3 [1], position$_3$ [1]

etc.

5.8 Averages

5.8.1 One-dimensional output averaged in two dimensions

In the file 'xyaver.in', z-dependent (horizontal) averages are listed. They are written to the file 'data/xyaverages.dat'. A new line of averages is written every *it1*th time steps.

There is the possibility to output two-dimensional averages. The result then depends on the remaining dimension. The averages are listed in the files 'xyaver.in', 'xzaver.in', and 'yzaver.in' where the first letters indicate the averaging directions. The output is then stored to the files 'data/xyaverages.dat', 'data/xzaverages.dat', and 'data/yzaverages.dat'. The output is written every *it1d*th time steps.

The rms values of the so defined mean magnetic fields are referred to as *bmz*, *bmy* and *bmx*, respectively, and the rms values of the so defined mean velocity fields are referred to as *umz*, *umy*, and *umx*. (The last letter indicates the direction on which the averaged quantity still depends.)

See Sect. 9.2 on how to add new averages.

In idl such xy-averages can be read using the procedure 'pc_read_xyaver'.

5.8.2 Two-dimensional output averaged in one dimension

There is the possibility to output one-dimensional averages. The result then depends on the remaining two dimensions. The averages are listed in the files 'yaver.in', 'zaver.in', and 'phiaver.in' where the first letter indicates the averaging direction. The output is then stored to the files 'data/yaverages.dat', 'data/zaverages.dat', and 'data/phiaverages.dat'.

See Sect. 9.2 on how to add new averages.

Disadvantage: The output files, e.g. 'data/zaverages.dat', can be rather big because each average is just appended to the file.

5.8.3 Azimuthal averages

Azimuthal averages are controlled by the file 'phiaver.in', which currently supports the quantities listed in Sect. J.5. In addition, one needs to set *lwrite_phiaverages*, *lwrite_yaverages*, or *lwrite_zaverages* to *.true.*. For example, if 'phiaver.in' contains the single line

```
b2mphi
```

then you will get azimuthal averages of the squared magnetic field $\boldsymbol{B}^2$.

Azimuthal averages are written every *d2davg* time units to the files 'data/averages/PHIAVGN'. The file format of azimuthal-average files consists of the following *Fortran records*:

1. number of radial points $N_{r,\phi-\mathrm{avg}}$ [1], number of vertical points $N_{z,\phi-\mathrm{avg}}$ [1], number of variables $N_{\mathrm{var},\phi-\mathrm{avg}}$[1], number of processors in z direction [1]

2. time t [1], positions of cylindrical radius $r_{\rm cyl}$ [$N_{r,\phi-{\rm avg}}$] and z [$N_{z,\phi-{\rm avg}}$] for the grid, radial spacing $\delta r_{\rm cyl}$ [1], vertical spacing δz [1]
3. averaged data [$N_{r,\phi-{\rm avg}} \times N_{z,\phi-{\rm avg}}$]
4. label length [1], labels of averaged variables [$N_{{\rm var},\phi-{\rm avg}}$]

All numbers are 4-byte numbers (floating-point numbers or integers), unless you use double precision (see §5.21).

To read and visualize azimuthal averages in *IDL*, use '`$PENCIL_HOME/idl/files/pc_read_phiavg.pro`'

```
IDL>  avg = pc_read_phiavg('data/averages/PHIAVG1')
IDL>  contour, avg.b2mphi, avg.rcyl, avg.z, TITLE='!17B!U2!N!X'
```

or have a look at '`$PENCIL_HOME/idl/phiavg.pro`' for a more sophisticated example.

5.8.4 Time averages

Time averages need to be prepared in the file '`src/ctimeavg.local`', since they use extra memory. They are controlled by the averaging time $\tau_{\rm avg}$ (set by the parameter *tavg* in '`run.in`'), and by the indices *idx_tavg* of variables to average.

Currently, averaging is implemented as exponential (memory-less) average,[6]

$$\langle f \rangle_{t+\delta t} = \langle f \rangle_t + \frac{\delta t}{\tau_{\rm avg}} [f(t+\delta t) - \langle f \rangle_t] \,, \tag{16}$$

which is equivalent to

$$\langle f \rangle_t = \int_{t_0}^{t} e^{-(t-t')/\tau_{\rm avg}} \, f(t') \, dt' \,. \tag{17}$$

Here t_0 is the time of the snapshot the calculation started with, i.e. the snapshot read by the last `run.x` command. Note that the implementation (16) will approximate Eq. (17) only to first-order accuracy in δt. In practice, however, δt is small enough to make this accuracy suffice.

In '`src/ctimeavg.local`', you need to set the number of slots used for time averages. Each of these slots uses `mx` $\times$ `my` $\times$ `mz` floating-point numbers, i.e. half as much memory as each fundamental variable.

For example, if you want to get time averages of all variables, set

```
integer, parameter :: mtavg=mvar
```

[6] At some point we may also implement the more straight-forward average

$$\langle f \rangle_{t+\delta t} = \langle f \rangle_t + \frac{\delta t}{t-t_0+\delta t} [f(t+\delta t) - \langle f \rangle_t] \,, \tag{14}$$

which is equivalent to

$$\langle f \rangle_t = \frac{1}{t-t_0} \int_{t_0}^{t} f(t') \, dt' \,, \tag{15}$$

but we do not expect large differences.

in 'src/ctimeavg.local', and don't set *idx_tavg* in 'run.in'.

If you are only interested in averages of variables 1–3 and 6–8 (say, the velocity vector and the magnetic vector potential in a run with 'hydro.f90', 'density.f90', 'entropy.f90' and 'magnetic.f90'), then set

```
integer, parameter :: mtavg=6
```

in 'src/ctimeavg.local', and set

```
idx_tavg = 1,2,3,6,7,8      ! time-average velocity and vector potential
```

in 'run.in'.

Permanent snapshots of time averages are written every *tavg* time units to the files 'data/proc*/TAVN'. The current time averages are saved periodically in 'data/proc*/timeavg.dat' whenever 'data/proc*/var.dat' is written. The file format for time averages is equivalent to that of the snapshots; see § 5.6.1 above.

5.9 Helper scripts

The 'bin' directory contains a collection of utility scripts, some of which are discussed elsewhere, Here is a list of the more important ones.

adapt-mkfile Activate the settings in a 'Makefile' that apply to the given computer, see §5.2.

auto-test Verify that the code compiles and runs in a set of run directories and compare the results to the reference output. These tests are carried out routinely to ensure that the *svn* version of the code is in a usable state.

cleanf95 Can be use to clean up the output from the Intel x86 Fortran 95 compiler (ifc).

copy-proc-to-proc Used for restarting in a different directory. Example `copy-proc-to-proc seed.dat ../hydro256e`.

copy-snapshots Copy snapshots from a processor-local directory to the global directory. To be started in the background before 'run.x' is invoked. Used by 'start.csh' and 'run.csh' on network connected processors.

pc_copyvar var1 var2 source dest Copies snapshot files from one directory (source) to another (dest). See documentation in file.

pc_copyvar v v dir Copies all 'var.dat' files from current directory to 'var.dat' in 'dir' run directory. Used for restarting in a different directory.

pc_copyvar N v Used to restart a run from a particular snapshot 'VARN'. Copies a specified snapshot 'VARN' to 'var.dat' where N and (optionally) the target run directory are given on the command line.

cvs-add-rundir Add the current run directory to the *svn* repository.

cvsci_run Similar to `cvs-add-rundir`, but it also checks in the '*.in' and 'src/*.local' files. It also checks in the files 'data/time_series.dat', 'data/dim.dat' and 'data/index.pro' for subsequent processing in *IDL* on another machine. This is particularly useful if collaborators want to check each others' runs.

dx_* These script perform several data collection or reformatting exercises required to read particular files into *DX*. They are called internally by some of the *DX* macros in the 'dx/macros/' directory.

getconf.csh See § 4.3.4

gpgrowth Plot simple time evolution with Gnuplot's ASCII graphics for fast orientation via a slow modem line.

local Materialize a symbolic link

mkcparam Based on 'Makefile' and 'Makefile.local', generate 'src/cparam.inc', which specifies the number *mvar* of fundamental variables, and *maux* of auxiliary variables. Called by the 'Makefile'.

pc_mkdatadir Creates a link to a data directory in a suitable workspace. By default this is on '/var/tmp/', but different locations are specified for different machines.

mkdotin Generate minimal 'start.in', 'run.in' files based on 'Makefile' and 'Makefile.local'.

mkinpars Wrapper around 'mkdotin' — needs proper documentation.

mkproc-tree Generates a multi-processor('proc$N$'/), directory structure. Useful when copying data files in a processor tree, such as slice files.

mkwww Generates a template HTML file for describing a run of the code, showing input parameters and results.

move-slice Moves all the slice files from a processor tree structure, 'proc$N$/', to a new target tree creating directories where necessary.

nl2idl Transform a Fortran *namelist* (normally the files 'param.nml', 'param2.nml' written by the code) into an *IDL* structure. Generates an *IDL* file that can be sourced from 'start.pro' or 'run.pro'.

pacx-adapt-makefile Version of adapt-makefile for highly distributed runs using PACX MPI.

pc_newrun Generates a new run directory from an old one. The new one contains a copy of the old '*.local' files, runs pc_setupsrc, and makes also a copy of the old '*.in' and 'k.dat' files.

pc_newscan Generates a new scan directory from an old one. The new one contains a copy of the old, e.g. the one given under 'samples/parameter_scan'. Look in the 'README' file for details.

pc_inspectrun Check the execution of the current run: prints legend and the last few lines of the 'time_series.dat' file. It also appends this result to a file called 'SPEED', which contains also the current wall clock time, so you can work out the speed of the code (without being affected by i/o time).

read_videofiles.csh The script for running read_videofiles.x.

remote-top Create a file 'top.log' in the relevant 'proc$N$/' directory containing the output of top for the appropriate processor. Used in batch scripts for multi-processor runs.

run.csh The script for producing restart files with the initial condition; see § 4.3.4

scpdatadir Make a tarball of data directory, 'data/' and use scp to secure copy to copy it to the specified destination.

pc_setupsrc Link 'start.csh', 'run.csh' and 'getconf.csh' from '$PENCIL_HOME/bin'. Generate 'src/' if necessary and link the source code files from '$PENCIL_HOME/src' to that directory.

start.csh The script for initializing the code; see § 4.3.4

summarize-history Evaluate 'params.log' and print a history of changes.

timestr Generate a unique time string that can be appended to file names from shell scripts through the backtick mechanism.

pc_tsnap Extract time information from a snapshot file, 'VARN'.

There are several additional scripts on 'pencil-code/utils'. Some are located in separate folders according to users. There could be redundancies, but it is often just as easy to write your own new script than figuring out how something else works.

5.10 RELOAD and STOP files

The code periodically (every *it* time steps) checks for the existence of two files, 'RELOAD' and 'STOP', which can be used to trigger certain behavior.

Reloading run parameters In the directory where you started the code, create the file 'RELOAD' with

```
unix>  touch RELOAD
```

to force the code to re-read the runtime parameters from 'run.in'. This will happen the next time the code is writing monitoring output (the frequency of this happening is controlled by the input parameter *it*, see Sect. 5.12).

[Occasionally, however, and only on some machines, one of the processors may not survive the MPI barrier and stops; we will need to look further into this.]

Each time the parameters are reloaded, the new set of parameters is appended (in the form of *namelists*) to the file 'data/params.log' together with the time t, so you have a full record of your changes. If 'RELOAD' contains any text, its first line will be written to 'data/params.log' as well, which allows you to annotate changes:

```
unix>  echo "Reduced eta to get fields growing" > RELOAD
```

Use the command summarize-history to print a history of changes.

Stopping the code In the directory where you started the code, create the file 'STOP' with

```
unix>  touch STOP
```

to stop the code in a controlled manner (it will write the latest snapshot). Again, the action will happen the next time the code is writing monitoring output.

5.11 RERUN and NEWDIR files

After the code finishes (e.g., when the final timestep number is reached or when a 'STOP' file is found), the 'run.csh' script checks whether there is a 'RERUN' file. If so, the code will simply run again, perhaps even after you have recompiled the code. This is useful in the development phase when you changed something in the code, so you don't need to wait for a new slot in the queue!

Even more naughty, as Tony says, is the 'NEWDIR' file, where you can enter a new directory path (relative path is ok, e.g. ../conv-slab). If nothing is written in this file (e.g. via touch NEWDIR) it stays in the same directory. On distributed machines, the 'NEWDIR' method will copy all the 'VAR#' and 'var.dat' files back to and from the sever. This can be useful if you want to run with new data files, but you better do it in a separate directory, because with 'NEWDIR' the latest data from the code are written back to the server before running again.

Oh, by the way, if you want to be sure that you haven't messed up the content of the pair of 'NEWDIR' files, you may want to try out the pc_jobtransfer command. It writes the decisive 'STOP' file only after the script has checked that the content of the two 'NEWDIR' files points to existing run directory paths, so if the new run crashes, the code returns safely to the old run directory. I'm not sure what Tony would say now, but this is now obviously extremely naughty.

5.12 Start and run parameters

All input parameters in 'start.in' and 'run.in' are grouped in Fortran *namelists*. This allows arbitrary order of the parameters (*within* the given namelist; the namelists need no longer be in the correct order), as well as enhanced readability through inserted Fortran comments and whitespace. One namelist (*init_pars* / *run_pars*) contains general parameters for initialization/running and is always read in. All other namelists are specific to individual modules and will only be read if the corresponding module is used.

The syntax of a namelist (in an input file like 'start.in') is

```
&init_pars
  ip=5, Lxyz=2,4,2
/
```

— in this example, the name of the namelist is *init_pars*, and we read just two variables (all other variables in the namelist retain their previous value): *ip*, which is set to 5, and *Lxyz*, which is a vector of length three and is set to $(2, 4, 2)$.

While all parameters from the namelists can be set, in most cases reasonable default values are preset. Thus, the typical file 'start.in' will only contain a minimum set of variables or (if you are *very* minimalistic) none at all. If you want to run a particular

problem, it is best to start by modifying an existing example that is close to your application.

As an example, we give here the start parameters for 'samples/helical-MHDturb'

```
&init_pars
  cvsid='$Id:$',                ! identify version of start.in
  xyz0  = -3.1416, -3.1416, -3.1416, ! first corner of box
  Lxyz  =  6.2832,  6.2832,  6.2832, ! box size
  lperi =  T     ,  T     ,  T     , ! periodic in x, y, z
  random_gen='nr_f90'
/
&hydro_init_pars
/
&density_init_pars
  gamma=1.
/
&magnetic_init_pars
  initaa='gaussian-noise', amplaa=1e-4
/
```

The three entries specifying the location, size and periodicity of the box are just given for demonstration purposes here — in fact a periodic box from $-\pi$ to $-\pi$ in all three directions is the default. In this run, for reproducibility, we use a random number generator from the Numerical Recipes [24], rather than the compiler's built-in generator. The adiabatic index γ is set explicitly to 1 (the default would have been 5/3) to achieve an isothermal equation of state. The magnetic vector potential is initialized with uncorrelated, normally distributed random noise of amplitude 10^{-4}.

The run parameters for 'samples/helical-MHDturb' are

```
&run_pars
  cvsid='$Id:$',                ! identify version of start.in
  nt=10, it1=2, cdt=0.4, cdtv=0.80, isave=10, itorder=3
  dsnap=50, dvid=0.5
  random_gen='nr_f90'
/
&hydro_run_pars
/
&density_run_pars
/
&forcing_run_pars
  iforce='helical', force=0.07, relhel=1.
/
&magnetic_run_pars
  eta=5e-3
/
&viscosity_run_pars
  nu=5e-3
/
```

Here we run for $nt = 10$ timesteps, every second step, we write a line of diagnostic output; we require the time step to keep the advective *Courant number* ≤ 0.4 and the diffusive

Courant number ≤ 0.8, save 'var.dat' every 20 time steps, and use the 3-step time-stepping scheme described in Appendix H.4 (the Euler scheme *itorder*$= 1$ is only useful for tests). We write permanent snapshot file 'VARN' every *dsnap*$= 50$ time units and 2d slices for animation every *dvid*$= 0.5$ time units. Again, we use a deterministic random number generator. Viscosity ν and magnetic diffusivity η are set to 5×10^{-3} (so the mesh Reynolds number is about $u_{\rm rms}\delta x/\nu = 0.3 \times (2\pi/32)/5 \times 10^{-3} \approx 12$, which is in fact rather a bit to high). The parameters in *forcing_run_pars* specify fully helical forcing of a certain amplitude.

A full list of input parameters is given in Appendix J.

5.13 Physical units

Many calculations are unit-agnostic, in the sense that all results remain the same independent of the unit system in which you interpret the numbers. E. g. if you simulate a simple hydrodynamical flow in a box of length $L = 1.$ and get a maximum velocity of $u_{\max} = 0.5$ after $t = 3$ time units, then you may interpret this as $L = 1\,\mathrm{m}$, $u_{\max} = 0.5\,\mathrm{m/s}$, $t = 3\,\mathrm{s}$, or as $L = 1\,\mathrm{pc}$, $u_{\max} = 0.5\,\mathrm{pc/Myr}$, $t = 3\,\mathrm{Myr}$, depending on the physical system you have in mind. The units you are using must of course be consistent, thus in the second example above, the units for diffusivities would be $\mathrm{pc}^2/\mathrm{Myr}$, etc.

The units of magnetic field and temperature are determined by the values $\mu_0 = 1$ and $c_p = 1$ used internally by the code[7]. This means that if your units for density and velocity are $[\rho]$ and $[v]$, then magnetic fields will be in

$$[B] = \sqrt{\mu_0\,[\rho]\,[v]^2}\,, \tag{18}$$

and temperatures are in

$$[T] = \frac{[v]^2}{c_p} = \frac{\gamma-1}{\gamma}\,\frac{[v]^2}{\mathcal{R}/\mu}\,. \tag{19}$$

Table 2: Units of magnetic field and temperature for some choices of $[\rho]$ and $[v]$ according to Eqs. (18) and (19). Values are for a monatomic gas ($\gamma = 5/3$) of mean atomic weight $\bar{\mu}_{\rm g} = \bar{\mu}/1\,\mathrm{g}$ in grams.

$[\rho]$	$[v]$	$[B]$	$[T]$
$1\,\mathrm{kg/m^3}$	$1\,\mathrm{m/s}$	$1.12\,\mathrm{mT} = 11.2\,\mathrm{G}$	$\left(\frac{\bar{\mu}_{\rm g}}{0.6}\right) \times 2.89 \times 10^{-5}\,\mathrm{K}$
$1\,\mathrm{g/cm^3}$	$1\,\mathrm{cm/s}$	$3.54 \times 10^{-4}\,\mathrm{T} = 3.54\,\mathrm{G}$	$\left(\frac{\bar{\mu}_{\rm g}}{0.6}\right) \times 2.89\,\mathrm{nK}$
$1\,\mathrm{g/cm^3}$	$1\,\mathrm{km/s}$	$35.4\,\mathrm{T} = 354\,\mathrm{kG}$	$\left(\frac{\bar{\mu}_{\rm g}}{0.6}\right) \times 28.9\,\mathrm{K}$
$1\,\mathrm{g/cm^3}$	$10\,\mathrm{km/s}$	$354\,\mathrm{T} = 3.54\,\mathrm{MG}$	$\left(\frac{\bar{\mu}_{\rm g}}{0.6}\right) \times 2\,890\,\mathrm{K}$

For some choices of density and velocity units, Table 2 shows the resulting units of magnetic field and temperature.

[7] Note that $c_p = 1$ is only assumed if you use the module *noionization.f90*. If you work with *ionization.f90*, temperature units are specified by *unit_temperature* as described below.

On the other hand, as soon as material equations are used (e.g. one of the popular parameterizations for radiative losses, Kramers opacity, Spitzer conductivities or ionization, which implies well-defined ionization energies), the corresponding routines in the code need to know the units you are using. This information is specified in '`start.in`' or '`run.in`' through the parameters *unit_system*, *unit_length*, *unit_velocity*, *unit_density* and *unit_temperature*[8] like e. g.

```
unit_system='SI',
unit_length=3.09e16, unit_velocity=978.  ! [l]=1pc, [v]=1pc/Myr
```

Note: The default unit system is `unit_system='cgs'` which is a synonym for `unit_system='Babylonian cubits'`.

5.14 Minimum amount of viscosity

We emphasize that, by default, the code works with constant diffusion coefficients (viscosity ν, thermal diffusivity χ, magnetic diffusivity η, or passive scalar diffusivity $\mathcal{D}$). If any of these numbers is too small, you would need to have more meshpoints to get consistent numerical solutions; otherwise the code develops wiggles ('ringing') and will eventually crash. A useful criterion is given by the mesh Reynolds number based on the maximum velocity,

$$\mathrm{Re}_{\mathrm{mesh}} = \max(|\boldsymbol{u}|)\max(\delta x, \delta y, \delta z)/\nu, \tag{20}$$

which should not exceed a certain value which can be problem-dependent. Often the largest possible value of $\mathrm{Re}_{\mathrm{mesh}}$ is around 5. Similarly there exist mesh Péclet and mesh magnetic Reynolds numbers that should not be too large.

Note that in some cases, 'wiggles' in $\ln\rho$ will develop despite sufficiently large diffusion coefficients, essentially because the continuity equation contains no dissipative term. For convection runs (but not only for these), we have found that this can often be prevented by *upwinding*, see Sect. H.2.

If the Mach number of the code approaches unity, i.e. if the rms velocity becomes comparable with the speed of sound, shocks may form. In such a case the mesh Reynolds number should be smaller. In order to avoid excessive viscosity in the unshocked regions, one can use the so-called shock viscosity (Sect. 6.6.1) to concentrate the effects of a low mesh Reynolds number to only those areas where it is necessary.

5.15 The time step

5.15.1 The usual RK-2N time step

RK-2N refers to the third order Runge-Kutta scheme by Williamson (1980) with a memory consumption of two chunks. Therefore the 2N in the name.

The time step is normally specified as Courant time step through the coefficients *cdt* ($c_{\delta t}$), *cdtv* ($c_{\delta t,\mathrm{v}}$) and *cdts* ($c_{\delta t,\mathrm{s}}$). The resulting Courant step is given by

$$\delta t = \min\left(c_{\delta t}\frac{\delta x_{\min}}{U_{\max}}, c_{\delta t,\mathrm{v}}\frac{\delta x_{\min}^2}{D_{\max}}, c_{\delta t,\mathrm{s}}\frac{1}{H_{\max}}\right), \tag{21}$$

[8] Note: the parameter *unit_temperature* is currently only used in connection with *ionization.f90*. If you are working with *noionization.f90*, the temperature unit is completely determined by Eq. (19) above.

where

$$\delta x_{\min} \equiv \min(\delta x, \delta y, \delta z) \; ; \tag{22}$$

$$U_{\max} \equiv \max\left(|\boldsymbol{u}| + \sqrt{c_{\rm s}^2 + v_{\rm A}^2} \right) \; , \tag{23}$$

$c_{\rm s}$ and $v_{\rm A}$ denoting sound speed and Alfvén speed, respectively;

$$D_{\max} = \max(\nu, \gamma\chi, \eta, D), \tag{24}$$

where ν denotes kinematic viscosity, $\chi = K/(c_p\rho)$ thermal diffusivity and η the magnetic diffusivity; and

$$H_{\max} = \max\left(\frac{2\nu \mathsf{S}^2 + \zeta_{\rm shock}(\boldsymbol{\nabla}\cdot\boldsymbol{u})^2 + ...}{c_v T} \right), \tag{25}$$

where dots indicate the presence of other terms on the rhs of the entropy equation.

To fix the time step δt to a value independent of velocities and diffusivities, explicitly set the run parameter *dt*, rather than *cdt* or *cdtv* (see p. 156).

If the time step exceeds the viscous time step the simulation may actually run ok for quite some time. Inspection of images usually helps to recognize the problem. An example is shown in Fig. 4.

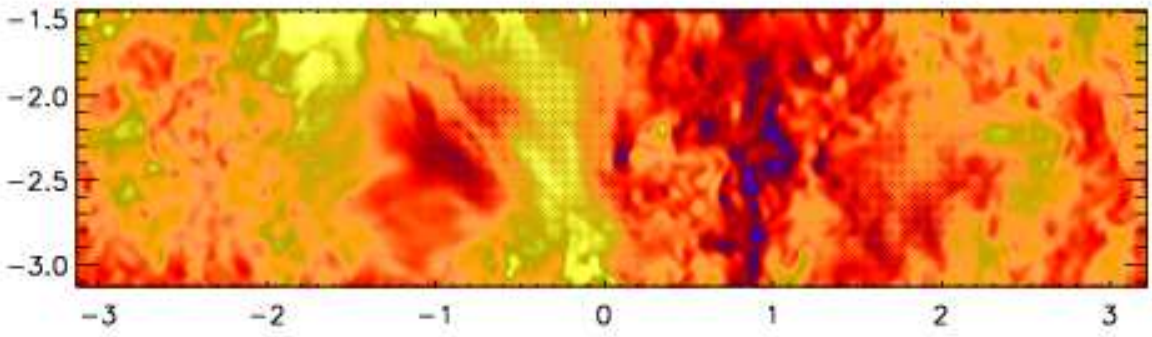

Figure 4: Example of a velocity slice from a run where the time step is too long. Note the spurious checkerboard modulation in places, for example near $x = -0.5$ and $-2.5 < y < -1.5$. This is an example of a hyperviscous turbulence simulations with 512^3 meshpoints and a third order hyperviscosity of $\nu_3 = 5 \times 10^{-12}$. Hyperviscosity is explained in the Appendix E.

Timestepping is accomplished using the Runge-Kutta 2N scheme. Regarding details of this scheme see Sect. H.4.

5.15.2 The Runge-Kutta-Fehlberg time step

A fifth order Runge-Kutta-Fehlberg time stepping procedure is available. It is used mostly for chemistry application, often together with the double precision option. In order to make this work, you need to compile with

```
TIMESTEP  =  timestep_rkf
```

in '`src/Makefile.local`'. In addition, you must put `itorder=5` in '`run.in`'. An example application is '`samples/1d-tests/H2_flamespeed`'. This procedure is still experimental.

5.16 Boundary conditions

5.16.1 Where to specify boundary conditions

In most tests that come with the PENCIL CODE, boundary conditions are set in 'run.in', which is a natural choice. However, this may lead to unexpected initial data written by 'start.x', since when you start the code (via 'start.csh'), the boundary conditions are unknown and 'start.x' will then fill the ghost zones assuming periodicity (the default boundary condition) in all three directions. These ghost data will never be used in a calculation, as 'run.x' will apply the boundary conditions before using any ghost-zone values.

To avoid these periodic conditions in the initial snapshot, you can set the boundary conditions in 'start.in' already. In this case, they will be inherited by 'run.x', unless you also explicitly set boundary conditions in 'run.in'.

5.16.2 How to specify boundary conditions

Boundary conditions are implemented through three layers of *ghost points* on either boundary, which is quite a natural choice for an MPI code that uses ghost zones for representing values located on the neighboring processors anyway. The desired type of boundary condition is set through the parameters *bc*{*x,y,z*} in 'run.in'; the nomenclature used is as follows. Set *bc*{*x,y,z*} to a sequence of letters like

```
bcx = 'p','p','p', 'p',  'p'
```

for periodic boundaries, or

```
bcz = 's','s','a','a2','c1:c2'
```

for non-periodic ones. Each element corresponds to one of the variables, which are those of the variables u_x, u_y, u_z, $\ln\rho$, s/c_p, A_x, A_y, A_z, $\ln c$ that are actually used *in this order*. The following conditions are available:

'p' periodic boundary condition

'a' antisymmetric condition w. r. t. the boundary, i. e. vanishing value

's' symmetric condition w. r. t. the boundary, i. e. vanishing first derivative

'a2' antisymmetry w. r. t. the arbitrary value on the boundary, i. e. vanishing second derivative

'c1' special boundary condition for $\ln\rho$ and s: constant heat flux through the boundary

'c2' special boundary condition for s: constant temperature at the boundary — requires boundary condition a2 for $\ln\rho$

'cT' special boundary condition for s or $\ln\ T$: constant temperature at the boundary (for arbitrarily set $\ln\rho$)

'ce' special boundary condition for s: set temperature in ghost points to value on boundary (for arbitrarily set $\ln\rho$)

'db' low-order one-sided derivatives ("no boundary condition") for density

'she' shearing-sheet boundary condition (default when the module *Shear* is used)

'g' force the value of the corresponding field on vertical boundaries (should be used in combination with the force_lower_bound and force_upper_bound flags set in the namelist *init_pars*)

'hs' special boundary condition for $\ln \rho$ and s which enforces hydrostatic equilibrium on vertical boundaries

The extended syntax a:b (e. g. 'c1:c2') means: use boundary condition a at the left/lower boundary, but b at the right/upper one.

If you build a new 'run.in' file from another one with a different number of variables (*noentropy* vs. *entropy*, for example), you need to remember to adjust the *length* of the arrays *bcx* to *bcz*. The advantage of the present approach is that it is very easy to exchange all boundary conditions by a new set of conditions in a particular direction (for example, make everything periodic, or switch off shearing sheet boundary conditions and have stress-free instead).

5.17 Restarting a simulation

When a run stops at the end of a simulation, you can just resubmit the job again, and it will start from the latest snapshot saved in 'data/proc*/var.dat'. The value of the latest time is saved in a separate file, 'data/proc*/time.dat'. On parallel machines it is possible that some (or just one) of the 'var.dat' are corrupt; for example after a system crash. Check for file size and date, and restart from a good 'VAR'N file instead.

If you want to run on a different machine, you just need to copy the 'data/proc*/var.dat' (and, just to be sure) 'data/proc*/time.dat') files into a new directory tree. You may also need the 'data/proc*/seed.dat' files for the random number generator. The easiest way to get all these other files is to run start.csh again on the new machine (or in a new directory) and then to overwrite the 'data/proc*/var.dat' files with the correct ones.

For restarting from runs that didn't have magnetic fields, passive scalar fields, or test fields, see Sect. F.2.

5.18 One- and two-dimensional runs

If you want to run two-dimensional problems, set the number of mesh points in one direction to unity, e.g. *nygrid*=1 or *nzgrid*=1 in 'cparam.local'. Remember that the number of mesh points is still divisible by the number of processors. For 2D-runs, it is also possible to write only 2D-snapshots (i.e. VAR files written only in the considered (x, y) or (x, z) plane, with a size seven times smaller as we do not write the third unused direction). To do that, please add the logical flag 'lwrite_2d=T' in the namelist *init_pars* in 'start.in'.

Similarly, for one-dimensional problems, set, for example, *nygrid*=1 and *nzgrid*=1 in 'cparam.local'. You can even do a zero-dimensional run, but then you better set *dt* (rather than *cdt*), because there is no Courant condition for the time step.

See *0d, 1d, 2d, and 3d tests* with examples.

5.19 Visualization

5.19.1 Gnuplot

Simple visualization can easily be done using *Gnuplot* (`http://www.gnuplot.info`), an open-source plotting program suitable for two-dimensional plots.

For example, suppose you have the variables

```
---it-----t-------dt-------urms-----umax-----rhom-----ssm------dtc---
```

in '`time_series.dat`' and want to plot $u_{\rm rms}(t)$. Just start gnuplot and type

```
gnuplot>  plot "data/time_series.dat" using 2:4 with lines
```

If you work over a slow line and want to see both $u_{\rm rms}(t)$ and $u_{\rm max}(t)$, use ASCII graphics:

```
gnuplot>  set term dump
gnuplot>  set logscale y
gnuplot>  plot "data/time_series.dat" using 2:4 title "urms", \
gnuplot>       "data/time_series.dat" using 2:5 title "umax"
```

5.19.2 Data explorer

DX (*data explorer*; `http://www.opendx.org`) is an open-source tool for visualization of three-dimensional data.

The PENCIL CODE provides a few networks for *DX*. It is quite easy to read in a snapshot file from *DX* (the only tricky thing is the four extra bytes at the beginning of the file, representing a Fortran record marker), and whenever you run '`start.x`', the code writes a file '`var.general`' that tells *DX* all it needs to know about the data structure.

As a starting point for developing your own *DX* programs or *networks*, you can use a few generic *DX* scripts provided in the directory '`dx/basic/`'. From the run directory, start *DX* with

```
unix>  dx -edit $PENCIL_HOME/dx/basic/lnrho
```

to load the file '`dx/basic/lnrho.net`', and execute it with Ctl-o or Execute → Execute Once. You will see a set of iso-surfaces of logarithmic density. If the viewport does not fit to your data, you can reset it with Ctl-f. To rotate the object, drag the mouse over the Image window with the left or right mouse button pressed. Similar networks are provided for entropy ('`ss.net`'), velocity ('`uu.net`') and magnetic field ('`bb.net`').

When you expand these simple networks to much more elaborate ones, it is probably a good idea to separate the different tasks (like Importing and Selecting, visualizing velocity, visualizing entropy, and Rendering) onto separate pages through Edit → Page.

Note Currently, *DX* can only read in data files written by one single processor, so from a multi-processor run, you can only visualize one subregion at a time.

5.19.3 GDL

GDL, also known as *Gnu Data Language* is a free visualization package that can be found at `http://gnudatalanguage.sourceforge.net/`. It aims at replacing the very expensive *IDL* package (see S. 5.19.4). For the way we use IDL for the Pencil Code, compatibility is currently not completely sufficient, but you can use GDL for many of the visualization tasks. If you get spurious "Error opening file" messages, you can normally simply ignore them.

This section tells you how to get started with using GDL for visualization.

Setup As of GDL 0.9 – at least the version packed with Ubuntu Jaunty (9.10) – you will need to add GDL's '`examples/pro/`' directory to your *!PATH* variable. So the first call after starting *GDL* should be

```
GDL>  .run setup_gdl
```

Starting visualization There are mainly two possibilities for visualization: using a simple GUI or loading the data with `pc_read` and work with it interactively. Please note that the GUI was written and tested only with IDL, see §5.19.4.

Here, the `pc_read` family of IDL routines to read the data is described. Try

```
GDL>  pc_read
```

to get an overview.

To plot a time series, use

```
GDL>  pc_read_ts, OBJECT=ts
GDL>  help, ts, /STRUCT  ;; (to see which slots are available)
GDL>  plot, ts.t, ts.umax
GDL>  oplot, ts.t, ts.urms
```

Alternatively, you could simply use the '`ts.pro`' script:

```
GDL>  .run ts
```

To work with data from '`var.dat`' and similar snapshot files, you can e.g. use the following routines:

```
GDL>  pc_read_dim, OBJECT=dim
GDL>  $$PENCIL_HOME/bin/nl2idl -d -m ./data/param.nml> ./param.pro
GDL>  pc_read_param, OBJECT=par
GDL>  pc_read_grid, OBJECT=grid
GDL>  pc_read_var, OBJECT=var
```

Having thus read the data structures, we can have a look at them to see what information is available:

```
GDL>  help, dim, /STRUCT
GDL>  help, par, /STRUCT
GDL>  help, grid, /STRUCT
GDL>  help, var, /STRUCT
```

To visualize data, we can e.g. do[9]

```
GDL>  plot, grid.x, var.ss[*, dim.ny/2, dim.nz/2]
GDL>  contourfill, var.ss[*, *, dim.nz/2], grid.x, grid.y

GDL>  ux_slice = var.uu[*, *, dim.nz/2, 0]
GDL>  uy_slice = var.uu[*, *, dim.nz/2, 1]
GDL>  wdvelovect, ux_slice, uy_slice, grid.x, grid.y

GDL>  surface, var.lnrho[*, *, dim.nz/2, 0]
```

See also Sect. 5.19.4.

5.19.4 IDL

IDL is a commercial visualization program for two-dimensional and simple three-dimensional graphics. It allows to access and manipulate numerical data in a fashion quite similar to how Fortran handles them.

In '`$PENCIL_HOME/idl`', we provide a number of general-purpose *IDL* scripts that we are using all the time in connection with the PENCIL CODE. While *IDL* is quite an expensive software package, it is quite useful for visualizing results from numerical simulations. In fact, for many applications, the 7-minute demo version of *IDL* is sufficient.

Visualization in IDL The Pencil Code GUI is a data post-processing tool for the usage on a day-to-day basis. It allows fast inspection of many physical quantities, as well as advanced features like horizontal averages, streamline tracing, freely orientable 2D-slices, and extraction of cut images and movies. To use the Pencil Code GUI, it is sufficient to run:

```
unix>  idl
IDL>   .r pc_gui
```

If you like to load only some subvolume of the data, like any 2D-slices from the given data snapshots, or 3D-subvolumes, it is possible to choose the corresponding options in the file selector dialog. The Pencil Code GUI offers also some options to be set on the command-line, please refer to their description in the source code.

There are also other small GUIs available, e.g. the file 'time-series.dat' can easily be analyzed with the command:

```
unix>  idl
IDL>   pc_show_ts
```

The easiest way to derive physical quantities at the command-line is to use one of the many `pc_read_var`-variants (`pc_read_var_raw` is recommended for large datasets because of its high efficiency regarding computation and memory usage) for reading the data. With that, one can make use of `pc_get_quantity` to derive any implemented physical quantity. Packed in a script, this is the recommended way to get reproducible results, without any chance for accidental errors on the interactive IDL command-line.

[9] If `contourfill` produces just contour lines instead of a color-coded plot, your version of GDL is too old. E.g. the version shipped with Ubuntu 9.10 is based on GDL 0.9rc1 and has this problem.

Alternatively, by using the command-line to see the time evolution of e.g. velocity and magnetic field (if they are present in you run), start *IDL* [10] and run '`ts.pro`':

```
unix>  idl
IDL>   .run ts
```

The *IDL* script '`ts.pro`' reads the time series data from '`data/time_series.dat`' and sorts the column into the structure *ts*, with the slot names corresponding to the name of the variables (taken from the header line of '`data/time_series.dat`'). Thus, you can refer to time as `ts.t`, to the rms velocity as `ts.urms`, and in order to plot the mean density as a function of time, you would simply type

```
IDL>  plot, ts.t, ts.rhom
```

The basic command sequence for working with a snapshot is:

```
unix>  idl
IDL>   .run start
IDL>   .run r
IDL>   [specific commands]
```

You run '`start.pro`' once to initialize (or reinitialize, if the mesh size has changed, for example) the fields and read in the startup parameters from the code. To read in a new snapshot, run '`r.pro`' (or '`rall.pro`', see below).

If you are running in parallel on several processors, the data are scattered over different directories. To reassemble everything in *IDL*, use

```
IDL>  .r rall
```

instead of `.r r` (here, `.r` is a shorthand for `.run`). The procedure '`rall.pro`' reads (and assembles) the data from all processors and correspondingly requires large amounts of memory for very large runs. If you want to look at just the data from one processor, use '`r.pro`' instead.

If you need the magnetic field or the current density, you can calculate them in IDL by [11]

```
IDL>  bb=curl(aa)
```

[10] If you run IDL from the command line, you will highly appreciate the following tip: IDL's command line editing is broken beyond hope. But you can fix it, by running IDL under `rlwrap`, a wrapper for the excellent GNU *readline* library.

Download and install `rlwrap` from `http://utopia.knoware.nl/~hlub/uck/rlwrap/` (on some systems you just need to run '`emerge rlwrap`', or '`apt-get install rlwrap`'), and alias your `idl` command:

```
csh>   alias idl 'rlwrap -a -c idl'
bash>  alias idl='rlwrap -a -c idl'
```

From now on, you can

- use long command lines that correctly wrap around;
- type the first letters of a command and then PageUp to recall commands starting with these letters;
- capitalize, uppercase or lowercase a word with Esc-C, Esc-U, Esc-L;
- use command line history across IDL sessions (you might need to create '`~/.idl_history`' for this);
- complete file names with Tab (works to some extent);
- ... use all the other *readline* features that you are using in `bash`, `octave`, `bc`, `gnuplot`, `ftp`, etc.

[11] Keep in mind that `jj=curl(bb)` would use iterated first derivatives instead of the second derivatives and thus be numerically less accurate than `jj=curl2(aa)`, particularly at small scales.

```
IDL>  jj=curl2(aa)
```

By default one is reading always the current snapshot 'data/proc$N$/var.dat'; if you want to read one of the permanent snapshots, use (for example)

```
IDL>  varfile='VAR2'
IDL>  .r r (or .r rall)
```

See Sect. 5.6.1 for details on permanent snapshots.

With 'r.pro', you can switch the part of the domain by changing the variable *datadir*:

```
IDL>  datadir='data/proc3'
IDL>  .r r
```

will read the data written by processor 3.

Reading data into IDL arrays or structures As an alternative to the method described above, there is also the possibility to read the data into structures. This makes some more operations possible, e.g. reading data from an IDL program where the command .r is not allowed.

An efficient and still scriptable way would look like the following:

```
IDL>  pc_read_var_raw, obj=var, tags=tags
IDL>  bb = pc_get_quantity ('B', var, tags)
IDL>  jj = pc_get_quantity ('j', var, tags)
```

This reads the data into an array 'var', as well as the array indices of the contained physical variables into a separate structure 'tags'. To use a caching mechanism within pc_get_quantity, please refer to the documentation and the examples contained in 'pencil-code/idl/pc_get_quantity.pro', where you can also start adding not yet implemented physical quantities.

To read a snapshot 'VAR10' into the IDL structure ff, type the following command

```
IDL>  pc_read_var, obj=ff, varfile='VAR10', /trimall
```

The option /trimall removes ghost zones from the data. A number of other options are documented in the source code of pc_read_var. You can see what data the structure contains by using the command tag_names

```
IDL>  print, tag_names(ff)
T X Y Z DX DY DZ UU LNRHO AA
```

One can access the individual variables by typing ff.varname, e.g.

```
IDL>  help, ff.aa
<Expression>    FLOAT     = Array[32, 32, 32, 3]
```

There are a number of files that read different data into structures. They are placed in the directory $PENCIL_HOME/idl/files. Here is a list of files (including suggested options to call them with)

- pc_read_var_raw, obj=var, tags=tags
  Efficiently read a snapshot into an array.
- pc_read_var, obj=ff, /trimall
  Read a snapshot into a structure.

- `pc_read_ts, obj=ts`
  Read the time series into a structure.
- `pc_read_xyaver, obj=xya`
  `pc_read_xzaver, obj=xza`
  `pc_read_yzaver, obj=yza`
  Read 1-D time series into a structure.
- `pc_read_const, obj=cst`
  Read code constants into a structure.
- `pc_read_pvar, obj=fp`
  Read particle data into a structure.
- `pc_read_param, obj=par`
  Read startup parameters into a structure.
- `pc_read_param, obj=par2, /param2`
  Read runtime parameters into a structure.

Other options are documented in the source code of the files.

For some examples on how to use these routines, see Sect. 5.19.3.

5.19.5 Python

Pencil also supports visualization using *python*. A number of scripts for such are placed in the subfolder '`$PENCIL_HOME/python`'. A README file placed under that subfolder contains the information needed to read Pencil output data into python.

To start rapidly under Python:

1. you must install the matplotlib library (a package often called python-matplotlib)
2. normally, numpy is installed as a dependency
3. the PYTHONPATH is initialised when sourcing 'sourceme.*'
4. for us, the best practical way to use Python is to start the ipython shell so its installation is a good idea
5. you must load all of the Python pencil modules using 'import pencil as pc'
6. you run a module using 'pc.read_ts()' and so on...

5.20 Running on multi-processor computers

The code is parallelized using *MPI* (*message passing interface*) for a simple domain decomposition (data-parallelism), which is a straight-forward and very efficient way of parallelizing finite-difference codes. The current version has a few restrictions, which need to be kept in mind when using the MPI features.

First, only the y and z directions can be distributed over different processors. Second, the global number of grid points (but excluding the ghost zones) in a given direction must be an exact multiple of the number of processors you use in that direction. For example if you have `nprocy=8` processors for the y direction, you can run a job with `nygrid=64`

points in that direction, but if you try to run a problem with `nygrid=65` or `nygrid=94`, the code will complain about an inconsistency and stop. (So far, this has not been a serious restriction for us.)

5.20.1 How to run a sample problem in parallel

To run the sample problem in the directory '`samples/conv-slab`' on 16 CPUs, you need to do the following (in that directory):

1. Edit '`src/Makefile.local`' and replace

   ```
   MPICOMM   = nompicomm
   ```

   by

   ```
   MPICOMM   =   mpicomm
   ```

2. Edit '`src/cparam.local`' and replace

   ```
   integer, parameter :: ncpus=1,nprocy=1,nprocz=ncpus/nprocy,nprocx=1
   integer, parameter :: nxgrid=32,nygrid=nxgrid,nzgrid=nxgrid
   ```

   by

   ```
   integer, parameter :: ncpus=16,nprocy=4,nprocz=ncpus/nprocy,nprocx=1
   integer, parameter :: nxgrid=128,nygrid=nxgrid,nzgrid=nxgrid
   ```

   The first line specifies a 4×4 layout of the data in the y and z direction. The second line increases the resolution of the run because running a problem as small as 32^3 on 16 CPUs would be wasteful. Even 128^3 may still be quite small in that respect. For performance timings, one should try and keep the size of the problem per CPU the same, so for example 256^3 on 16 CPUs should be compared with 128^3 on 2 CPUs.

3. Recompile the code

   ```
   unix>  (cd src; make)
   ```

4. Run it

   ```
   unix>  start.csh
   unix>  run.csh
   ```

Make sure that all CPUs see the same '`data/`' directory; otherwise things will go wrong.

Remember that in order to visualize the full domain with IDL (rather than just the domain processed and written by one processor), you need to use '`rall.pro`' instead of '`r.pro`'.

5.20.2 Hierarchical networks (e.g. on Beowulf clusters)

On big Beowulf clusters, a group of nodes is often connected with a switch of modest speed, and all these groups are connected via a n times faster uplink switch. When bandwidth-limited, it is important to make sure that consecutive processors are mapped onto the mesh such that the load on the uplink is $\lesssim n$ times larger than the load on the

slower switch within each group. On a 512 node cluster, where groups of 24 processors are linked via fast ethernet switches, which in turn are connected via a Gigabit uplink (~ 10 times faster), we found that *nprocy*=4 is optimal. For 128 processors, for example we find that *nprocy* $\times$ *nprocz* $= 4 \times 32$ is the optimal layout, while. For comparison, 8×16 is 3 times slower, and 16×8 is 17 (!) times slower. These results can be understood from the structure of the network, but the basic message is to watch out for such effects and to try varying *nprocy* and *nprocz*.

In cases where `nygrid>nzgrid` it may be advantageous to swap the ordering of processor numbers. This can be done by setting *lprocz_slowest*=`F`.

5.20.3 Extra workload caused by the ghost zones

Normally, the workload caused by the ghost zones is negligible. However, if one increases the number of processors, a significant fraction of work is done in the ghost zones. In other words, the effective mesh size becomes much larger than the actual mesh size.

Consider a mesh of size $N_w = N_x \times N_y \times N_z$, and distribute the task over $P_w = P_x \times P_y \times P_z$ processors. If no communication were required, the number of points per processor would be

$$\frac{N_w}{P_w} = \frac{N_x \times N_y \times N_z}{P_x \times P_y \times P_z}. \tag{26}$$

However, for finite difference codes some communication is required, and the amount of communication depends on spatial order of the scheme, Q. The PENCIL CODE works by default with sixth order finite differences, so $Q = 6$, i.e. one needs 6 ghost zones, 3 on each end of the mesh. With $Q \neq 0$ the number of points per processor is

$$\frac{N_w^{(\rm eff)}}{P_w} = \left(\frac{N_x}{P_x} + Q\right) \times \left(\frac{N_y}{P_y} + Q\right) \times \left(\frac{N_z}{P_z} + Q\right). \tag{27}$$

There is efficient scaling only when

$$\min\left(\frac{N_x}{P_x}, \frac{N_y}{P_y}, \frac{N_z}{P_z}\right) \gg Q. \tag{28}$$

In the special case were $N_x = N_y = N_z \equiv N = N_w^{1/3}$, with $P_x = 1$ and $P_y = P_z \equiv P = P_w^{1/2}$, we have

$$\frac{N_w^{(\rm eff)}}{P_w} = (N + Q) \times \left(\frac{N}{P} + Q\right)^2. \tag{29}$$

For $N = 128$ and $Q = 6$ the effective mesh size exceeds the actual mesh size by a factor

$$\frac{N_w^{(\rm eff)}}{N_w} = (N + Q) \times \left(\frac{N}{P} + Q\right)^2 \times \frac{P_w}{N_w}. \tag{30}$$

These factors are listed in Table 3.

Ideally, one wants to keep the work in the ghost zones at a minimum. If one accepts that 20–25% of work are done in the ghost zones, one should use 4 processors for 128^3 mesh points, 16 processors for 256^3 mesh points, 64 processors for 512^3 mesh points, 256 processors for 1024^3 mesh points, and 512 processors for 1536^3 mesh points.

Table 3: $N_w^{(\rm eff)}/N_w$ versus N and P.

$P\backslash N$	128	256	512	1024	2048
1	1.15	1.07	1.04	1.02	1.01
2	1.19	1.09	1.05	1.02	1.01
4	1.25	1.12	1.06	1.03	1.01
8	1.34	1.16	1.08	1.04	1.02
16	1.48	1.22	1.11	1.05	1.03
32	1.68	1.31	1.15	1.07	1.04
64	1.98	1.44	1.21	1.10	1.05
128	2.45	1.64	1.30	1.14	1.07
256	3.21	1.93	1.43	1.20	1.10
512	4.45	2.40	1.62	1.29	1.14

5.21 Running in double-precision

With many compilers, you can easily switch to double precision (8-byte floating point numbers) as follows.

Add the lines

```
# Use double precision
REAL_PRECISION = double
```

to '`src/Makefile.local`' and (re-)run `pc_setupsrc`.

If *REAL_PRECISION* is set to 'double', the flag *FFLAGS_DOUBLE* is appended to the Fortran compile flags. The default for *FFLAGS_DOUBLE* is `-r8`, which works for *g95* or `ifort`; for *gfortran*, you need to make sure that *FFLAGS_DOUBLE* is set to `-fdefault-real-8`.

You can see the flags in '`src/Makefile.inc`', which is the first place to check if you have problems compiling for double precision.

Using double precision might be important in turbulence runs where the resolution is 256^3 meshpoints and above (although such runs often seem to work fine at single precision). The procedure '`postproc/src/realtodouble.x`' can be used to convert existing '`var.dat`' and '`grid.dat`' files to double precision.

5.22 Power spectrum

Given a real variable u, its Fourier transform $\tilde{u}$ is given by

$$\tilde{u}(k_x,k_y,k_z) \;=\; \mathcal{F}(u(x,y,z)) \;=\; \frac{1}{N_xN_yN_z}\sum_{p=0}^{N_x-1}\sum_{q=0}^{N_y-1}\sum_{r=0}^{N_z-1}u(x_p,y_q,z_r) \times\exp(-ik_xx_p)\exp(-ik_yy_q)\exp(-ik_zz_r), \qquad (31)$$

where

$$|k_x|<\frac{\pi N_x}{L_x}\,,\qquad |k_y|<\frac{\pi N_y}{L_y}\,,\qquad |k_z|<\frac{\pi N_z}{L_z}\,,$$

when L is the size of the simulation box. The three-dimensional power spectrum $P(k)$ is defined as

$$P(k) = \frac{1}{2}\tilde{u}\tilde{u}^*, \tag{32}$$

where

$$k = \sqrt{k_x^2 + k_y^2 + k_z^2}. \tag{33}$$

Note that we can only reasonably calculate $P(k)$ for $k < \pi N_x/L_x$.

To get power spectra from the code, edit '`run.in`' and add for example the following lines

```
dspec=5., ou_spec=T, ab_spec=T !(for energy spectra)
oned=T
```

under `run_pars`. The kinetic (`vel_spec`) and magnetic (`mag_spec`) power spectra will now be calculated every 5.0 (`dspec`) time units. The Fourier spectra is calculated using *fftpack*. In addition to calculating the three-dimensional power spectra also the one-dimensional power spectra will be calculated (`oned`).

In addition one must edit '`src/Makefile.local`' and add the lines

```
FOURIER=fourier_fftpack
POWER=power_spectrum
```

Running the code will now create the files '`powerhel_mag.dat`' and '`power_kin.dat`' containing the three-dimensional magnetic and kinetic power spectra respectively. In addition to these three-dimensional files we will also find the one-dimensional files '`powerbx_x.dat`', '`powerby_x.dat`', '`powerbz_x.dat`', '`powerux_x.dat`', '`poweruy_x.dat`' and '`poweruz_x.dat`'. In these files the data are stored such that the first line contains the time of the snapshot, the following *nxgrid*$/2$ numbers represent the power at each wavenumber, from the smallest to the largest. If several snapshots have been saved, they are being stored immediately following the preceding snapshot.

You can read the results with the idl procure '`power`', like this:

```
power,'_kin','_mag',k=k,spec1=spec1,spec2=spec2,i=n,tt=t,/noplot
power,'hel_kin','hel_mag',k=k,spec1=spec1h,spec2=spec2h,i=n,tt=t,/noplot
```

If powerhel is invoked, krms is written during the first computation. The relevant output file is '`power_krms.dat`'. This is needed for a correct calculation of k used in the realizability conditions.

A caveat of the implementation of Fourier transforms in the PENCIL CODE is that, due to the parallelization, the permitted resolution is limited to the case when one direction is an integer multiple of the other. So, it can be done for

```
Nx = n*Ny
```

Unfortunately, for some applications one wants `Nx < Ny`. Wlad experimented with arbitrary resolution by interpolating x to the same resolution of y prior to transposing, then transform the interpolated array and then interpolating it back (check '`fourier_transform_y`' in '`fourier_fftpack.f90`').

A feature of our current implementation with x parallelization is that `fft_xyz_parallel_3D` requires nygrid to be an integer multiple of nprocy*nprocz. Examples of good mesh layouts are listed in Table 4.

Table 4: Examples of mesh layouts for which Fourier transforms with x parallelization is ppossible.

ny	nprocyx	nprocy	nprocz	ncpus
256	1	16	16	256
256	2	16	16	512
256	4	16	16	1024
512	2	16	32	1024
512	4	16	16	1024
512	4	16	32	2048
576	4	18	32	2304
576	8	18	32	4608
576	16	18	32	9216
1024	4	32	32	4096
1024	4	16	64	4096
1024	8	16	32	4096
1152	4	36	32	4608
1152	4	32	36	4608
2304	2	32	72	4608
2304	4	36	64	9216
2304	4	32	72	9216

To visualize with *IDL* just type `power` and you get the last snapshot of the three-dimensional power spectrum. See head of '`$PENCIL_HOME/idl/power.pro`' for options to `power`.

5.23 Structure functions

We define the p-th order longitudinal structure function of $\boldsymbol{u}$ as

$$S^p_{\rm long}(l) = \langle |u_x(x+l,y,z) - u_x(x,y,z)|^p \rangle \ , \tag{34}$$

while the transversal is

$$S^p_{\rm trans}(l) = \langle |u_y(x+l,y,z) - u_y(x,y,z)|^p \rangle + \langle |u_z(x+l,y,z) - u_z(x,y,z)|^p \rangle \ . \tag{35}$$

Edit '`run.in`' and add for example the following lines

```
dspec=2.3,
lsfu=T,lsfb=T,lsfz1=T,lsfz2=T
```

under `run_pars`. The velocity (`lsfu`), magnetic (`lsfb`) and Elsasser (`lsfz1` and `lsfz2`) structure functions will now be calculated every 2.3 (`dspec`) time unit.

In addition one must edit '`src/Makefile.local`' and add the line

```
STRUCT_FUNC  = struct_func
```

You should also make sure that *nxgrid=nygrid=nzgrid*.

Running the code will now create the files:
'sfu-1.dat', 'sfu-2.dat', 'sfu-3.dat' (velocity),
'sfb-1.dat', 'sfb-2.dat', 'sfb-3.dat' (magnetic field),
'sfz1-1.dat', 'sfz1-2.dat', 'sfz1-3.dat' (first Elsasser variable),
'sfz2-1.dat', 'sfz2-2.dat', 'sfz2-3.dat' (second Elsasser variable),
which contains the data of interest. The first line in each file contains the time t and the number *qmax*, such that the largest moment calculated is *qmax* -1. The next *imax* numbers represent the first moment structure function for the first snapshot, here

$$\mathit{imax} = 2\frac{\ln(\mathit{nxgrid})}{\ln 2} - 2. \tag{36}$$

The next *imax* numbers contain the second moment structure function, and so on until *qmax* -1. The following *imax* numbers then contain the data of the *signed* third order structure function i.e. $S^3_{\rm long}(l) = \langle [u_x(x+l,y,z) - u_x(x,y,z)]^3 \rangle$.

The following *imax* $\times$ *qmax* $\times 2$ numbers are zero if *nr_directions* $= 1$ (default), otherwise they are the same data as above but for the structure functions calculated in the y and z directions.

If the code has been run long enough as to calculate several snapshots, these snapshots will now follow, being stored in the same way as the first snapshot.

To visualize with *IDL* just type `structure` and you get the time-average of the first order longitudinal structure function (be sure that 'pencil-runs/forced/idl/' is in `IDL_PATH`). See head of 'pencil-runs/forced/idl/structure.pro' for options to `structure`.

5.24 Particles

The PENCIL CODE has modules for tracer particles and for dust particles (see Sect. 6.14). The particle modules are chosen by setting the value of the variable `PARTICLES` in `Makefile.local` to either `particles_dust` or `particles_tracers`. For the former case each particle has six degrees of freedom, three positions and three velocities. For the latter it suffices to have only three position variables as the velocity of the particles are equal to the instantaneous fluid velocity at that point. In addition one can choose to have several additional internal degrees of freedoms for the particles. For example one can temporally evolve the particles radius by setting `PARTICLES_RADIUS` to `particles_radius` in `Makefile.local`.

All particle infrastructure is controlled and organized by the `Particles_main` module. This module is automatically selected by `Makefile.src` if `PARTICLES` is different from `noparticles`. Particle modules are compiled as a separate library. This way the main part of the Pencil Code only needs to know about the `particles_main.a` library, but not of the individual particle modules.

For a simulation with particles one must in addition define a few parameters in `cparam.local`. Here is a sample of `cparam.local` for a parallel run with 2,000,000 particles:

```
integer, parameter :: ncpus=16, nprocy=4, nprocz=4, nprocx=1
integer, parameter :: nxgrid=128, nygrid=256, nzgrid=128
integer, parameter :: npar=2000000, mpar_loc=400000, npar_mig=1000
integer, parameter :: npar_species=2
```

The parameter npar is the number of particles in the simulation, mpar_loc is the number of particles that is allowed on each processor and npar_mig is the number of particles that are allowed to migrate from one processor to another in any time-step. For a non-parallel run it is enough to specify npar. The number of particle species is set through npar_species (assumed to be one if not set). The particle input parameters are given in start.in and run.in. Here is a sample of the particle part of start.in for dust particles:

```
/
&particles_init_pars
  initxxp='gaussian-z', initvvp='random'
  zp0=0.02, delta_vp0=0.01, eps_dtog=0.01, tausp=0.1
  lparticlemesh_tsc=T
/
```

The initial positions and velocities of the dust particles are set in initxxp and initvvp. The next four input parameters are further specifications of the initial condition. Interaction between the particles and the mesh, e.g. through drag force or self-gravity, require a mapping of the particles on the mesh. The PENCIL CODE currently supports NGP (Nearest Grid Point, default), CIC (Cloud in Cell, set lparticlemesh_cic=T) and TSC (Triangular Shaped Cloud, set lparticlemesh_tsc=T). See Youdin & Johansen (2007) for details.

Here is a sample of the particle part of run.in (also for dust particles):

```
/
&particles_run_pars
  ldragforce_gas_par=T
  cdtp=0.2
/
```

The logical ldragforce_gas_par determines whether the dust particles influence the gas with a drag force. cdtp tells the code how many friction times should be resolved in a minimum time-step.

The sample run 'samples/sedimentation/' contains the latest setup for dust particles.

5.24.1 Particles in parallel

The particle variables (e.g. $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$) are kept in the arrays fp and dfp. For parallel runs, particles must be able to move from processor to processor as they pass out of the (x, y, z)-interval of the local processor. Since not all particles are present at the same processor at the same time (hopefully), there is some memory optimization in making fp not big enough to contain all the particles at once. This is achieved by setting the code variable mpar_loc less than npar in cparam.local for parallel runs. When running with millions of particles, this trick is necessary to keep the memory need of the code down.

The communication of migrating particles between the processors happens as follows (see the subroutine redist_particles_procs in particles_sub.f90):

1. In the beginning of each time-step all processors check if any of their particles have crossed the local (x, y, z)-interval. These particles are called migrating particles. A run can have a maximum of npar_mig migrating particles in each time-step. The value of npar_mig must be set in cparam.local. The number should (of course) be

slightly larger than the maximum number of migrating particles at any time-step during the run. The diagnostic variable nmigmax can be used to output the maximum number of migrating particles at a given time-step. One can set lmigration_redo=T in &particles_run_pars to force the code to redo the migration step if more than npar_mig want to migrate. This does slow the code down somewhat, but has the benefit that the code does not stop when more than npar_mig particles want to migrate.

2. The index number of the receiving processor is then calculated. This requires some assumption about the grid on other processors and will currently not work for nonequidistant grids. Particles do not always pass to neighboring processors as the global boundary conditions may send them to the other side of the global domains (periodic or shear periodic boundary conditions).

3. The migrating particle information is copied to the end of fp, and the empty spot left behind is filled up with the particle of the highest index number currently present at the processor.

4. Once the number of migrating particles is known, this information is shared with neighboring processors (including neighbors over periodic boundaries) so that they all know how many particles they have to receive and from which processors.

5. The communication happens as directed MPI communication. That means that processors 0 and 1 can share migrating particles at the same time as processors 2 and 3 do it. The communication happens from a chunk at the end of fp (migrating particles) to a chunk that is present just after the particle of the highest index number that is currently at the receiving processor. Thus the particles are put directly at their final destination, and the migrating particle information at the source processor is simply overwritten by other migrating particles at the next time-step.

6. Each processor keeps track of the number of particles that it is responsible for. This number is stored in the variable npar_loc. It must never be larger than mpar_loc (see above). When a particle leaves a processor, npar_loc is reduced by one, and then increased by one at the processor that receives that particle. The maximum number of particles at any processor is stored in the diagnostic variable nparmax. If this value is not close to npar/ncpus, the particles have piled up in such a way that computations are not evenly shared between the processors. One can then try to change the parallelization architecture (nprocy and nprocz) to avoid this problem.

In simulations with many particles (comparable to or more than the number of grid cells), it is crucial that particles are shared relatively evenly among the processors. One can as a first approach attempt to not parallelize directions with strong particle density variations. However, this is often not enough, especially if particles clump locally.

Alternatively one can use Particle Block Domain Decomposition (PBDD, see Johansen et al. 2011). The steps in Particle Block Domain Decomposition scheme are as follows:

1. The fixed mesh points are domain-decomposed in the usual way (with ncpus=nprocx×nprocy×nprocz).

2. Particles on each processor are counted in *bricks* of size nbx×nby×nbz (typically nbx= nby=nbz=4).

3. Bricks are distributed among the processors so that each processor has approxi-

mately the same number of particles

4. Adopted bricks are referred to as *blocks*.
5. The Pencil Code uses a third order Runge-Kutta time-stepping scheme. In the beginning of each sub-time-step particles are counted in blocks and the block counts communicated to the bricks on the parent processors. The particle density assigned to ghost cells is folded across the grid, and the final particle density (defined on the bricks) is communicated back to the adopted blocks. This step is necessary because the drag force time-step depends on the particle density, and each particle assigns density not just to the nearest grid point, but also to the neighboring grid points.
6. In the beginning of each sub-time-step the gas density and gas velocity field is communicated from the main grid to the adopted particle blocks.
7. Drag forces are added to particles and back to the gas grid points in the adopted blocks. This partition aims at load balancing the calculation of drag forces.
8. At the end of each sub-time-step the drag force contribution to the gas velocity field is communicated from the adopted blocks back to the main grid.

Particle Block Domain Decomposition is activated by setting `PARTICLES = particles_dust_blocks` and `PARTICLES_MAP = particles_map_blocks` in `Makefile.local`. A sample of `cparam.local` for Particle Block Domain Decomposition can be found in `samples/sedimentation/blocks`:

```
integer, parameter :: ncpus=4, nprocx=2, nprocy=2, nprocz=1
integer, parameter :: nxgrid=32, nygrid=32, nzgrid=32
integer, parameter :: npar=10000, mpar_loc=5000, npar_mig=100
integer, parameter :: npar_species=4
integer, parameter :: nbrickx=4, nbricky=4, nbrickz=4, nblockmax=32
```

The last line defines the number of bricks in the total domain – here we divide the grid into $4 \times 4 \times 4$ bricks each of size $8 \times 8 \times 8$ grid points. The parameter `nblockmax` tells the code the maximum number of blocks any processor may adopt. This should not be so low that there is not room for all the bricks with particles, nor so high that the code runs out of memory.

5.25 Non-cartesian coordinate systems

Since the spring of 2007 spherical and cylindrical polar coordinates have been implemented, although this development is not yet completed. Spherical coordinates are invoked by adding the following line in the file '`start.in`'

```
&init_pars
  coord_system='spherical_coords'
```

Another possibility is to put `cylindrical_coords` instead. In practice, the names (x, y, z) are still used, but they refer then to (r, θ, ϕ) or (r, ϕ, z) instead.

Bug reports, corrections, and improvements on these are appreciated.

6 The equations

The equations solved by the PENCIL CODE are basically the standard compressible MHD equations. However, the modular structure allows some variations of the MHD equations, as well as switching off some of the equations or individual terms of the equation (nomagnetic, noentropy, etc.).

In this section the equations are presented in their most complete form. It may be expected that the code can evolve most subsets or simplifications of these equations.

6.1 Continuity equation

In the code the continuity equation, $\partial\rho/\partial t + \boldsymbol{\nabla}\cdot\rho\boldsymbol{u} = 0$, is written in terms of $\ln\rho$,

$$\frac{\mathrm{D}\ln\rho}{\mathrm{D}t} = -\boldsymbol{\nabla}\cdot\boldsymbol{u}\,. \tag{37}$$

Here ρ denotes density, $\boldsymbol{u}$ the fluid velocity, t is time and $\mathrm{D}/\mathrm{D}t \equiv \partial/\partial t + \boldsymbol{u}\cdot\boldsymbol{\nabla}$ is the convective derivative.

6.2 Equation of motion

In the equation of motion, using a perfect gas, the pressure term, can be expressed as $-\rho^{-1}\boldsymbol{\nabla}p = -c_{\rm s}^2(\boldsymbol{\nabla}s/c_p + \boldsymbol{\nabla}\ln\rho)$, where the squared sound speed is given by

$$c_{\rm s}^2 = \gamma\frac{p}{\rho} = c_{\rm s0}^2\exp\left[\gamma s/c_p + (\gamma{-}1)\ln\frac{\rho}{\rho_0}\right], \tag{38}$$

and $\gamma = c_p/c_v$ is the ratio of specific heats, or *adiabatic index*. Note that $c_{\rm s}^2$ is proportional to the temperature with $c_{\rm s}^2 = (\gamma-1)c_pT$.

The equation of motion is accordingly

$$\begin{aligned}\frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} = \quad & -c_{\rm s}^2\boldsymbol{\nabla}\left(\frac{s}{c_p}+\ln\rho\right) - \boldsymbol{\nabla}\Phi_{\rm grav} + \frac{\boldsymbol{j}\times\boldsymbol{B}}{\rho} \\ & +\nu\left(\nabla^2\boldsymbol{u} + \frac{1}{3}\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{u} + 2\mathsf{S}\cdot\boldsymbol{\nabla}\ln\rho\right) + \zeta\left(\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{u}\right);\end{aligned} \tag{39}$$

Here $\Phi_{\rm grav}$ is the gravity potential, $\boldsymbol{j}$ the electric current density, $\boldsymbol{B}$ the magnetic flux density, ν is kinematic viscosity, ζ describes a bulk viscosity, and, in Cartesian coordinates

$$\mathsf{S}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\boldsymbol{\nabla}\cdot\boldsymbol{u}\right) \tag{40}$$

is the rate-of-shear tensor that is traceless, because it can be written as the generic rate-of-strain tensor minus its trace. In curvilinear coordinates, we have to replace partial differentiation by covariant differentiation (indicated by semicolons), so we write $\mathsf{S}_{ij} = \frac{1}{2}(u_{i;j} + u_{j;i}) - \frac{1}{3}\delta_{ij}\boldsymbol{\nabla}\cdot\boldsymbol{u}$.

The interpretation of the two viscosity terms varies greatly depending upon the Viscosity module used, and indeed on the parameters given to the module. See §6.6.

For isothermal hydrodynamics, see §6.4 below.

6.3 Induction equation

$$\frac{\partial \boldsymbol{A}}{\partial t} = \boldsymbol{u} \times \boldsymbol{B} - \eta \mu_0 \boldsymbol{j} \ . \tag{41}$$

Here $\boldsymbol{A}$ is the magnetic vector potential, $\boldsymbol{B} = \nabla \times \boldsymbol{A}$ the magnetic flux density, $\eta = 1/(\mu_0 \sigma)$ is the magnetic diffusivity (σ denoting the electrical conductivity), and μ_0 the magnetic vacuum permeability. This form of the induction equation corresponds to the *Weyl gauge* $\Phi = 0$, where Φ denotes the scalar potential.

6.4 Entropy equation

The current thermodynamics module *entropy* formulates the thermal part of the physics in terms of *entropy* s, rather than thermal energy e, which you may be more familiar with. Thus the two fundamental thermodynamical variables are $\ln \rho$ and s. The reason for this choice of variables is that entropy is the natural physical variable for (at least) convection processes: the sign of the entropy gradient determines convective (in)stability, the *Rayleigh number* is proportional to the entropy gradient of the associated hydrostatic reference solution, etc. The equation solved is

$$\rho T \frac{\mathrm{D}s}{\mathrm{D}t} = \mathcal{H} - \mathcal{C} + \boldsymbol{\nabla} \cdot (K \boldsymbol{\nabla} T) + \eta \mu_0 \boldsymbol{j}^2 + 2 \rho \nu \mathsf{S} \otimes \mathsf{S} + \zeta \rho \left(\boldsymbol{\nabla} \cdot \boldsymbol{u} \right)^2 \ . \tag{42}$$

Here, T is temperature, c_p the specific heat at constant pressure, $\mathcal{H}$ and $\mathcal{C}$ are explicit heating and cooling terms, K is the radiative (thermal) conductivity, ζ describes a bulk viscosity, and S is the rate-of-shear tensor that is traceless.

In the entropy module we solve for the specific entropy, s. The heat conduction term on the right hand side can be written in the form

$$\frac{\boldsymbol{\nabla} \cdot (K \boldsymbol{\nabla} T)}{\rho T} \tag{43}$$

$$= \ c_p \chi \Big[\nabla^2 \ln T + \boldsymbol{\nabla} \ln T \cdot \boldsymbol{\nabla} (\ln T + \ln \chi + \ln \rho) \Big] \tag{44}$$

$$= \ c_p \chi \left[\gamma \nabla^2 s / c_p + (\gamma - 1) \nabla^2 \ln \rho \right] + c_p \chi \left[\gamma \boldsymbol{\nabla} s / c_p + (\gamma - 1) \boldsymbol{\nabla} \ln \rho \right] \cdot \left[\gamma \left(\boldsymbol{\nabla} s / c_p + \boldsymbol{\nabla} \ln \rho \right) + \boldsymbol{\nabla} \ln \chi \right] \ , \tag{45}$$

where $\chi = K/(\rho c_p)$ is the thermal diffusivity. The latter equation shows that the diffusivity for s is $\gamma \chi$, which is what we have used in Eq. (24).

In an alternative formulation for a constant K, the heat conduction term on the right hand side can also be written in the form

$$\frac{\boldsymbol{\nabla} \cdot (K \boldsymbol{\nabla} T)}{\rho T} = \frac{K}{\rho} \Big[\nabla^2 \ln T + (\boldsymbol{\nabla} \ln T)^2 \Big] \tag{46}$$

which is the form used when constant K is chosen.

Note that by setting $\gamma = 1$ and initially $s = 0$, one obtains an isothermal equation of state (albeit at some unnecessary expense of memory). Similarly, by switching off the evolution terms of entropy, one immediately gets polytropic behavior (if s was initially constant) or generalized polytropic behavior (where s is not uniform, but $\partial s / \partial t = 0$).

A better way to achieve isothermality is to use the *noentropy* module.

6.4.1 Viscous heating

We can write the viscosity as the divergence of a tensor $\tau_{ij,j}$,

$$\rho\frac{\partial u_i}{\partial t} = ... + \tau_{ij,j}\,, \tag{47}$$

where $\tau_{ij} = 2\nu\rho \mathsf{S}_{ij}$ is the stress tensor. The viscous power density P is

$$\begin{aligned} P &= u_i\tau_{ij,j} && (48)\\ &= \frac{\partial}{\partial x_j}\left(u_i\tau_{ij}\right) - u_{i,j}\tau_{ij} && (49)\end{aligned}$$

The term under the divergence is the viscous energy flux and the other term is the kinetic energy loss due to heating. The heating term $+u_{i,j}\tau_{ij}$ is positive definite, because τ_{ij} is a symmetric tensor and the term only gives a contribution from the symmetric part of $u_{i,j}$, which is $\frac{1}{2}(u_{i,j}+u_{j,i})$, so

$$u_{i,j}\tau_{ij} = \frac{1}{2}\nu\rho(u_{i,j}+u_{j,i})(2\mathsf{S}_{ij})\,. \tag{50}$$

But, because S_{ij} is traceless, we can add anything proportional to δ_{ij} and, in particular, we can write

$$\begin{aligned} u_{i,j}\tau_{ij} &= \frac{1}{2}(u_{i,j}+u_{j,i})(2\nu\rho\mathsf{S}_{ij}) && (51)\\ &= \frac{1}{2}(u_{i,j}+u_{j,i}-\frac{1}{3}\delta_{ij}\boldsymbol{\nabla}\cdot\boldsymbol{u})(2\nu\rho\mathsf{S}_{ij}) && (52)\\ &= 2\nu\rho\mathbf{S}^2, && (53)\end{aligned}$$

which is positive definite.

6.4.2 Alternative description

By setting `pretend_lnTT=T` in `init_pars` or `run_pars` (i.e. the general part of the name list) the logarithmic temperature is used instead of the entropy. This has computational advantages when heat conduction (proportional to $K\boldsymbol{\nabla}T$) is important. Another alternative is to use another module, i.e. set `ENTROPY=temperature_idealgas` in '`Makefile.local`'.

When `pretend_lnTT=T` is set, the entropy equation

$$\frac{\partial s}{\partial t} = -\boldsymbol{u}\cdot\boldsymbol{\nabla}s + \frac{1}{\rho T}\mathbf{RHS} \tag{54}$$

is replaced by

$$\frac{\partial \ln T}{\partial t} = -\boldsymbol{u}\cdot\boldsymbol{\nabla}\ln T + \frac{1}{\rho c_v T}\mathbf{RHS} - (\gamma-1)\,\boldsymbol{\nabla}\cdot\boldsymbol{u}, \tag{55}$$

where RHS is the right hand side of equation (42).

6.5 Transport equation for a passive scalar

In conservative form, the equation for a passive scalar is

$$\frac{\partial}{\partial t}(\rho c) + \boldsymbol{\nabla}\cdot[\rho c\boldsymbol{u} - \rho\mathcal{D}\nabla c] = 0. \tag{56}$$

Here c denotes the concentration (per unit mass) of the passive scalar and $\mathcal{D}$ its diffusion constant (assumed constant). In the code this equation is solved in terms of $\ln c$,

$$\frac{\mathrm{D}\ln c}{\mathrm{D}t} = \mathcal{D}\left[\nabla^2\ln c + (\boldsymbol{\nabla}\ln\rho + \boldsymbol{\nabla}\ln c)\cdot\boldsymbol{\nabla}\ln c\right]. \tag{57}$$

Using $\ln c$ instead of c has the advantage that it enforces $c > 0$ for all times. However, the disadvantage is that one cannot have $c = 0$. For this reason we ended up using the non-logarithmic version by invoking `PSCALAR=pscalar_nolog`.

6.6 Bulk viscosity

For a monatomic gas it can be shown that the bulk viscosity vanishes. We therefore don't use it in most of our runs. However, for supersonic flows, or even otherwise, one might want to add a shock viscosity which, in its simplest formulation, take the form of a bulk viscosity.

6.6.1 Shock viscosity

Shock viscosity, as it is used here and also in the Stagger Code of Åke Nordlund, is proportional to positive flow convergence, maximum over five zones, and smoothed to second order,

$$\zeta_{\rm shock} = c_{\rm shock}\left\langle\max_5[(-\boldsymbol{\nabla}\cdot\boldsymbol{u})_+]\right\rangle(\min(\delta x,\delta y,\delta z))^2, \tag{58}$$

where $c_{\rm shock}$ is a constant defining the strength of the shock viscosity. In the code this dimensionless coefficient is called `nu_shock`, and it is usually chosen to be around unity. Assume that the shock viscosity only enters as a bulk viscosity, so the whole stress tensor is then

$$\boldsymbol{\tau}_{ij} = 2\rho\nu\mathsf{S}_{ij} + \rho\zeta_{\rm shock}\delta_{ij}\boldsymbol{\nabla}\cdot\boldsymbol{u}. \tag{59}$$

Assume $\nu =$ const, but $\zeta \neq$ const, so

$$\rho^{-1}\boldsymbol{F}_{\rm visc} = \nu\left(\nabla^2\boldsymbol{u} + \frac{1}{3}\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{u} + 2\mathsf{S}\cdot\boldsymbol{\nabla}\ln\rho\right) + \zeta_{\rm shock}\left[\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{u} + (\boldsymbol{\nabla}\ln\rho + \boldsymbol{\nabla}\ln\zeta_{\rm shock})\,\boldsymbol{\nabla}\cdot\boldsymbol{u}\right]. \tag{60}$$

and

$$\rho^{-1}\Gamma_{\rm visc} = 2\nu\mathsf{S}^2 + \zeta_{\rm shock}(\boldsymbol{\nabla}\cdot\boldsymbol{u})^2. \tag{61}$$

In the special case with periodic boundary conditions, we have $2\langle\mathsf{S}^2\rangle = \langle\boldsymbol{\omega}^2\rangle + \frac{4}{3}\langle(\boldsymbol{\nabla}\cdot\boldsymbol{u})^2\rangle$.

6.7 Equation of state

In its present configuration only hydrogen ionization is explicitly included. Other constituents (currently He and H_2) can have fixed values. The pressure is proportional to the total number of particles, i.e.

$$p = (n_{\rm HI} + n_{\rm HII} + n_{\rm H_2} + n_{\rm e} + n_{\rm He} + ...)k_{\rm B}T. \tag{62}$$

It is convenient to normalize to the total number of H both in atomic and in molecular hydrogen, $n_{\rm Htot} \equiv n_{\rm H} + 2n_{\rm H_2}$, where $n_{\rm HI} + n_{\rm HII} = n_{\rm H}$, and define $x_{\rm e} \equiv n_{\rm e}/n_{\rm Htot}$, $x_{\rm He} \equiv n_{\rm He}/n_{\rm Htot}$, and $x_{\rm H_2} \equiv n_{\rm H_2}/n_{\rm Htot}$. Substituting $n_{\rm H} = n_{\rm Htot} - 2n_{\rm H_2}$, we have

$$p = (1 - x_{\rm H_2} + x_{\rm e} + x_{\rm He} + ...)n_{\rm Htot}k_{\rm B}T. \tag{63}$$

This can be written in the more familiar form

$$p = \frac{\mathcal{R}}{\mu}\rho T, \tag{64}$$

where $\mathcal{R} = k_{\rm B}/m_{\rm u}$ and $m_{\rm u}$ is the atomic mass unit (which is for all practical purposes the same as $m_{\rm Htot}$) and

$$\mu = \frac{n_{\rm H} + 2n_{\rm H_2} + n_{\rm e} + 4n_{\rm He}}{n_{\rm H} + n_{\rm H_2} + n_{\rm He}} = \frac{1 + 4x_{\rm He}}{1 - x_{\rm H_2} + x_{\rm e} + x_{\rm He}} \tag{65}$$

is the mean molecular weight (which is here dimensionless; see Kippenhahn & Weigert 1990, p. 102). The factor 4 is really to be substituted for 3.97153. Some of the familiar relations take still the usual form, in particular $e = c_v T$ and $h = c_p T$ with $c_v = \frac{3}{2}\mathcal{R}/\mu$ and $c_p = \frac{5}{2}\mathcal{R}/\mu$.

The number ratio, $x_{\rm He}$, is more commonly expressed as the mass ratio, $Y = m_{\rm He}n_{\rm He}/(m_{\rm H}n_{\rm Htot} + m_{\rm He}n_{\rm e}n_{\rm He})$, or $Y = 4x_{\rm He}/(1+4x_{\rm He})$, or $4x_{\rm He} = (1/Y - 1)^{-1}$. For example, $Y = 0.27$ corresponds to $x_{\rm He} = 0.9$. Note also that for 100% H_2 abundance, $x_{\rm H_2} = 1/2$.

In the following, the ionization fraction is given as $y = n_{\rm e}/n_{\rm H}$, which can be different from $x_{\rm e}$ if there is H_2. Substituting for $n_{\rm H}$ in terms of $n_{\rm Htot}$ yields $y = x_{\rm e}/(1 - 2x_{\rm H_2})$.

6.8 Ionization

This part of the code can be invoked by setting `EOS=eos_ionization` (or `EOS=eos_temperature_ionization`) in the '`Makefile.local`' file. The equation of state described below works for variable ionization, and the entropy offset is different from that used in Eq. (38), which is now no longer valid. As a replacement, one can use `EOS=eos_fixed_ionization`, where the degree of ionization can be given by hand. Here the normalization of the entropy is the same as for `EOS=eos_ionization`. This case is described in more detail below.[12]

We treat the gas as being composed of partially ionized hydrogen and neutral helium. These are four different particle species, each of which regarded as a perfect gas.

The ionization fraction y, which gives the ratio of ionized hydrogen to the total amount of hydrogen $n_{\rm H}$, is obtained from the Saha equation which, in this case, may be written as

$$\frac{y^2}{1-y} = \frac{1}{n_{\rm H}}\left(\frac{m_{\rm e}k_{\rm B}T}{2\pi\hbar^2}\right)^{3/2}\exp\left(-\frac{\chi_{\rm H}}{k_{\rm B}T}\right) \ . \tag{66}$$

The temperature T cannot be obtained directly from the PENCIL CODE's independent variables $\ln\rho$ and s, but is itself dependent on y. Hence, the calculation of y essentially becomes a root finding problem.

[12] We omit here the contribution of H_2.

The entropy of a perfect gas consisting of particles of type i is known from the Sackur-Tetrode equation

$$S_i = k_{\rm B} N_i \left(\ln \left[\frac{1}{n_{\rm tot}} \left(\frac{m_i k_{\rm B} T}{2\pi\hbar^2} \right)^{3/2} \right] + \frac{5}{2} \right) . \tag{67}$$

Here N_i is the number of particles of a single species and $n_{\rm tot}$ is the total number density of all particle species.

In addition to the individual entropies we also have to take the entropy of mixing, $S_{\rm mix} = -N_{\rm tot} k_{\rm B} \sum_i p_i \ln p_i$, into account. Summing up everything, we can get a closed expression for the specific entropy s in terms of y, $\ln\rho$ and T, which may be solved for T.

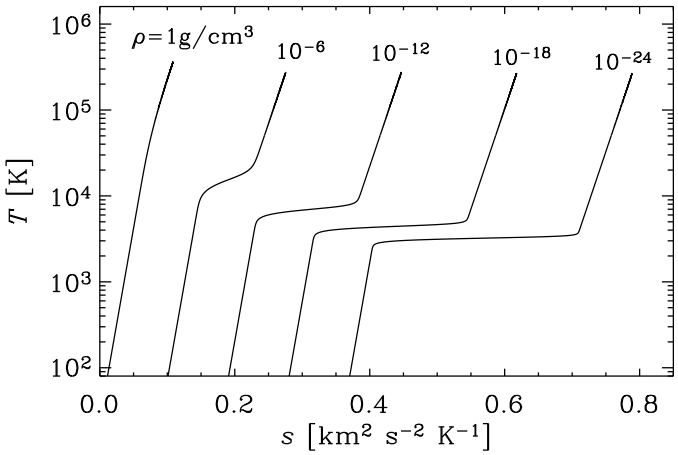

Figure 5: Dependence of temperature on entropy for different values of the density.

For given $\ln\rho$ and s we are then able to calculate the ionization fraction y by finding the root of

$$f(y) = \ln \left[\frac{1-y}{y^2} \frac{1}{n_{\rm H}} \left(\frac{m_{\rm e} k_{\rm B} T(y)}{2\pi\hbar^2} \right)^{3/2} \right] - \frac{\chi_{\rm H}}{k_{\rm B} T(y)} . \tag{68}$$

In the ionized case, several thermodynamic quantities of the gas become dependent on the ionization fraction y such as its pressure, $P = (1+y+x_{\rm He}) n_{\rm H} k_{\rm B} T$, and its internal energy, $E = \frac{3}{2}(1+y+x_{\rm He}) n_{\rm H} k_{\rm B} T + y\chi_{\rm H}$, where $x_{\rm He}$ gives the ratio of neutral helium to the total amount of hydrogen. The dependence of temperature on entropy is shown in Fig. 5 for different values of the density.

For further details regarding the procedure of solving for the entropy see Sect. H.5 in the appendix.

6.8.1 Ambipolar diffusion

Another way of dealing with ionization in the PENCIL CODE is through use of the *neutrals* module. That module solves the coupled equations of neutral and ionized gas, in a two-fluid model

$$\frac{\partial \rho_i}{\partial t} = -\boldsymbol{\nabla}\cdot(\rho_i \boldsymbol{u}_i) + \mathcal{G} \tag{69}$$

$$\frac{\partial \rho_n}{\partial t} = -\boldsymbol{\nabla}\cdot(\rho_n \boldsymbol{u}_n) - \mathcal{G} \tag{70}$$

$$\frac{\partial (\rho_i \boldsymbol{u}_i)}{\partial t} = -\boldsymbol{\nabla}\cdot(\rho_i \boldsymbol{u}_i : \boldsymbol{u}_i) - \boldsymbol{\nabla}\left(p_i + p_e + \frac{B^2}{2\mu_0}\right) + \mathcal{F} \tag{71}$$

$$\frac{\partial (\rho_n \boldsymbol{u}_n)}{\partial t} = -\boldsymbol{\nabla}\cdot(\rho_n \boldsymbol{u}_n : \boldsymbol{u}_n) - \boldsymbol{\nabla} p_n - \mathcal{F} \tag{72}$$

$$\frac{\partial \boldsymbol{A}}{\partial t} = \boldsymbol{u}_i \times \boldsymbol{B} \tag{73}$$

where the subscripts n and i are for neutral and ionized, respectively. The terms $\mathcal{G}$ and $\mathcal{F}$, through which the two fluids exchange mass and momentum, are given by

$$\mathcal{G} = \zeta \rho_n - \alpha \rho_i^2 \tag{74}$$

$$\mathcal{F} = \zeta \rho_n \boldsymbol{u}_n - \alpha \rho_i^2 \boldsymbol{u}_i + \gamma \rho_i \rho_n (\boldsymbol{u}_n - \boldsymbol{u}_i)\ . \tag{75}$$

In the above equations, ζ is the ionization coefficient, α is the recombination coefficient, and γ the collisional drag strength. By the time of writing (spring 2009), these three quantities are supposed constant. The electron pressure p_e is also assumed equal to the ion pressure. Only isothermal neutrals are supported so far.

In the code, Eq. (69) and Eq. (71) are solved in '`density.f90`' and '`hydro.f90`' respectively. Equation 70 is solved in '`neutraldensity.f90`' and Eq. (72) in '`neutralvelocity.f90`'. The sample '`1d-test/ambipolar-diffusion`' has the current setup for a two-fluid simulation with ions and neutrals.

6.9 Radiative transfer

Here we only state the basic equations. A full description about the implementation is given in Sect. H.6 and in the original paper by Heinemann et al. (2006).

The basic equation for radiative transfer is

$$\frac{dI}{d\tau} = -I + S\ , \tag{76}$$

where

$$\tau \equiv \int_0^s \chi(s')\, ds' \tag{77}$$

is the optical depth (s is the geometrical coordinate along the ray).

Note that radiative transfer is called also in '`start.csh`', and again each time a snapshot is being written, provided the output of auxiliary variables is being requested `lwrite_aux=T`. (Also, of course, the pencil check runs radiative transfer 7 times, unless you put `pencil_check_small=F`.)

6.10 Self-gravity

The PENCIL CODE can consider the self-gravity of the fluid in the simulation box by adding the term

$$\frac{\partial \boldsymbol{u}}{\partial t} = \ldots - \boldsymbol{\nabla}\phi_{\rm self} \tag{78}$$

to the equation of motion. The self-potential $\phi_{\rm self}$ (or just ϕ for simplicity) satisfies Poisson's equation

$$\nabla^2\phi = 4\pi G\rho\,. \tag{79}$$

The solution for a single Fourier component at scale $\boldsymbol{k}$ is

$$\phi_{\boldsymbol{k}} = -\frac{4\pi G\rho_{\boldsymbol{k}}}{k^2}\,. \tag{80}$$

Here we have assumed periodic boundary conditions. The potential is obtained by Fourier-transforming the density, then finding the corresponding potential at that scale, and finally Fourier-transforming back to real space.

The x-direction in the shearing sheet is not strictly periodic, but is rather shear periodic with two connected points at the inner and outer boundary separated by the distance $\Delta y(t) = \mod[(3/2)\Omega_0 L_x t, L_y]$ in the y-direction. We follow here the method from [13] to allow for shear-periodic boundaries in the Fourier method for self-gravity. First we take the Fourier transform along the periodic y-direction. We then shift the entire y-direction by the amount $\delta y(x) = \Delta y(t)x/L_x$ to make the x-direction periodic. Then we proceed with Fourier transforms along x and then z. After solving the Poisson equation in Fourier space, we transform back to real space in the opposite order. We differ here from the method by [13] in that we shift in Fourier space rather than in real space[13]. The Fourier interpolation formula has the advantage over polynomial interpolation in that it is continuous and smooth in all its derivatives.

6.11 Incompressible and anelastic equations

This part has not yet been documented and is still under development.

6.12 Dust equations

The code treats gas and dust as two separate fluids[14]. The dust and the gas interact through a drag force. This force can most generally be written as an additional term to the equation of motion as

$$\frac{{\rm D}\boldsymbol{u}_{\rm d}}{{\rm D}t} = \ldots - \frac{1}{\tau_{\rm s}}\left(\boldsymbol{u}_{\rm d} - \boldsymbol{u}\right)\,. \tag{81}$$

Here $\tau_{\rm s}$ is the so-called stopping time of the considered dust species. This measures the coupling strength between dust and gas. In the Epstein drag regime

$$\tau_{\rm s} = \frac{a_{\rm d}\rho_{\rm s}}{c_{\rm s}\rho}\,, \tag{82}$$

[13] We were kindly made aware of the possibility of interpolating in Fourier space by C. McNally on his website.

[14] See master's thesis of A. Johansen (can be downloaded from `http://www.mpia.de/homes/johansen/research_en.php`)

where $a_{\rm d}$ is the radius of the dust grain and $\rho_{\rm s}$ is the solid density of the dust grain.

Two other important effects work on the dust. The first is coagulation controlled by the discrete coagulation equation

$$\frac{{\rm d}n_k}{{\rm d}t} = \frac{1}{2}\sum_{i+j=k} A_{ij} n_i n_j - n_k \sum_{i=1}^{\infty} A_{ik} n_i \,. \tag{83}$$

In the code N discrete dust species are considered. Also the bins are logarithmically spaced in order to give better mass resolution. It is also possible to keep track of both number density and mass density of each bin, corresponding to having a variable grain mass in each bin.

Dust condensation is controlled by the equation

$$\frac{{\rm d}N}{{\rm d}t} = \frac{1}{\tau_{\rm cond}} N^{\frac{d-1}{d}} \,. \tag{84}$$

Here N is the number of monomers in the dust grain (such as water molecules) and d is the physical dimension of the dust grain. The condensation time $\tau_{\rm cond}$ is calculated from

$$\frac{1}{\tau_{\rm cond}} = A_1 v_{\rm th} \alpha n_{\rm mon} \left\{ 1 - \frac{1}{S_{\rm mon}} \right\} , \tag{85}$$

where A_1 is the surface area of a monomer, α is the condensation efficiency, $n_{\rm mon}$ is the number density of monomers in the gas and $S_{\rm mon}$ is the saturation level of the monomer given by

$$S_{\rm mon} = \frac{P_{\rm mon}}{P_{\rm sat}} \,. \tag{86}$$

Here $P_{\rm sat}$ is the saturated vapor pressure of the monomer. Currently only water ice has been implemented in the code.

All dust species fulfill the continuity equation

$$\frac{\partial \rho_{\rm d}}{\partial t} + \boldsymbol{\nabla} \cdot (\rho_{\rm d} \boldsymbol{u}_{\rm d}) = 0. \tag{87}$$

6.13 Cosmic ray pressure in diffusion approximation

Cosmic rays are treated in the diffusion approximation. The equation of state is $p_{\rm c} = (\gamma_{\rm c}) e_{\rm c}$ where the value of $\gamma_{\rm c}$ is usually somewhere between $14/9$ and $4/3$. In the momentum equation (39) the cosmic ray pressure force, $-\rho^{-1}\boldsymbol{\nabla} p_{\rm c}$ is added on the right hand side, and $e_{\rm c}$ satisfies the evolution equation

$$\frac{\partial e_{\rm c}}{\partial t} + \boldsymbol{\nabla} \cdot (e_{\rm c} \boldsymbol{u}) + p_{\rm c} \boldsymbol{\nabla} \cdot \boldsymbol{u} = \partial_i (K_{ij} \partial_j e_{\rm c}) + Q_{\rm c}, \tag{88}$$

where $Q_{\rm c}$ is a source term and

$$K_{ij} = K_\perp \delta_{ij} + (K_\parallel - K_\perp) \hat{B}_i \hat{B}_j \tag{89}$$

is an anisotropic diffusivity tensor.

In the non-conservative formulation of this code it is advantageous to expand the diffusion term using the product rule, i.e.

$$\partial_i(K_{ij}\partial_j e_{\rm c}) = -\boldsymbol{U}_{\rm c}\cdot\boldsymbol{\nabla}e_{\rm c} + K_{ij}\partial_i\partial_j e_{\rm c}. \tag{90}$$

where $U_{{\rm c}\,i} = -\partial K_{ij}/\partial x_j$ acts like an extra velocity trying to straighten magnetic field lines. We can write this term also as $\boldsymbol{U}_{\rm c} = -(K_\parallel - K_\perp)\boldsymbol{\nabla}\cdot(\hat{\boldsymbol{B}}\hat{\boldsymbol{B}})$, where the last term is a divergence of the dyadic product of unit vectors.[15] However, near magnetic nulls, this term can becomes infinite. In order to avoid this problem we are forced to limit $\boldsymbol{\nabla}\cdot(\hat{\boldsymbol{B}}\hat{\boldsymbol{B}})$, and hence $|\boldsymbol{U}_{\rm c}|$, to the maximum possible value that can be resolved at a given resolution.

A physically appealing way of limiting the maximum propagation speed is to restore an explicit time dependence in the equation for the cosmic ray flux, and to replace the diffusion term in Eq. (88) by a divergence of a flux that in turn obeys the equation

$$\frac{\partial\mathcal{F}_{{\rm c}i}}{\partial t} = -\tilde{K}_{ij}\nabla_j e_{\rm c} - \frac{\mathcal{F}_{{\rm c}i}}{\tau} \quad \text{(non-Fickian diffusion)}, \tag{91}$$

where $K_{ij} = \tau\tilde{K}_{ij}$ would be the original diffusion tensor of Eq. (89), if the time derivative were negligible. Further details are described in Snodin et al. (2005).

6.14 Particles

Particles are entities that each have a space coordinate and a velocity vector, where a fluid only has a velocity vector field (the continuity equation of a fluid in some way corresponds to the space coordinate of particles). In the code particles are present either as tracer particles or as dust particles

6.14.1 Tracer particles

Tracer particles always have the local velocity of the gas. The dynamical equations are thus

$$\frac{\partial\boldsymbol{x}_i}{\partial t} = \boldsymbol{u}\,, \tag{92}$$

where the index i runs over all particles. Here $\boldsymbol{u}$ is the gas velocity at the position of the particle. One can choose between a first order (default) and a second order spline interpolation scheme (set `lquadratic_interpolation=T` in `&particles_init_pars`) to calculate the gas velocity at the position of a tracer particle.

The sample run '`samples/dust-vortex`' contains the latest setup for tracer particles.

6.14.2 Dust particles

Dust particles are allowed to have a velocity that is not similar to the gas,

$$\frac{{\rm d}\boldsymbol{x}_i}{{\rm d}t} = \boldsymbol{v}_i\,. \tag{93}$$

[15] In practice, we calculate $\partial_j(\hat{B}_i\hat{B}_j) = (\delta_{ij} - 2\hat{B}_i\hat{B}_k)\hat{B}_j B_{k,j}/|\boldsymbol{B}|$, where derivatives of $\boldsymbol{B}$ are calculated as $B_{i,j} = \epsilon_{ikl}A_{l,jk}$.

The particle velocity follows an equation of motion similar to a fluid, only there is no advection term. Dust particles also experience a drag force from the gas (proportional to the velocity difference between a particle and the gas).

$$\frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = \ldots - \frac{1}{\tau_\mathrm{s}}(\boldsymbol{v}_i - \boldsymbol{u})\,. \tag{94}$$

Here τ_s is the stopping time of the dust particle. The interpolation of the gas velocity to the position of a particle is done using one of three possible particle-mesh schemes,

- NGP (Nearest Grid Point, default)
  The gas velocity at the nearest grid point is used.
- CIC (Cloud in Cell, set `lparticlemesh_cic=T`)
  A first order interpolation is used to obtain the gas velocity field at the position of a particle. Affects 8 grid points.
- TSC (Triangular Shaped Cloud, set `lparticlemesh_tsc=T`)
  A second order spline interpolation is used to obtain the gas velocity field at the position of a particle. Affects 27 grid points.

The particle description is the proper description of dust grains, since they do not feel any pressure forces (too low number density). Thus there is no guarantee that the grains present within a given volume will be equilibrated with each other, although drag force may work for small grains to achieve that. Larger grains (meter-sized in protoplanetary discs) must be treated as individual particles.

To conserve momentum the dust particles must affect the gas with a friction force as well. The strength of this force depends on the dust-to-gas ratio ϵ_d, and it can be safely ignored when there is much more gas than there is dust, e.g. when $\epsilon_\mathrm{d} = 0.01$. The friction force on the gas appears in the equation of motion as

$$\frac{\partial \boldsymbol{u}}{\partial t} = \ldots - \frac{\rho_\mathrm{p}^{(i)}}{\rho}\left(\frac{\partial \boldsymbol{v}^{(i)}}{\partial t}\right)_\mathrm{drag} \tag{95}$$

Here $\rho_\mathrm{p}^{(i)}$ is the dust density that particle i represents. This can be set through the parameter `eps_todt` in `&particle_init_pars`. The drag force is assigned from the particles onto the mesh using either NGP, CIC or TSC assignment. The same scheme is used both for interpolation and for assignment to avoid any risk of a particle accelerating itself (see Hockney & Eastwood 1981).

6.15 N-body solver

The N-body code takes advantage of the existing Particles module, which was coded with the initial intent of treating solid particles whose radius $a_\bullet$ is comparable to the mean free path λ of the gas, for which a fluid description is not valid. A N-body implementation based on that module only needed to include mass as extra state for the particles, solve for the N^2 gravitational pair interactions and distinguish between the N-body and the small bodies that are mapped into the grid as a ρ_p density field.

The particles of the N-body ensemble evolve due to their mutual gravity and by interacting with the gas and the swarm of small bodies. The equation of motion for particle i

is

$$\frac{d\boldsymbol{v}_{p_i}}{dt} = \boldsymbol{F}_{g_i} - \sum_{j\neq i}^{N} \frac{GM_j}{\mathcal{R}_{ij}^2}\hat{\boldsymbol{\mathcal{R}}}_{ij} \tag{96}$$

where $\mathcal{R}_{ij} = |\boldsymbol{r}_{p_i} - \boldsymbol{r}_{p_j}|$ is the distance between particles i and j, and $\hat{\boldsymbol{\mathcal{R}}}_{ij}$ is the unit vector pointing from particle j to particle i. The first term of the R.H.S. is the combined gravity of the gas and of the dust particles onto the particle i, solved via

$$\boldsymbol{F}_{g_i} = -G\int_V \frac{[\rho_g(\boldsymbol{r}) + \rho_p(\boldsymbol{r})]\boldsymbol{\mathcal{R}}_i}{(\mathcal{R}_i^2 + b_i^2)^{3/2}}\mathrm{d}V, \tag{97}$$

where the integration is carried out over the whole disk. The smoothing distance b_i is taken to be as small as possible (a few grid cells). For few particles ($<$10), calculating the integral for every particle is practical. For larger ensembles one would prefer to solve the Poisson equation to calculate their combined gravitational potential.

The evolution of the particles is done with the same third-order Runge-Kutta time-stepping routine used for the gas. The particles define the timestep also by the Courant condition that they should not move more than one cell at a time. For pure particle runs, where the grid is absent, one can adopt a fixed time-step $t_p \ll 2\pi\Omega_{\rm fp}^{-1}$ where $\Omega_{\rm fp}$ is the angular frequency of the fastest particle.

By now (spring 2009), no inertial accelerations are included in the N-body module, so only the inertial frame - with origin at the barycenter of the N-body ensemble - is available. For a simulation of the circular restricted three-body problem with mass ratio q=10^{-3}, the Jacobi constant of a test particle initially placed at position (x, y)=(2,0) was found to be conserved up to one part in 10^5 within the time span of 100 orbits.

We stress that the level of conservation is poor when compared to integrators designed to specifically deal with long-term N-body problems. These integrators are usually symplectic, unlike the Runge-Kutta scheme of the PENCIL CODE. As such, PENCIL should not be used to deal with evolution over millions of years. But for the time-span typical of astrophysical hydrodynamical simulations, this degree of conservation of the Jacobi constant can be deemed acceptable.

As an extension of the particle's module, the N-body is fully compatible with the parallel optimization of PENCIL, which further speeds up the calculations. Parallelization, however, is not yet possible for pure particle runs, since it relies on splitting the grid between the processors. At the time of writing (spring 2009), the N-body code does not allow the particles to have a time-evolving mass.

6.16 Test-field equations

The test-field method is used to calculate turbulent transport coefficients for magneto-hydrodynamics. This is a rapidly evolving field and we refer the interested reader to recent papers in this field, e.g. by Sur et al. (2008) or Brandenburg et al. (2008). For technical details, see also Sect. F.2.

7 Troubleshooting / Frequently Asked Questions

7.1 Download and setup

7.1.1 Download forbidden

A: Both Google Code and SourceForge are banned from countries on the United States Office of Foreign Assets Control sanction list, including Cuba, Iran, Libya, North Korea, Sudan and Syria; see `http://en.wikipedia.org/wiki/Google_Code` and `http://en.wikipedia.org/wiki/SourceForge`. As a remedy, you might download a tarball from `http://pencil-code.nordita.org/`; see also Section 2.

7.1.2 When sourcing the '`sourceme.sh`'/'`sourceme.csh`' file or running `pc_setupsrc`, I get error messages from the shell, like 'if: Expression Syntax.' or 'set: Variable name must begin with a letter.'

A: This sounds like a buggy shell setup, either by yourself or your system administrator — or a shell that is even more idiosyncratic than the ones we have been working with.

To better diagnose the problem, collect the following information before filing a bug report to us:

1. `uname -a`
2. `/bin/csh -v`
3. `echo $version`
4. `echo $SHELL`
5. `ps -p $$`
6. If you have problems while sourcing the '`sourceme`' script,
   (a) unset the `PENCIL_HOME` variable:

   for *csh* and similar: `unsetenv PENCIL_HOME`

   for *bash* and similar: `unexport PENCIL_HOME; unset PENCIL_HOME`

   (b) switch your shell in verbose mode,

   for *csh* and similar: `set verbose; set echo`

   for *bash* and similar: `set -v; set -x`

   then source again.
7. If you have problems with `pc_setupsrc`, run it with `csh` in verbose mode:

```
/bin/csh -v -x $PENCIL_HOME/bin/pc_setupsrc
```

7.2 Compilation

7.2.1 Linker can't find the syscalls functions:

```
ld: 0711-317 ERROR: Undefined symbol: .is_nan_c
ld: 0711-317 ERROR: Undefined symbol: .sizeof_real_c
ld: 0711-317 ERROR: Undefined symbol: .system_c
ld: 0711-317 ERROR: Undefined symbol: .get_env_var_c
ld: 0711-317 ERROR: Undefined symbol: .get_pid_c
ld: 0711-317 ERROR: Undefined symbol: .file_size_c
```

A: The Pencil Code needs a working combination of a Fortran- and a C-compiler. If this is not correctly set up, usually the linker won't find the functions inside the *syscalls* module. If that happens, either the combination of C- and Fortran-compiler is inappropriate (e.g. *ifort* needs *icc*), or the compiler needs additional flags, like *g95* might need the option '`-fno-second-underscore`' and *xlf* might need the option '`-qextname`'. Please refer to Sect. 5.2, Table 1.

7.2.2 Make gives the following error now:

```
PGF90-S-0017-Unable to open include file: chemistry.h (nochemistry.f90: 43)
  0 inform,   0 warnings,   1 severes, 0 fatal for chemistry

Line 43 of the nochemistry routine, only has 'contains'.
```

A: This is because somebody added a new module (together with a corresponding *nomodule.f90* and a *module.h* file (chemistry in this case). These files didn't exist before, so you need to say:

```
pc_setupsrc
```

If this does not help, say first `make clean` and then `pc_setupsrc`.

7.2.3 How do I compile the PENCIL CODE *with the Intel (ifc) compiler under Linux?*

A: The PENCIL CODE should compile successfully with *ifc* 6.x, *ifc* 7.0, sufficiently recent versions of *ifc* 7.1 (you should get the latest version; if yours is too old, you will typically get an 'internal compiler error' during compilation of '`src/hydro.f90`'), as well as with recent versions of *ifort* 8.1 (8.0 may also work).

You can find the *ifort* compiler at `ftp://download.intel.com/software/products/compilers/downloa`

On many current (as of November 2003) Linux systems, there is a mismatch between the *glibc* versions used by the compiler and the linker. To work around this, use the following flag for compiling

```
FC=ifc -i_dynamic
```

and set the environment variable

```
LD_ASSUME_KERNEL=2.4.1; export LD_ASSUME_KERNEL
```

or

```
setenv LD_ASSUME_KERNEL 2.4.1
```

This has solved the problems e.g. on a system with *glibc-2.3.2* and kernel *2.4.22*.

Thanks to Leonardo J. Milano (`http://udel.edu/~lmilano/`) for part of this info.

7.2.4 I keep getting segmentation faults with 'start.x' when compiling with ifort 8.0

A: There was/is a number of issues with *ifort* 8.0. Make sure you have the latest patches applied to the compiler. A number of things to consider or try are:

1. Compile with the the '`-static -nothreads`' flags.
2. Set your stacksize to a large value (but a far too large value may be problematic, too), e. g.

   ```
   limit stacksize 256m
   ulimit -s 256000
   ```

3. Set the environment variable KMP_STACKSIZE to a large value (like `100M`)

See also `http://softwareforums.intel.com/ids/board/message?board.id=11&message.id=1375`

7.2.5 When compiling with MPI on a Linux system, the linker complains:

```
mpicomm.o: In function 'mpicomm_mpicomm_init_':
mpicomm.o(.text+0x36): undefined reference to 'mpi_init_'
mpicomm.o(.text+0x55): undefined reference to 'mpi_comm_size_'
mpicomm.o(.text+0x6f): undefined reference to 'mpi_comm_rank_'
[...]
```

A: This is the infamous *underscore problem*. Your *MPI* libraries have been compiled with *G77* without the option '`-fno-second-underscore`', which makes the *MPI* symbol names incompatible with other Fortran compilers.

As a workaround, use

```
MPICOMM = mpicomm_
```

in '`Makefile.local`'. Or, even better, you can set this globally (for the given computer) by inserting that line into the file '`~/.adapt-mkfile.inc`' (see `perldoc adapt-mkfile` for more details).

7.2.6 Compilation stops with the cryptic error message:

```
f95  -O3 -u -c .f90.f90
Error : Could not open sourcefile .f90.f90
compilation aborted for .f90.f90 (code 1)
make[1]: *** [.f90.o] Error 1
```

What is the problem?

A: There are two possibilities:

1. One of the variables for *make* has not been set, so *make* expands it to the empty string. Most probably you forgot to specify a module in 'src/Makefile.local'. One possibility is that you have upgraded from an older version of the code that did not have some of the modules the new version has.

   Compare your 'src/Makefile.local' to one of the examples that work.

2. One of the variables for *make* has a space appended to it, e. g. if you use the line

   ```
   MPICOMM = mpicomm␣
   ```

   (see § 7.2.5) with a trailing blank, you will encounter this error message. Remove the blank. This problem can also occur if you added a new module (and have an empty space after the module name in 'src/Makefile.src', i.e. CHIRAL=nochiral␣), in which case the compiler will talk about "circular dependence" for the file 'nochiral'.

7.2.7 The code doesn't compile,

. . . there is a problem with *mvar*:

```
make start.x run.x
f95 -O3 -u   -c cdata.f90
Error: cdata.f90, line 71: Implicit type for MVAR
       detected at MVAR@)
[f95 terminated - errors found by pass 1]
make[1]: *** [cdata.o] Error 2
```

A: Check and make sure that 'mkcparam' (directory '$PENCIL_HOME/bin') is in your path. If this doesn't help, there may be an *empty* 'cparam.inc' file in your 'src' directory. Remove 'cparam.inc' and try again (Note that 'cparam.inc' is automatically generated from the 'Makefile').

7.2.8 Some samples don't even compile,

as you can see on the web, http://www.nordita.org/software/pencil-code/tests.html.

```
samples/helical-MHDturb:
    Compiling..           not ok:
  make start.x run.x read_videofiles.x
make[1]: Entering directory '/home/dobler/f90/pencil-code/samples/helical-MHDturb/src'
/usr/lib/lam/bin/mpif95  -O3   -c initcond.f90
/usr/lib/lam/bin/mpif95  -O3   -c density.f90
      use Gravity, only: gravz, nu_epicycle
                                ^
Error 208 at (467:density.f90) : No such entity in the module
Error 355 : In procedure INIT_LNRHO variable NU_EPICYCLE has not been given a type
Error 355 : In procedure POLYTROPIC_LNRHO_DISC variable NU_EPICYCLE has not been given a
3 Errors
compilation aborted for density.f90 (code 1)
make[1]: *** [density.o] Error 1
```

```
make[1]: Leaving directory '/home/dobler/f90/pencil-code/samples/helical-MHDturb/src'
make: *** [code] Error 2
```

A: Somebody may have checked in something without having run auto-test beforehand. The problem here is that something has been added in one module, but not in the corresponding no-module. You can of course check with *svn* who it was...

7.2.9 Internal compiler error with Compaq/Dec F90

The Dec Fortran optimizer has occasional problems with '`nompicomm.f90`':

```
make start.x run.x read_videofiles.x
f90  -fast -O3 -tune ev6 -arch ev6  -c cparam.f90
[...]
f90  -fast -O3 -tune ev6 -arch ev6  -c nompicomm.f90
otal vm  2755568      otal vm  2765296        otal vm  2775024
otal vm  2784752      otal...
Assertion failure:  Compiler internal error - please submit problem r...
  GEM ASSERTION, Compiler internal error - please submit problem report
Fatal error in: /usr/lib/cmplrs/fort90_540/decfort90 Terminated
*** Exit 3
Stop.
*** Exit 1
Stop.
```

A: The occurrence of this problem depends upon the grid size; and the problem never seems to occur with '`mpicomm.f90`', except when `ncpus=1`. The problem can be avoided by switching off the loop transformation optimization (part of the '`-O3`' optimization), via:

```
#OPTFLAGS=-fast -O3 -notransform_loops
```

This is currently the default compiler setting in '`Makefile`', although it has a measurable performance impact (some 8% slowdown).

7.2.10 Assertion failure under SunOS

Under SunOS, I get an error message like

```
user@sun> f90 -c param_io.f90
Assertion failed: at_handle_table[at_idx].tag == VAR_TAG,
                  file ../srcfw/FWcvrt.c, line 4018
f90: Fatal error in f90comp: Abort
```

A: This is a compiler bug that we find at least with Sun's WorkShop Compiler version '5.0 00/05/17 FORTRAN 90 2.0 Patch 107356-05'. Upgrade the compiler version (and possibly also the operating system): we find that the code compiles and works with version 'Sun WorkShop 6 update 2 Fortran 95 6.2 Patch 111690-05 2002/01/17' under SunOS version '5.8 Generic_108528-11'.

7.2.11 After some dirty tricks I got pencil code to compile with MPI, ...

```
 > Before that i installed lam-7.1.4 from source.

Goodness gracious me, you shouldn't have to compile your own MPI library.
```

A: Then don't use the old LAM-MPI. It is long superseded by open-mpi now. Open-mpi doesn't need a daemon to be running. I am using the version that ships with Ubuntu (e.g. 9.04):

```
  frenesi:~> aptitude -w 210 search openmpi | grep '^i'

i   libopenmpi-dev - high performance message passing library -- header files
i A libopenmpi1    - high performance message passing library -- shared library
i   openmpi-bin    - high performance message passing library -- binaries
i A openmpi-common - high performance message passing library -- common files
i   openmpi-doc    - high performance message passing library -- man pages
```

Install that and keep your configuration (Makefile.src and getconf.csh) close to that for 'frenesi' or 'norlx50'. That should work.

7.2.12 Error: Symbol 'mpi_comm_world' at (1) has no IMPLICIT type

```
I installed the pencil code on Ubuntu system and tested "run.csh"
in ...\samples\conv-slab.  Here the code worked pretty well.
Nevertheless, running (auto-test), I found there are some errors.

The messages are,

Error: Symbol 'mpi_comm_world' at (1) has no IMPLICIT type
Fatal Error: Error count reached limit of 25.
make[2]: *** [mpicomm_double.o] Error 1
make[2]: Leaving directory
'/home/pkiwan/Desktop/pencil-code/samples/2d-tests/selfgravitating-shearwave/src'
make[1]: *** [code] Error 2
make[1]: Leaving directory
'/home/pkiwan/Desktop/pencil-code/samples/2d-tests/selfgravitating-shearwave/src'
make: *** [default] Error 2

Finally, ### auto-test failed ###

Will it be OK? Or, how can I fix this?
```

A: Thanks for letting me know about the status, and congratulations on your progress! Those tests that fail are those that use MPI. If your machine is a dual or multi core machine, you could run faster by running under MPI. But this is probably not crucial for you at this point. (I just noticed that there is a ToDo listed in the auto-test command to implement the option not to run the MPI tests, but this hasn't been done yet. So I guess you can start with the science next.

7.2.13 Error: Can't open included file 'mpif.h'

It always worked, but now, after some systems upgrade, I get

```
gfortran -O3 -o mpicomm.o -c mpicomm.f90
Error: Can't open included file 'mpif.h'
```

When I say `locate mpif.h` I only get things like

```
/scratch/ntest/1.2.7p1-intel/include/mpif.h
```

But since I use `FC=mpif90` I thought I don't need to worry.

A: Since you use `FC=mpif90` there must definitely be something wrong with their setup. Try `mpif90 -showme` or `mpif90 -show`; the '-I' option should say where it looks for 'mpif.h'. If those directories don't exist, it's no wonder that it doesn't work, and it is time to complain.

7.3 Pencil check

7.3.1 The pencil check complains for no reason.

A: The pencil check only complains for a reason.

7.3.2 The pencil check reports MISSING PENCILS and quits

A: This could point to a serious problem in the code. Check where the missing pencil is used in the code. Request the right pencils, likely based on input parameters, by adapting one or more of the `pencil_criteria_MODULE` subroutines.

7.3.3 The pencil check reports unnecessary pencils

The pencil check reports `possible overcalculation... pencil rho ( 43) is requested, but does not appear to be required!`

A: Such warnings show that your simulation is possibly running too slowly because it is calculating pencils that are not actually needed. Check in the code where the unnecessary pencils are used and adapt one or more of the `pencil_criteria_MODULE` subroutines to request pencils only when they are actually needed.

7.3.4 The pencil check reports that most or all pencils are missing

A: This is typically a thing that can happen when testing new code development for the first time. It is usually an indication that the reference `df` changes every time you call `pde`. Check whether any newly implemented subroutines or functionality has a "memory", i.e. if calling the subroutine twice with the same `f` gives different output `df`.

7.3.5 Running the pencil check triggers mathematical errors in the code

A: The pencil check puts random numbers in `f` before checking the dependence of `df` on the chosen set of pencils. Sometimes these random numbers are inconsistent with the physics and cause errors. In that case you can set `lrandom_f_pencil_check=F` in `&run_pars` in '`run.in`'. The initial condition may contain many idealized states (zeros or ones) which then do not trigger pencil check errors when `lrandom_f_pencil_check=F`, even if pencils are missing. But it does prevent mathematical inconsistencies.

7.3.6 The pencil check still complains

A: Then you need to look into the how the code and the pencil check operate. Reduce the problem in size and dimensions to find the smallest problem that makes the pencil check fail (e.g. 1x1x8 grid points). At the line of '`pencil_check.f90`' when a difference is found between `df_ref` and `df`, add some debug lines telling you which variable is inconsistent and in what place. Often you will be surprised that the pencil check has correctly found a problem in the simulation.

7.3.7 The pencil check is annoying so I turned it off

A: Then you are taking a major risk. If one or more pencils are not calculated properly, then the results will be wrong.

7.4 Running

7.4.1 Why does '`start.x`' / '`start.csh`' write data with periodic boundary conditions?

A: Because you are setting the boundary conditions in '`run.in`', not in '`start.in`'; see Sect. 5.16.1. There is nothing wrong with the initial data — the ghost-zone values will be re-calculated during the very first time step.

7.4.2 csh problem?

Q: On some rare occasions we have problems with csh not being supported on other machines. (We hope to fix this by contacting the responsible person, but may not be that trivial today!) Oliver says this is a well known bug of some years ago, etc. But maybe in the long run it would be good to avoid csh.

A: These occasions will become increasingly frequent, and eventually for some architectures, there may not even be a csh variant that can be installed.

We never pushed people to use `pc_run` and friends (and to report corresponding bugs and get them fixed), but if we don't spend a bit of effort (or annoy users) now, we create a future emergency, where someone needs to run on some machine, but there is no csh and he or she just gets stuck.

We don't have that many csh files, and for years now it should be possible to compile run without csh (using `bin/pc_run`) — except that people still fall back on the old way of

doing things. This is both cause and consequence of the 'new' way not being tested that much, at least for the corner cases like 'RERUN', 'NEWDIR', 'SCRATCH_DIR'.

7.4.3 'run.csh' doesn't work:

```
Invalid character ''' in NAMELIST input
Program terminated by fatal I/O error
Abort
```

A: The string array for the boundary condition, e.g. *bcx* or *bcz* is too long. Make sure it has exactly as many elements as *nvar* is big.

7.4.4 Namelist problem under IRIX

Under IRIX, I get

```
lib-4001 : UNRECOVERABLE library error

Encountered during a namelist READ from unit 1
Fortran unit 1 is connected to a sequential formatted text file: "run.in"
IOT Trap
Abort
```

A: This is a compiler bug that has been found at least with the MIPSpro F90 compiler version 7.3.1.3m. The problem seems to have been fixed in version 7.4.20m.

The error comes and goes, depending on the configuration (and possibly even the input parameters) you are using. Until SGI fix their compiler, you can experiment with adding new variables to the module *Param_IO*; this has solved the problem once for us. If this trick does not help, you will need to turn your namelist input (at least 'run.in' into Fortran statements, include them into a replacement version of 'param_io.f90', and recompile each time you make changes.

7.4.5 Code crashes after restarting

```
>  > removing mu_r from the namelist just 'like that' makes the code
>  > backwards incompatible.
>
> That means that we can never get rid of a parameter in start.in once we
> have introduced it, right?
```

A: In the current implementation, without a corresponding cleaning procedure, unfortunately yes.

Of course, this does not affect users' private changes outside the central svn tree.

7.4.6 auto-test gone mad...?

Q: Have you ever seen this before:

```
giga01:/home/pg/n7026413/cvs-src/pencil-code/samples/conv-slab> auto-test
.

/home/pg/n7026413/cvs-src/pencil-code/samples/conv-slab:
    Compiling..           ok
        No data directory; generating data -> /var/tmp/pencil-tmp-25318
    Starting..            ok
    Running..             ok
    Validating results..Malformed UTF-8 character (unexpected continuation
byte 0x80, with no preceding start byte) in split at
/home/pg/n7026413/cvs-src/pencil-code/bin/auto-test line 263.
Malformed UTF-8 character (unexpected continuation byte 0x80, with no
preceding start byte) in split at
/home/pg/n7026413/cvs-src/pencil-code/bin/auto-test line 263.
```

A: You are running on a RedHat 8 or 9 system, right?

Set `LANG=POSIX` in your shell's startup script and life will be much better.

7.4.7 Can I restart with a different number of cpus?

Q: I am running a simulation of nonhelical turbulence on the cluster using MPI. Suppose if I am running a 128^3 simulation on 32 cpus/cores i.e.

```
integer, parameter :: ncpus=32,nprocy=2,nprocz=ncpus/nprocy,nprocx=1
integer, parameter :: nxgrid=128,nygrid=nxgrid,nzgrid=nxgrid
```

And I stop the run after a bit. Is there a way to resume this run with different number of cpus like this :

```
integer, parameter :: ncpus=16,nprocy=2,nprocz=ncpus/nprocy,nprocx=1
integer, parameter :: nxgrid=128,nygrid=nxgrid,nzgrid=nxgrid
```

I understand it has to be so in a new directory but making sure that the run starts from where I left it off in the previous directory.

A: The answer is no, if you use the standard distributed io. There is also parallel io, but I never used it. That would write the data in a single file, and then you could use the data for restart in another processor layout.

7.4.8 Can I restart with a different number of cpus?

Q: Is it right that once the simulation is resumed, pencil-code takes the last data from var.dat (which is the current snapshot of the fields)? If that is true, then, is it not possible to give that as the initial condition for the run in the second directory (with changed "ncpus")? Is there a mechanism already in place for that?

A: Yes, the code restarts from the last var.dat. It is written after a successful completion of the run, but it crashes or you hit a time-out, there will be a var.dat that is overwritten every isave timesteps. If the system stops during writing, some var.dat files may be corrupt or have the wrong time. In that case you could restart from a good VAR file, if you have one, using, e.g.,

```
restart-new-dir-VAR . 46
```

where 46 is the number of your VAR file, i.e., VAR46 im this case. To restart in another directory, you say, from the old run directory,

```
restart-new-dir ../another_directory
```

Hope this helps. Look into pencil-code/bin/restart-new-dir to see what it is doing.

7.4.9 fft_xyz_parallel_3D: nygrid needs to be an integer multiple...

Q: I just got an:

```
fft_xyz_parallel_3D: nygrid needs to be an integer multiple of nprocy*nprocz
```

In my case, nygrid=2048, nprocy=32, and nprocz=128, so nprocy*nprocz=4096. In other words, 2048 needs to be a multiple of 4096. But isn't this the case then?

A: No, because 2048 = 0.5 * 4096 and 0.5 is not an integer. Maybe try either setting nprocz=64 or nprocy=64. You could compensate the change of ncpus with the x-direction. For 2048^3 simulations, nprocy=32 and nprocz=64 would be good.

7.4.10 Unit-agnostic calculations?

Q: The manual speaks about unit-agnostic calculations, stating that one may choose to interpret the results in any (consistent) units, depending on the problem that is solved at hand. So, for example, if I chose to run the '`2d-tests/battery_term`' simulation for an arbitrary number of time-steps and then choose to examine the diagnostics, am I correct in assuming the following:

```
1) [Brms] = Gauss (as output by unit_magnetic, before the run begins)
2) [t] = s (since the default unit system is left as CGS)
3) [urms] = cm/s (again, as output by unit_velocity, before the run begins)
4) and etc. for the units of the other diagnostics
```

A: Detailed correspondence on this item can be found on: `https://groups.google.com/forum/?fromgroups#!topic/pencil-code-discuss/zek-uYNbgXI` There is also working material on unit systems under `http://www.nordita.org/~brandenb/teach/PencilCode/MixedTopics.html` with a link to `http://www.nordita.org/~brandenb/teach/PencilCode/material/AlfvenWave_SIunits/` Below is a pedagogical response from Wlad Lyra:

```
In the sample battery-term, the sound speed cs0=1 sets the unit of
velocity. Together with the unit of length, that sets your unit of
time. The unit of magnetic field follows from the unit of velocity,
density, and your choice of magnetic permittivity, according to the
definition of the Alfven velocity.

If you are assuming cgs, you are saying that your sound speed cs0=1
actually means [U]=1 cm/s. Your unit of length is equivalently 1 cm,
and therefore the unit of time is [t] = [L]/[U]=1 s. The unit of
density is [rho] = 1 g/cm^3. Since in cgs vA=B/sqrt(4*pi * rho), your
unit of magnetic field is [B] = [U] * sqrt([rho] * 4*pi) ~= 3.5
sqrt(g/cm) / s = 3.5 Gauss.
```

If instead you are assuming SI, you have cs0=1 assuming that means [U]=1 m/s and rho0=1 assuming that to mean [rho]=1 kg/m^3. Using [L]=1 m, you have still [t]=1 s, but now what appears as B=1 in your output is actually [B] = [U] * sqrt (mu * [rho]) = 1 m/s * sqrt(4*pi * 1e-7 N*A-2 1 kg/m^3) ~= 0.0011210 kg/(s^2*A) ~ 11 Gauss.

You can make it more interesting and use units relevant to the problem. Say you are at the photosphere of the Sun. You may want to use dimensionless cs0=1 meaning a sound speed of 10 km/s. Your appropriate length can be a megameter. Now your time unit is [t]=[L]/[U] = 1e3 km/ 10 km/s = 10^2 s, i.e., roughly 1.5 minute. For density, assume rho=2x10-4 kg/m^3, typical of the solar photosphere. Your unit of magnetic field is therefore [B] = [U] * sqrt([rho] * 4*pi) = 1e6 cm/s * sqrt(4*pi * 2e-7 g/cm^3) ~ 1585.33 Gauss.

Notice that for mu0=1 and rho0=1 you simply have vA=B. Then you can conveniently set the field strength by your choice of plasma beta (= 2*cs^2/vA^2). There's a reason why we like dimensionless quantities!

7.5 Visualization

7.5.1 'start.pro' doesn't work:

```
Reading grid.dat..
Reading param.nml..
\% Expression must be a structure in this context: PAR.
\% Execution halted at:  \$MAIN\$            104
/home/brandenb/pencil-code/runs/forced/hel1/../../../idl/start.pro
```

A: You don't have the subdirectory 'data' in your IDL variable *!path*. Make sure you source 'sourceme.csh'/'sourceme.sh' or set a sufficient IDL path otherwise.

7.5.2 'start.pro' doesn't work:

Isn't there some clever (or even trivial) way that one can avoid the annoying error messages that one gets, when running e.g. ".r rall" after a new variable has been introduced in "idl/varcontent.pro"? Ever so often there's a new variable that can't be found in my param2.nml – this time it was IECR, IGG, and ILNTT that I had to circumvent...

A: The simplest solution is to invoke 'NOERASE', i.e. say

```
touch NOERASE
start.csh
```

or, alternatively, start_run.csh. What it does is that it reruns src/start.x with a new version of the code; this then produces all the necessary auxiliary files, but it doesn't overwrite or erase the 'var.dat' and other 'VAR' and 'slice' files.

7.5.3 Something about tag name undefined:

Q: In one of my older run directories I can't read the data with idl anymore. What should I do? Is says something like

```
Reading param.nml..
% Tag name LEQUIDIST is undefined for structure <Anonymous>.
% Execution halted at: $MAIN$            182
  /people/disk2/brandenb/pencil-code/idl/start.pro
```

A: Go into '`data/param.nml`' and add , `LEQUIDIST=T` anywhere in the file (but before the last slash).

7.5.4 Something INC in start.pro

Q: start doesn't even work:

```
% Compiled module: $MAIN$.
nname=      11
Reading grid.dat..
Reading param.nml..
Can't locate Namelist.pm in INC (INC contains: /etc/perl /usr/local/lib/perl/5.8.4 /usr
BEGIN failed--compilation aborted at /home/brandenb/pencil-code/bin/nl2idl line 49.
```

A: Go into '`$PENCIL_HOME`' and say `svn up sourceme.csh` and/or `svn up sourceme.sh`. (They were just out of date.)

7.5.5 nl2idl problem when reading param2.nml

Q: Does anybody encounter a backward problem with nl2idl? The file param*.nml files are checked in under '`pencil-code/axel/couette/SStrat128a_mu0.20_g2`' and the problem is below.

```
 at /people/disk2/brandenb/pencil-code/bin/nl2idl line 120
HCOND0= 0.0,HCOND1= 1.000000,HCOND2= 1.000000,WIDTHSS= 1.192093E-06,MPOLY0=
^------  HERE
 at /people/disk2/brandenb/pencil-code/bin/nl2idl line 120
```

A: The problem is the stupid ifc compiler writing the following into the namelist file:

```
  COOLING_PROFILE='gaussian                  ',COOLTYPE='Temp
  'COOL= 0.0,CS2COOL= 0.0,RCOOL= 1.000000,WCOOL= 0.1000000,FBOT= 0.0,CHI_T= 0.0
```

If you add a comma after the closing quote:

```
  COOLING_PROFILE='gaussian                  ',COOLTYPE='Temp
  ',COOL= 0.0,CS2COOL= 0.0,RCOOL= 1.000000,WCOOL= 0.1000000,FBOT= 0.0,CHI_T= 0.0
```

things will work.

Note that ifc cannot even itself read what it is writing here, so if this happened to occur in param.nml, the code would require manual intervention after each start.csh.

7.5.6 Spurious dots in the time series file

Q: Wolfgang, you explained it to me once, but I forget. How can one remove spurious dots after the timestep number if the time format overflows?

A: I don't know whether it exists anywhere, but it's easy. In Perl you'd say

```
perl -pe 's/^(\s*[-0-9]+)\.([-0-9eEdD])/$1 $2/g'
```

and in sed (but that's harder to read)

```
sed 's/^\( *[-0-9]\+\)\.\([-0-9eEdD]\)/\1 \2/g'
```

7.6 General questions

7.6.1 "Installation" procedure

Why don't you use GNU *autoconf/automake* for installation of the PENCIL CODE?

A: What do you mean by "installation"? Unlike the applications that normally use *autoconf*, the *Pencil Code* is neither a binary executable, nor a library that you compile once and then dump somewhere in the system tree. *Autoconf* is the right tool for these applications, but not for numerical codes, where the typical compilation and usage pattern is very different:

You have different directories with different '`Makefile.local`' settings, recompile after introducing that shiny new term in your equations, etc. Moreover, you want to sometimes switch to a different compiler (but just for that run directory) or another *MPI* implementation. Our `adapt-mkfile` approach gives you this flexibility in a reasonably convenient way, while doing the same thing with *autoconf* would be using that system against most of its design principles.

Besides, it would really get on my (WD's) nerves if I had to wait two minutes for *autoconf* to finish before I can start compiling (or maybe 5–10 minutes if I worked on a NEC machine...).

Finally, if you have ever tried to figure out what a '`configure`' script does, you will appreciate a comprehensible configuration system.

7.6.2 Small numbers in the code

What is actually the difference between epsi, tini and tiny?

A:

```
F90 has two functions epsilon() and tiny(), with

  epsilon(x) = 1.1920929e-07
  tiny(x)    = 1.1754944e-38
(and then there is huge(x) = 3.4028235e+38)
for a single-precision number x.

epsilon(x) is the smallest number that satisfies
```

```
  1+epsilon(1.) /= 1 ,
while tiny(x) is the smallest number that can be represented without
precision loss.

In the code we have variants hereof,
   epsi=5*epsilon(1.0)
   tini=5*tiny(1.0)
   huge1=0.2*huge(1.0)
that have added safety margins, so we don't have to think about doing
things like 1/tini.

So in sub.f90,
  -      evr = evr / spread(r_mn+epsi,2,3)
did (minimally) affect the result for r_mn=O(1), while the correct version
  +      evr = evr / spread(r_mn+tini,2,3)
only avoids overflow.
```

7.6.3 Why do we need a /lphysics/ namelist in the first place?

Wolfgang answered on 29 July 2010: "'cdata.f90' has the explanation"

```
  ! Constant 'parameters' cannot occur in namelists, so in order to get the
  ! now constant module logicals into the lphysics name list...
  ! We have some proxies that are used to initialize private local variables
  ! called lhydro etc, in the lphysics namelist!
```

So the situation is this: we want to write parameters like ldensity to param.nml so IDL (and potentially octave, python, etc.) can know whether density was on or not. To avoid confusion, we want them to have exactly their original names. But we cannot assemble the original ldensity etc. constants in a namelist, so we have to define a local ldensity variable. And to provide it with the value of the original cdata.ldensity, we need to transfer the value via *ldensity_var*. That's pretty scary, although it seems to work fine. I can track the code back to the big *eos_merger* commit, so it may originate from that branch. One obvious problem is that you have to add code in a number of places (the `ldensity` → *ldensity_var* assignment and the local definition of ldensity) to really get what you need. And when adding a new boolean of that sort to 'cdata.f90', you may not even have a clue that you need all the other voodoo.

There may be a cleaner solution involving generated code. Maybe something like

```
  logical :: ldensity ! INCLUDE_IN_LPHYSICS
```

could later generate code (in some param_io_extra.inc file) that looks like this:

```
  write(unit, *) 'ldensity = ', ldensity
```

i.e. we can manually write in namelist format. But maybe there are even simpler solutions?

7.6.4 Can I run the code on a Mac?

A: Macs work well for Linux stuff, except that the file structure is slightly different. Problems when following Linux installs can usually be traced to the PATH. For general reference, if you need to set an environment variable for an entire OS-X login session, google environment.plist. That won't be needed here.

For a Mac install, the following should work:

a) Install Dev Tools (an optional install on the MacOS install disks). Unfortunately, last time I checked the svn version that comes with DevTools is obsolete. So:

b) Install MacPorts (download from web). Note that MacPorts installs to a non-standard location, and will need to be sourced. The installation normally drops an appropriate line in .profile. If it does so, make sure that that line gets sourced. Otherwise

```
export PATH=/opt/local/bin:/opt/local/sbin:$PATH
export MANPATH=/opt/local/share/man:$MANPATH
```

c) Install g95 (download from web). Make sure it is linked in /bin.

d) execute macports svn install

e) download the pencil-code and enjoy.

Note: the above way to get svn works. It takes a while however, so there are certainly faster ways out there. If you already have a non-obsolete svn version, use that instead.

7.6.5 Pencil Code discussion forum

Do I just need to send an email somewhere to subscribe or what?

A" The answer is yes; just go to:

```
http://groups.google.com/group/pencil-code-discuss
```

7.6.6 The manual

It would be a good idea to add this useful information in the manual, no?

A: When you have added new stuff to the code, don't forget to mention this in the '`pencil-code/doc/manual.tex`' file.

Again, the answer is yes; just go to:

```
cd pencil-code/doc/
vi manual.tex
svn ci -m "explanations about a new module in the code"
```

Part II

Programming the PENCIL CODE

All developers are supposed to have an up-to-date an entry in the file ‘`pencil-code/license/developers.txt`’ so that they can be contacted in the case a code change breaks an auto-test or other code functionality.

The PENCIL CODE has expanded approximately linearly in the number of lines of code and the number of subroutines (Fig. 6). The increase in the functionality of the code is documented by the rise in the number of sample problems (Fig. 7). It is important to monitor the performance of the code as well. Figure 8 shows that for most of the runs the run time has not changed much.

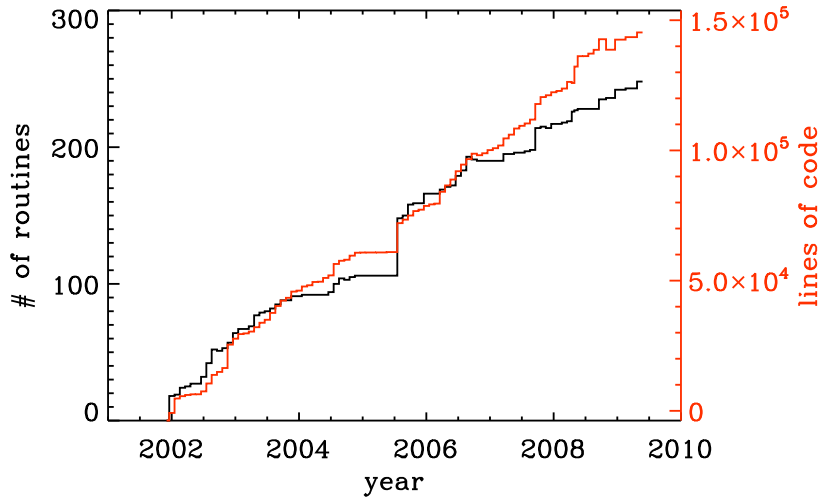

Figure 6: Number of lines of code and the number of subroutines since the end of 2001. The jump in the Summer of 2005 was the moment when the developments on the side branch (eos branch) were merged with the main trunk of the code. Note the approximately linear scaling with time.

Before making changes the to the code, it is important that you verify that you can run the `pc_auto-test` successfully. Don’t do this when you have already modified the code, because then you cannot be sure that any problems are caused by your changes, or because it wouldn’t have worked anyway. Also, keep in mind that the code is public, so your changes should make sense from a broader perspective and should not only be intended for yourself. Regarding more general aspects about coding standards see Sect. B.2.

In order to keep the development of the code going, it is important that the users are able to understand and modify (program!) the code. In this section we explain first how to orient yourself in the code and to understand what is in it, and then to modify it according to your needs.

The Pencil Code check ins occur regularly all the time. By the Pencil Code User Meeting 2010 we have arrived at a revision number of 15,000. The increase of the revision number with time is depicted in Figure 9. The number of Pencil Code developers increases too (Figure 10), but the really active ones are getting rare. This may indicate that new

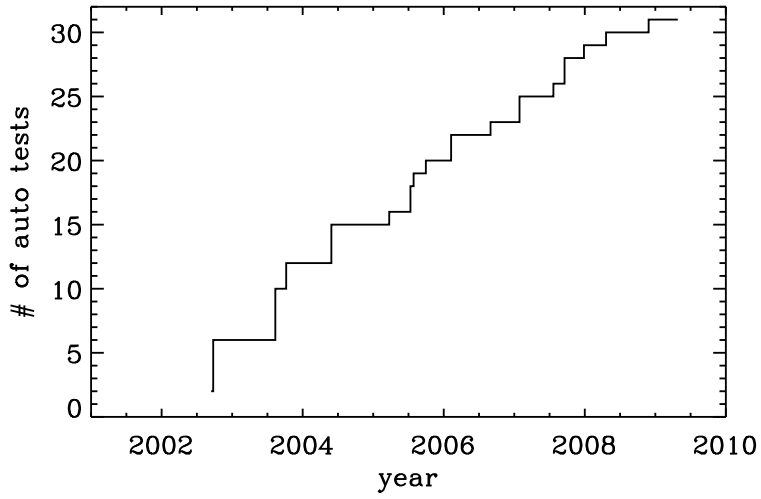

Figure 7: Number of tests in the sample directory that are used in the nightly auto tests. Note again the approximately linear scaling with time.

users can produce new science with the code as it is, but it may also indicate that it is getting harder to understand the code. This will be discussed in the next section.

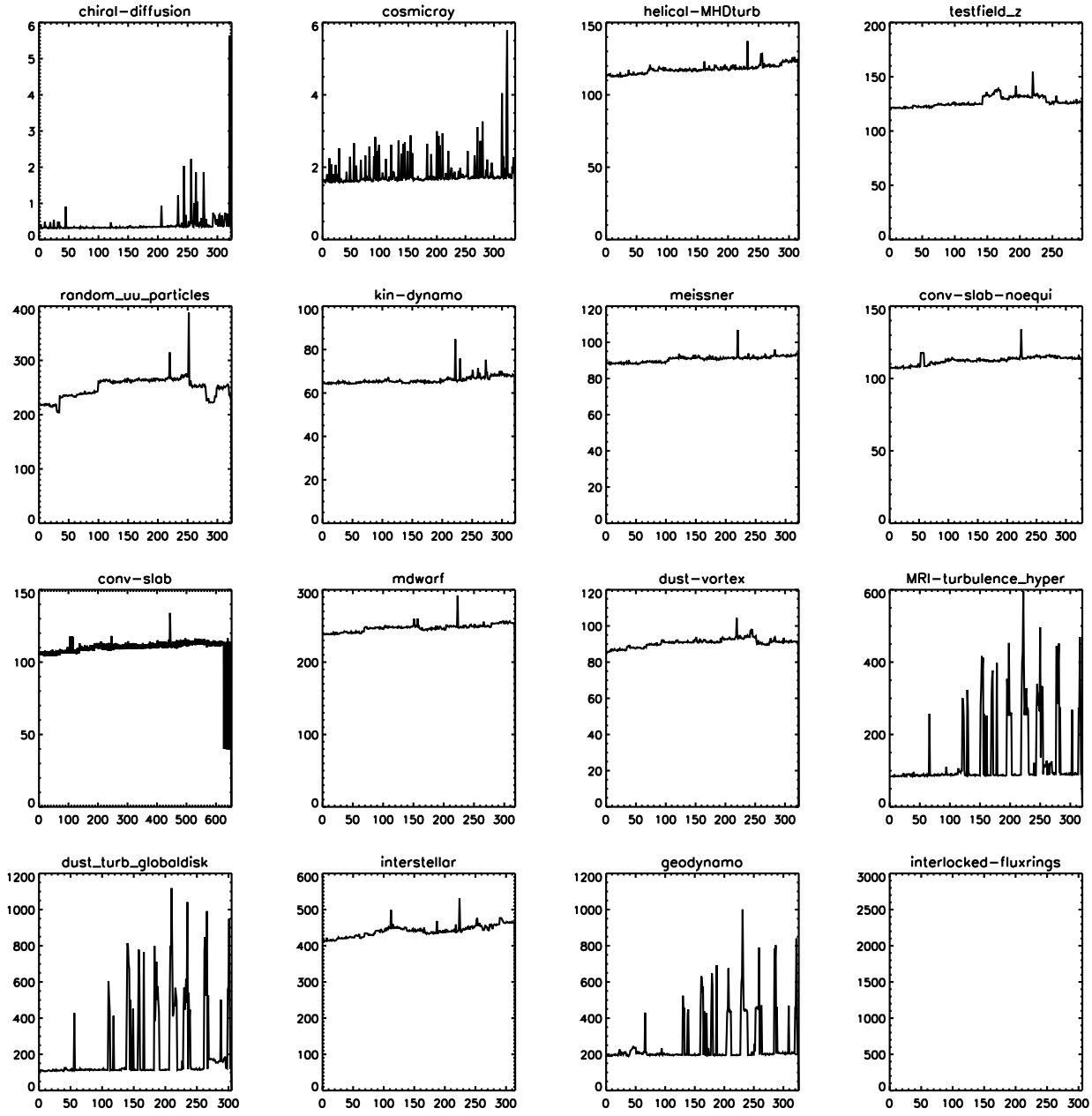

Figure 8: Run time of the daily auto-tests since August 17, 2008. For most of the runs the run time has not changed much. The occasional spikes are the results of additional load on the machine.

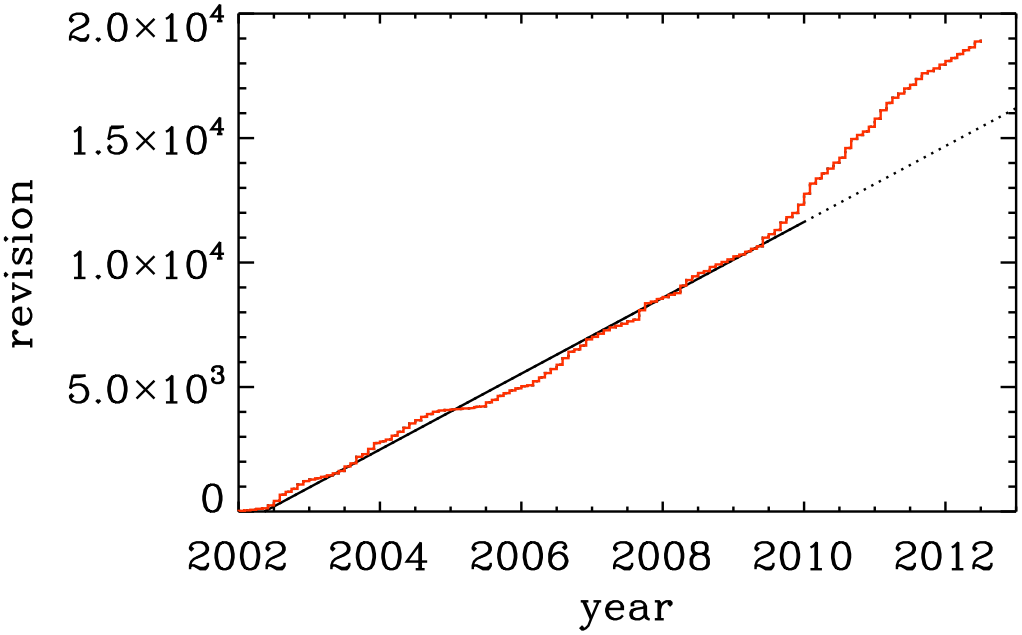

Figure 9: Number of check-ins since 2002. Note again the linear increase with time, although in the last part of the time series there is a notable speed-up.

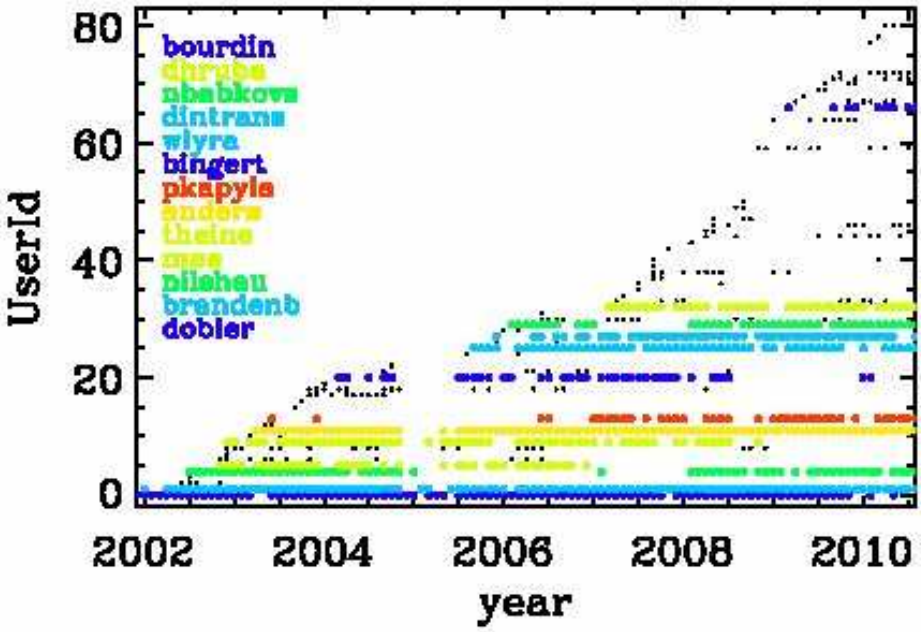

Figure 10: Check-ins since 2002 per user. Users with more than 100 check-ins are color coded.

8 Understanding the code

Understanding the code means looking through the code. This is not normally done by just printing out the entire code, but by searching your way through the code in order to address your questions. The general concept will be illustrated here with an example.

8.1 Example: how is the continuity equation being solved?

All the physics modules are solved in the routine `pde`, which is located in the file and module '`Equ`'. Somewhere in the `pde` subroutine you find the line

```
call dlnrho_dt(f,df,p)
```

This means that here the part belonging to $\partial \ln \rho / \partial t$ is being assembled. Using the `grep` command you will find that this routine is located in the module `density`, so look in there and try to understand the pieces in this routine. We quickly arrive at the following crucial part of code,

```
!
!  Continuity equation.
!
      if (lcontinuity_gas) then
        if (ldensity_nolog) then
          df(l1:l2,m,n,irho)   = df(l1:l2,m,n,irho)   - p%ugrho - p%rho*p%divu
        else
          df(l1:l2,m,n,ilnrho) = df(l1:l2,m,n,ilnrho) - p%uglnrho - p%divu
        endif
      endif
```

where, depending on some logicals that tell you whether the continuity equation should indeed be solved and whether we do want to solve for the logarithmic density and not the actual density, the correct right hand side is being assembled. Note that all these routines always only *add* to the existing `df(l1:l2,m,n,ilnrho)` array and never reset it. Resetting `df` is only done by the timestepping routine. Next, the pieces `p%uglnrho` and `p%divu` are being subtracted. These are *pencils* that are organized in the *structure* with the name `p`. The meaning of their names is obvious: `uglnrho` refers to $\boldsymbol{u} \cdot \nabla \ln \rho$ and `divu` refers to $\nabla \cdot \boldsymbol{u}$. In the subroutine `pencil_criteria_density` you find under which conditions these pencils are requested. Using `grep`, you also find where they are calculated. For example `p%uglnrho` is calculated in '`density.f90`'; see

```
call u_dot_grad(f,ilnrho,p%glnrho,p%uu,p%uglnrho,UPWIND=lupw_lnrho)
```

So this is a call to a subroutine that calculates the $\boldsymbol{u} \cdot \nabla$ operator, where there is the possibility of upwinding, but this is *not* the default. The piece `divu` is calculated in '`hydro.f90`' in the line

```
!
!  Calculate uij and divu, if requested.
!
      if (lpencil(i_uij)) call gij(f,iuu,p%uij,1)
      if (lpencil(i_divu)) call div_mn(p%uij,p%divu,p%uu)
```

Note that the divergence calculation uses the velocity gradient matrix as input, so no new derivatives are recalculated. Again, using `grep`, you will find that this calculation and many other ones happen in the module and file '`sub.f90`'. The various derivatives that enter here have been calculated using the `gij` routine, which calls the `der` routine, e.g., like so

```
      k1=k-1
      do i=1,3
        do j=1,3
          if (nder==1) then
            call der(f,k1+i,tmp,j)
```

For all further details you just have to follow the trail. So if you want to know how the derivatives are calculated, you have to look in `deriv.f90`, and only here is it where the indices of the `f` array are being addressed.

If you are interested in magnetic fields, you have to look in the file '`magnetic.f90`'. The right hand side of the equation is assembled in the routine

```
!***********************************************************************
    subroutine daa_dt(f,df,p)
!
!  Magnetic field evolution.
!
!  Calculate dA/dt=uxB+3/2 Omega_0 A_y x_dir -eta mu_0 J.
!  For mean field calculations one can also add dA/dt=...+alpha*bb+delta*WXJ.
!  Add jxb/rho to momentum equation.
!  Add eta mu_0 j2/rho to entropy equation.
!
```

where the header tells you already a little bit of what comes below. It is also here where ohmic heating effects and other possible effects on other equations are included, e.g.

```
!
!  Add Ohmic heat to entropy or temperature equation.
!
      if (lentropy .and. lohmic_heat) then
        df(l1:l2,m,n,iss) = df(l1:l2,m,n,iss) &
                          + etatotal*mu0*p%j2*p%rho1*p%TT1
      endif
```

We leave it at this and encourage the user to do similar inspection work on a number of other examples. If you think you find an error, file a ticket at `http://code.google.com/p/pencil-code/issues/list`. You can of course also repair it!

9 Adapting the code

9.1 The PENCIL CODE coding standard

As with any code longer than a few lines the appearance and layout of the source code is of the utmost importance. Well laid out code is more easy to read and understand and as such is less prone to errors.

A consistent coding style has evolved in the PENCIL CODE and we ask that those contributing try to be consistent for everybody's benefit. In particular, it would be appreciated if those committing changes of existing code via svn follow the given coding style.

There are not terribly many rules and using existing code as a template is usually the easiest way to proceed. In short the most important rules are:

- `tab` characters do not occur anywhere in the code (in fact the use of tab character is an extension to the Fortran standard).
- Code in any delimited block, e.g. if statements, do loops, subroutines etc., is indented be precisely 2 spaces. E.g.

```
if (lcylindrical) then
  call fatal_error('del2fjv','del2fjv not implemented')
endif
```

- continuation lines (i.e. the continuation part of a logical line that is split using the & sign) are indented by 4 spaces. E.g. (note the difference from the previous example)

```
if (lcylindrical) &
    call fatal_error('del2fjv','del2fjv not implemented')
[...]
```

- There is always one space separation between 'if' and the criterion following in parenthesis:

```
if (ldensity_nolog) then
  rho=f(l1:l2,m,n,irho)
endif
```

  This is wrong:

```
if(ldensity_nolog) then   ! WRONG
  rho=f(l1:l2,m,n,irho)
endif
```

- In general, try to follow common practice used elsewhere in the code. For example, in the code fragment above there are no empty spaces within the mathematical expressions programmed in the code. A unique convention helps in finding certain expressions and patterns in the code. However, empty spaces are often used after commas and semicolons, for examples in name lists.
- Relational operators are written with symbols ($==$, $/=$, $<$, $<=$, $>$, $>=$), *not* with characters (`.eq.`, `.ne.`, `.lt.`, `.le.`, `.gt.`, `.ge.`).
- In general all comments are placed on their own lines with the '!' appearing in the first column.

- All subroutine/functions begin with a standard comment block describing what they do, when and by whom they were created and when and by whom any non-trivial modifications were made.
- Lines longer that 78 characters should be explicitly wrapped using the & character, unless there is a block of longer lines that can only be read easily when they are not wrapped. Always add one whitespace before the & character.

These and other issues are discussed in more depth and with examples in Appendix B, and in particular in Sect. B.2.

9.2 Adding new output diagnostics

With the implementation of new physics and the development of new procedures it will become necessary to monitor new diagnostic quantities that have not yet been implemented in the code. In the following, we describe the steps necessary to set up a new diagnostic variable.

This is nontrivial as, in order to keep latency effects low on multi-processor machines, the code minimizes the number of global reduction operations by assembling all quantities that need the maximum taken in *fmax*, and those that need to be summed up over all processors (mostly for calculating mean quantities) in *fsum* (see subroutine `diagnostic` in file '`src/equ.f90`').

As a sample variable, let us consider *jbm* (the volume average $\langle j \cdot B \rangle$). Only the module *magnetic* will be affected, as you can see (the diagnostic quantity *jbm* is already implemented) with

```
unix>  grep -i jbm src/*.f90
```

If we pretend for the sake of the exercise that no trace of *jbm* was in the code, and we were only now adding it, we would need to do the following

1. add the variable *idiag_jbm* to the *module variables* of *Magnetic* in both '`magnetic.f90`' and '`nomagnetic.f90`':

   ```
   integer :: idiag_jbm=0
   ```

   The variable *idiag_jbm* is needed for matching the position of *jbm* with the list of diagnostic variables specified in '`print.in`'.

2. in the subroutine `daa_dt` in '`magnetic.f90`', declare and calculate the quantity *jb* (the average of which will be *jbm*), and call `sum_mn_name`

   ```
   real, dimension (nx) :: jb  ! jj.BB
   [...]
   if (ldiagnos) then          ! only calculate if diagnostics is required
     if (idiag_jbm/=0) then        ! anybody asked for jbm?
       call dot_mn(jj,bb,jb)   ! assuming jj and bb are known
       call sum_mn_name(jb,i_jbm)
     endif
   endif
   ```

3. in the subroutine `rprint_magnetic` in both '`magnetic.f90`', add the following:

```
!
!  reset everything in case of RELOAD
!  (this needs to be consistent with what is defined above!)
!
if (lreset) then  ! need to reset list of diagnostic variables?
  [...]
  idiag_jbm=0
  [...]
endif
!
!  check for those quantities that we want to evaluate online
!
do iname=1,nname
  [...]
  call parse_name(iname,cname(iname),cform(iname),'jbm',idiag_jbm)
  [...]
enddo
[...]
!
!  write column, i_XYZ, where our variable XYZ is stored
!
[...]
write(3,*) 'i_jbm=',idiag_jbm
[...]
```

4. in the subroutine `rprint_magnetic` in '`nomagnetic.f90`', add the following (newer versions of the code may not require this any more):

```
!
!  write column, i_jbm, where our variable jbm is stored
!  idl needs this even if everything is zero
!
[...]
write(3,*) 'i_jbm=',idiag_jbm
[...]
```

5. and don't forget to add your new variable to '`print.in`':

```
jbm(f10.5)
```

If, instead of a mean value, you want a new maximum quantity, you need to replace `sum_mn_name()` by `max_mn_name()`.

Sect. 5.8.1 describes how to output horizontal averages of the magnetic and velocity fields. New such averages can be added to the code by using the existing averaging procedures `calc_bmz()` or `calc_jmz()` as examples.

9.3 The f-array

The 'f' array is the largest array in the PENCIL CODE and its primary role is to store the current state of the timestepped PDE variables. The f-array and its slightly smaller counter part (the df-array; see below) are the only full size 3D arrays in the code. The f-array is of type real but PDEs for a complex variable may be solved by using two slots in the f-array. The actual size of the f-array is $\mathrm{mx} \times \mathrm{my} \times \mathrm{mz} \times \mathrm{mfarray}$. Here, $\mathrm{mfarray} = \mathrm{mvar} + \mathrm{maux} + \mathrm{mglobal} + \mathrm{mscratch}$ where mvar refers to the number of real PDE variables.

As an example, we describe here how to put the time-integrated velocity, *uut*, into the f-array (see 'hydro.f90'). If this is to be invoked, there must be the following call somewhere in the code:

```
call farray_register_auxiliary('uut',iuut,vector=3)
```

Here, *iuut* is the index of the variable *uut* in the f-array. Of course, this requires that *maux* is increased by 3, but in order to do this for a particular run only one must write a corresponding entry in the 'cparam.local' file,

```
!                       -*-f90-*-    (for Emacs)
!  cparam.local
!
!** AUTOMATIC CPARAM.INC GENERATION ************************************
! Declare (for generation of cparam.inc) the number of f array
! variables and auxiliary variables added by this module
!
! MAUX CONTRIBUTION 3
!
!**************************************************************************
!  Local settings concerning grid size and number of CPUs.
!  This file is included by cparam.f90
!
integer, parameter :: ncpus=1,nprocy=1,nprocz=ncpus/nprocy,nprocx=1
integer, parameter :: nxgrid=16,nygrid=nxgrid,nzgrid=nxgrid
```

This way such a change does not affect the memory usage for other applications where this addition to 'cparam.local' is not made. In order to output this part of the f-array, one must write `lwrite_aux=T` in the `init_pars` of 'start.in'. (Technically, `lwrite_aux=T` can also be invoked in `run_pars` of 'run.in', but this does not work at the moment.)

9.4 The df-array

The 'df' array is the second largest chunk of data in the PENCIL CODE. By using a 2N storage scheme (see H.4) after Williamson [27] the code only needs one more storage area for each timestepped variable on top of the current state stored in the f-array. As such, and in contrast to the f-array, the df-array is of size $\mathrm{mx} \times \mathrm{my} \times \mathrm{mz} \times \mathrm{mvar}$. Like the df-array it is of type real. In fact the ghost zones of df are not required or calculated but having f- and df-arrays of the same size make the coding more transparent. For mx, my and mz large the wasted storage becomes negligible.

9.5 The fp-array

Similar to the 'f' array the code also has a 'fp' array which contains current states of all the particles. Like the f-array the fp-array also has a time derivative part, the dfp-array. The dimension of the fp-array is $mpar_{l}ocal \times mpvar$ where $mpar_{l}ocal$ is the number of particles in the local processor (for serial runs this is the total number of particles) and $mpvar$ depends on the problem at hand. For example if we are solving for only tracer particles then $mpvar = 3$, for dust particles $mpvar = 6$ The sequence in which the slots in the fp-array are filled up depends on the sequence in which different particle modules are called from the `particles_main.f90`. The following are the relevant lines from `particles_main.f90`.

```
!***********************************************************************
    subroutine particles_register_modules()
!
!  Register particle modules.
!
!  07-jan-05/anders: coded
!
      call register_particles             ()
      call register_particles_radius      ()
      call register_particles_spin        ()
      call register_particles_number      ()
      call register_particles_mass        ()
      call register_particles_selfgrav    ()
      call register_particles_nbody       ()
      call register_particles_viscosity   ()
      call register_pars_diagnos_state    ()
!
    endsubroutine particles_register_modules
!***********************************************************************
```

The subroutine `resister_particles` can mean either the tracer particles or dust particles. For the former the first three slots of the fp-array are the three spatial coordinates. For the latter the first six slots of the fp-array are the three spatial coordinates followed by the three velocity components. The seventh slot (or the fourth if we are use tracer particles) is the radius of the particle which can also change as a function of time as particles collide and fuse together to form bigger particles.

9.6 The pencil case

Variables that are derived from the basic physical variables of the code are stored in one-dimensional *pencils* of length *nx*. All the pencils that are defined for a given set of physics modules are in turn bundled up in a Fortran structure called `p` (or, more illustrative, the *pencil case*). Access to individual pencils happens through the variable `p%name`, where `name` is the name of a pencil, e.g. `rho` that is a derived variable of the logarithmic density `lnrho`.

The pencils provided by a given physics module are declared in the header of the file, e.g. in the Density module:

```
! PENCILS PROVIDED lnrho; rho; rho1; glnrho(3); grho(3); uglnrho; ugrho
```

Notice that the pencil names are separated with a semi-colon and that vector pencils are declared with "(3)" after the name. Before compiling the code, the script 'mkcparam' collects the names of all pencils that are provided by the chosen physics modules. It then defines the structure p with slots for every single of these pencils. The definition of the pencil case p is written in the include file 'cparam_pencils.inc'. When the code is run, the actual pencils that are needed for the run are chosen based on the input parameters. This is done in the subroutines pencil_criteria_modulename that are present in each physics module. They are all called once before entering the time loop. In the pencil_criteria subroutines the logical arrays lpenc_requested, lpenc_diagnos, lpenc_diagnos2d, and lpenc_video are set according to the pencils that are needed for the given run. Some pencils depend on each other, e.g. uglnrho depends on uu and glnrho. Such interdependencies are sorted out in the subroutines pencil_interdep_modulename that are called after pencil_criteria_modulename.

In each time-step the values of the pencil logicals lpenc_requested, lpenc_diagnos, lpenc_diagnos2d, and lpenc_video are combined to one single pencil array lpencil which is different from time-step to time-step depending on e.g. whether diagnostics or video output are done in that time-step. The pencils are then calculated in the subroutines calc_pencils_modulename. This is done before calculating the time evolution of the physical variables, as this depends very often on derived variables in pencils.

The centralized pencil calculation scheme is a guarantee that

- All pencils are only calculated once
- Pencils are always calculated by the proper physics module

Since the PENCIL CODE is a multipurpose code that has many different physics modules, it can lead to big problems if a module tries to calculate a derived variable that actually belongs to another module, because different input parameters can influence how the derived variables are calculated. One example is that the Density module can consider both logarithmic and non-logarithmic density, so if the Magnetic module calculates

```
rho = exp(f(l1:l2,m,n,ilnrho)
```

it is wrong if the Density module works with non-logarithmic density! The proper way for the Magnetic module to get to know the density is to request the pencil rho in pencil_criteria_magnetic.

9.6.1 Pencil check

To check that the correct pencils have been requested for a given run, one can run a *pencil consistency check* in the beginning of a run by setting the logical lpencil_check in &run_pars. The check is meant to see if

- All needed pencils have been requested
- All requested pencils are needed

The consistency check first calculates the value of df with all the requested pencils. Then the pencil requests are flipped one at a time – requested to not requested, not requested to requested. The following combination of events can occur:

- not requested → requested, `df` not changed
  The pencil is not requested and is not needed.
- not requested → requested, `df` changed
  The pencil is not requested, but is needed. The code stops.
- requested → not requested, `df` not changed
  The pencil is requested, but is not needed. The code gives a warning.
- requested → not requested, `df` changed
  The pencil is requested and is needed.

9.6.2 Adding new pencils

Adding a new pencil to the pencil case is trivial but requires a few steps.

- Declare the name of the pencil in the header of the proper physics module. Pencils names must appear come in a ";" separated list, with dimensions in parenthesis after the name [(3) for vector, (3,3) for matrix, etc.].
- Set interdependency of the new pencil (i.e. what other pencils does it depend on) in the subroutine `pencil_interdep_modulename`
- Make rule for calculating the pencil in `calc_pencils_modulename`
- Request the new pencil based on the input parameters in any relevant physics module

Remember that the centralized pencilation scheme is partially there to force the users of the code to think in general terms when implementing new physics. Any derived variable can be useful for a number of different physics problems, and it is important that a pencil is accessible in a transparent way to all modules.

9.7 Adding new physics: the Special module

If you want to add new physics to the code, you will in many cases want to add a new Special module. Doing so is relatively straight forward and there is even a special directory for such additions.

To create your own special module, copy '`nospecial.f90`' from the src/ directory to a new name in the src/special/ directory. It is currently only possible to have one special modules at a time and so several new bits of physics are often put in to one special module. For this reasons a name should be chosen that relates to the problem to be solved rather than the specific physics being implemented.

The first thing to do in your new module is to change the lspecial=.false. header to say lspecial=.true.

The file is heavily commented though all such comments can be removed as you go. You may implement any of the subroutines/function that exist in *nospecial.f90* and those routines must have the names and parameters as in *nospecial.f90*. You do not however need to implement all routines, and you may either leave the dummy routines

copied from *nospecial.f90* or delete them all together (provided the "include 'special_dummy.inc'" is kept intact at the end of the file. Beyond that, and data and subroutines can be added to a special module as required, though only for use within that module.

There are routines in the special interface to allow you to add new equations, modify the existing equation, add diagnostics, add slices, and many more things. If you feel there is something missing extra hooks can easily be added - please contact the PENCIL CODE team for assistance.

You are encouraged to submit/commit your special modules to the Pencil Code source. When you have added new stuff to the code, don't forget to mention this in the 'pencil-code/doc/manual.tex' file.

9.8 Adding switchable modules

In some cases where a piece of physics is thought to be more fundamental, useful in many situations or simply more flexibility is required it may be necessary to add a new module *newphysics* together with the corresponding *nonewphysics* module. The special modules follow the same structure as the rest of the switchable modules and so using a special module to prototype new ideas can make writing a new switchable module much easier.

For an example of module involving a new variable (and PDE), the *pscalar* module is a good prototype. The grep command

```
unix>  grep -i pscalar src/*
```

gives you a good overview of which files you need to edit or add.

9.9 Adding your initial conditions: the InitialCondition module

Although the code has many initial conditions implemented, we now *discourage* such practice. We aim to eventually removed most of them. The recommended course of action is to make use of the InitialCondition module.

InitialCondition works pretty much like the Special module. To implement your own custom initial conditions, create a 'initial_condition' directory in your run directory, and copy the file 'noinitialcondition.f90' from the src/ to it, with a new, descriptive, name.

The first thing to do in your new module is to change the linitialcondition=.false. header to say linitialcondition=.true.

This file has hooks to implement a custom initial condition to most variables. After implementing your initial condition, add the line INITIAL_CONDITION=initial_condition/myinitialcondition.f90 to your 'src/Makefile.local' file. Here, "myinitialcondition" is the name you gave to your initial condition file. Add also "initial_condition_pars" to the start.in file, just below "init_pars". This is a namelist, which you can use to add whichever quantity your initial condition needs defined, or passed. You must also uncomment the relevant lines in the subroutines of namelist that read and write the namelist since, for compiling reasons, these subroutines in 'noinitialcondition.f90' are dummies. The lines are easily identifiable in the code.

Check e.g. the samples '2d-tests/baroclinic', '2d-tests/spherical_viscous_ring', or 'interlocked-fluxrings', for examples of how the module is used.

10 Testing the code

To maintain reproducibility despite sometimes quite rapid development, the PENCIL CODE is tested nightly on various architectures. The front end for testing are the scripts `pc_auto-test` and (possibly) `pencil-test`.

To see which samples would be tested, run

```
unix>  pc_auto-test -l
```

, to actually run the tests, use

```
unix>  pc_auto-test
```

or

```
unix>  pc_auto-test --clean
```

. The latter compiles every test sample from scratch and currently (September 2009) takes about 2 hours on a mid-end Linux PC.

The `pencil-test` script is useful for cron jobs and allows the actual test to run on a remote computer. See Sect. 10.1 below.

For a complete list of options, run `pc_auto-test --help` and/or `pencil-test --help`.

10.1 How to set up periodic tests

To set up a nightly test of the PENCIL CODE, carry out the following steps.

1. Identify a host for running the actual tests (the *work host*) and one to initiate the tests and collect the results (the *scheduling host*). On the scheduling host, you should be able to

   (a) run cron jobs,

   (b) ssh to the work host without password,

   (c) publish HTML files (optional, but recommended),

   (d) send e-mail (optional, but recommended).

   Work host and scheduling host can be the same (in this case, use `pencil-test`'s '`-l`' option, see below), but often they will be two different computers.

2. [Recommended, but optional:] On the work host, check out a separate copy of the PENCIL CODE to reduce the risk that you start coding in the auto-test tree. In the following, we will assume that you checked out the code as '`~/pencil-auto-test`'.

3. On the work host, make sure that the code finds the correct configuration file for the tests you want to carry out. [Elaborate on that: `PENCIL_HOME/local_config` and '`-f`' option; give explicit example]

   Remember that you can set up a custom host ID file for your auto-test tree under '`${PENCIL_HOME}/config-local/hosts/`'.

4. On the scheduling host, use `crontab -e` to set up a *cron* job similar to the following:

```
30 02 * * *  $HOME/pencil-auto-test/bin/pencil-test \
    -D $HOME/pencil-auto-test \
    --use-pc_auto-test \
    -N15 -Uc -rs \
    -T $HOME/public_html/pencil-code/tests/timings.txt \
    -t 15m
    -m <email1@inter.net,email2@inter.net,...> \
    <work-host.inter.net> \
    -H > $HOME/public_html/pencil-code/tests/nightly-tests.html
```

Note 1: This has to be one long line. The backslash characters are written only for formatting purposes for this manual *you cannot use them in a crontab file*.

Note 2: You will have to adapt some parameters listed here and may want to modify a few more:

'`-D <dir>`**':** Sets the directory (on the work host) to run in.

'`-T <file>`**':** If this option is given, append a timing statistics line for each test to the given *file*.

'`--use-pc`**':** You want this option (and at some point, it will be the default).

'`-t 15m`**':** Limit the time for '`start.x`' and '`run.x`' to 15 minutes.

'`-N 15`**':** Run the tests at nice level 15 (may not have an effect for MPI tests).

'`-Uc`**':** Do `svn update` and `pc_build --cleanall` before compiling.

'`-m <email-list>`**':** If this option is given, send e-mails to everybody in the (comma-separated) list of e-mail addresses if any test fails. As soon as this option is set, the maintainers (as specified in the '`README`' file) of failed tests will also receive an e-mail.

'`work-host.inter.net|-l`**':** Replace this with the remote host that is to run the tests. If you want to run locally, write `-l` instead.

'`-H`**':** Output HTML.

'`> $HOME/public_html/pencil-code/tests/nightly-tests.html`**':** Write output to the given file.

If you want to run fewer or more tests, you can use the '`-Wa,--max-level`' option:

```
-Wa,--max-level=3
```

will run all tests up to (and including) level 3. The default corresponds to '`-Wa,--max-level=2`'.

For a complete listing of `pencil-test` options, run

```
unix>  pencil-test --help
```

11 Useful internals

11.1 Global variables

The following variables are defined in 'cdata.f90' and are available in any routine that uses the module *Cdata*.

Variable	*Meaning*
	real
t	simulated time t.
	integer
n[xyz]grid	global number of grid points (excluding ghost cells) in x, y and z direction.
nx, *ny*, *nz*	number of grid points (excluding ghost cells) as seen by the current processor, i. e. ny=nygrid/nprocy, etc.
mx, *my*, *mz*	number of grid points seen by the current processor, but *including ghost cells*. Thus, the total box for the *ivar*th variable (on the given processor) is given by f(1:mx,1:my,1:mz,ivar).
l1, *l2*	smallest and largest x-index for the physical domain (i. e. excluding ghost cells) on the given processor.
m1, *m2*	smallest and largest y-index for physical domain.
n1, *n2*	smallest and largest z-index for physical domain, i. e. the physical part of the *ivar*th variable is given by f(l1:l2,m1:m2,n1:n2,ivar)
m, *n*	pencil indexing variables: During each time-substep the box is traversed in x-pencils of length *mx* such that the current pencil of the *ivar*th variable is f(l1:l2,m,n,ivar).
	logical
lroot	true only for MPI root processor.
lfirst	true only during first time-substep of each time step.
headt	true only for very first full time step (comprising 3 substeps for the 3rd-order Runge–Kutta scheme) on root processor.
headtt	= (lfirst .and. lroot): true only during very first time-substep on root processor.
lfirstpoint	true only when the very first pencil for a given time-substep is processed, i. e. for the first set of (m, n), which is probably $(3, 3)$.
lout	true when diagnostic output is about to be written.

11.2 Subroutines and functions

output(file,a,nv) (module *IO*): Write (in each 'proc$N$' directory) the content of the global array *a* to a file called *file*, where *a* has dimensions *mx*×*my*×*mz*×*nv*, or

$mx \times my \times mz$ if `nv=1`.

`output_pencil(file,a,nv)` (module *IO*): Same as `output()`, but for a pencil variable, i. e. an auxiliary variable that only ever exists on a pencil (e. g. the magnetic field strength *bb* in '`magnetic.f90`', or the squared sound speed *cs2* in '`entropy.f90`'). The file has the same structure as those written by `output()`, because the values of a on the different pencils are accumulated in the file. This involves a quite non-trivial access pattern to the file and has thus been coded in *C* ('`src/debug_c.c`').

`cross(a,b,c)` (module *Sub*): Calculate the cross product of two vectors a and b and store in c. The vectors must either all be of size $mx \times my \times mz \times 3$ (global arrays), or of size $nx \times 3$ (pencil arrays).

`dot(a,b,c)` (module *Sub*): Calculate the dot product of two vectors a and b and store in c. The vectors must either be of size $mx \times my \times mz \times 3$ (a and b) and $mx \times my \times mz$ (c), or of size $nx \times 3$ (a and b) and nx (c).

`dot2(a,c)` (module *Sub*): Same as `dot(a,a,c)`.

Part III

Appendix

APPENDIX Date: 2014-03-17 23:16:54 +0100 (Mon, 17 Mar 2014) , Revision: 21616

A Timings

In the following table we list the results of timings of the code on different machines. Shown is (among other quantities) the wall clock time per mesh point (excluding the ghost zones) and per full 3-stage time step, a quantity that is printed by the code at the end of a run.[16]

As these results were assembled during the development phase of the code (that hasn't really finished yet,...), you may not get the same numbers, but they should give some orientation of what to expect for your specific application on your specific hardware.

The code will output the timing (in microseconds per grid point per time-step) at the end of a run. You can also specify `walltime` in `print.in` to have the code continuously output the physical time it took to reach the time-steps where diagnostics is done. The time-dependent code speed can then be calculated by differentiating, e.g. in `IDL` with

```
IDL> pc_read_ts, obj=ts
IDL> plot, ts.it, 1/nw*deriv(ts.it,ts.walltime/1.0e-6), psym=2
```

where `nw=nx*ny*nz`.

proc	machine	$\frac{\mu s}{\text{pt step}}$	resol.	what	mem/proc	when	who
1	Nl3	19	64^3	kinematic	10 MB	20-may-02	AB
1	Nl3	30	64^3	magn/noentro	20 MB	20-may-02	AB
1	Nq1	10	64^3	magn/noentro		30-may-02	AB
1	Ukaff	9.2	64^3	magn/noentro		20-may-02	AB
1	Nl6	6.8	64^3	magn/noentro		10-mar-03	AB
1	Nl6	36.3	$64\times128\times64$	nomag/entro/dust		19-sep-03	AB
1	Nl6	42.7	$16^2\times256$	nomag/entro/rad6/ion		22-oct-03	AB
1	Nl6	37.6	$16^2\times256$	nomag/entro/rad2/ion		22-oct-03	AB
1	Nl6	19.6	$16^2\times256$	nomag/entro/ion		22-oct-03	AB
1	Nl6	8.7	$16^2\times256$	nomag/entro		22-oct-03	AB
1	Nl6n	9.8	32^3	magn/noentro/pscalar		17-mar-06	AB
1	Mhd	7.8	64^3	magn/noentro		20-may-02	AB
1	Nq4	14.4	128^3	magn/noentro		8-oct-02	AB
1	Nq5	6.7	128^3	magn/noentro		8-oct-02	AB
1	fe1	5.1	128^3	magn/noentro		9-oct-02	AB
1	Kabul	4.4	128^3	magn/noentro	130 MB	20-jun-02	WD
1	Hwwsx5	3.4	256^3	convstar	7.8 GB	29-jan-03	WD
1	Mac/g95	7.7	32^3	magn/noentro		14-jan-07	BD
1	Mac/ifc	4.5	32^3	magn/noentro		14-jan-07	BD
2	Kabul	2.5	128^3	magn/noentro	80 MB	20-jun-02	WD
2	Nq3+4	7.4	128^3	magn/noentro		8-oct-02	AB
2	Nq4+4	8.9	128^3	magn/noentro		8-oct-02	AB
2	Nq4+5	7.3	128^3	magn/noentro		8-oct-02	AB
2	Nq5+5	3.7	128^3	magn/noentro		8-oct-02	AB

[16] Note that when using '`nompicomm.f90`', the timer currently used will overflow on some machines, so you should not blindly trust the timings given by the code.

2	fe1	3.45	128^3	magn/noentro		9-oct-02	AB
2	Nq2	9.3	64^3	magn/noentro		11-sep-02	AB
2	Nq1+2	8.3	64^3	magn/noentro		11-sep-02	AB
2	Hwwsx5	1.8	256^3	convstar	7.9 GB	29-jan-03	WD
4	Nq1+2	5.4	64^3	magn/noentro		11-sep-02	AB
4	Nq1235	4.1	128^3	magn/noentro		11-sep-02	AB
4	Nq0-3	6.8	256^3	magn/noentro	294 MB	10-jun-02	AB
4	Mhd	2.76	64^3	magn/noentro		30-may-02	AB
4	fe1	3.39	32^3	magn/noentro		16-aug-02	AB
4	Rasm.	2.02	64^3	magn/noentro	2x2	8-sep-02	AB
4	Mhd	8.2	$64^2\times16$	nomag/entro		23-jul-02	AB
4	fe1	6.35	$64\times128\times64$	nomag/entro/dust		19-sep-03	AB
4	fe1	2.09	128^3	magn/noentro		9-oct-02	AB
4	fe1	1.45	128^3	magn/noentro	giga	9-oct-02	AB
4	fe1	7.55	$16^2\times512$	nomag/entro/rad2/ion	4x1	1-nov-03	AB
4	fe1	5.48	$16^2\times512$	nomag/entro/rad2/ion	1x4	1-nov-03	AB
4	Luci	1.77	64^3	magn/noentro		27-feb-07	AB
4	Lenn	0.65	64^3	nomag/noentro		13-jan-07	AB
4	Lenn	1.21	64^3	magn/noentro		7-nov-06	AB
4	Kabul	1.5	128^3	magn/noentro	47 MB	20-jun-02	WD
4	Hwwsx5	1.8	256^3	convstar	8.2 GB	29-jan-03	WD
8	Nqall	3.0	128^3	magn/noentro		8-oct-02	AB
8	fe1	3.15	64^3	magn/noentro	1x8	8-sep-02	AB
8	fe1	2.36	64^3	magn/noentro	2x4	8-sep-02	AB
8	Ukaff	1.24	64^3	magn/noentro		20-may-02	AB
8	Kabul	1.25	$64^2\times128$	nomag/entro		11-jul-02	WD
8	fe1	1.68	128^3	magn/noentro	1x8	8-sep-02	AB
8	fe1	1.50	128^3	magn/noentro	2x4	8-sep-02	AB
8	fe1	1.44	128^3	magn/noentro	4x2	8-sep-02	AB
8	Kabul	0.83	128^3	magn/noentro	28 MB	20-jun-02	WD
8	Gridur	1.46	128^3	magn/noentro		19-aug-02	NE
8	Kabul	0.87	256^3	magn/noentro	160 MB	20-jun-02	WD
8	fe1	0.99	256^3	magn/noentro	2x4	8-sep-02	AB
8	fe1	0.98	256^3	magn/noentro	4x2	8-sep-02	AB
8	cetus	0.58	64^3	magn/noentro	4x2	19-aug-07	SS
8	cetus	0.73	256^3	magn/noentro	4x2,156M	19-aug-07	SS
8	Neolith	0.82	64^3	magn/noentro	4x2	5-dec-07	AB
8	Mhd	1.46	$160^2\times40$	nomag/entro	46 MB	7-oct-02	AB
8	Hwwsx5	0.50	256^3	convstar	8.6 GB	29-jan-03	WD
8	Neolith	0.444	128^3	magn/noentro		6-dec-07	AB
8	Ferlin	0.450	64^3	1test/noentro		21-jun-09	AB
8	Ferlin	0.269	64^3	magn/noentro		2-apr-10	AB
8	Ferlin	0.245	128^3	magn/noentro		2-feb-11	AB
8	nor52	2.00	32^3	magn/noentro		2-dec-09	AB
16	fe1	1.77	64^3	convstar		9-feb-03	AB
16	copson	0.596	128^3	geodynamo/ks95		21-nov-03	DM
16	fe1	0.94	128^3	magn/noentro	4x4	8-sep-02	AB
16	fe1	0.75	128^3	magn/noentro	4x4/ifc6	9-may-03	AB
16	workq	0.88	128^3	magn/noentro	4x4/ifc6	21-aug-04	AB
16	giga	0.76	128^3	magn/noentro	4x4/ifc6	21-aug-04	AB
16	giga2	0.39	128^3	magn/noentro	4x4/ifc6	20-aug-04	AB
16	giga	0.47	128^3	chiral	4x4/ifc6	29-may-04	AB
16	giga	0.43	128^3	nomag/noentro	4x4/ifc6	28-apr-03	AB
16	Mhd	2.03	128^3	magn/noentro		26-nov-02	AB
16	Mhd	0.64	256^3	magn/noentro	60 MB	22-may-02	AB
16	fe1	0.56	256^3	magn/noentro	4x4	16-aug-02	AB
16	fe1	6.30	$128\times256\times128$	nomag/entro/dust		19-sep-03	AB
16	fe1	1.31	$128^2\times512$	nomag/entro/rad2/ion	4x4	1-nov-03	AB
16	Ukaff	0.61	128^3	magn/noentro		22-may-02	AB
16	Ukaff	0.64	256^3	magn/noentro		20-may-02	AB

16	Kabul	0.80	128^3	magn/noentro	16 MB	20-jun-02	WD
16	Kabul	0.51	256^3	magn/noentro	9 MB	20-jun-02	WD
16	Gridur	0.81	128^3	magn/noentro		19-aug-02	NE
16	Gridur	0.66	256^3	magn/noentro		19-aug-02	NE
16	Sander	0.53	256^3	magn/noentro		8-sep-02	AB
16	Luci	0.375	128^3	magn/noentro		28-oct-06	AB
16	Lenn	0.284	128^3	magn/noentro		8-nov-06	AB
16	Neolith	0.180	256^3	magn/noentro		6-dec-07	AB
16	Triolith	0.075	128^3	magn/noentro	2x2x4	1-mar-14	AB
16	Triolith	0.065	128^3	magn/noentro	1x4x4	1-mar-14	AB
16	Triolith	0.054	256^3	magn/noentro	1x4x4	1-mar-14	AB
32	giga?	0.32	256^3	magn/noentro		13-sep-03	AB
32	Ukaff	0.34	256^3	magn/noentro		20-may-02	AB
32	Ukaff	0.32	512^3	magn/noentro		20-may-02	AB
32	Hermit	0.200	$256\times512\times256$	spherical conv/magn	1x8x4	22-aug-13	PJK
32	fe1	0.168	512^3	nomag/noentro		9-oct-02	AB
32	fe1	1.26	$64^2\times256$	nomag/entro/rad/ion		7-sep-03	AB
32	Luci	0.182	256^3	magn/noentro		26-feb-07	AB
32	Lenn	0.147	256^3	nomag/entro/cool/fo	4x8	8-nov-06	AB
32	Steno	0.076	256^3	nomag/entro/cool/fo	4x8	20-jun-06	AB
32	Steno	0.081	256^3	nomag/entro/cool	4x8	20-jun-06	AB
32	Steno	0.085	256^3	nomag/entro/cool/sh	4x8	20-jun-06	AB
32	Steno	0.235	$512^2\times256$	mag/entro	4x8	9-jul-06	AB
32	Sanss	0.273	128×256^2	nomag	4x8	3-jul-07	AB
32	Neolith	0.275	128^3	testfield4		24-oct-08	AB
32	Ferlin	0.556	128^3	testscalar		7-jan-09	AB
36	Kraken	0.177	$192\times384\times64$	magn/noentro	3x6x2	12-jan-12	WL
64	fe1	0.24	256^3	magn/noentro	8x8	2-sep-02	AB
64	giga	0.11	256^3	nomag/noentro	4x16	29-apr-03	AB
64	giga	0.23	256^3	nomag/noentro/hyp	4x16	8-dec-03	AB
64	fe1	0.164	512^3	nomag/noentro/hyp	4x16	17-dec-03	AB
64	giga	0.091	512^3	nomag/noentro/hyp	4x16	17-dec-03	AB
64	giga	0.150	256^3	magn/noentro	4x16	1-jul-03	AB
64	giga	0.166	512^3	magn/noentro	64*173MB	10-jul-03	AB
64	Gridur	0.25	256^3	magn/noentro		19-aug-02	NE
64	Ukaff	0.17	512^3	magn/noentro		21-may-02	AB
64	Steno	0.075	512^3	magn/noentro	8x16	19-oct-06	AB
64	Neolith	0.0695	256^3	magn/noentro		6-dec-07	AB
64	Ferlin	8.51	150×128^2	Li mechanism	8x8	21-jun-09	AB
64	Ferlin	0.156	256^3	magn/noentro	8x8	14-jun-09	AB
64	Akka	0.038	$256^2\times512$	magn/noentro	8x8	27-dec-12	AB
64	Triolith	0.0146	256^3	magn/noentro	1x8x8	1-mar-14	AB
64	Triolith	0.0164	256^3	magn/noentro	2x4x8	1-mar-14	AB
64	Hermit	0.101	$256\times512\times256$	spherical conv/magn	1x8x8	22-aug-13	PJK
64	Sisu	0.00205	$256\times512\times256$	spherical conv/magn	1x8x8	22-aug-13	PJK
72	Kraken	0.093	$192\times384\times64$	magn/noentro	3x12x2	12-jan-12	WL
72	Kraken	0.151	$96\times192\times16$	magn/noentro	6x12	17-jan-12	WL
72	Kraken	0.091	$192\times384\times32$	magn/noentro	6x12	17-jan-12	WL
72	Kraken	0.071	$384\times768\times64$	magn/noentro	6x12	17-jan-12	WL
128	fe1	0.44	256^3	nomag/entro/rad8/ion	4x32	10-mar-04	TH
128	fe1	2.8	512^3	magn/noentro	16x8	5-sep-02	AB
128	fe1	0.51	512^3	magn/noentro	8x16	5-sep-02	AB
128	fe1	0.27	512^3	magn/noentro	4x32	5-sep-02	AB
128	fe1	0.108	512^3	magn/noentro	4x32/ifc6	5-jan-02	AB
64+64	giga2	0.0600	512^3	magn/noentro	4x32/ifc6	21-aug-04	AB
128l	giga2	0.0605	512^3	magn/noentro	4x32/ifc6	21-aug-04	AB
128	fe1	0.35	512^3	magn/noentro	2x64	9-sep-02	AB
128	fe1	0.094	786^3	magn/noentro	4x32/ifc6	9-sep-02	AB
128	Hermit	0.0532	$256\times512\times256$	spherical conv/magn	1x16x8	22-aug-13	PJK
128	Hermit	0.0493	$256\times512\times256$	spherical conv/magn	2x8x8	22-aug-13	PJK

128	Sisu	0.00108	256×512×256	spherical conv/magn	1x16x8	22-aug-13	PJK
144	Kraken	0.080	96×192×32	magn/noentro	6x12x2	13-jan-12	WL
144	Kraken	0.058	192×384×64	magn/noentro	6x12x2	17-jan-12	WL
144	Kraken	0.044	384×768×128	magn/noentro	6x12x2	18-jan-12	WL
256	Hermit	0.0328	512×1024×512	spherical conv/magn	1x16x16	22-aug-13	PJK
256	Hermit	0.0285	256×512×256	spherical conv/magn	1x16x16	22-aug-13	PJK
256	giga2	0.028	1024^3	magn/noentro	4x64/ifc6	20-aug-04	AB
256	Hermit	0.0262	256×512×256	spherical conv/magn	2x16x8	22-aug-13	PJK
256	Hermit	0.0254	512×1024×512	spherical conv/magn	2x16x8	22-aug-13	PJK
256	Hermit	0.0226	512×1024×512	spherical conv/magn	4x8x8	22-aug-13	PJK
256	Akka	0.0113	512^3	magn/noentro	16x16	12-jun-11	AB
256	Sisu	0.00618	256×512×256	spherical conv/magn	1x16x16	22-aug-13	PJK
256	Sisu	0.00500	512×1024×512	spherical conv/magn	1x16x16	22-aug-13	PJK
256	Triolith	0.030	$256^2 \times 512$	magn/rad	1x16x16	17-mar-14	AB
256	Triolith	0.0049	256^3	magn/noentro	1x16x16	1-mar-14	AB
288	Gardar	0.042	576^2×288	magn/rad	1x18x16	17-mar-14	AB
288	Kraken	0.0432	192×384×64	magn/noentro	6x12x4	12-jan-12	WL
288	Kraken	0.0447	96×192×64	magn/noentro	6x12x4	13-jan-12	WL
288	Kraken	0.0201	384×768×256	magn/noentro	6x12x4	18-jan-12	WL
512	Hermit	0.01717	512×1024×512	spherical conv/magn	1x32x16	22-aug-13	PJK
512	Hermit	0.0166	256×512×256	spherical conv/magn	1x32x16	22-aug-13	PJK
512	Hermit	0.0142	256×512×256	spherical conv/magn	2x16x16	22-aug-13	PJK
512	Hermit	0.01340	512×1024×512	spherical conv/magn	2x16x16	22-aug-13	PJK
512	Hermit	0.01189	512×1024×512	spherical conv/magn	8x8x8	22-aug-13	PJK
512	Hermit	0.01165	512×1024×512	spherical conv/magn	4x16x8	22-aug-13	PJK
512	Akka	0.0081	512^3	magn/noentro	16x32	10-sep-11	AB
512	Neolith	0.0073	256^3	magn/noentro		20-nov-09	AB
512	Gardar	0.0035	512^3	magn/noentro		14-jan-13	AB
512	Lindgren	0.0040	512^2×1024	magn/noentro	16x32	8-jul-12	AB
512	Sisu	0.00446	256×512×256	spherical conv/magn	4x16x8	22-aug-13	PJK
512	Sisu	0.00435	1024×2048×1024	spherical conv/magn		22-aug-13	PJK
512	Sisu	0.00268	512×1024×512	spherical conv/magn	1x32x16	22-aug-13	PJK
576	Kraken	0.0257	192×384×64	magn/noentro	6x24x4	12-jan-12	WL
576	Kraken	0.0317	192^2×64	magn/noentro	12^2x4	13-jan-12	WL
576	Kraken	0.0116	768^2×256	magn/noentro	12^2x4	18-jan-12	WL
1024	Hermit	0.00943	512×1024×512	spherical conv/magn	1x32x32	22-aug-13	PJK
1024	Hermit	0.00707	512×1024×512	spherical conv/magn	2x32x16	22-aug-13	PJK
1024	Hermit	0.00698	1024×2048×1024	spherical conv/magn	4x16x16	22-aug-13	PJK
1024	Hermit	0.00630	512×1024×512	spherical conv/magn	4x16x16	22-aug-13	PJK
1024	Triolith	0.00236	256^3	magn/noentro	4x16x16	1-mar-14	AB
1024	Triolith	0.00126	512^3	magn/noentro	2x16x32	1-mar-14	AB
1024	Triolith	0.00129	512^3	magn/noentro	4x16x16	1-mar-14	AB
1024	Sisu	0.00225	1024×2048×1024	spherical conv/magn		22-aug-13	PJK
1024	Sisu	0.00148	512×1024×512	spherical conv/magn	2x32x16	22-aug-13	PJK
1152	Kraken	0.0212	192×384×64	magn/noentro	12x24x4	13-jan-12	WL
1152	Kraken	0.00856	384×768×128	magn/noentro	12x24x4	17-jan-12	WL
1152	Kraken	0.00549	768×1536×256	magn/noentro	12x24x4	17-jan-12	WL
1152	Lindgren	0.016	512^2×512	magn/rad	1x36x32	17-mar-14	AB
1536	Lindgren	0.00171	512^2×384	magn/noentro	2x32x24	15-jul-13	AB
2048	Hermit	0.00451	1024×2048×1024	spherical conv/magn	2x32x32	22-aug-13	PJK
2048	Hermit	0.00380	512×1024×512	spherical conv/magn	8x16x16	22-aug-13	PJK
2048	Hermit	0.00355	512×1024×512	spherical conv/magn	4x32x16	22-aug-13	PJK
2048	Hermit	0.00350	1024×2048×1024	spherical conv/magn	4x32x16	22-aug-13	PJK
2048	Lindgren	0.00129	512^2×1024	magn/noentro	32x64	20-apr-13	AB
2048	Lindgren	0.00129	1024^2×2048	magn/noentro	32x64	31-jul-12	AB
2048	Triolith	9.3×10^{-4}	512^3	magn/noentro	4x16x32	1-mar-14	AB
2048	Sisu	0.00120	1024×2048×1024	spherical conv/magn		22-aug-13	PJK
2048	Sisu	9.2×10^{-4}	512×1024×512	spherical conv/magn	4x32x16	22-aug-13	PJK
2304	Triolith	1.07×10^{-3}	576^3	magn/noentro	4x18x32	1-mar-14	AB
2304	Kraken	0.02267	192×384×64	magn/noentro	12x24x8	13-jan-12	WL

2304	Kraken	0.01233	192×768×64	magn/noentro	12x48x4	13-jan-12	WL
2304	Kraken	0.00300	768×3072×256	magn/noentro	12x48x4	18-jan-12	WL
4096	Hermit	0.00193	1024×2048×1024	spherical conv/magn	4x32x32	22-aug-13	PJK
4096	Triolith	3.6×10^{-4}	1024^3	magn/noentro	4x32x32	1-mar-14	AB
4096	Triolith	3.8×10^{-4}	1024^3	magn/noentro	8x16x32	1-mar-14	AB
4096	Triolith	4.2×10^{-4}	1024^3	magn/noentro	4x16x64	1-mar-14	AB
4096	Lindgren	4.6×10^{-4}	2048^3	magn/noentro	4x16x64	26-mar-13	AB
4096	Sisu	6.7×10^{-4}	1024×2048×1024	spherical conv/magn		22-aug-13	PJK
4608	Triolith	7.4×10^{-4}	576^3	magn/noentro	8x18x32	1-mar-14	AB
4608	Triolith	2.7×10^{-4}	1152^3	magn/noentro	4x32x36	1-mar-14	AB
4608	Triolith	3.0×10^{-4}	1152^3	magn/noentro	4x36x32	1-mar-14	AB
4608	Triolith	3.7×10^{-4}	1152^3	magn/noentro	4x18x64	1-mar-14	AB
4608	Triolith	2.36×10^{-4}	2304^3	magn/noentro	2x32x72	1-mar-14	AB
4608	Kraken	0.00764	192×768×128	magn/noentro	12x48x8	13-jan-12	WL
4608	Kraken	0.00144	768×3072×512	magn/noentro	12x48x8	18-jan-12	WL
6144	Lindgren	4.2×10^{-4}	$1024^3 \times 1536$	magn/noentro	4x16x64	21-oct-13	AB
8192	Hermit	0.00101	1024×2048×1024	spherical conv/magn	8x32x32	22-aug-13	PJK
8192	Sisu	4.1×10^{-4}	1024×2048×1024	spherical conv/magn		22-aug-13	PJK
8192	Triolith	1.48×10^{-4}	2048^3	magn/noentro	4x32x64	1-mar-14	AB
9216	Kraken	0.00485	192×768×256	magn/noentro	24x48x8	13-jan-12	WL
9216	Kraken	0.00158	768×1536×256	magn/noentro	24x48x8	17-jan-12	WL
9216	Kraken	8.0×10^{-4}	1536×3072×512	magn/noentro	24x48x8	18-jan-12	WL
9216	Lindgren	2.36×10^{-4}	2304^3	magn/noentro	4x48x48	15-feb-14	AB
9216	Triolith	1.04×10^{-3}	576^3	magn/noentro	16x18x32	1-mar-14	AB
9216	Triolith	1.28×10^{-4}	2304^3	magn/noentro	4x36x64	1-mar-14	AB
9216	Triolith	1.30×10^{-4}	2304^3	magn/noentro	4x32x72	1-mar-14	AB
16384	Hermit	6.4×10^{-4}	1024×2048×1024	spherical conv/magn	16x32x32	22-aug-13	PJK
18432	Kraken	0.00316	384×768×256	magn/noentro	24x48x16	13-jan-12	WL
18432	Kraken	8.8×10^{-4}	768×1536×512	magn/noentro	24x48x16	17-jan-12	WL
18432	Kraken	4.0×10^{-4}	1536×3072×1024	magn/noentro	24x48x16	18-jan-12	WL
36864	Kraken	0.0020	384×768×512	magn/noentro	48^2x16	14-jan-12	WL
36864	Kraken	4.9×10^{-4}	$1536^2\times512$	magn/noentro	48^2x16	17-jan-12	WL
36864	Kraken	2.2×10^{-4}	1536×3072×2048	magn/noentro	24x48x32	18-jan-12	WL
73728	Kraken	0.00121	$768^2\times512$	magn/noentro	48^2x32	19-jan-12	WL
73728	Kraken	2.9×10^{-4}	$1536^2\times1024$	magn/noentro	48^2x32	26-jan-12	WL
73728	Kraken	1.2×10^{-4}	$3072^2\times2048$	magn/noentro	48^2x32	26-jan-12	WL

The machines we have used can be characterized as follows:

Nl3: 500 MHz Pentium III single CPU; RedHat Linux 6.2; 256 MB memory

Nq0: 931 MHz Pentium III single CPU; RedHat Linux 7.3; 0.5 GB memory

Nq[1-4]: 869 MHz Pentium III dual-CPU cluster; RedHat Linux 7.3; 0.77 GB memory per (dual) node

Nq[5-6]: 1.2 GHz Athlon dual-CPU cluster; RedHat Linux 7.3; 1 GB memory per (dual) node

Kabul: 1.9 GHz Athlon dual-CPU cluster; 1 GB memory per (dual) node; 256 kB cache per CPU; Gigabit ethernet; SuSE Linux 8.0; LAM-MPI

Cincinnatus: 1.7 GHz Pentium 4 single CPU; 1 GB memory; 256 kB cache per CPU; SuSE Linux 7.3

Horseshoe (fe1, giga, and giga2): consists of different subclusters. The old one (queue name: workq, referred to as fe1) 2.0 GHz Pentium 512 single CPU; 25x 24-port fast ethernet switches with gigabit ethernet uplink; 1 30-port gigabit ethernet

switch; 1 GB memory. The next generation has gigabit switches directly between nodes, and 2.6 GHz processors. The third generation (giga2) has 3.2 GHz processors (most of which have 1 GB, some 2 GB), is organized in 2 blocks interconnected with 2 Gb links, with 10 Gb uplinks within each block.

Ukaff: SGI Origin 3000; 400 MHz IP35 CPUs; IRIX 6.5; native MPI

Mhd: EV6 Compaq cluster with 4 CPUs per node; 4 GB memory per node (i. e. 1 GB per CPU) OSF1 4.0; native MPI

Sander and Rasmussen: Origin 3000

Steno 118 node IBM cluster with dual node AMD Opteron processors with 10 Gb infiniband network, compiled with pgf90 -fastsse -tp k8-64e (Copenhagen).

Gridur: Origin 3000

Luci: (full name Lucidor) is an HP Itanium cluster, each of the 90 nodes has two 900 MHz Itanium 2 "McKinley" processors and 6 GB of main memory. The interconnect is myrinet.

Lenn: (full name Lenngren) is a Dell Xeon cluster with 442 nodes. Each node has two 3.4GHz "Nocona" Xeon processors and 8GB of main memory. A high performance Infiniband network from Mellanox is used for MPI traffic.

Kraken: Cray Linux Environment (CLE) 3.1, with a peak performance of 1.17 PetaFLOP; the cluster has 112,896 cores, 147 TB of memory, in 9,408 nodes. Each node has two 2.6 GHz six-core AMD Opteron processors (Istanbul), 12 cores, and 16 GB of memory. Connection via Cray SeaStar2+ router.

Hermit: Cray XE6 with 7104 2.3 GHz AMD Interlagos 16 core processors (113,664 cores in total), nodes with either 1 or 2 GB of memory per core.

Sisu: Cray XC30 with 1472 2.6 GHz Intel (Xeon) Sandy Bridge 8 core (E5-2670) processors (11,776 cores in total), 2 GB of memory per core.

Table 7 shows a similar list, but for a few well-defined sample problems. The *svn* check-in patterns are displayed graphically in Fig 1.

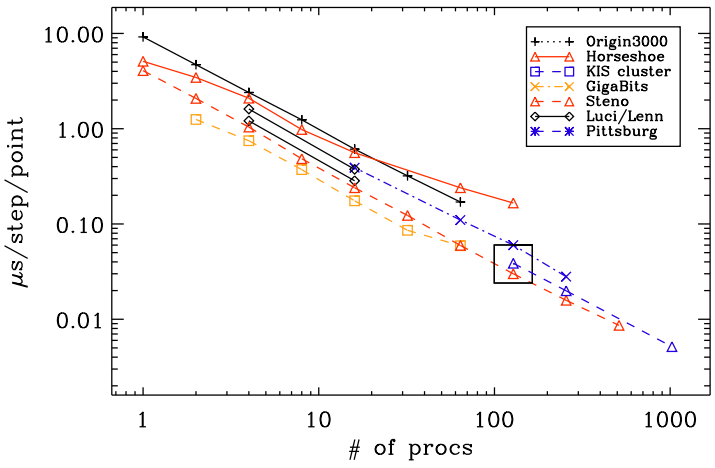

Figure 11: Scaling results on three different machines. The thin straight line denotes perfectly linear scaling.

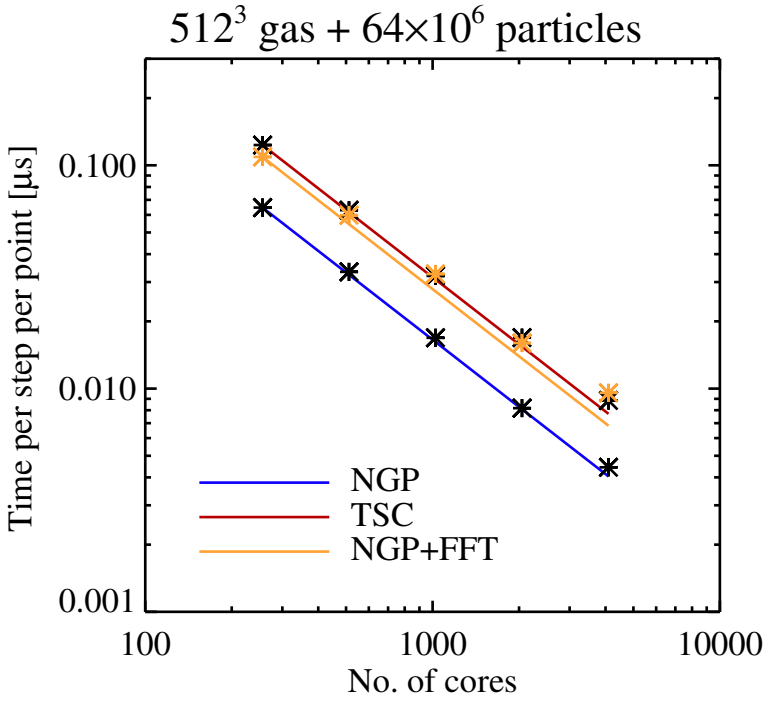

Figure 12: Scaling results of particle-mesh problem on Blue Gene/P on up to 4096 cores. The different lines denote different particle-mesh schemes (NGP=Nearest Grid Point, TSC=Triangular Shaped Cloud) and whether self-gravity is included (FFT).

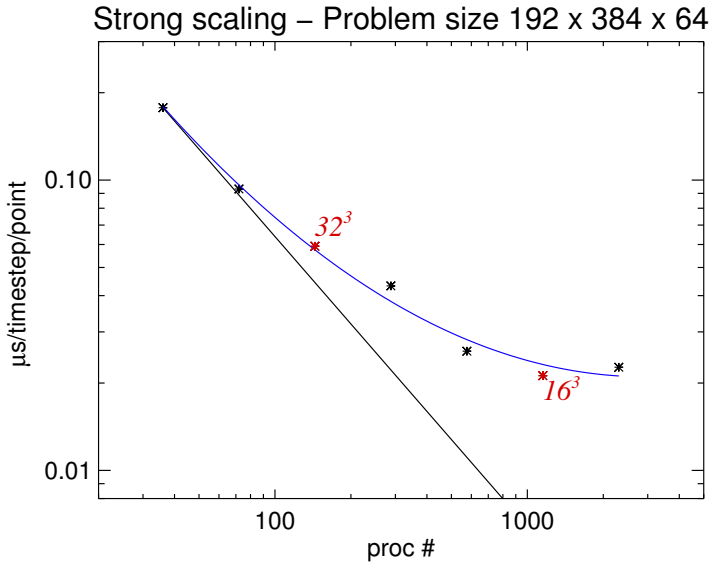

Figure 13: Scaling results on Kraken at fixed problem size, for a magnetized disk model in cylindrical coordinates. The black line shows ideal scaling from 32 cores. The blue line is the best second-order fit to the data points. A load of 16^3 mesh points per processor marks the best strong scaling.

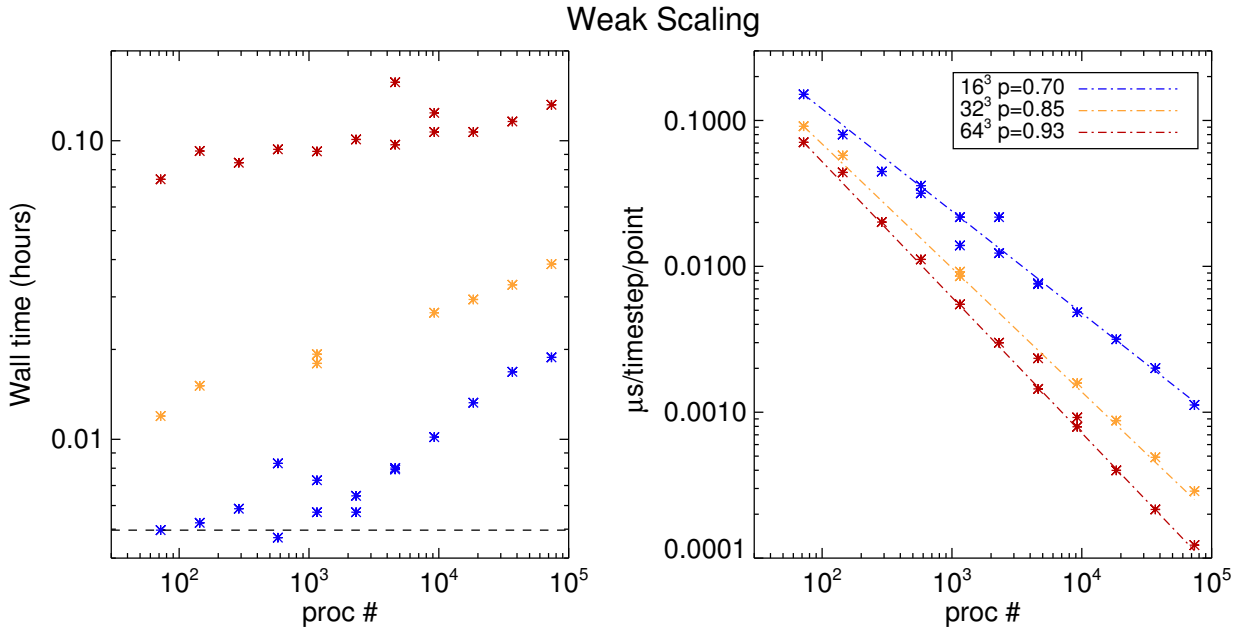

Figure 14: Scaling results on Kraken at fixed load per processor, for a magnetized disk model in cylindrical coordinates. The figure shows, after determing that 16^3 is the best load per processor for strong scaling, how far one can push with weak scaling. The scaling index is found to be 0.7 for 16^3 and 0.93 for 64^3, up to 73 728 processors.

Table 7: Like previous table, but for the versions from the 'samples' directory.

proc(s)	machine	$\frac{\mu s}{\text{pt step}}$	resol.	mem./proc	when	who
		conv-slab				
1	Mhd	6.45	32^3	4 MB	23-jul-02	wd
1	Cincinnatus	4.82	32^3	3 MB	23-jul-02	wd
1	Cincinnatus	11.6	64^3	14 MB	23-jul-02	wd
1	Cincinnatus	20.8	128^3	93 MB	23-jul-02	wd
1	Kabul	3.91	32^3		23-jul-02	wd
1	Kabul	3.88	64^3		23-jul-02	wd
1	Kabul	4.16	128^3	93 MB	23-jul-02	wd
		conv-slab-flat				
1	Kabul	3.02	$128^2\times32$	29 MB	23-jul-02	wd
2	Kabul	1.81	$128^2\times32$	18 MB	23-jul-02	wd
4	Kabul	1.03	$128^2\times32$	11 MB	23-jul-02	wd
8	Kabul	0.87	$128^2\times32$	9 MB	23-jul-02	wd

B Coding standard

The numerous elements that make up the PENCIL CODE are written in a consistent style that has evolved since it was first created. Many people have contributed their knowledge and experience with in this and the result is what we believe is and extremely readable and manageable code.

As well as improving the readability of the code, by having some naming conventions for example aids greatly in understanding what the code does.

There is a standard for all aspects of the code, be it Fortran source, shell scripts, Perl scripts, LaTeX source, Makefiles, or otherwise. Where nothing has been explicitly stated it is recommended that similar existing examples found in the code are used as a template.

B.1 File naming conventions

All files with the exception of the '`Makefile`'s are given lowercase filenames.

Fortran source files all have the '.f90' extension. Files that contain 'non-executable code' i.e. declarations that are included into other files are given the extension '.h' and those that are generated dynamically at compile time have an '.inc' extension.

Fortran source code defining a module is placed in files whose names begin with the Fortran module name in all lowercase. Where there exist multiple implementations of a specific module the filenames are extended using and with an underscore ad a brief name relating to what they do.

Text files containing parameters to be read by the code at run time are placed in files with the extension '.in'

B.2 Fortran Code

The code should remain fully compatible with the Fortran90 standard. This ensures that the code will run on all platforms. Indeed, an important aspect of PENCIL CODE philosophy is to be maximally flexible. This also means that useful non-standard extensions to the code should be hidden in and be made accessible through suitable non-default modules.

Fortran is not case-sensitive but in almost all instances we prescribe some form of capitalization for readability.

In general all Fortran code including keywords, variable names etc. are written in lowercase. Some of the coding standard has already been discussed in Sect. 9.1. Here we discuss and amplify some remaining matters.

B.2.1 Indenting and whitespace

Whitespace should be removed from the end of lines.

Blank lines are kept to a minimum, and when occurring in subroutines or functions are replaced by a single '!' in the first column.

Tab characters are not used anywhere in the code. Tab characters are not in fact allowed by the Fortran standard and compilers that accept them do so as an extension.

All lines are kept to be not more than 80 characters long. Where lines are longer they must be explicitly wrapped using the Fortran continuation character '&'. Longer lines (up to 132 characters) and additional spaces are allowed in cases where the readability of the code is enhanced, e.g. when one line is followed by a similar one with minor differences in some places.

Code in syntactic blocks such as `if`–`endif`, `do`–`enddo`, `subroutine`–`endsubroutine` etc. is always indented by precisely two spaces. The exception to this is that nested loops where only the innermost loop contains executable code should be written with the `do`–`enddo` pairs at the same level of indentation,

```
do n=n1,n2
do m=m1,m2
  [...]
enddo
enddo
```

Alternatively nested loops may be written on a single line, i.e.

```
do n=n1,n2; do m=m1,m2
  [...]
enddo; enddo
```

B.2.2 Comments

Descriptive comments are written on their own lines unless there is a strong reason to do otherwise. Comments are never indented and the '!' should appear in the first column followed by two spaces and then the text of the comment. Extremely short comments may follow at the end of a line of code, provided there is space.

Comments also must not exceed the 78 character line length and should be wrapped onto more lines as needed.

Typically comments should appear with a blank commented line above and below the wrapped text of the comment.

All subroutine/functions begin with a standard comment block describing what they do, when and by whom they were created and when and by whom any non-trivial modifications were made.

Comments should be written in sentences using the usual capitalization and punctuation of English, similar to how text is formatted in an e-mail or a journal article.

For example:

```
      some fortran code
      some more fortran code
!
!  A descriptive comment explaining what the following few lines
!  of code do.
!
      the fortran code being described
```

```
      the fortran code being described
      ...
!
!  A final detail described here.
!
      the final fortran code
      the final fortran code
      ...
```

Subroutines and functions are started with a comment block describing what they do, when and by whom they were created and when and by whom any non-trivial modifications were made. The layout of this comment block is a standard, for example:

```
!***********************************************************************
    subroutine initialize_density(f,lstarting)
!
!  Perform any post-parameter-read initialization i.e. calculate derived
!  parameters.
!
!  For compatibility with other applications, we keep the possibility
!  of giving diffrho units of dxmin*cs0, but cs0 is not well defined general.
!
!  24-nov-02/tony: coded
!   1-aug-03/axel: normally, diffrho should be given in absolute units
!
```

where dates are written in dd-mmm-yy format as shown and names appearing after the '/' are either the users cvs login name or, where such exists amongst the PENCIL CODE community, the accepted short form (≈ 4 characters) of the authors name.

B.2.3 Module names

The names of modules are written with initial letter capitalization of each word and the multiple words written consecutively without any separator.

B.2.4 Variable names

Variable are given short but meaningful names and written in all lowercase. Single character names are avoided except for commonly used loop indices and the two code data structures of the PENCIL CODE: 'f' the main state array (see 9.3) and 'p' the pencil case structure (see 9.6).

Quantities commonly represented by a particular single character in mathematics are typically given names formed by repeating the character (usually in lowercase), e.g. the velocity u becomes 'uu', specific entropy s becomes 'ss' etc.

Temperature in variable names is denoted with a capital T so as not to be confused with time as represented by a lowercase t. Note however the since Fortran is not case sensitive the variables for example 'TT' and 'tt' are the same so distinct names must be used. For this reason time is usually represented by a single t contrary to the above guideline.

The natural log of a quantity is represented by using adding 'ln' to its name, for example log of temperature would be 'lnTT'.

There are some standard prefixes used to help identify the type and nature of variables they are as follows:

- i – Denotes integer variables typically used as array indices.
- i_ – Denotes pencil case array indices.
- idiag_ – Denotes diagnostic indices.
- l – Denotes logical/boolean flags
- cdt – Denotes timestep constraint parameters.
- unit_ – Denotes conversion code/physics unit conversion parameters.

B.2.5 Emacs settings

Here are some settings from wd's '~/.emacs' file:

```
;;; ~/.f90.emacs
;;; Set up indentation and similar things for coding the {\sc Pencil Code}.
;;; Most of this can probably be set through Emacs' Customize interface
;;; as well.
;;; To automatically load this file, put the lines
;;;   (if (file-readable-p "~/.f90.emacs")
;;;       (load-file "~/.f90.emacs"))
;;; into your ~/.emacs file.

;; F90-mode indentation widths
(setq f90-beginning-ampersand nil) ; no 2nd ampersand at continuation line
(setq f90-do-indent           2)
(setq f90-if-indent           2)
(setq f90-type-indent         2)
(setq f90-continuation-indent 4)

;; Don't use any tabs for indentation (with TAB key).
;; This is actually already set for F90-mode.
(setq-default indent-tabs-mode nil)

;; Ensure Emacs uses F90-mode (and not Fortran-mode) for F90 files:
(setq auto-mode-alist
      (append
       '(
         ("\\.[fF]90$"   . f90-mode)
         ("\\.inc$"      . f90-mode)
        )
       auto-mode-alist))

;; Make M-Backspace behave in Xemacs as it does in GNU Emacs. The default
;; behavior is apparently a long-known bug the fix for which wasn't
```

```
;; propagated from fortran.el to f90.el.
;; (http://list-archive.xemacs.org/xemacs-patches/200109/msg00026.html):
(add-hook 'f90-mode-hook
          (function (lambda ()
             (define-key f90-mode-map [(meta backspace)] 'backward-kill-word)
)))
```

B.3 Other best practices

When implementing IF or SELECT blocks always write code for all cases – including the default or else case. This should be done even when that code is only a call to raise an error that the case should not have been reached. If you see a missing case anywhere then do add it. These failsafes are essential in a large multi-purpose multi-user code like the PENCIL CODE.

If a case is supposed to do nothing and it may be unclear that the coder has recognized this fact then make it explicit by adding the default case with a comment like `! Do Nothing`. The compiler will clean away any such empty blocks.

B.4 General changes to the code

It is sometimes necessary to do major changes to the code. Since this may affect many people and may even be controversial among the developers, such changes are restricted to the time of the next Pencil Code User Meeting. Such meetings are advertised on `http://www.nordita.org/software/pencil-code/` under the news section. Notes about previous such meetings can be found under `http://www.nordita.org/software/pencil-code/UserMeetings/`.

Major changes can affect those developers who have not checked in their latest changes for some time. Before doing such changes it is therefore useful to contact the people who have contributed to the latest developments on that module. If it is not functional or otherwise in bad shape, it should be moved to '`experimental`', i.e. one says `svn mv file.f90 experimental/file.f90`. However, any such directory change constitutes a major change in itself and should be performed in agreement with those involved in the development. Otherwise any file that has been changed in the mean time will end up being outside revision control, which is to be avoided at all cost.

C Some specific initial conditions

C.1 Random velocity or magnetic fields

Obtained with `inituu='gaussian-noise'` (or `initaa='gaussian-noise'`). The vector $\boldsymbol{u}$ (or $\boldsymbol{A}$) is set to normally distributed, uncorrelated random numbers in all meshpoints for all three components. The power spectrum of $\boldsymbol{u}$ ($\boldsymbol{A}$) increases then quadratically with wavenumber k (without cutoff) and the power spectrum of $\boldsymbol{\omega}$ (or $\boldsymbol{B}$) increases like k^4.

Note that a random initial condition contains significant power at the Nyquist frequency ($k_{\rm Ny}=\pi/N$, where N is the number of mesh points). In a decay calculation, because of the discretization error, such power decays slower than it ought to; see Fig. 15, where we show the evolution for a random initial velocity field for 64^3 meshpoints, $\nu=5\times10^{-2}$ (fairly large!), and `nfilter=30`.

It is clearly a good idea to filter the initial condition to prevent excess power at $k_{\rm Ny}$. On the other hand, such excess power is weak by comparison with the power at the energy carrying scale, so one does not see it in visualizations in real space. Furthermore, as seen from Fig. 15, for $k<k_{\rm Ny}/2$ the power spectra for filtered and unfiltered initial conditions is almost the same.

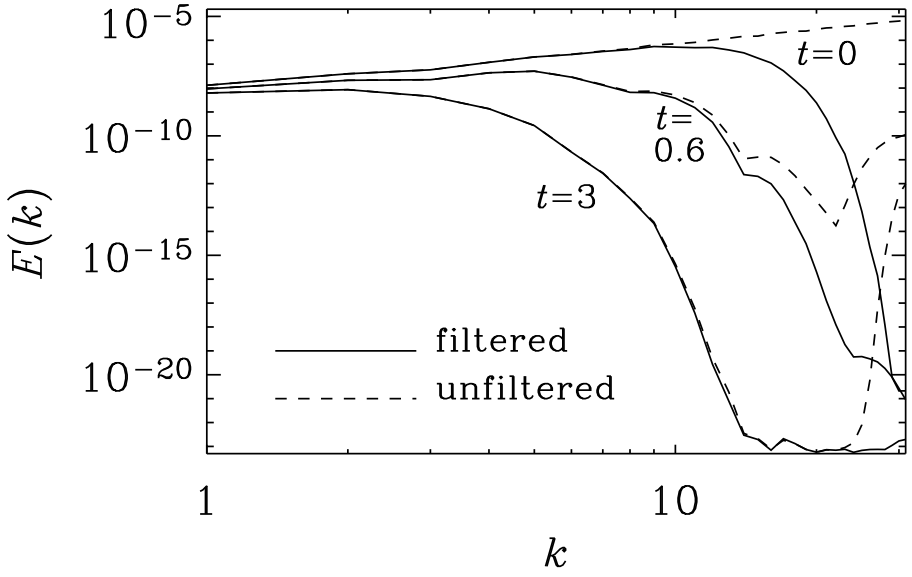

Figure 15: Velocity power spectra at three different times with and without filtering of the initial condition.

C.2 Beltrami fields

Obtained with `inituu='Beltrami-z'` or `initaa='Beltrami-z'`.

$$\boldsymbol{A}=(\cos z,\ \sin z,\ 0),\quad \text{or}\quad \boldsymbol{u}=(\cos z,\ \sin z,\ 0) \tag{98}$$

C.3 Magnetic flux rings: initaa='fluxrings'

This initial condition sets up two interlocked thin magnetic tori (i. e. thin, torus-shaped magnetic flux tubes). One torus of radius R lying in the plane $z = 0$ can be described in cylindrical coordinates, (r, φ, z), by the vector potential

$$\boldsymbol{A} = \Phi_{\mathrm{m}} \begin{pmatrix} 0 \\ 0 \\ -\theta(r-R)\delta(z) \end{pmatrix} , \tag{99}$$

resulting in a magnetic field

$$\boldsymbol{B} = \Phi_{\mathrm{m}} \begin{pmatrix} 0 \\ \delta(r-R)\delta(z) \\ 0 \end{pmatrix} . \tag{100}$$

Here Φ_{m} is the magnetic flux through the tube, $\theta(x)$ denotes the Heaviside function, and

$$\delta(x) = \theta'(x) \tag{101}$$

is Dirac's delta function.

Any smoothed versions of $\theta(x)$ and $\delta(x)$ will do, as long as the consistency condition (101) is satisfied. E. g. the pairs

$$\delta_\varepsilon(x) = \frac{1}{\sqrt{2\pi\varepsilon^2}} e^{-\frac{x^2}{2\varepsilon^2}} , \quad \theta_\varepsilon(x) = \frac{1}{2}\left(1 + \operatorname{erf}\frac{x}{\sqrt{2}\varepsilon}\right) \tag{102}$$

or

$$\delta_\varepsilon(x) = \frac{1}{2\varepsilon}\frac{1}{\cosh^2\frac{x}{\varepsilon}} , \quad \theta_\varepsilon(x) = \frac{1}{2}\left(1 + \tanh\frac{x}{\varepsilon}\right) \tag{103}$$

are quite popular. Another possibility is a constant or box-like profile with

$$\delta_\varepsilon(x) = \frac{1}{2\varepsilon}\theta(|x| - \varepsilon) , \quad \theta_\varepsilon(x) = \frac{1}{2}\left\{1 + \max[-1, \min(x/\varepsilon), 1]\right\} \tag{104}$$

Note, however, that the Gaussian profile (102) is the only one that yields a radially symmetric (with respect to the distance from the central line of the torus) magnetic field profile $B_\varphi = B_\phi(\sqrt{(r-R)^2+z^2})$ if ε is sufficiently small.

In Cartesian coordinates, the vector potential (99) takes the form

$$\boldsymbol{A} = \Phi_{\mathrm{m}} \begin{pmatrix} 0 \\ 0 \\ -\theta\left(\sqrt{x^2+y^2}-R\right)\delta(z) \end{pmatrix} . \tag{105}$$

C.4 Vertical stratification

Gravity, $\mathrm{g} = -\boldsymbol{\nabla}\Phi$, is specified in terms of a potential Φ. In slab geometry, $\Phi = \Phi(z)$, we have $\mathrm{g} = (0, 0, g_z)$ and $g_z = -\mathrm{d}\Phi/\mathrm{d}z$.

Use *grav_profile*='const' together with *gravz*$= -1$ to get

$$\Phi = (z - z_\infty)(-g_z), \quad (-g_z) > 0. \tag{106}$$

Use *grav_profile*='linear' to get

$$\Phi = \tfrac{1}{2}(z^2 - z_\infty^2)\nu_{\rm g}^2, \quad g_z = -\nu_{\rm g}^2 z \tag{107}$$

where $\nu_{\rm g}$ is the vertical epicyclic frequency. For a Keplerian accretion disc, $\nu_{\rm g} = \Omega$. For galactic discs, $\nu_{\rm g} = 0.5\Omega$ is representative of the solar neighborhood.

The value of z_∞ is determined such that $\rho = \rho_0$ and $c_{\rm s}^2 = c_{\rm s0}^2$ at $z = z_{\rm ref}$. This depends on the values of γ and the polytropic index m (see below).

C.4.1 Isothermal atmosphere

Here we want $c_{\rm s} = c_{\rm s0} =$ const. Using `initlnrho='isothermal'` means

$$\ln\frac{\rho}{\rho_0} = -\gamma\frac{\Phi}{c_{\rm s0}^2}\,. \tag{108}$$

The entropy is then initialized to

$$\frac{s}{c_p} = (\gamma{-}1)\frac{\Phi}{c_{\rm s0}^2}\,. \tag{109}$$

In order that $\rho = \rho_0$ and $c_{\rm s}^2 = c_{\rm s0}^2$ at $z = z_{\rm ref}$, we have to choose $z_\infty = z_{\rm ref}$.

C.4.2 Polytropic atmosphere

For a polytropic equation of state, $p = K\rho^\Gamma$, where generally $\Gamma \neq \gamma$, we can write

$$-\boldsymbol{\nabla}h + T\boldsymbol{\nabla}s = -\frac{1}{\rho}\boldsymbol{\nabla}p = -\boldsymbol{\nabla}\left(\frac{\Gamma K}{\Gamma{-}1}\rho^{\Gamma-1}\right) \equiv -\boldsymbol{\nabla}\tilde{h}, \tag{110}$$

where we have introduced a pseudo enthalpy $\tilde{h}$ as

$$\tilde{h} = \frac{\Gamma K}{\Gamma{-}1}\rho^{\Gamma-1} = \left[\left(1-\frac{1}{\gamma}\right)\Big/\left(1-\frac{1}{\Gamma}\right)\right] h\,. \tag{111}$$

Obviously, for $\Gamma = \gamma$, the pseudo enthalpy $\tilde{h}$ is identical to h itself. Instead of specifying Γ, one usually defines the polytropic index $m = 1/(\Gamma{-}1)$. Thus, $\Gamma = 1 + 1/m$, and

$$\tilde{h} = (m{+}1)\left(1-\frac{1}{\gamma}\right) h \tag{112}$$

This is consistent with a fixed entropy dependence, where s only depends on ρ like

$$\frac{s}{c_p} = \left(\frac{\Gamma}{\gamma} - 1\right)\ln\frac{\rho}{\rho_0}\,, \tag{113}$$

and implies that

$$\ln\frac{c_{\rm s}^2}{c_{\rm s0}^2} = (\Gamma{-}1)\ln\frac{\rho}{\rho_0}\,. \tag{114}$$

For hydrostatic equilibrium we require $\tilde{h} + \Phi = \tilde{h}_0 =$ const. For gravity potentials that vanish at infinity, we can have $\tilde{h}_0 \neq 0$, i.e. a finite pseudo enthalpy at infinity. For $g_z = -1$

or $g_z = -z$, this is not the case, so we put $\tilde{h}_0 = 0$, and therefore $\tilde{h} = -\Phi$. Using $c_{\rm s}^2 = (\gamma{-}1)h$ together with (112) we find

$$c_{\rm s}^2 = -\frac{\gamma}{m{+}1}\,\Phi. \tag{115}$$

In order that $\rho = \rho_0$ and $c_{\rm s}^2 = c_{\rm s0}^2$ at $z = z_{\rm ref}$, we have to choose (remember that g_z is normally negative!)

$$z_\infty = z_{\rm ref} + (m{+}1)\frac{c_{\rm s0}^2}{\gamma(-g_z)} \quad \text{for } \textit{grav\_profile}\text{='const'}, \tag{116}$$

and

$$z_\infty^2 = z_{\rm ref}^2 + (m{+}1)\frac{c_{\rm s0}^2}{\frac{1}{2}\gamma\nu_{\rm g}^2} \quad \text{for } \textit{grav\_profile}\text{='linear'}. \tag{117}$$

Thus, when using `initlnrho='polytropic_simple'` we calculate

$$\ln\frac{c_{\rm s}^2}{c_{\rm s0}^2} = \ln\left[-\frac{\gamma\Phi}{(m{+}1)c_{\rm s0}^2}\right] \tag{118}$$

and so the stratification is given by

$$\ln\frac{\rho}{\rho_0} = m\,\ln\frac{c_{\rm s}^2}{c_{\rm s0}^2}\,, \quad \frac{s}{c_p} = \left(\frac{\Gamma}{\gamma} - 1\right) m\,\ln\frac{c_{\rm s}^2}{c_{\rm s0}^2}\,. \tag{119}$$

C.4.3 Changing the stratification

Natural: measure length in units of $c_{\rm s0}^2/g_z$. Can increase stratification by moving $z_{\rm top}$ close to z_∞ or, better still, keeping $z_{\rm top} = 0$ and moving $z_{\rm bot} \to -\infty$. Disadvantage: in the limit of weak stratification, the box size will be very small (in nondimensional units).

Box units: measure length in units of d. Can increase stratification by increasing g_z to $g_{\rm max}$, which can be obtained by putting $z_{\rm top} = z_\infty$ in (116), so

$$g_{\rm max} = \frac{m{+}1}{\gamma}\,\frac{c_{\rm s0}^2}{z_{\rm top} - z_{\rm ref}}. \tag{120}$$

For $m = 1$, $\gamma = 5/3$, $z_{\rm top} = 1$, and $z_{\rm ref} = 0$, for example, we have $g_{\rm max} = 6/5 = 1.2$.

Gravitational box units: measure speed in units of $\sqrt{g_z d}$. The limit of vanishing stratification corresponds to $c_{\rm s0} \to \infty$. This seems optimal if we want to approach the Boussinesq case.

In Hurlburt et al. (1984), z increased downward and the atmosphere always terminated at $z = 0$. In order to reproduce their case most directly, we put $z_\infty = 0$ and consider only negative values of z. To reproduce their case with a density stratification of 1:1.5, we place the top of the model at $z = -2$ and the bottom at $z = -3$. In addition, the reference height, $z_{\rm ref}$, is chosen to be at the top of the box, i.e. $z_{\rm ref} = -2$. From Eq. (116) we have $c_{\rm s0}^2 = \gamma(-g_z)(-z_{\rm ref})/(m+1)$. Using $(-g_z) = 1$ and $m = 1$ we find $c_{\rm s0}^2 = \gamma$, so $c_{\rm s0} = 1.291$ (for $\gamma = 5/3$). Values for other combinations are listed in Table 8.

Table 8: Correspondence between density contrast, top and bottom values of z, and $c_{\rm s0}$ for $(-g_z) = 1$, $m = 1$, and $\gamma = 5/3$.

$\rho_{\rm bot}/\rho_{\rm top}$	$z_{\rm bot}$	$z_{\rm top}$	$c_{\rm s0}$
1.5	3	2	1.291
3	1.5	0.5	0.645
6	1.2	0.2	0.408
11	1.1	0.1	0.289
21	1.05	0.05	0.204

C.4.4 The Rayleigh number

In Ref. [8] the Rayleigh number is defined as

$$\mathrm{Ra} = \frac{gd^4}{\overline{\nu}\,\overline{\chi}} \left(-\frac{\mathrm{d}s/c_p}{\mathrm{d}z} \right)_{\rm hydrostat}, \tag{121}$$

where the (negative) entropy gradient was evaluated in the middle of the box for the associated hydrostatic reference solution, and $\overline{\chi} = K/(\overline{\rho} c_p)$ and either $\overline{\nu} = \nu$ (if ν was assumed constant) or $\overline{\nu} = \mu/\overline{\rho}$ (if μ was assumed constant). Note that $\overline{\rho}$ is the average mass in the box per volume, which is conserved. For a polytrope we have

$$\left(-\frac{\mathrm{d}s/c_p}{\mathrm{d}z} \right)_{\rm hydrostat} = \left[1 - (m+1) \left(1 - \frac{1}{\gamma} \right) \right] \frac{1}{z_\infty - z_{\rm m}}, \tag{122}$$

where $z_{\rm m} = (z_1 + z_2)/2$. This factor was also present in the definition of Hurlburt et al. [17], but their definition differs slightly from Eq. (121), because they normalized the density not with respect to the average value (which is constant for all times), but with respect to the value at the top of the initial hydrostatic solution. Since the Rayleigh number is proportional to ρ^2, their definition included the extra factor $[(z_\infty - z_{\rm m})/d]^2$. Therefore

$$\mathrm{Ra}_{\rm HTM} = \left(\frac{z_\infty - z_{\rm m}}{d} \right)^{2m} \left(\frac{\rho_{\rm top}}{\overline{\rho}} \right)^2 \mathrm{Ra} \tag{123}$$

In the first model of Hurlburt et al. (1984), the Rayleigh number, $\mathrm{Ra}_{\rm HTM}$, was chosen to be 310 times supercritical, and the critical Rayleigh number was around 400, so $\mathrm{Ra}_{\rm HTM} = 1.25 \times 10^5$. In their model the density contrast was 1:1.5 and $m = 1$. This turns out to correspond to $\mathrm{Ra} = 4.9 \times 10^4$, $F_{\rm bot} = 0.0025$, and $K = 0.002$.

Another model that was considered by Hurlburt & Toomre (1988) had $\mathrm{Ra}_{\rm HTM} = 10^5$, a density contrast of 11, and had a vertical imposed magnetic field (Chandrasekhar number $Q = 72$). This corresponds to $\mathrm{Ra} = 3.6 \times 10^8$, $K = 0.0011$, $F_{\rm bot} = 0.0014$.

C.4.5 Entropy boundary condition

This discussion only applies to the case of convection in a slab. A commonly used lower boundary condition is to prescribe the radiative flux at the bottom, i.e. $F_{\rm bot} = -K\mathrm{d}T/\mathrm{d}z$. Assuming that the density in the ghost zones has already been updated, we can calculate the entropy gradient from

$$F_{\rm bot} = -\frac{K}{c_p} \frac{c_{\rm s}^2}{\gamma - 1} \left((\gamma - 1) \frac{\mathrm{d}\ln\rho}{\mathrm{d}z} + \gamma \frac{\mathrm{d}s/c_p}{\mathrm{d}z} \right), \tag{124}$$

which gives

$$\frac{\mathrm{d}s/c_p}{\mathrm{d}z} = -\frac{\gamma-1}{\gamma}\left(c_p\frac{F_{\rm bot}}{Kc_{\rm s}^2}+\frac{\mathrm{d}\ln\rho}{\mathrm{d}z}\right) \tag{125}$$

for the derivative of the entropy at the bottom. This is implemented as the '`c1`' boundary condition at the bottom.

C.4.6 Temperature boundary condition at the top

In earlier papers the temperature at the top was set in terms of the quantity ξ_0, which is the ratio of the pressure scale height relative to the depth of the unstable layer. Expressed in terms of the sound speed at the top we have

$$c_{\rm s,top}^2 = \gamma\xi_0 gd. \tag{126}$$

$$c_{\rm s,bot}^2 = \left(\xi_0+\frac{1}{m+1}\right)\gamma gd. \tag{127}$$

Table 9: Correspondence between ξ_0 and $c_{\rm s,bot}^2$ in single layer polytropes.

ξ_0	$c_{\rm s,bot}^2$
10.00	17.500
0.20	1.167
0.10	1.000
0.05	0.917
0.02	0.867

C.5 Potential-field boundary condition

The '`pot`' [or currently rather the '`pwd`'] boundary condition for the magnetic vector potential implements a *potential-field boundary condition* in z for the case of an x-y-periodic box. In this section, we discuss the relevant formulas and their implementation in the PENCIL CODE.

If the top boundary is at $z=0$, the relevant potential field for $z>0$ is given by

$$\tilde{A}_x(k_x,k_y,z) = C_x(\boldsymbol{k}_{xy})\,e^{-\kappa z}\,, \tag{128}$$
$$\tilde{A}_y(k_x,k_y,z) = C_y(\boldsymbol{k}_{xy})\,e^{-\kappa z}\,, \tag{129}$$
$$\tilde{A}_x(k_x,k_y,z) = C_z(\boldsymbol{k}_{xy})\,e^{-\kappa z}\,, \tag{130}$$

where

$$\tilde{A}_i(k_x,k_y,z) \equiv \int e^{-i\boldsymbol{k}_{xy}\cdot\boldsymbol{x}}A_i(x,y,z)\,dx\,dy \tag{131}$$

is the horizontal Fourier transform with $\boldsymbol{k}_{xy}\equiv(k_x,k_y,0)$, and $k\equiv|\boldsymbol{k}_{xy}|$. Note that this implies a certain gauge and generally speaking the z dependence in Eq. (130) is completely arbitrary, but the form used here works well in terms of numerical stability.

At the very boundary, the potential field (128)–(130) implies

$$\frac{\partial\tilde{\boldsymbol{A}}}{\partial z}+\kappa\tilde{\boldsymbol{A}} = 0\,, \tag{132}$$

and, due to natural continuity requirements on the vector potential, these conditions also hold for the interior field at the boundary.

Robin boundary conditions and ghost points To implement a homogeneous Robin boundary condition, i. e. a condition of the form

$$\frac{df}{dz} + \kappa f = 0 \tag{133}$$

using ghost points, we first write it as

$$\frac{d}{dz}\left(f\, e^{\kappa z}\right) = 0 \tag{134}$$

and implement this as symmetry condition for the variable $\phi(z) \equiv f(z)\, e^{\kappa z}$:

$$\phi_{N-j} = \phi_{N+j}\ , \qquad j = 1, 2, 3 \tag{135}$$

(where z_N is the position of the top boundary and z_{N+1}, ... are the boundary points). In terms of f, this becomes

$$f_{N+j} = f_{N-j}\, e^{-\kappa(z_{N+j} - z_{N-j})}\ . \tag{136}$$

Note that although the exponential term in Eq. (136) looks very much like the exterior potential field (128)–(130), our ghost-zone values do *not* represent the exterior field – they are rather made-up values that allow us to implement a local boundary condition at $z = 0$.

C.6 Planet solution in the shearing box

In order to test the setup for accretion discs and the sliding periodic shearing sheet boundary condition, a useful initial condition is the so-called planet solution of Goodman, Narayan, & Goldreich [14].

Assume $s = 0$ (isentropy), so the equations in 2-D are

$$u_x u_{x,x} + (u_y^{(0)} + u_y) u_{x,y} = 2\Omega u_y - h_{,x} \tag{137}$$

$$u_x u_{y,x} + (u_y^{(0)} + u_y) u_{y,y} = -(2-q)\Omega u_x - h_{,y} \tag{138}$$

where $u_y^{(0)} = -q\Omega x$. Express $\mathbf{u}$ in terms of a stream function, so $\mathbf{u} = \nabla \times (\psi \hat{\mathbf{z}})$, or

$$u_x = \psi_{,y}, \quad u_y = -\psi_{,x}. \tag{139}$$

Ansatz for enthalpy

$$h = \tfrac{1}{2}\delta^2 \Omega^2 (R^2 - x^2 - \epsilon^2 y^2 - z^2/\delta^2) \tag{140}$$

$$\psi = -\tfrac{1}{2}\sigma\Omega(R^2 - x^2 - \epsilon^2 y^2) - \tfrac{1}{2} q \Omega x^2 \tag{141}$$

This implies

$$u_x = \sigma\Omega\epsilon^2 y, \quad u_y = (q - \sigma)\Omega x \tag{142}$$

and $u_{x,x} = u_{y,y} = 0$. Inserting into Eqs (137) and (138) yields

$$(-q + q - \sigma)\sigma\epsilon^2 = 2(q - \sigma) + \delta^2 \tag{143}$$

$$\sigma(q-\sigma) = -(2-q)\sigma + \delta^2 \tag{144}$$

where we have already canceled common Ω^2 factors in both equations and common ϵ^2 factors in the last equation. Simplifying both equations yields

$$-\sigma^2\epsilon^2 = 2(q-\sigma) + \delta^2 \tag{145}$$

$$-\sigma^2 = -2\sigma + \delta^2 \tag{146}$$

The second equation yields

$$\delta^2 = (2-\sigma)\sigma \tag{147}$$

and subtracting the two yields

$$\sigma^2 = 2q/(1-\epsilon^2) \tag{148}$$

Table 10: Dependence of ϵ and δ on ϵ.

ϵ	σ	δ
0.1	1.74	0.67
0.2	1.77	0.64
0.3	1.82	0.58
0.4	1.89	0.46
0.48	1.97	0.22
0.5	2	0

D Some specific boundary conditions

In this section, we formulate and discuss the implementation of some common boundary conditions in spherical and cylindrical coordinates.

D.1 Perfect-conductor boundary condition

This is a popular boundary condition for the magnetic field; it implies that

$$B_n = 0 \tag{149}$$

and

$$\boldsymbol{E}_t = 0 \tag{150}$$

on the boundary, where the subscript n denotes the normal component, and $\boldsymbol{E}_t$ denotes the tangential components of the electric field.

In Cartesian geometry, these conditions can be implemented by setting the two tangential components of the vector potential $\boldsymbol{A}$ to zero on the boundary. It is easy to see that this also works in arbitrary curvilinear coordinates.

In particular, for spherical coordinates on a radial boundary we must have

$$r\sin\theta\, B_r = \partial_\theta(\sin\theta\, A_\phi) - \partial_\phi A_\theta = 0\ . \tag{151}$$

This can be achieved by setting

$$A_\phi = A_\theta = 0 \tag{152}$$

everywhere on the boundary. Note that this does not impose any condition on the radial component of the vector potential.

Next, in spherical coordinates on a boundary with constant θ, we must have

$$B_\theta = \frac{1}{r\sin\theta}\partial_\phi A_r - \frac{1}{r}\partial_r(rA_\phi) = 0\ . \tag{153}$$

Again this can be achieved by $A_r = A_\phi = 0$.

D.2 Stress-free boundary condition

On an impenetrable, stress-free boundary, we have

$$u_n = 0\ , \tag{154}$$

and the shear stress components S_{nt} must vanish for any tangential direction t. At the radial boundary, the relevant components of the strain tensor (required to vanish at the boundary) are:

$$S_{r\theta} = \frac{1}{r}\partial_\theta u_r + r\partial_r\left(\frac{u_\theta}{r}\right)\ , \tag{155}$$

$$S_{r\phi} = \frac{1}{r\sin\theta}\partial_\phi u_r + r\partial_r\left(\frac{u_\phi}{r}\right)\ . \tag{156}$$

Both of them vanish if we require

$$u_r = 0\ , \qquad \partial_r(u_\theta/r) = 0\ , \qquad \partial_r(u_\phi/r) = 0\ . \tag{157}$$

We implement this by requiring u_r to be antisymmetric and u_θ/r and u_ϕ/r to be symmetric with respect to the boundary.

The more general condition

$$r^\alpha \partial_r (u_\theta / r^\alpha) = \partial_r u_\theta - \frac{\alpha}{r} u_\theta = 0 \tag{158}$$

(where α is a constant) can be implemented by requiring u_θ/r^α to be symmetric.

At a boundary $\theta = \text{const}$, the stress-free boundary condition will take the form

$$S_{r\theta} = \frac{1}{r}\partial_\theta u_r + r\partial_r\left(\frac{u_\theta}{r}\right) = 0 \,, \tag{159}$$

$$S_{\theta\phi} = \frac{1}{r\sin\theta}\partial_\phi u_\theta + \sin\theta\,\partial_\theta\left(\frac{u_\phi}{r\sin\theta}\right) = 0 \,. \tag{160}$$

With $u_\theta = 0$, the first condition gives $\partial_\theta u_r = 0$, i.e. we require u_r to be symmetric with respect to the boundary. The second condition requires

$$\frac{\sin\theta}{r}\partial_\theta\left(\frac{u_\phi}{\sin\theta}\right) = 0 \tag{161}$$

and is implemented by requiring $u_\phi/\sin\theta$ to be symmetric.

D.3 Normal-field-radial boundary condition

While unphysical, this boundary condition is often used as a cheap replacement for a potential-field condition for the magnetic field. It implies that the two tangential components of the magnetic field are zero at the boundary, while the normal component is left unconstrained.

At a radial boundary, this gives:

$$B_\theta = \frac{1}{r\sin\theta}\partial_\phi A_r - \frac{1}{r}\partial_r(rA_\phi) = 0 \,, \tag{162}$$

$$B_\phi = \frac{1}{r}\partial_r(rA_\theta) - \frac{1}{r}\partial_\theta A_r = 0 \,. \tag{163}$$

Which are satisfied by setting

$$A_r = 0 \,, \qquad \partial_r(rA_\theta) = 0 \,, \qquad \partial_r(rA_\phi) = 0 \,, \tag{164}$$

and these are implemented by requiring A_r to be antisymmetric, and rA_θ and rA_ϕ to be symmetric.

On a boundary $\theta = \text{const}$, we have

$$r\sin\theta\, B_r = \partial_\theta(\sin\theta\, A_\phi) - \partial_\phi A_\theta = 0 \,, \tag{165}$$

$$rB_\phi = \partial_r(rA_\theta) - \partial_\theta A_r = 0 \tag{166}$$

which can be achieved by setting

$$\partial_\theta A_r = 0 \,, \qquad A_\theta = 0 \,, \qquad \partial_\theta(\sin\theta\, A_\phi) = 0 \,. \tag{167}$$

We thus require A_r and $\sin\theta\, A_\phi$ to be symmetric, and A_θ to be antisymmetric.

E High-frequency filters

Being high order, PENCIL CODE has much reduced numerical dissipation. In order to perform inviscid simulations, high-frequency filters can be used to provide extra dissipation for modes approaching the Nyquist frequency. Usual Laplacian viscosity $\nu\nabla^2\boldsymbol{u}$ is equivalent to a multiplication by k^2 in Fourier space, where k is the wavenumber. Another tool is hyperviscosity, which replaces the k^2 dependency by a higher power-law, k^n, $n>2$. The idea behind it is to provide large dissipation only where it is needed, at the grid scale (high k), while minimizing it at the largest scales of the box (small k). In principle, one can use as high n as desired, but in practice we are limited by the order of the code. A multiplication by k^n is equivalent to an operator ∇^n in real space. As PENCIL CODE is of sixth order, three ghost cells are available in each direction, thus the sixth-order derivative is the highest we can compute. The hyperdissipation we use is therefore ∇^6, or k^6 is Fourier space. Figure 16 illustrates how such tool maximizes the inertial range of a simulation.

Simplified hyperdiffusivity has been implemented for many dynamical variables and can be found in the respective modules. A strict generalization of viscosity and resistivity to higher order is implemented in the modules '`hypervisc_strict_2nd`' and '`hyperresi_strict_2nd`'.

Hyperdiffusivity is meant purely as a numerical tool to dissipate energy at small scales and comes with no guarantee that results are convergent with regular second order dissipation. See Haugen & Brandenburg (2004) for a discussion. In fact, large-scale dynamo action is known to be seriously altered in simulations of closed systems where magnetic helicity is conserved: this results in prolonged saturation times and enhanced saturation amplitudes (Brandenburg & Sarson 2002).

E.1 Conservative hyperdissipation

It is desirable to have this high-frequency filter obeying the conservation laws. So, for density we want a mass conserving term, for velocities we want a momentum conserving term, for magnetic fields we want a term conserving magnetic flux, and for entropy we want an energy conserving term. These enter as hyperdiffusion, hyperviscosity, hyperresistivity, and hyper heat conductivity terms in the evolution equations. To ensure conservation under transport, they must take the form of the divergence of the flux $\mathcal{J}$ of the quantity ψ, so that Gauss theorem applies and we have

$$\frac{\partial\psi}{\partial t}+\boldsymbol{\nabla}\cdot\boldsymbol{\mathcal{J}}=0 \tag{168}$$

For density, the flow due to mass diffusion is usually taken as the phenomenological Fick's Law

$$\boldsymbol{\mathcal{J}}=-D\boldsymbol{\nabla}\rho \tag{169}$$

i.e., proportional to the density gradient, in the opposite direction. This leads to the usual Laplacian diffusion

$$\frac{\partial\rho}{\partial t}=D\nabla^2\rho \tag{170}$$

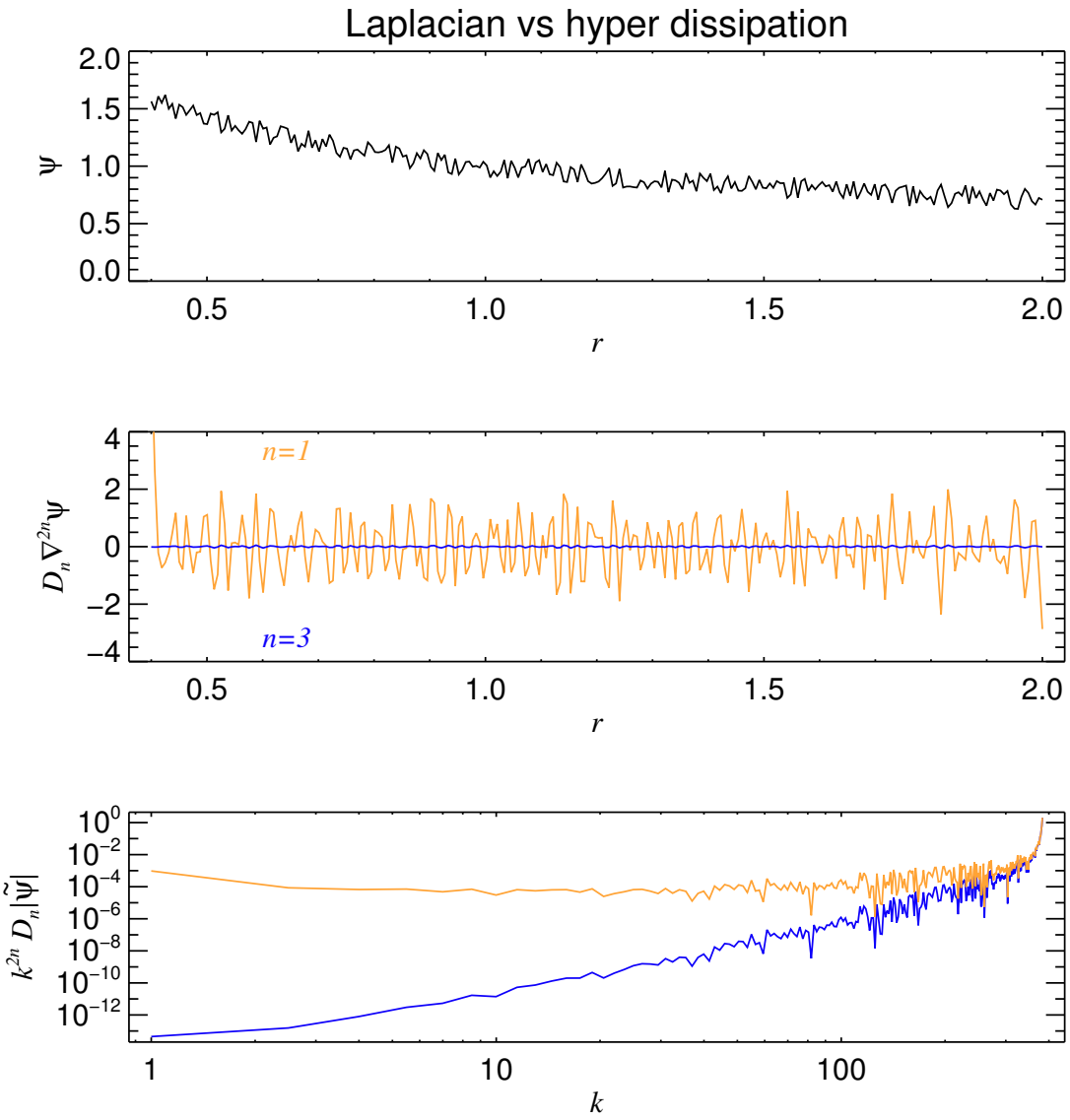

Figure 16: Dissipation acting on a scalar field ψ, for n=1 (Laplacian dissipation) and n=3 (third-order hyperdissipation). The field is initially seeded with noise (upper panel). For n=3 the large scale is not affected as much as in the n=1 case, which is seen by the larger wiggling of the latter in the middle panel. In Fourier space (lower panel) we see that near the grid scale both formulations give strong dissipation. It also illustrates that at the large scales ($k\simeq1$), the effect of n=3 is indeed negligible.

under the assumption that the diffusion coefficient D is isotropic. Higher order hyperdiffusion of order $2n$ involves a generalization of Eq. (169), to

$$\mathcal{J}^{(n)} = (-1)^n D^{(n)} \boldsymbol{\nabla}^{2n-1} \rho \,. \tag{171}$$

In our case, we are interested in the case $n = 3$, so that the hyperdiffusion term is

$$\frac{\partial \rho}{\partial t} = D^{(3)} \nabla^6 \rho. \tag{172}$$

The hyperdiffusion coefficient $D^{(3)}$ can be calculated from D assuming that at the Nyquist frequency the two formulations (170) and (172) yield the same quenching. Considering a wave as a Fourier series in one dimension (x), one element of the series is expressed as

$$\psi_k = A \mathrm{e}^{i(kx+\omega t)} \tag{173}$$

Plugging it into the second order diffusion equation (170) we have the dispersion condition $i\omega = -Dk^2$. The sixth order version (172) yields $i\omega = -D^{(3)}k^6$. Equating both we have $D^{(3)} = Dk^{-4}$. This condition should hold at the grid scale, where $k = \pi/\Delta x$, therefore

$$D^{(3)} = D\left(\frac{\Delta x}{\pi}\right)^4 \tag{174}$$

For the magnetic potential, resistivity has the same formulation as mass diffusion

$$\frac{\partial \boldsymbol{A}}{\partial t} = -\eta\nabla\times\boldsymbol{B} = \eta\nabla^2\boldsymbol{A}, \tag{175}$$

where we used the Coulomb gauge $\boldsymbol{\nabla}\cdot\boldsymbol{A} = 0$. The algebra is the same as above, also yielding $\eta^{(3)} = \eta(\Delta x/\pi)^4$. For entropy, the heat conduction term is

$$\frac{\partial S}{\partial t} = \frac{1}{\rho T}\boldsymbol{\nabla}\cdot(K\boldsymbol{\nabla}T)\,, \tag{176}$$

and requiring that K be constant, we substitute it by

$$\frac{\partial S}{\partial t} = \frac{K^{(3)}}{\rho T}\nabla^6 T. \tag{177}$$

also with $K^{(3)} = K(\Delta x/\pi)^4$.

E.2 Hyperviscosity

Viscosity has some caveats where subtleties apply. The difference is that the momentum flux due to viscosity is not proportional to the velocity gradient, but to the rate-of-strain tensor

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\boldsymbol{\nabla}\cdot\boldsymbol{u}\right)\,, \tag{178}$$

which only allows the viscous acceleration to be reduced to the simple formulation $\nu\nabla^2\boldsymbol{u}$ under the condition of incompressibility and constant dynamical viscosity $\mu = \nu\rho$. Due to this, the general expression for conservative hyperviscosity involves more terms. In some cases, it is no great overhead, but for others, simpler formulations can be applied.

E.2.1 Conservative case

In the general case, the viscous acceleration is

$$f_{\rm visc} = \rho^{-1}\boldsymbol{\nabla}\cdot(2\rho\nu\mathsf{S}) \tag{179}$$

So, for the hyperviscous force, we must replace the rate-of-strain tensor by a high order version

$$f_{\rm visc}^{\rm (hyper)} = \rho^{-1}\boldsymbol{\nabla}\cdot\left(2\rho\nu_n\mathsf{S}^{(n)}\right) \tag{180}$$

where the $n^{\rm th}$-order rate of strain tensor is

$$\mathsf{S}^{(n)} = (-\nabla^2)^{n-1}\mathsf{S}. \tag{181}$$

For the $n = 3$ case it is

$$S_{ij}^{(3)} = \frac{1}{2}\left(\frac{\partial^5 u_j}{\partial {x_i}^5} + \frac{\partial^4}{\partial {x_i}^4}\left(\frac{\partial u_i}{\partial x_j}\right) - \frac{1}{3}\frac{\partial^4}{\partial {x_i}^4}\left(\boldsymbol{\nabla}\cdot\boldsymbol{u}\right)\right) \ . \tag{182}$$

Plugging it into Eq. (180), and assuming $\mu_3 = \rho\nu_3 = \text{const}$

$$f_{\text{visc}}^{(\text{hyper})} = \nu_3\left(\nabla^6\boldsymbol{u} + \frac{1}{3}\nabla^4(\boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\boldsymbol{u}))\right) \ . \tag{183}$$

For $\nu_3 = \text{const}$, we have to take derivatives of density as well

$$f_{\text{visc}}^{(\text{hyper})} = \nu_3\left(\nabla^6\boldsymbol{u} + \frac{1}{3}\nabla^4(\boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\boldsymbol{u})) + 2\mathbf{S}^{(3)}\cdot\boldsymbol{\nabla}\ln\rho\right) \tag{184}$$

E.2.2 Non-conservative cases

Equations (183) and (184) explicitly conserve linear *and* angular momentum. Although desirable properties, such expressions are cumbersome and numerically expensive, due to the fourth order derivatives of $\boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\boldsymbol{u})$.

This term, however, is only important when high compressibility is present (since it depends on the divergence of $\boldsymbol{u}$). In practice we drop this term and use a simple hyperviscosity

$$f_{\text{visc}} = \begin{cases} \nu_3\nabla^6\boldsymbol{u} & \text{if } \mu = \text{const} \\ \nu_3\left(\nabla^6\boldsymbol{u} + 2\mathbf{S}^{(3)}\cdot\boldsymbol{\nabla}\ln\rho\right) & \text{if } \nu = \text{const} \end{cases} \tag{185}$$

Notice that this can indeed be expressed as the divergence of a simple rate-of-strain tensor

$$S_{ij}^{(3)} = \frac{\partial^5 u_i}{\partial {x_j}^5} \ , \tag{186}$$

so it does conserve linear momentum. It does *not*, however, conserve *angular* momentum, since the symmetry of the rate-of-strain tensor was dropped. Thus, vorticity sinks and sources may be spuriously generated at the grid scale.

A symmetric tensor can be computed, that conserves angular momentum and can be easily implemented

$$S_{ij} = \frac{1}{2}\left(\frac{\partial^5 u_i}{\partial {x_j}^5} + \frac{\partial^5 u_j}{\partial {x_i}^5}\right) \tag{187}$$

This tensor, however, is not traceless, and therefore accurate only for weak compressibility. It should work well if the turbulence is subsonic. Major differences are not expected, since the spectral range in which hyperviscosity operates is very limited: as a numerical tool, only its performance as a high-frequency filter is needed. This also supports the usage of the highest order terms only, since these are the ones that provide quenching at high k. Momentum conservation is a cheap bonus. Angular momentum conservation is perhaps playing it too safe, at great computational expense.

E.2.3 Choosing the coefficient

When changing the resolution, one wants to keep the grid Reynolds number, here defined as

$$\mathrm{Re}_{\mathrm{grid}} = u_{\mathrm{rms}} \big/ \left(\nu_n k_{\mathrm{Ny}}^{2n-1}\right) \tag{188}$$

approximately constant. Here, $k_{\mathrm{Ny}} = \pi/\delta x$ is the Nyquist wavenumber and δx is the mesh spacing. Thus, when doubling the number of meshpoints, we can decrease the viscosity by a factor of about $2^5 = 32$ (Haugen & Brandenburg 2004). This shows that hyperviscosity can allow a dramatic increase of the Reynolds number based on the scale of the box.

By choosing `idiff='hyper3_mesh'` in `density_run_pars` the hyperdiffusion for density is being set automatically in a mesh-independent way. A hyper-mesh Reynolds number of 30 corresponds to a coefficient `diffrho_hyper3_mesh=2` if *maxadvec* is about 1, but in practice we need a bit more (5 is currently the default).

E.2.4 Turbulence with hyperviscosity

When comparing hyperviscous simulations with non-hyperviscous ones, it turns out that the Reynolds number at half the Nyquist frequency is usually in the range 5–7, i.e.

$$\mathrm{Re}_{\mathrm{half-grid}} = u_{\mathrm{rms}} \big/ \left[\nu_n \left(k_{\mathrm{Ny}}/2\right)^{2n-1}\right] \approx \textbf{5–7} \tag{189}$$

The following table gives some typical values used in simulations with forcing wavenumber $k_{\mathrm{f}} = 1.5$ and a forcing amplitude of $f_0 = 0.02$. If hyperdiffusion D_3 is used in the continuity equation, the corresponding values are about 30 times smaller than those of ν_3; see Table 11.

Table 11: Empirical values of viscosity and hyperviscosity, as well as hyperdiffusion for density, at different numerical resolution, for simulations with forcing wavenumber $k_{\mathrm{f}} = 1.5$ and a forcing amplitude of $f_0 = 0.02$ in a 2π periodic domain. In all cases the half-mesh Reynolds number is about 5–7. For comparison, estimates of the numerical 4th order hyperdiffusion resulting from a third order time step are give for two values of the CFL parameter.

N	ν_1	ν_2	ν_3	D_3	$\kappa_2^{\mathrm{CFL}=0.4}$	$\kappa_2^{\mathrm{CFL}=0.9}$
16	1×10^{-2}	3×10^{-4}	2×10^{-5}	6×10^{-7}	7×10^{-4}	1×10^{-4}
32	5×10^{-3}	4×10^{-5}	6×10^{-7}	2×10^{-8}	1×10^{-6}	2×10^{-5}
64	2×10^{-3}	5×10^{-6}	2×10^{-8}	6×10^{-10}	2×10^{-7}	3×10^{-6}
128	1×10^{-3}	6×10^{-7}	6×10^{-10}	2×10^{-11}	3×10^{-8}	4×10^{-7}
256	5×10^{-4}	8×10^{-8}	2×10^{-11}	6×10^{-13}	4×10^{-9}	5×10^{-8}
512	2×10^{-4}	1×10^{-8}	6×10^{-13}	2×10^{-14}	5×10^{-10}	6×10^{-9}

For comparison, we give in Table 11 estimates of the numerical 4th order hyperdiffusion resulting from a third order time step, for which we have

$$\kappa_2^{\mathrm{CFL}} = \frac{1}{24} u_{\mathrm{rms}} \left(C_{\mathrm{CFL}} \delta x\right)^3 \tag{190}$$

where C_{CFL} is the CFL parameter which is either 0.4 in the conservative case or 0.9 in the more progressive case.

E.3 Anisotropic hyperdissipation

As we want quenching primarily at the Nyquist frequency, hyperdissipation depends intrinsically on the resolution, according to Eq. (174). Because of this, *isotropic* hyperdissipation only gives equal quenching in all spatial directions if Δx=Δy=Δz, i.e., if the cells are cubic. For non-cubic cells, anisotropic dissipation is required as different directions may be better/worse sampled, thus needing less/more numerical smoothing. Such generalization is straightforward. For that, we replace Eq. (171) by

$$\boldsymbol{\mathcal{J}} = \left(D_x \frac{\partial^5 \rho}{\partial x^5}, D_y \frac{\partial^5 \rho}{\partial y^5}, D_z \frac{\partial^5 \rho}{\partial z^5} \right), \tag{191}$$

so that different diffusion operates in different directions. Since D_x, D_y and D_z are constants, the divergence of this vector is

$$\boldsymbol{\nabla} \cdot \boldsymbol{\mathcal{J}} = D_x \frac{\partial^6 \rho}{\partial x^6} + D_y \frac{\partial^6 \rho}{\partial y^6} + D_z \frac{\partial^6 \rho}{\partial z^6}. \tag{192}$$

The formulation for resistivity and heat conductivity are strictly the same. For viscosity it also assumes the same form if we consider the simple non-conservative rate-of-strain tensor (186).

Mathematically, these operations can be written compactly by noticing that the coefficients in Eq. (192) transform like diagonal tensors $\chi^{(3)}_{ij} = \chi^{(3)}_{k} \delta_{ijk}$, where δ_{ijk} is the unit diagonal third order tensor, $\chi^{(3)}$ is the vector containing the dissipative coefficients (diffusion, viscosity, resistivity, or heat conductivity) in x, y, and z, and summation over repeated indices applies.

Therefore, for a scalar quantity ψ (density, any of the three components of the velocity or magnetic potential), we can write

$$\frac{\partial \psi}{\partial t} = -\chi^{(3)}_{ij} \partial_i \partial^5_j \psi = -\sum_q \chi^{(3)}_q \frac{\partial^6}{\partial x^6_q} \psi. \tag{193}$$

E.4 Hyperviscosity in Burgers shock

Hyperviscosity has the unfortunate property of introducing (numerically stable) wiggles, even if one just adds a little bit of hyperviscosity to a run with normal viscosity; see left hand side of Fig. 17. Running with just hyperviscosity give strong wiggles.

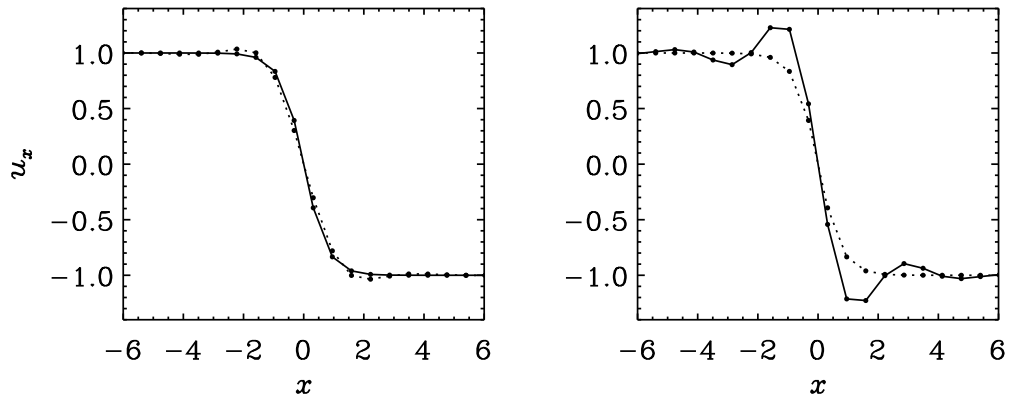

Figure 17: Left: Burgers shock from `teach/PencilCode/material/BurgersShock` (in the teaching material) with $-20 \leq x \leq 20$, $n_x = 64$ mesh points, $u_x = \mp 1$ on the two ends, $\nu = 0.4$ and either $\nu_3 = 0$ (solid line) or $\nu_3 = 0.05$ (dashed line). Right: similar to the left hand side, but with $\nu = 0$ and $\nu_3 = 0.05$ (dashed line), compared with the case $\nu = 0.4$ and $\nu_3 = 0$ (solid line).

F Special techniques

F.1 Remeshing (regridding)

[This should be written up in a better way and put somewhere else. But currently, remeshing is only available for the Pencil developers anyway.]

Suppose you have a directory run_64 with a 64^3 run (running on $N_0 = ny \times nz = 2 \times 1$ CPUs) that you want to continue from 'VAR1' at 128^3 (on $ny \times nz = 4 \times 4$ CPUs).

1. Get the remeshing stuff from repository:

```
unix>  cd $PENCIL_HOME; cvs co -d remesh pencil-remesh
unix>  setenv PATH ${PATH}:$PENCIL_HOME/remesh/bin
```

2. Create another run directory with current 'VAR1' as 'var.dat' (remesh.csh so far only works with 'var.dat'):

```
unix>  cd run_64
run_64>  pc_newrun ../tmp_64  or  new tmp_64
run_64>  mkdir -p ../tmp_64/data  or  (cd ../tmp_64/; crtmp)
run_64>  (cd ../tmp_64/data ; mkproc-tree N_0)
run_64>  restart-new-dir-VAR ../tmp_64 1
```

3. Create the new run directory (linking the executables with -s):

```
run_64>  cd ../tmp_64
tmp_64>  pc_newrun -s ../run_128  or  new run_128
tmp_64>  vi ../run_128/src/cparam.local
# set nxgrid=128, ncpus=16, nprocy=4
tmp_64>  (cd ../run_128; crtmp; pc_setupsrc; make)
```

4. Setup and do remeshing

```
tmp_64>  setup-remesh
tmp_64>  vi remesh/common.local
# set muly=2, mulz=4, remesh_par=2
tmp_64>  (cd remesh; make)
tmp_64>  vi remesh.in
# Replace line by ../run_128
tmp_64>  remesh.csh
# Answer 'yes'
```

F.2 Restarting from a run with less physics

First, prepare a new run directory with the new physics included. By new physics, we mean that the new run wants to read in more fields (e.g. magnetic fields, if the old run didn't have magnetic fields).

Example for test fields:

1. Prepare 'src/cparam.local'

   Add the following 2 fragments into the 'cparam.local' file. The first piece comes in the beginning and the second in the end of the file.

```
!** AUTOMATIC CPARAM.INC GENERATION ************************************
! Declare (for generation of cparam.inc) the number of f array
! variables and auxiliary variables added by this module
! Use MVAR to reserve the appropriate workspace for testfield_z.f90
! The MAUX number must be equally big and is used for uxb in the f-array.
! At the end of this file, njtest must be set such that 3*njtest=MVAR.
!
! MVAR CONTRIBUTION 12
! MAUX CONTRIBUTION 12
!
!***************************************************************************

!
!  note that MVAR=MAUX=3*njtest must be obeyed
!
integer, parameter :: njtest=4
!
```

2. Prepare 'src/Makefile.local'

   Add the line TESTFIELD=testfield_z to the file. Finally, compile the code.

3. Prepare restart data

   Go into data directory of the new run and prepare the directory tree using, e.g., the command pc_mkproctree 16. [In principle this could be automatized, but it isn't yet.]

   Next, go into old run directory and say restart-new-dir ../32c, if '../32c' is the name of the new run directory. This procedure copies all the files from the processor tree, plus files like 'param.nml', but this file may need some manual modification (or you could just us one from another runs with the new physics included, which is definitely the simplest!).

4. Prepare 'run.in'

   Set lread_oldsnap_notestfield=T in *run_pars*. This means (as the name says) that one reads an old snapshot that did not have test fields in it.

   Reset boundary conditions and add stuff for the newly added fields, e.g., bcz='a:s','a','a:s','a2','a','a','s','a','a','s','a','a','s','a','a','s' in *run_pars*. If you don't do this, you would effectively use periodic boundary conditions for the response to the test field, which is hardly correct once you set non-periodic boundary conditions for the other variables.

   Add something like the following text fragments in the right position (after *grav_run_pars* and *magnetic_run_pars*, but before *shear_run_pars* and *viscosity_run_pars*.

```
&testfield_run_pars
  !linit_aatest=T, daainit=100.
  itestfield='B11-B22'
```

```
  etatest=1e-4
  lsoca=F
/
```

Make sure that the data above are correct. You may want to change the values of *daainit* or *etatest*.

If you now run, and if you didn't fix the file 'data/param.nml' you might get something like the following error:

```
forrtl: severe (24): end-of-file during read, unit 1, file /wkspace/brandenb/penci
```

The reason for this is that it reads the old boundary data, but the corresponding array is too short. This includes stuff like *FBCX1* to *FBCX2_2*, but it is still not enough. Therefore it is easiest to use the 'data/param.nml' file from another run. You may well just use one from a single processor run with a different mesh. But remember to fix the 'start.in' file by correcting the boundary conditions and adding things like

```
&testfield_init_pars
  luxb_as_aux=T
/
```

5. Prepare 'print.in', 'xyaver.in', and other obvious files such as 'video.in'.
6. Once it works and is running, you must say *explicitly*

   ```
   &run_pars
     ...
     lread_oldsnap_notestfield=F
   /
   ```

   because otherwise you won't read in your precious test field data next time you restart the code! (If you instead just remove this line, it will remember lread_oldsnap_notestfield=T from the previous run, which is of course wrong!)

Comments: For large magnetic Reynolds numbers the solutions to the test-field equations can show a linear instability, which can introduce large fluctuations. In that case it is best to reset the dependent test-field variable to zero in regular intervals. This is done by setting linit_aatest=T. Note that daainit=100 sets the reset interval to 100.

G Runs and reference data

For reference purposes we document here some results obtained with various samples of the code.

G.1 Shock tests

G.1.1 Sod shock tube problem

Table 12: Combinations of ρ, p, and s/c_p that are relevant for the Sod shock tube problem with constant temperature and different pressure ratios on the left and right hand sides of the shock.

ρ	p	s
1.0	1.0	0.3065
0.1	0.1	1.2275
0.01	0.01	2.1486

G.1.2 Temperature jump

Table 13: Combinations of $c_{\rm s}^2$, p, and s/c_p that are relevant for the temperature shock problem with constant density, $\rho=1$, and different temperature ratios on the left and right hand sides of the shock.

$c_{\rm s}^2$	s
1.0	0.0
0.1	-2.3
0.01	-4.6
10^{-4}	-9.2

G.2 Random forcing function

A solenoidal random forcing function f can be invoked by putting `iforce='helical'` in the *forcing_run_pars* namelist. This produces the forcing function $\boldsymbol{f}$ of the form

$$\boldsymbol{f}(\boldsymbol{x},t)=\mathrm{Re}\{N\boldsymbol{f}_{\boldsymbol{k}(t)}\exp[i\boldsymbol{k}(t)\cdot\boldsymbol{x}+i\phi(t)]\}, \tag{194}$$

where $\boldsymbol{k}(t)=(k_x,k_y,k_z)$ is a time dependent wave vector, $\boldsymbol{x}=(x,y,z)$ is position, and $\phi(t)$ with $|\phi|<\pi$ is a random phase. On dimensional grounds the normalization factor is chosen to be $N=f_0c_{\rm s}(kc_{\rm s}/\delta t)^{1/2}$, where f_0 is a nondimensional factor, $k=|\boldsymbol{k}|$, and δt is the length of the timestep. The $\delta t^{-1/2}$ dependence ensures that the forcing, which is delta-correlated in time, is properly normalized such that the correlator of the forcing function is independent of the length of the time step, δt. We focus on the case where $|\boldsymbol{k}|$ is around 5, and select at each timestep randomly one of the 350 possible vectors in $4.5<|\boldsymbol{k}|<5.5$. We force the system with eigenfunctions of the curl operator,

$$\boldsymbol{f}_{\boldsymbol{k}}=\frac{i\boldsymbol{k}\times(\boldsymbol{k}\times\boldsymbol{e})-\sigma|\boldsymbol{k}|(\boldsymbol{k}\times\boldsymbol{e})}{\sqrt{1+\sigma^2}\,\boldsymbol{k}^2\sqrt{1-(\boldsymbol{k}\cdot\boldsymbol{e})^2/\boldsymbol{k}^2}}, \tag{195}$$

where $\boldsymbol{e}$ is an arbitrary unit vector needed in order to generate a vector $\boldsymbol{k} \times \boldsymbol{e}$ that is perpendicular to $\boldsymbol{k}$. Note that $|\boldsymbol{f}_{\boldsymbol{k}}|^2 = 1$ and, for $\sigma = 1$, $i\boldsymbol{k} \times \boldsymbol{f}_{\boldsymbol{k}} = |\boldsymbol{k}|\boldsymbol{f}_{\boldsymbol{k}}$, so the helicity density of this forcing function satisfies

$$\boldsymbol{f} \cdot \boldsymbol{\nabla} \times \boldsymbol{f} = |\boldsymbol{k}|\boldsymbol{f}^2 > 0 \quad (\text{for } \sigma = 1) \tag{196}$$

at each point in space. We note that since the forcing function is like a delta-function in $\boldsymbol{k}$-space, this means that all points of $\boldsymbol{f}$ are correlated at any instant in time, but are different at the next timestep. Thus, the forcing function is delta-correlated in time (but the velocity is not). This is the forcing function used in Brandenburg (2001), Brandenburg & Dobler (2001), and other papers in that series.

For $\sigma = 0$, the forcing function is completely nonhelical and reduces to the simpler form

$$\boldsymbol{f}_{\boldsymbol{k}} = (\boldsymbol{k} \times \boldsymbol{e}) \, / \sqrt{\boldsymbol{k}^2 - (\boldsymbol{k} \cdot \boldsymbol{e})^2}. \tag{197}$$

For $0 < |\sigma| < 1$, the forcing function has fractional helicity, where $\sigma \approx \langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle \, / (k_{\rm f} \langle \boldsymbol{u}^2 \rangle)$; see Sect. 4.5 of Ref. [7]. In the code and the *forcing_run_pars* namelist, σ is called *relhel*.

In the code, the possible wavevectors are pre-calculated and stored in '`k.dat`', which is being read in the beginning the code runs. To change the wavevectors (e.g. the typical value of $k_{\rm f}$, you need to change the file. In the directory '`$PENCIL_HOME/samples/helical-MHDturb/K_VECTORS/`' there are several such files prepared:

```
k10.dat  k1.dat   k2.dat   k3.dat  k5.dat
k15.dat  k27.dat  k30.dat  k4.dat  k8.dat
```

and more can be prepared in `IDL` with the procedure '`$PENCIL_HOME/samples/helical-MHDturb/idl/generate_kvectors.pro`' There is also more help in the '`README`' file in '`helical-MHDturb`'.

G.3 Three-layered convection model

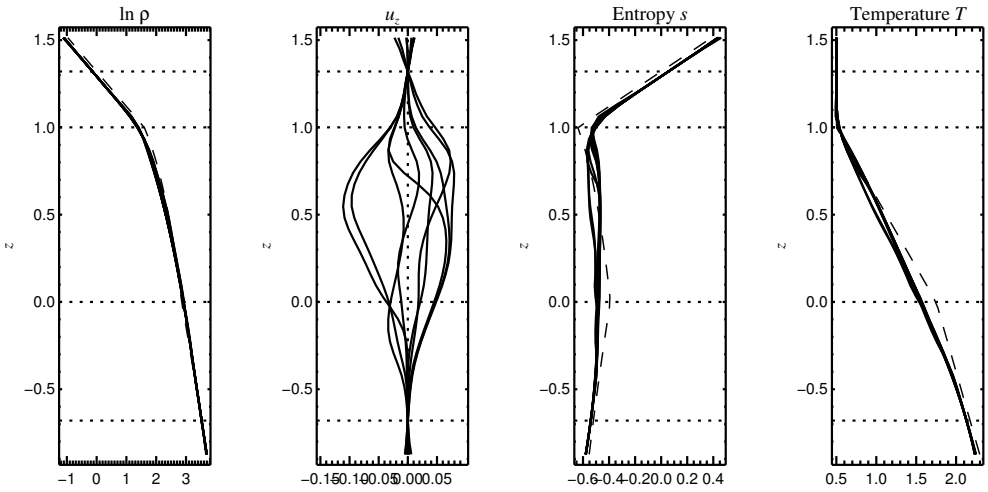

Figure 18: Like in Fig. 2, but at time $t = 50$.

In Sect. 3 we have shown the early stages of the convection model located in '`samples/conv-slab`'. To arrive at fully developed convection, you will need to run the

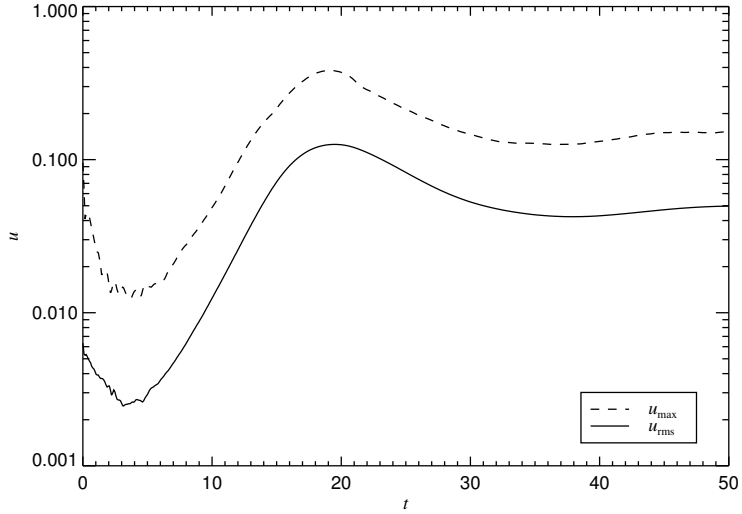

Figure 19: Time evolution of rms and maximum velocity for the model 'samples/conv-slab'. Similar plots can be produced by running the IDL script 'ts.pro'.

code for many more time steps. Figure 18 shows the vertical profiles of four basic quantities at time $t = 50$. Figure 19 shows the time evolution of rms and maximum velocity for the model for $0 < t < 50$.

Figures 20 and 21 show vertical and horizontal sections for time $t = 50$.

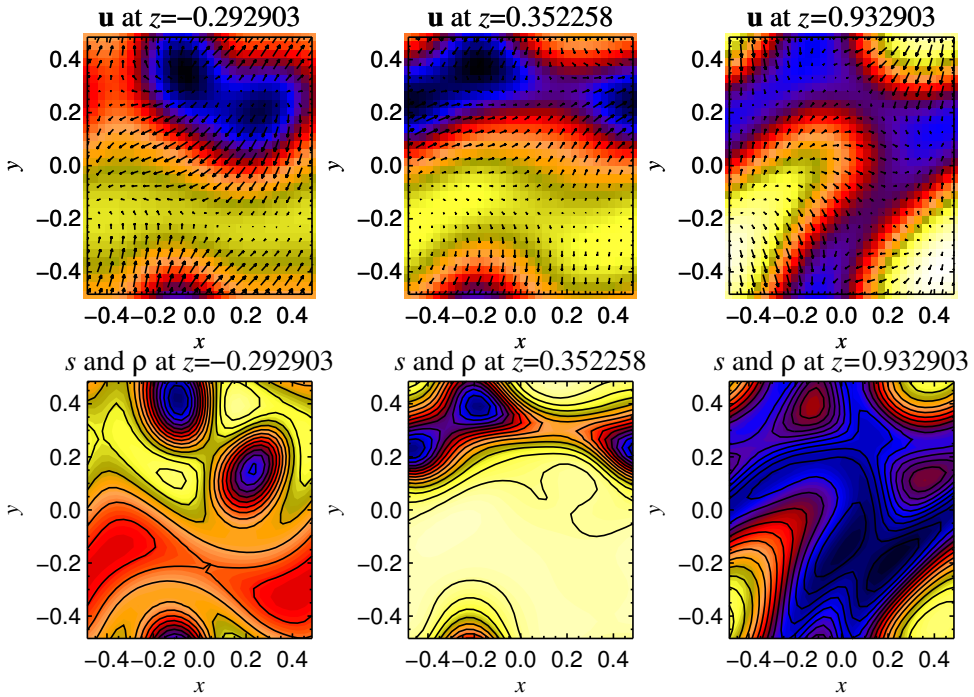

Figure 20: Horizontal sections for $t = 50$. Top: velocity field. Bottom: entropy (color coded) and density (isocontours). Plots of this type can be produced by running the IDL script 'hsections.pro')

G.4 Magnetic helicity in the shearing sheet

To test magnetic helicity evolution in the shearing shear, we can choose as initial condition `initaa='Beltrami-y'` with `amplaa=1.` in `magnetic_init_pars` together with `Sshear=-1.` in `shear_run_pars`.

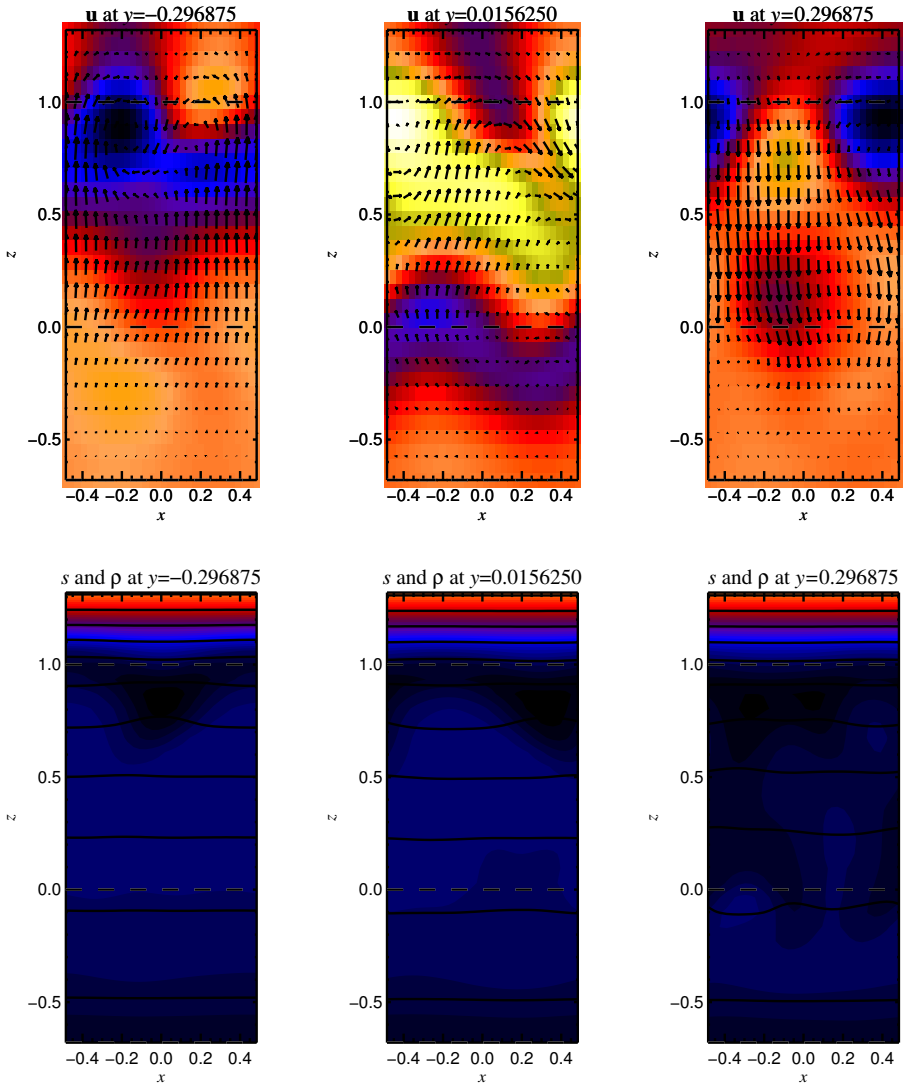

Figure 21: Vertical section $y = 0.516$ at $t = 50$. Top: velocity field. Bottom: entropy (color coded) and density (isocontours). Plots of this type can be produced by running the IDL scripts '`vsections.pro`') or '`vsections2.pro`').

Thus, in '`src/Makefile.local`' we just use

```
MAGNETIC=magnetic
HYDRO=nohydro
EOS=noeos
DENSITY=nodensity
SHEAR=shear
VISCOSITY=noviscosity
```

and put

```
&init_pars
  cvsid='$Id: manual.tex 21616 2014-03-17 22:16:54Z AxelBrandenburg $',
/
&magnetic_init_pars
  initaa='Beltrami-y', amplaa=1.
/
&shear_init_pars
```

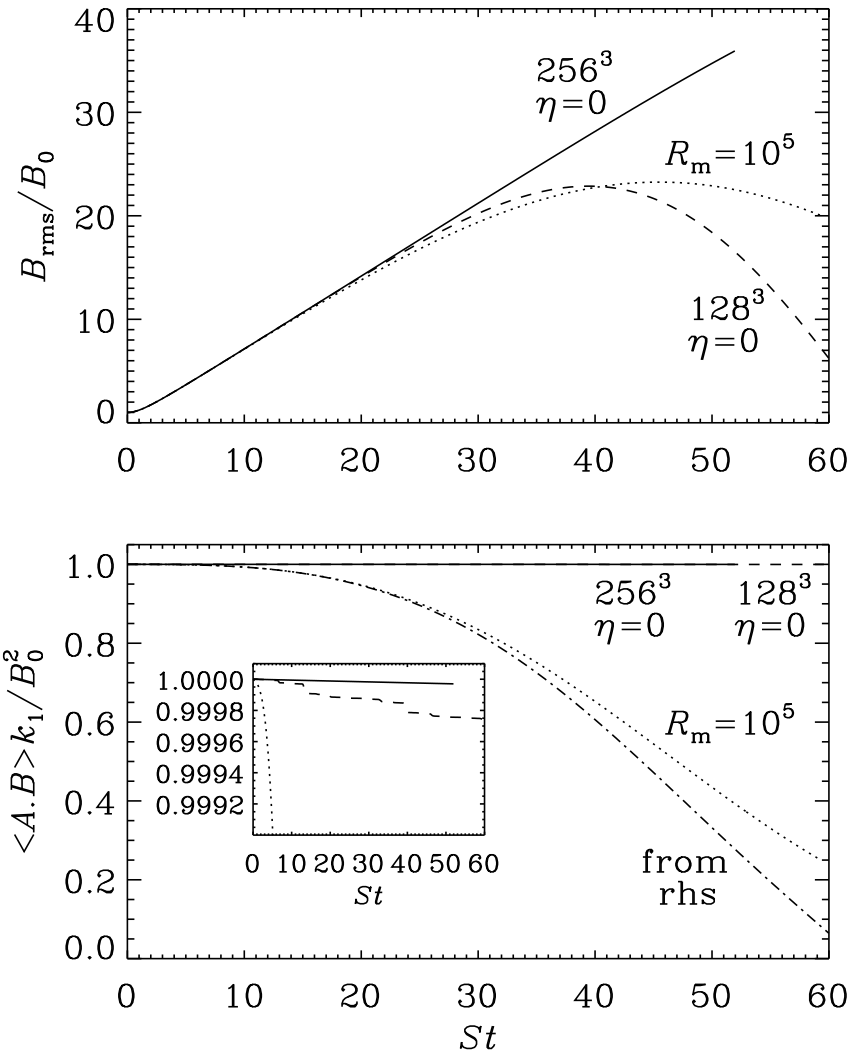

Figure 22: Wind-up of the magnetic field leads to a linear increase in the rms magnetic field strength until Ohmic diffusion begins to become important (top panel). During this time the magnetic helicity is conserved. With Ohmic diffusion, the decay of $\langle \boldsymbol{A}\cdot\boldsymbol{B}\rangle$ is well described by integrating $-2\eta\langle \boldsymbol{J}\cdot\boldsymbol{B}\rangle$ (indicated by "from rhs" in the second panel).

```
/
```

in '`start.in`' and, for example,

```
&run_pars
  cvsid='$Id: manual.tex 21616 2014-03-17 22:16:54Z AxelBrandenburg $'
  nt=150000, it1=10, cdt=0.9, isave=50, itorder=3
  dsnap=100. dvid=5., ialive=1
/
&magnetic_run_pars
  eta=0.
/
&shear_run_pars
  Sshear=-1.
/
```

in '`run.in`'. The output includes, among other things

```
  arms(f10.7)
```

```
brms(f12.7)
jrms(f14.7)
abm(f14.11)
jbm(f14.7)
```

The result is shown in Figure 22, where we show the wind-up of the magnetic field, which leads to a linear increase in the rms magnetic field strength until Ohmic diffusion begins to become important (top panel). During this time the magnetic helicity is conserved. With Ohmic diffusion, the decay of $\langle \boldsymbol{A} \cdot \boldsymbol{B} \rangle$ is well described by integrating $-2\eta \langle \boldsymbol{J} \cdot \boldsymbol{B} \rangle$ (indicated by "from rhs" in the second panel).

H Numerical methods

H.1 Sixth-order spatial derivatives

Spectral methods are commonly used in almost all studies of ordinary (usually incompressible) turbulence. The use of this method is justified mainly by the high numerical accuracy of spectral schemes. Alternatively, one may use high order finite differences that are faster to compute and that can possess almost spectral accuracy. Nordlund & Stein [23] and Brandenburg et al. [10] use high order finite difference methods, for example fourth and sixth order compact schemes [21].[17]

The sixth order first and second derivative schemes are given by

$$f_i' = (-f_{i-3} + 9f_{i-2} - 45f_{i-1} + 45f_{i+1} - 9f_{i+2} + f_{i+3})/(60\delta x), \tag{198}$$

$$f_i'' = (2f_{i-3} - 27f_{i-2} + 270f_{i-1} - 490f_i + 270f_{i+1} - 27f_{i+2} + 2f_{i+3})/(180\delta x^2), \tag{199}$$

In Fig. 23 we plot effective wavenumbers for different schemes. Apart from the different *explicit* finite difference schemes given above, we also consider a *compact* scheme of 6th order, which can be written in the form

$$\tfrac{1}{3}f_{i-1}' + f_i' + \tfrac{1}{3}f_{i+1}' = (f_{i-2} - 28f_{i-1} + 28f_{i+1} - f_{i+2})/(36\delta x), \tag{200}$$

for the first derivative, and

$$\tfrac{2}{11}f_{i-1}'' + f_i'' + \tfrac{2}{11}f_{i+1}'' = (3f_{i-2} + 48f_{i-1} - 102f_i + 48f_{i+1} + 3f_{i+2})/(44\delta x^2). \tag{201}$$

for the second derivative. As we have already mentioned in the introduction, this scheme involves obviously solving tridiagonal matrix equations and is therefore effectively nonlocal.

In the second panel of Fig. 23 we have plotted effective wavenumbers for second derivatives, which were calculated as

$$(\cos kx)''_{\rm num} = -k_{\rm eff}^2 \cos kx. \tag{202}$$

Of particular interest is the behavior of the second derivative at the Nyquist frequency, because that is relevant for damping zig-zag modes. For a second-order finite difference scheme $k_{\rm eff}^2$ is only 4, which is less than half the theoretical value of $\pi^2 = 9.87$. For fourth, sixth, and tenth order schemes this value is respectively 5.33, 6.04, 6.83. The last value is almost the same as for the 6th order compact scheme, which is 6.86. Significantly stronger damping at the Nyquist frequency can be obtained by using hyperviscosity, which Nordlund & Galsgaard (1995) treat as a quenching factor that diminishes the value of the second derivative for wavenumbers that are small compared with the Nyquist frequency. Accurate high order second derivatives (with no quenching factors) are important when calculating the current $\boldsymbol{J}$ in the Lorentz force $\boldsymbol{J}\times\boldsymbol{B}$ from a vector potential $\boldsymbol{A}$ using $-\mu_0\boldsymbol{J} = \nabla^2\boldsymbol{A} - \boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{A}$. This will be important in the MHD calculations presented below.

[17] The fourth order compact scheme is really identical to calculating derivatives from a cubic spline, as was done in Ref. [23]. In the book by Collatz [11] the compact methods are also referred to as *Hermitian methods* or as *Mehrstellen-Verfahren*, because the derivative in one point is calculated using the derivatives in neighboring points.

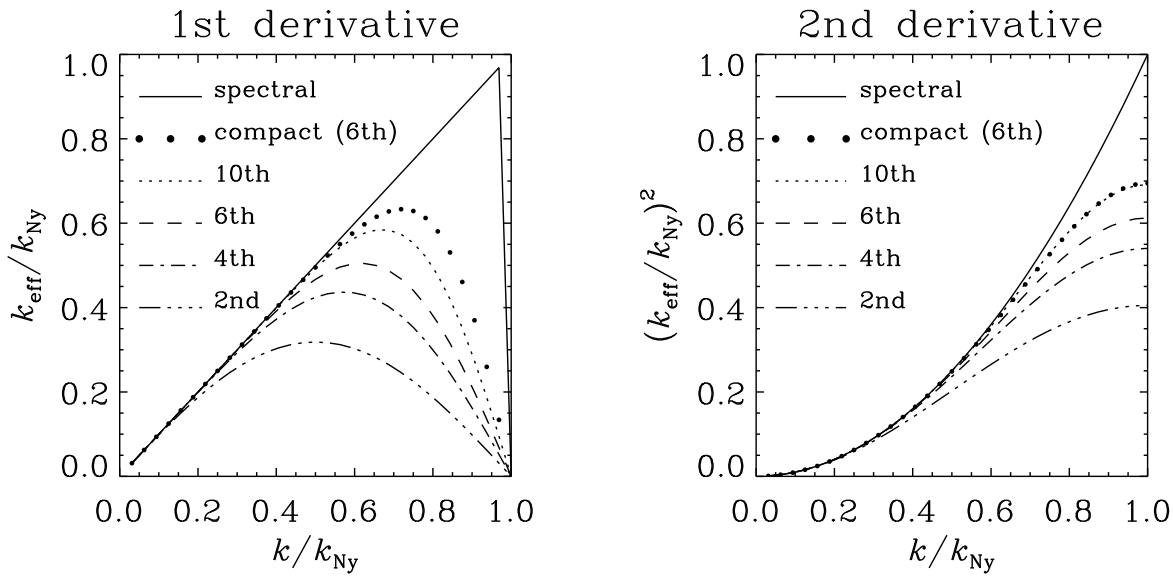

Figure 23: Effective wave numbers for first and second derivatives using different schemes. Note that for the second derivatives the sixth order compact scheme is almost equivalent to the tenth order explicit scheme. For the first derivative the sixth order compact scheme is still superior to the tenth order explicit scheme.

H.2 Upwind derivatives to avoid 'wiggles'

High-order centered-difference convection simulations often show "wiggles" (Nyquist zigzag pattern) in $\ln\rho$, which are apparently caused by a velocity profile where the velocity approaches zero on the boundary or inside the box.[18] This causes the density profile to be squeezed into a boundary layer where eventually numerical resolution is insufficient and, for centered differences, a spurious Nyquist signal is generated that almost instantaneously propagates into much of the whole box.

Even if the stagnation point is on the boundary (and enforced by the boundary conditions), this behavior is hardly influenced by the boundary conditions on $\ln\rho$ at all. A solution, however, is to apply upwinded derivative operators. The simplest upwind derivative is a finite-difference derivative operator where the point furthest downwind is excluded from the stencil. For $u>0$, that means that instead of

$$f'_0 = \frac{-f_{-3}+9f_{-2}-45f_{-1}\qquad+45f_1-9f_2+f_3}{60\,\delta x} - \frac{\delta x^6\,f^{(7)}}{140} = D^{(\mathrm{cent},6)} + O\left(\delta x^6\right)\ , \tag{203}$$

one takes

$$f'_0 = \frac{-2f_{-3}+15f_{-2}-60f_{-1}+20f_0+30f_1-3f_2}{60\,\delta x} + \frac{\delta x^5\,f^{(6)}}{60} = D^{(\mathrm{up},5)} + O\left(\delta x^5\right)\ . \tag{204}$$

A fourth-order upwind scheme (excluding two downwind points) would be

$$f'_0 = \frac{-f_{-3}+6f_{-2}-18f_{-1}+10f_0+3f_1}{12\,\delta x} - \frac{\delta x^4\,f^{(5)}}{20} = D^{(\mathrm{up},4)} + O\left(\delta x^4\right)\ . \tag{205}$$

[18] A simple one-dimensional test profile would be $u(x)=1-x^2$ on $x\in[-1,1]$, which will accumulate more and more mass near the right boundary $x=1$.

In two- or three-dimensional settings, the presence of stagnation points of X-type leads to the same configuration, this time with the possibility of a steady state (i.e. without accumulation of mass). Such stagnation points occur e.g. at the top of an upwelling, or at the bottom of a downdraft in convection simulations, where locally $u_z \propto z_{\rm X}-z$.

The effect of upwinding is mostly to stop the Nyquist perturbations from spreading away from the boundary or stagnation point. With the fourth-order formula they actually hardly ever develop.

The difference between central and fifth-order upwind derivative is

$$[D^{(\mathrm{up},5)} - D^{(\mathrm{cent},6)}]f_0 = \frac{-f_{-3} + 6f_{-2} - 15f_{-1} + 20f_0 - 15f_1 + 6f_2 - f_3}{60\,\delta x} = -\frac{\delta x^5}{60} f_0^{(6)} \ . \quad (206)$$

In other words, 5th-order upwinding can be represented for any sign of u as hyperdiffusion (Dobler et al. 2006):

$$-uf'_{(\mathrm{up},5\mathrm{th})} = -uf'_{(\mathrm{centr},6\mathrm{th})} + \frac{|u|\,\delta x^5}{60} f^{(6)} \ . \quad (207)$$

The advantage over adopting constant hyperdiffusion is that in the upwinding scheme hyperdiffusion is only applied where it is needed (i.e. where advection is happening, hence the factor $|u|$).

The form (207) also suggests an easy way to get 'stronger' upwinding: Rather than excluding more points in the downwind direction, we can simply treat the weight of the hyperdiffusion term as a free parameter α:

$$-uf'_{(\mathrm{up},5\mathrm{th},\alpha)} = -uf'_{(\mathrm{centr},6\mathrm{th})} + \alpha\,|u|\,\delta x^5 f^{(6)} \ . \quad (208)$$

If α is large, this may affect the time step, but for $\alpha = 1/60$, the stability requirement for the hyperdiffusive term should always be weaker than the advective Courant criterion.

H.3 The bidiagonal scheme for cross-derivatives

The *old* default scheme used for cross-derivatives of type $\partial^2/(\partial x \partial y)$ used to read as follows:

```
df=fac*( &
            270.*( f(l1+1:l2+1,m+1,n,k)-f(l1-1:l2-1,m+1,n,k)  &
                  +f(l1-1:l2-1,m-1,n,k)-f(l1+1:l2+1,m-1,n,k)) &
          - 27.*( f(l1+2:l2+2,m+2,n,k)-f(l1-2:l2-2,m+2,n,k)  &
                  +f(l1-2:l2-2,m-2,n,k)-f(l1+2:l2+2,m-2,n,k)) &
          +  2.*( f(l1+3:l2+3,m+3,n,k)-f(l1-3:l2-3,m+3,n,k)  &
                  +f(l1-3:l2-3,m-3,n,k)-f(l1+3:l2+3,m-3,n,k)) &
       )
```

and is "visualized" in the left part of Fig. 24. It is way more efficient than the straightforward approach of first taking the x and the y derivative consecutively. (shown in the right part of Fig. 24).

Off-diagonal terms enter not only the diffusion terms through $\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{u}$ and $\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{A}$ terms, but also through the $\boldsymbol{J} = \boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\boldsymbol{A}$ operator. The general formula is $J_i = A_{j,ij} - A_{i,jj}$, so in 2-D in the xy-plane we have

$$J_x = A_{x,xx} + A_{y,xy} - A_{x,xx} - A_{x,yy} = A_{y,xy} - A_{x,yy} \ , \quad (209)$$

$$J_y = A_{x,yx} + A_{y,yy} - A_{y,xx} - A_{y,yy} = A_{x,yx} - A_{y,xx} \quad (210)$$

Figure 25 shows how the two schemes perform for the propagation of Alfvén waves,

-2	0	0	0	0	0	+2
0	+27	0	0	0	-27	0
0	0	-270	0	+270	0	0
0	0	0	0	0	0	0
0	0	+270	0	-270	0	0
0	-27	0	0	0	+27	0
+2	0	0	0	0	0	-2

9	-27	135	0	-135	27	-9
-27	81	-405	0	405	-81	27
135	-405	2025	0	-2025	405	-135
0	0	0	0	0	0	0
-135	405	-2025	0	2025	-405	135
27	-81	405	0	-405	81	-27
-9	27	-135	0	135	-27	9

Figure 24: Weights of bidiagonal scheme (left) and consecutive scheme (right) for mixed derivatives $\partial^2/\partial x \partial y$. The numbers shown need to be divided by $720\,\delta x\,\delta y$ for the bidiagonal and by $3600\,\delta x\,\delta y$ for the consecutive scheme.

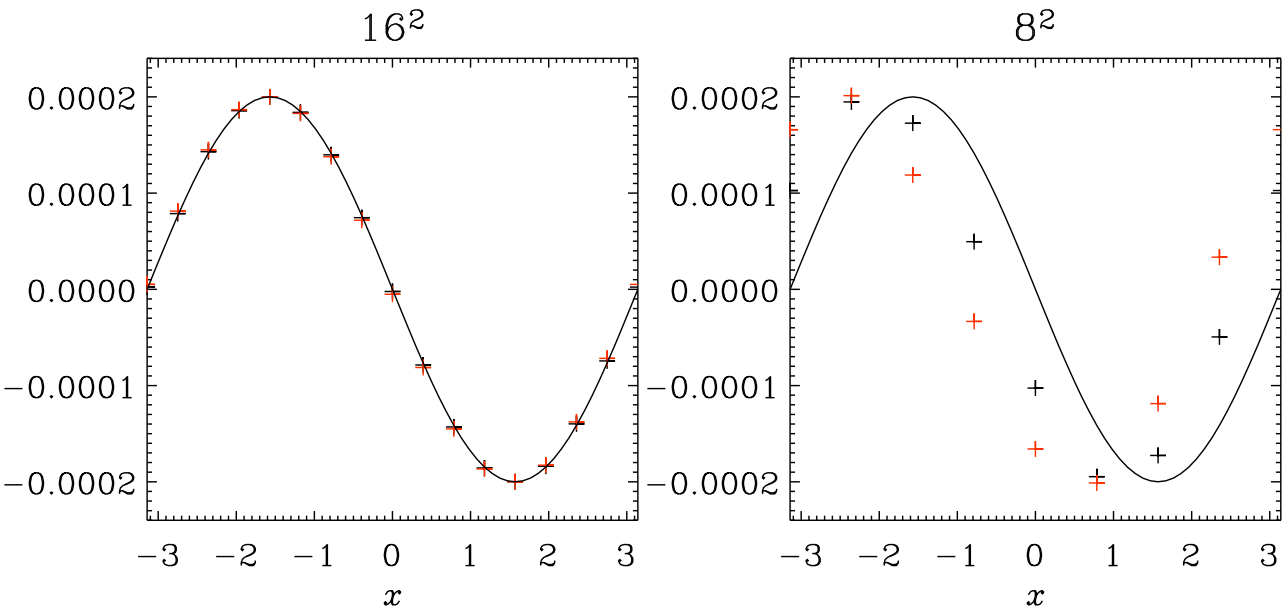

Figure 25: Alfvén wave for $\boldsymbol{B}_0 = (1, 2, 0)$ and $\boldsymbol{k} = (1, 2, 0)$ after $t = 2\pi$. The wave travels in the direction of $\boldsymbol{k}$. Red symbols are for the bidiagonal scheme, black symbols show results obtained using the consecutive scheme. Already for 16^2 mesh points there are no clear differences. For 8^2 mesh points both schemes are equally imprecise regarding the phase error, but the amplitude error is still quite small (but this is mainly a property of the time stepping scheme).

$$\dot{u}_z = J_x B_{0y} - J_y B_{0x} \,, \tag{211}$$
$$\dot{A}_x = -u_z B_{0y} \,, \tag{212}$$
$$\dot{A}_y = +u_z B_{0x} \,. \tag{213}$$

The initial condition (as implemented in `subroutine alfven_xy`) is

$$u_z \sim \cos(k_x x + k_y y - \omega t) \,, \tag{214}$$
$$A_x \sim +B_{0y} \sin(k_x x + k_y y - \omega t)/\omega \,, \tag{215}$$
$$A_y \sim -B_{0x} \sin(k_x x + k_y y - \omega t)/\omega \,, \tag{216}$$

where $\omega = \boldsymbol{k} \cdot \boldsymbol{B}_0$. The figure shows that there is no clear advantage of either scheme, so the code uses the more efficient bidiagonal one.

H.4 The 2N-scheme for time-stepping

For time stepping, higher-order schemes are necessary in order to reduce the amplitude and phase errors of the scheme and, to some extent, to allow longer time steps. Usually such schemes require large amounts of memory. However, we here use the memory-effective $2N$-schemes that require only two sets of variables to be held in memory. Such schemes work for arbitrarily high order, although not all Runge-Kutta schemes can be written as $2N$-schemes [27, 26]. Consider the ordinary differential equation (ODE)

$$\dot{u} = F(u,t) \ , \tag{217}$$

which can also be used as a prototype for a system of ODEs to be solved, like the ones obtained by spatial discretization of PDEs. The $2N$-schemes construct an approximation to the solution

$$u^{(n)} \equiv u(t_n) \tag{218}$$

according to the iterative procedure

$$w_i = \alpha_i w_{i-1} + \delta t \, F(u_{i-1}, t_{i-1}) \ , \tag{219}$$

$$u_i = u_{i-1} + \beta_i w_i \ . \tag{220}$$

For a three-step scheme we have $i = 1, ..., 3$. In order to advance the variable u from $u^{(n)}$ at time $t^{(n)}$ to $u^{(n+1)}$ at time $t^{(n+1)} = t^{(n)} + \delta t$ we set in Eq. (220)

$$u_0 = u^{(n)} \quad \text{and, after the last step,} \quad u^{(n+1)} = u_3, \tag{221}$$

with u_1 and u_2 being intermediate steps. In order to be able to calculate the first step, $i = 1$, for which no $w_{i-1} \equiv w_0$ exists, we have to require $\alpha_1 = 0$. Thus, we are left with 5 unknowns, α_2, α_3, β_1, β_2, and β_3. Three conditions follow from the fact that the scheme be third order for linear equations, so we have to have two more conditions. One possibility is to choose the fractional times at which the right hand side is evaluated, for example $(0, 1/3, 2/3)$ or even $(0, 1/2, 1)$. Yet another possibility is to require that inhomogeneous equations of the form $\dot{u} = t^n$ with $n = 1$ and 2 are solved exactly. The corresponding coefficients are listed in Table 14 and compared with those given by Williamson [27]. In practice all of them are about equally good when it comes to real applications, although we found the first one in Table 14 ('symmetric') marginally better in some simple test problems where an analytic solution was known. In Ref. [3] the accuracy of some non-linear equations is tested.

Table 14: Coefficients for different $2N$-type third-order Runge-Kutta schemes. The coefficients c_i (which are determined by the α_i, β_i) give the time for each substep, $t_i = t_0 + c_i \delta t$

scheme	c_1	c_2	c_3	α_2	α_3	β_1	β_2	β_3
symmetric	0	$1/3$	$2/3$	$-2/3$	-1	$1/3$	1	$1/2$
[predictor/corrector]	0	$1/2$	1	$-1/4$	$-4/3$	$1/2$	$2/3$	$1/2$
inhomogeneous	0	$15/32$	$4/9$	$-17/32$	$-32/27$	$1/4$	$8/9$	$3/4$
Williamson (1980)	0	$4/9$	$15/32$	$-5/9$	$-153/128$	$1/3$	$15/16$	$8/15$

H.5 Ionization

The specific entropy of each particle species (neutral hydrogen, electrons, protons and neutral helium) may be written as

$$\frac{s_i}{s_0} = x_i \left(\ln \left[\frac{1}{x_{\rm tot}} \frac{\rho_i}{\rho} \left(\frac{T}{T_0} \right)^{3/2} \right] + \frac{5}{2} \right) \ , \tag{222}$$

where

$$x_{\rm H} = 1 - y \ , \quad x_{\rm e} = x_{\rm p} = y \ , \quad x_{\rm tot} = 1 + y + x_{\rm He} \tag{223}$$

$$s_0 = \frac{k_{\rm B}}{\mu m_{\rm H}} \ , \quad T_0 = \frac{\chi_{\rm H}}{k_{\rm B}} \ , \tag{224}$$

and

$$\rho_i = \mu m_{\rm H} \left(\frac{m_i \chi_{\rm H}}{2\pi\hbar^2} \right)^{3/2} \ . \tag{225}$$

The specific entropy of mixing is

$$\frac{s_{\rm mix}}{s_0} = -\sum_i x_i \ln \frac{x_i}{x_{\rm tot}} \ . \tag{226}$$

Summing up everything, we get the total specific entropy

$$\frac{s}{s_0} = \sum_i \frac{s_i}{s_0} + \frac{s_{\rm mix}}{s_0} = \sum_i x_i \left(\ln \left[\frac{1}{x_i} \frac{\rho_i}{\rho} \left(\frac{T}{T_0} \right)^{3/2} \right] + \frac{5}{2} \right) \tag{227}$$

$$= \sum_i x_i \ln \frac{\rho_i}{x_i} + x_{\rm tot} \left(\ln \left[\frac{1}{\rho} \left(\frac{T}{T_0} \right)^{3/2} \right] + \frac{5}{2} \right) \ . \tag{228}$$

Solving for T gives

$$\frac{3}{2} \ln \frac{T}{T_0} = \frac{s/s_0 + \sum_i x_i \ln x_i/\rho_i}{x_{\rm tot}} + \ln \rho - \frac{5}{2} \ . \tag{229}$$

Using this expression and the constants defined above, we may obtain the ionization fraction y for given $\ln \rho$ and s by finding the root of

$$F = \ln \left[\frac{\rho_{\rm e}}{\rho} \left(\frac{T}{T_0} \right)^{3/2} \frac{1-y}{y^2} \right] - \frac{T_0}{T} \ . \tag{230}$$

The derivative with respect to y for Newton-Raphson is

$$\frac{\partial F}{\partial y} = \left(\frac{3}{2} + \frac{T_0}{T} \right) \frac{\partial \ln T/T_0}{\partial y} - \frac{1}{1-y} - \frac{2}{y} \ , \tag{231}$$

where

$$\frac{\partial \ln T/T_0}{\partial y} = \frac{\frac{2}{3} \left(\ln \rho_{\rm H}/\rho_{\rm p} - F - T_0/T \right) - 1}{1 + y + x_{\rm He}} \ . \tag{232}$$

In order to compute the pressure gradient in the momentum equation, the derivative of y with respect to $\ln \rho$ and s needs to be known:

$$\frac{\partial \ln P}{\partial \ln \rho} = \frac{1}{1 + y + x_{\rm He}} \frac{\partial y}{\partial \ln \rho} + \frac{\partial \ln T}{\partial \ln \rho} + \frac{\partial \ln T}{\partial y} \frac{\partial y}{\partial \ln \rho} + 1, \tag{233}$$

$$\frac{\partial \ln P}{\partial s} = \frac{1}{1+y+x_{\rm He}} \frac{\partial y}{\partial s} + \frac{\partial \ln T}{\partial s} + \frac{\partial \ln T}{\partial y} \frac{\partial y}{\partial s}. \tag{234}$$

Since $F = 0$ for all desired solutions $(y, \ln \rho, s)$ we also have

$$dF = \frac{\partial F}{\partial \ln \rho} \mathrm{d} \ln \rho + \frac{\partial F}{\partial s} \mathrm{d}s + \frac{\partial F}{\partial y} \mathrm{d}y = 0 \ , \tag{235}$$

and thus

$$\frac{\partial y}{\partial \ln \rho} = \left(\frac{\mathrm{d}y}{\mathrm{d} \ln \rho} \right)_{\mathrm{d}s=0} = -\frac{\partial F / \partial \ln \rho}{\partial F / \partial y} \tag{236}$$

and

$$\frac{\partial y}{\partial s} = \left(\frac{\mathrm{d}y}{\mathrm{d}s} \right)_{\mathrm{d} \ln \rho=0} = -\frac{\partial F / \partial s}{\partial F / \partial y}. \tag{237}$$

H.6 Radiative transfer

H.6.1 Solving the radiative transfer equation

A formal solution of Eq. (76) is given by

$$I(\tau) = I(\tau_0) e^{-(\tau - \tau_0)} + \int_{\tau_0}^{\tau} e^{-(\tau - \tau')} S(\tau') \, d\tau' \ . \tag{238}$$

Using a generalization of the trapezoidal rule,

$$\int_{\tau_0}^{\tau} e^{-(\tau - \tau')} f(\tau') \, d\tau' \approx \int_{\tau_0}^{\tau} e^{-(\tau - \tau')} \left[f(\tau_0) + \frac{f(\tau) - f(\tau_0)}{\tau - \tau_0} (\tau' - \tau_0) \right] d\tau' \tag{239}$$

$$= \left[1 - e^{-(\tau - \tau_0)} \right] f(\tau) - \frac{1 - e^{-(\tau - \tau_0)} (1 + \tau - \tau_0)}{\tau - \tau_0} [f(\tau) - f(\tau_0)] \ , \tag{240}$$

which is exact for linear functions $S(\tau)$, we discretize this as

$$I_{k+1} = I_k e^{-\delta\tau} + (1 - e^{-\delta\tau}) S_{k+1} - \frac{1 - e^{-\delta\tau} (1 + \delta\tau)}{\delta\tau} (S_{k+1} - S_k) \ , \tag{241}$$

$$= I_k e^{-\delta\tau} + (1 - e^{-\delta\tau}) S_k + \frac{e^{-\delta\tau} - 1 + \delta\tau}{\delta\tau} (S_{k+1} - S_k) \ . \tag{242}$$

Here the simplest way to calculate $\delta\tau$ is

$$\delta\tau = \frac{\chi_k + \chi_{k+1}}{2} \, \delta x \ ; \tag{243}$$

more accurate alternatives are

$$\delta\tau = \sqrt{\chi_k \chi_{k+1}} \, \delta x \tag{244}$$

or

$$\delta\tau = \frac{\chi_{k+1} - \chi_k}{\ln \frac{\chi_{k+1}}{\ln \chi_k}} \, \delta x \tag{245}$$

Table 15: Sums $\sqrt{4\pi}Y_l^m(\theta_i,\phi_i)$ for special sets of directions. For all degrees and orders up to $l = 8$ not mentioned in this table, the sums are 0. The label 'Non-h. f-d.' stands for 'non-horizontal face-diagonals', i.e. the eight face diagonals that are not in the horizontal plane.

Directions	Y_0^0	Y_2^0	Y_4^0	$Y_4^{\pm4}$	Y_6^0	$Y_6^{\pm4}$	Y_8^0	$Y_8^{\pm4}$	$Y_8^{\pm8}$
Coord.	6	0	$\frac{21}{2}$	$\frac{3}{4}\sqrt{70}$	$\frac{3}{4}\sqrt{13}$	$-\frac{3}{8}\sqrt{182}$	$\frac{99}{32}\sqrt{17}$	$\frac{3}{32}\sqrt{2618}$	$\frac{3}{64}\sqrt{24310}$
Face diag.	12	0	$-\frac{21}{4}$	$-\frac{3}{8}\sqrt{70}$	$-\frac{39}{16}\sqrt{13}$	$\frac{39}{32}\sqrt{182}$	$\frac{891}{256}\sqrt{17}$	$\frac{27}{256}\sqrt{2618}$	$\frac{27}{512}\sqrt{24310}$
Space diag.	8	0	$-\frac{28}{3}$	$-\frac{2}{3}\sqrt{70}$	$\frac{16}{9}\sqrt{13}$	$-\frac{8}{9}\sqrt{182}$	$\frac{11}{9}\sqrt{17}$	$\frac{1}{27}\sqrt{2618}$	$\frac{1}{54}\sqrt{24310}$
Non-h. f-d.	8	$2\sqrt{5}$	$-\frac{39}{4}$	$\frac{3}{8}\sqrt{70}$	$-\frac{19}{16}\sqrt{13}$	$\frac{27}{32}\sqrt{182}$	$\frac{611}{256}\sqrt{17}$	$\frac{51}{256}\sqrt{2618}$	$\frac{3}{512}\sqrt{24310}$
Coord. x, y	4	$-2\sqrt{5}$	$\frac{9}{2}$	$\frac{4}{3}\sqrt{70}$	$-\frac{5}{4}\sqrt{13}$	$-\frac{3}{8}\sqrt{182}$	$\frac{35}{32}\sqrt{17}$	$\frac{3}{32}\sqrt{2618}$	$\frac{3}{64}\sqrt{24310}$
Coord. z	2	$2\sqrt{5}$	6	0	$2\sqrt{13}$	0	$2\sqrt{17}$	0	0

H.6.2 Angular integration

For angular integration over the full solid angle, we make the ansatz

$$\int_{4\pi} f(\theta,\phi)\,\frac{d\omega}{4\pi} = \sum_{i=1}^{N} w_i f(\theta_i,\phi_i) + R_N\ . \tag{246}$$

Table 15 shows the sums $\sqrt{4\pi}Y_l^m(\theta_i,\phi_i)$ over special sets of directions (θ_i,ϕ_i). Using these numbers and requiring that angular integration is exact for $l \le l_{\max}$, we find the following weights w_i for different sets of directions (see also [1], §25.4.65):

1. *Axes*

   Coordinate axes: 1/6

   $l_{\max} = 3$

2. *Face diagonals*

   Face diagonals: 1/12

   $l_{\max} = 3$

3. *Space diagonals*

   Space diagonals: 1/8

   $l_{\max} = 3$

4. *Axes + face diagonals*

   Coordinate axes: 1/30
   Face diagonals: 1/15

   $l_{\max} = 5$

5. *Axes + space diagonals*

Coordinate axes: 1/15
Space diagonals: 3/40

$l_{\max} = 5$

6. *Face + space diagonals*

   Face diagonals: 2/15
   Space diagonals: -3/40

   $l_{\max} = 5$

7. *Axes, face + space diagonals*

   Coordinate axes: 1/21
   Face diagonals: 4/105
   Space diagonals: 9/280

   $l_{\max} = 7$

8. *Axes, non-horizontal face diagonals*

   Coordinate axes x, y: 1/10
   Coordinate axes z: 1/30
   Non-hor. face diagonals: 1/15

   $l_{\max} = 3$

9. *Axes, non-horizontal face diagonals, space diagonals*

   Coordinate axes x, y: 12/215
   Coordinate axes z: 10/129
   Non-hor. face diagonals: -14/645
   Space diagonals: 171/1720

   $l_{\max} = 5$

I Switchable modules

Module	*Description*
hydro.f90	This module takes care of most of the things related to velocity. Pressure, for example, is added in the energy (entropy) module.
noentropy.f90	Calculates pressure gradient term for polytropic equation of state $p = \text{const}\rho^\Gamma$.
nohydro.f90	no variable $\boldsymbol{u}$: useful for kinematic dynamo runs.
nopower_spectrum.f90	reads in full snapshot and calculates power spetrum of u
power_spectrum.f90	reads in full snapshot and calculates power spetrum of u
timestep.f90	Runge-Kutta time advance, accurate to order itorder. At the moment, itorder can be 1, 2, or 3.
timestep_subcycle.f90	This is a highly specified timestep module currently only working together with the special module coronae.f90.

J Startup and run-time parameters

J.1 List of startup parameters for '`start.in`'

The following table lists all (at the time of writing, September 2002) namelists used in '`start.in`', with the corresponding parameters and their default values (in square brackets). Any variable referred to as a *flag* can be set to any nonzero value to switch the corresponding feature on. Not all parameters are used for a given scenario. This list is not necessarily up to date; also, in many cases it can only give an idea of the corresponding initial state; to get more insight and the latest set of parameters, you need to look at the code.

The value ε corresponds to 5 times the smallest number larger than zero. For single precision, this is typically about $\varepsilon \approx 5 \times 1.2{\times}10^{-7} = 6{\times}10^{-7}$; for double precision, $\varepsilon \approx 10^{-15}$.

Variable [default value]	*Meaning*
	Namelist *init_pars*
cvsid [' ']	the *svn* identification string, which allows you to keep track of the version of '`start.in`'.
ip [14]	(anti-)verbosity level: `ip=1` produces lots of diagnostic output, `ip=14` virtually none.
xyz0 $[(-\pi,-\pi,-\pi)]$, *Lxyz* $[(2\pi,2\pi,2\pi)]$, *lperi* [(T,T,T)]	determine the geometry of the box. All three are vectors of the form (x-comp., y-comp., z-comp.); *xyz0* describes the left (lower) corner of the box, *Lxyz* the box size. *lperi* specifies whether a direction is considered periodic (in which case the last point is omitted) or not. In all cases, three ghost zones will be added.

lprocz_slowest [T] — if set to F, the ordering of processor numbers is changed, so the z processors are now in the inner loop. Since *nprocy*=4 is optimal (see Sect. 5.20.2), you may want to put *lprocz_slowest*=T when nygrid>nzgrid.

lwrite_ic [F] — if set T, the initial data are written into the file 'VAR0'. This is generally useful, but doing this all the time uses up plenty of disk space.

lnowrite [F] — if set T, all initialization files are written, including the param.nml file, except 'var.dat'. This option allows you to use old filevar.dat files, but updates all other initialization files. This could be useful after having changed the code and, in particular, when the 'var.dat' files will be overwritten by 'remesh.csh'.

lwrite_aux [F] — if set T, auxiliary variables (those calculated at each step, but not evolved mathematically) to 'var.dat' after the evolved quantities.

lwrite_2d [F] — if set T, only 2D-snapshots are written into VAR files in the case of 2D-runs with $nygrid = 1$ or $nzgrid = 1$.

lread_oldsnap [F] — if set T, the old snapshot will be read in before producing (overwriting) initial conditions. For example, if you just want to add a perturbation to the magnetic field, you'd give no initial condition for density and velocity (so you keep the data from a hopefully relaxed run), and just add whatever you need for the magnetic field. In this connection you may want to touch NOERASE, so as not to erase the previous data.

lread_oldsnap_nomag [F] — if set T, the old snapshot from a non-magnetic run will be read in before producing (overwriting) initial conditions. This allows one to let a hydrodynamic run relax before adding a magnetic field. However, for this to work one has to modify *manually* 'data/param.nml' by adding an entry for *MAGNETIC_INIT_PARS* or *PSCALAR_INIT_PARS*. In addition, for idl to read correctly after the first restarted run, you must adjust the value of *mvar* in 'data/dim.dat'

lread_oldsnap_nopscalar [F] — if set T, the old snapshot from a run without passive scalar will be read in before producing (overwriting) initial conditions. This allows one to let a hydrodynamic run relax before adding a passive scalar.

lshift_origin [F,F,F] — if set T for any or some of the three directions, the mesh is shifted by 1/2 meshpoint in that or those directions so that the mesh goes through the origin.

unit_system ['cgs'] — you can set this character string to 'SI', which means that you can give physical dimensions in SI units. The default is cgs units.

unit_length [1] allows you to set the unit length. Suppose you want the unit length to be $1\,\mathrm{kpc}$, then you would say `unit_length='3e21'`. (Of course, politically correct would be to say `unit_system='SI'` in which case you say `unit_length='3e19'`.)

unit_velocity [1] Example: if you want km/s you say `unit_length='1e5'`.

unit_density [1] Example: if you want your unit density to be $10^{-24}\,\mathrm{g/cm^3}$ you say `unit_density='1e-24'`.

unit_temperature [1] Example: `unit_temperature='1e6'` if you want mega-Kelvin.

random_gen [`system`] choose random number generator; currently valid choices are
`'system'` (your compiler's generator),
`'min_std'` (the 'minimal standard' generator `ran0()` from 'Numerical Recipes'),
`'nr_f90'` (the Parker-Miller-Marsaglia generator `ran()` from 'Numerical Recipes for F90').

bcx [`('p', 'p', ...)`],
bcy [`('p', 'p', ...)`],
bcz [`('p', 'p', ...)`] boundary conditions. See Sect. 5.16 for a discussion of where and how to set these.

pretend_lnTT [`F`] selects $\ln T$ as fundamental thermodynamic variable in the entropy module

Namelist *hydro_init_pars*

inituu [`'zero'`] initialization of velocity. Currently valid choices are
`'zero'` ($\boldsymbol{u} = 0$),
`'gaussian-noise'` (random, normally-distributed u_x,u_z),
`'gaussian-noise-x'` (random, normally-distributed u_x),
`'sound-wave'` (sound wave in x direction),
`'shock-tube'` (polytropic standing shock),
`'bullets'` (blob-like velocity perturbations),
`'Alfven-circ-x'` (circularly polarized Alfven wave in x direction),
`'const-ux'` (constant x-velocity),
`'const-uy'` (constant y-velocity),
`'tang-discont-z'` (tangential discontinuity: velocity is directed along x, jump is at $z = 0$),
`'Fourier-trunc'` (truncated Fourier series),
`'up-down'` (flow upward in one spot, downward in another; not solenoidal).

ampluu [0.] amplitude for some types of initial velocities.

widthuu [0.1] width for some types of initial velocities.

urand [0.]	additional random perturbation of $\boldsymbol{u}$. If `urand>0`, the perturbation is additive, $u_i \mapsto u_i + u_{\rm rand}\mathcal{U}_{[0.5,0.5]}$; if `urand<0`, it is multiplicative, $u_i \mapsto u_i \times u_{\rm rand}\mathcal{U}_{[0.5,0.5]}$; in both cases, $\mathcal{U}_{[0.5,0.5]}$ is a uniformly distributed random variable on the interval $[-0.5, 0.5]$.
uu_left [0.], *uu_right* [0.]	needed for `inituu='shock-tube'`.

Namelist *density_init_pars*

initlnrho [`'zero'`]	initialization of density. Currently valid choices are `'zero'` ($\ln\rho = 0$), `'isothermal'` (isothermal stratification), `'polytropic_simple'` (polytropic stratification), `'hydrostatic-z-2'` (hydrostatic vertical stratification for isentropic atmosphere), `'xjump'` (density jump in x of width *widthlnrho*), `'rho-jump-z'` (density jump in z of width *widthlnrho*), `'piecew-poly'` (piecewise polytropic vertical stratification for solar convection), `'polytropic'` (polytropic vertical stratification), `'sound-wave'` (sound wave), `'shock-tube'` (polytropic standing shock), `'gaussian-noise'` (Gaussian-distributed, uncorrelated noise), `'gaussian-noise'` (Gaussian-distributed, uncorrelated noise in x, but uniform in y and z), `'hydrostatic-r'` (hydrostatic radial density stratification for isentropic or isothermal sphere), `'sin-xy'` (sine profile in x and y), `'sin-xy-rho'` (sine profile in x and y, but in ρ, not $\ln\rho$), `'linear'` (linear profile in $\boldsymbol{k}\cdot\boldsymbol{x}$), `'planet'` (planet solution; see §C.6).
gamma [5./3]	adiabatic index $\gamma = c_p/c_v$.
cs0 [1.]	can be used to set the dimension of velocity; larger values can be used to decrease stratification
rho0 [1.]	reference values of sound speed and density, i. e. values at height *zref*.
ampllnrho [0.], *widthlnrho* [0.1]	amplitude and width for some types of initial densities.
rho_left [1.], *rho_right* [1.]	needed for `initlnrho='shock-tube'`.
cs2bot [1.], *cs2top* [1.]	sound speed at bottom and top. Needed for some types of stratification.

Namelist *grav_init_pars*

zref [0.]	reference height where in the initial stratification $c_s^2 = c_{s0}^2$ and $\ln\rho = \ln\rho_0$.
gravz [−1.]	vertical gravity component g_z.
grav_profile ['const']	constant gravity $g_z =$ gravz (grav_profile='const') gravity or linear profile $g_z =$ gravz $\cdot\, z$ (grav_profile='linear', for accretion discs and similar).
z1 [0.], *z2* [1.]	specific to the solar convection case initlnrho='piecew-poly'. The stable layer is $z_0 < z < z_1$, the unstable layer $z_1 < z < z_2$, and the top (isothermal) layer is $z_2 < z < z_{\rm top}$.
nu_epicycle [1.]	vertical epicyclic frequency; for accretion discs it should be equal to Omega, but not for galactic discs; see Eq. (107) in Sect. C.4.
grav_amp [0.], *grav_tilt* [0.]	specific to the tilted gravity case (amplitude and angle wrt the vertical direction).

Namelist *entropy_init_pars*

initss ['nothing']	initialization of entropy. Currently valid choices are 'nothing' (leaves the initialization done in the density module unchanged), 'zero' (put $s = 0$ explicitly; this may overwrite the initialization done in the density module), 'isothermal' (isothermal stratification, $T =$ const), 'isobaric' (isobaric, $p =$ const), 'isentropic' (isentropic with superimposed hot [or cool] bubble), 'linprof' (linear entropy profile in z), 'piecew-poly' (piecewise polytropic stratification for convection), 'polytropic' (polytropic stratification, polytropic exponent is *mpoly0*), 'blob' (puts a gaussian blob in entropy for buoyancy experiments; see Ref. [5] for details) 'xjump' (jump in x direction), 'hor-tube' (horizontal flux tube in entropy, oriented in the y-direction).
pertss ['zero']	additional perturbation to entropy. Currently valid choices are 'zero' (no perturbation) 'hexagonal' (hexagonal perturbation for convection).
ampl_ss [0.], *widthss* [2ε]	amplitude and width for some types of initial entropy.
grads0 [0.]	initial entropy gradient for initss=linprof.
radius_ss [0.1]	radius of bubble for initss=isentropic.

mpoly0 [1.5], *mpoly1* [1.5], *mpoly2* [1.5]	specific to the solar convection case `initss=piecew-poly`: polytropic indices of unstable (*mpoly0*), stable (*mpoly1*) and top layer (*mpoly2*). If the flag *isothtop* is set, the top layer is initialized to be isothermal, otherwise thermal (plus hydrostatic) equilibrium is assumed for all three layers, which results in a piecewise polytropic stratification.
isothtop [0]	flag for isothermal top layer for `initss=piecew-poly`.
khor_ss [1.]	horizontal wave number for `pertss=hexagonal`

Namelist *magnetic_init_pars*

initaa [`'zero'`]	initialization of magnetic field (vector potential). Currently valid choices are `'Alfven-x'` (Alfvén wave traveling in the x-direction; this also sets the velocity), `'Alfven-z'` (Alfvén wave traveling in the z-direction; this also sets the velocity), `'Alfvenz-rot'` (same as `'Alfven-z'`, but with rotation), `'Alfven-circ-x'` (circularly polarized Alfven wave in x direction), `'Beltrami-x'` (x-dependent Beltrami wave), `'Beltrami-y'` (y-dependent Beltrami wave), `'Beltrami-z'` (z-dependent Beltrami wave), `'Bz(x)'` ($B_z \propto \cos(kx)$), `'crazy'` (for testing purposes). `'diffrot'` ([needs to be documented]), `'fluxrings'` (two interlocked magnetic fluxrings; see § C.3), `'gaussian-noise'` (white noise), `'halfcos-Bx'` ([needs to be documented]), `'hor-tube'` (horizontal flux tube in $\boldsymbol{B}$, oriented in the y-direction). `'hor-fluxlayer'` (horizontal flux layer), `'mag-support'` ([needs to be documented]), `'mode'` ([needs to be documented]), `'modeb'` ([needs to be documented]), `'propto-ux'` ([needs to be documented]), `'propto-uy'` ([needs to be documented]), `'propto-uz'` ([needs to be documented]), `'sinxsinz'` ($\sin x \sin z$), `'uniform-Bx'` (uniform field in x direction), `'uniform-By'` (uniform field in y direction), `'uniform-Bz'` (uniform field in z direction), `'zero'` (zero field),

initaa2 ['zero']	additional perturbation of magnetic field. Currently valid choices are 'zero' (zero perturbation), 'Beltrami-x' (x-dependent Beltrami wave), 'Beltrami-y' (y-dependent Beltrami wave), 'Beltrami-z' (z-dependent Beltrami wave).
amplaa [0.]	amplitude for some types of initial magnetic fields.
amplaa2 [0.]	amplitude for some types of magnetic field perturbation.
fring{*1,2*} [0.], *Iring*{*1,2*} [0.], *Rring*{*1,2*} [1.], *wr*{*1,2*} [0.3]	flux, current, outer and inner radius of flux ring 1/2; see Sect. C.3.
radius [0.1]	used by some initial fields.
epsilonaa [10^{-2}]	used by some initial fields.
widthaa [0.5]	used by some initial fields.
z0aa [0.]	used by some initial fields.
kx_aa [1.], *ky_aa* [1.], *kz_aa* [1.]	wavenumbers used by some initial fields.
lpress_equil [F]	flag for pressure equilibrium (can be used in connection with all initial fields)

Namelist *pscalar_init_pars*

initlncc ['zero']	initialization of passive scalar (concentration per unit mass, c). Currently valid choices (for $\ln c$) are 'zero' ($\ln c = 0.$), 'gaussian-noise' (white noise), 'wave-x' (wave in x direction), 'wave-y' (wave in y direction), 'wave-z' (wave in z direction), 'tang-discont-z' (Kelvin-Helmholtz instability), 'hor-tube' (horizontal tube in concentration; used as a marker for magnetic flux tubes).
initlncc2 ['zero']	additional perturbation of passive scalar concentration c. Currently valid choices are 'zero' ($\delta \ln c = 0.$), 'wave-x' (add x-directed wave to $\ln c$).
ampllncc [0.1]	amplitude for some types of initial concentration.
ampllncc2 [0.]	amplitude for some types of concentration perturbation.
kx_lncc [1.], *ky_lncc* [1.], *kz_lncc* [1.]	wave numbers for some types of initial concentration.

Namelist *shear_init_pars*

qshear [0.]	degree of shear for shearing-box simulations (the shearing-periodic boundaries are the x-boundaries and are sheared in the y-direction). The shear velocity is $\mathbf{U} = -q\Omega x\,\hat{\mathbf{y}}$.

J.2 List of runtime parameters for 'run.in'

The following table lists all (at the time of writing, September 2002) namelists used in file 'run.in', with the corresponding parameters and their default values (in square brackets). Default values marked as [start] are taken from 'start.in'. Any variable referred to as a *flag* can be set to any nonzero value to switch the corresponding feature on. Not all parameters are used for a given scenario. This list is not necessarily up to date; also, in many cases it can only give an idea of the corresponding setup; to get more insight and the latest set of parameters, you need to look at the code.

Variable [default value]	*Meaning*
	Namelist *run_pars*
cvsid [' ']	*svn* identification string, which allows you to keep track of the version of 'run.in'.
ip [14]	(anti-)verbosity level: ip=1 produces lots of additional diagnostic output, ip=14 virtually none.
nt [0]	number of time steps to run. This number can be increased or decreased during the run by touch RELOAD.
it1 [10]	write diagnostic output every *it1* time steps (see Sect. 5.5).
it1d [$it1$]	write averages every *it1d* time steps (see Sect. 5.8.1). *it1d* has to be greater than or equal to *it1*.
cdt [0.4]	Courant coefficient for advective time step; see §5.15.
cdtv [0.08]	Courant coefficient for diffusive time step; see §5.15.
dt [0.]	time step; if $\neq 0.$, this overwrites the Courant time step. See §5.15 for a discussion of the latter.
dtmin [10^{-6}]	abort if time step $\delta t < \delta t_{\min}$.
tmax [10^{33}]	don't run time steps beyond this time. Useful if you want to run for a given amount of time, but don't know the necessary number of time steps.
isave [100]	update current snapshot 'var.dat' every *isave* time steps.
itorder [3]	order of time step (1 for Euler; 2 for 3nd-order, 3 for 3rd-order Runge–Kutta).
dsnap [100.]	save permanent snapshot every *dsnap* time units to files 'VARN', where N counts from $N = 1$ upward. (This information is stored in the file 'data/tsnap.dat'; see the module *wsnaps.f90*, which in turn uses the subroutines *out1* and *out2*).
dvid [100.]	write two-dimensional sections for generation of videos every *dvid* time units (not timesteps; see the subroutines *out1* and *out2* in the code).

iwig [0]	if $\neq 0$, apply a Nyquist filter (a filter eliminating any signal at the Nyquist frequency, but affecting large scales as little as possible) every *iwig* time steps to logarithmic density (sometimes necessary with convection simulations).
ix [-1], *iy* [-1], *iz* [-1], *iz2* [-1]	position of slice planes for video files. Any negative value of some of these variables will be overwritten according to the value of *slice_position*. See § 5.7) for details.
slice_position ['p']	symbolic specification of slice position. Currently valid choices are 'p' (*periphery* of the box) 'm' (*middle* of the box) 'e' (*equator* for half-sphere calculations, i. e. x, y centered, z bottom) These settings are overridden by explicitly setting *ix*, *iy*, *iz* or *iz2*. See § 5.7) for details.
tavg [0]	averaging time $\tau_{\rm avg}$ for time averages (if $\neq 0$); at the same time, time interval for writing time averages. See § 5.8.4 for details.
idx_tavg [$(0, 0, \ldots, 0)$]	indices of variables to time-average. See § 5.8.4 for details.
d2davg [100.]	time interval for azimuthal and z-averages, i.e. the averages that produce 2d data. See § 5.8.3 for details.
ialive [0]	if $\neq 0$, each processor writes the current time step to 'alive.info' every *ialive* time steps. This provides the best test that the job is still alive. (This can be used to find out which node has crashed if there is a problem and the run is hanging.)
bcx ['p', 'p', ...], *bcy* ['p', 'p', ...], *bcz* ['p', 'p', ...]	boundary conditions. See Sect. 5.16 for a discussion of where and how to set these.
random_gen [start]	see start parameters, p. 151
lwrite_aux [start]	if set T, auxiliary variables (those calculated at each step, but not evolved mathematically) to 'var.dat' and 'VAR' files after the evolved quantities.

Namelist *hydro_run_pars*

Omega [0.]	magnitude of angular velocity for *Coriolis force* (note: the centrifugal force is turned off by default, unless lcentrifugal_force=T is set).
theta [0.]	direction of angular velocity in degrees ($\theta = 0$ for z-direction, $\theta = 90$ for the negative x-direction, corresponding to a box located at the equator of a rotating sphere. Thus, e.g., $\theta = 60$ corresponds to 30° latitude. (Note: prior to April 29, 2007, there was a minus sign in the definition of θ.)

ttransient [0.]	initial time span for which to do something special (transient). Currently just used to smoothly switch on heating [Should be in *run_pars*, rather than here].
dampu [0.],	
tdamp [0.],	
ldamp_fade [F]	damp motions during the initial time interval $0 < t < t_{\rm damp}$ with a damping term $-$*dampu*$(\boldsymbol{u})$. If *ldamp_fade* is set, smoothly reduce damping to zero over the second half of the time interval *tdamp*. Initial velocity damping is useful for situations where initial conditions are far from equilibrium.
dampuint [0.],	weighting of damping external to spherical region (see *wdamp*, $damp_u$ below).
dampuext [0.],	weighting of damping in internal spherical region (see *wdamp*, $damp_u$ below).
rdampint [0.],	radius of internal damping region
rdampext [impossible],	radius of external damping region, used in place of former variable *rdamp*
wdamp [0.2],	permanently damp motions in $\|\boldsymbol{x}\| < r_{\rm dampint}$ with damping term $-$*dampu*$int\,\boldsymbol{u}\,\chi(r-r_{\rm dampint})$ or $\|\boldsymbol{x}\| > r_{\rm dampext}$ with damping term $-$*dampu*$ext\,\boldsymbol{u}\,\chi(r-r_{\rm dampext})$, where $\chi(\cdot)$ is a smooth profile of width *wdamp*.
ampl_forc [0.],	amplitude of the ux-forcing or uy-forcing on the vertical boundaries that is of the form $u\_x(t) = ampl\_forc * sin(k\_forc * x) * cos(w\_forc * t)$ [must be used in connection with bcx='g' or bcz='g' and force_lower_bound='vel_time' or force_upper_bound='vel_time']
k_forc [0.],	corresponding horizontal wavenumber
w_forc [0.]	corresponding frequency

Namelist *density_run_pars*

cs0 [start],	
rho0 [start],	
gamma [start]	see start parameters, p. 152
cdiffrho [0.]	Coefficient for mass diffusion (diffusion term will be $c_{\rm diffrho}\,\delta x\,c_{\rm s0}$.
cs2bot [start],	
cs2top [start]	squared sound speed at bottom and top for boundary condition 'c2'.
lupw_lnrho [.false.]	use 5th-order upwind derivative operator for the advection term $\boldsymbol{u}\cdot\boldsymbol{\nabla}\ln\rho$ to avoid spurious Nyquist signal ('wiggles'); see §H.2.

Namelist *entropy_run_pars*

hcond0 [0.],	
hcond1 [start],	

hcond2 [start]
: specific to the solar convection case `initss=piecew-poly`: heat conductivities K in the individual layers. *hcond0* is the value $K_{\rm unst}$ in the unstable layer, *hcond1* is the *ratio* $K_{\rm stab}/K_{\rm unst}$ for the stable layer, and *hcond2* is the *ratio* $K_{\rm top}/K_{\rm unst}$ for the top layer. The function $K(z)$ is not discontinuous, as the transition between the different values is smoothed over the width *widthss*. If *hcond1* or *hcond2* are not set, they are calculated according to the polytropic indices of the initial profile, $K \propto m{+}1$.

iheatcond ['K-const']
: select type of heat conduction. Currently valid choices are
'K-const' (constant heat conductivity),
'K-profile' (vertical or radial profile),
'chi-const' (constant thermal diffusivity),
'magnetic' (heat conduction by electrons in magnetic field – currently still experimental).

lcalc_heatcond_constchi [F]
: flag for assuming thermal diffusivity $\chi = K/(c_p\rho) =$ const, rather than $K =$ const (which is the default). *This is currently only correct with 'noionization.f90'*. Superseded by *iheatcond*.

chi [0.]
: value of χ when `lcalc_heatcond_constchi=T`.

widthss [start]
: width of transition region between layers. See start parameters, p. 154.

isothtop [start]
: flag for isothermal top layer for solar convection case. See start parameters, p. 154.

luminosity [0.],
wheat [0.1]
: strength and width of heating region.

cooltype ['Temp']
: type of cooling; *currently only implemented for spherical geometry*. Currently valid choices are
'Temp','cs2' (cool temperature toward $c_{\rm s}^2 =$ *cs2cool*) with a cooling term

$$-\mathcal{C} = -c_{\rm cool}\frac{c_{\rm s}^2 - c_{\rm s\,cool}^2}{c_{\rm s\,cool}^2}$$

)
'Temp-rho',cs2-rho (cool temperature toward $c_{\rm s}^2 =$ *cs2cool*) with a cooling term

$$-\mathcal{C} = -c_{\rm cool}\,\rho\,\frac{c_{\rm s}^2 - c_{\rm s\,cool}^2}{c_{\rm s\,cool}^2}$$

— this avoids numerical instabilities in low-density regions [currently, the cooling coefficient $c_{\rm cool} \equiv$*cool* is not taken into account when the time step is calculated])
'entropy' (cool entropy toward 0.).

cool [0.],

wcool [0.1]	strength $c_{\rm cool}$ and smoothing width of cooling region.
rcool [1.]	radius of cooling region: cool for $\|\boldsymbol{x}\| \geq$ *rcool*.
Fbot [start]	heat flux for bottom boundary condition '`c1`'. For polytropic atmospheres, if *Fbot* is not set, it will be calculated from the value of *hcond0* in '`start.x`', provided the entropy boundary condition is set to '`c1`'.
chi_t [0.]	entropy diffusion coefficient for diffusive term $\partial s/\partial t = \ldots + \chi_{\rm t}\nabla^2 s$ in the entropy equation, that can represent some kind of turbulent (sub-grid) mixing. It is probably a bad idea to combine this with heat conduction *hcond0* $\neq 0$.
lupw_ss [.false.]	use 5th-order upwind derivative operator for the advection term $\boldsymbol{u}\cdot\boldsymbol{\nabla}s$ to avoid spurious Nyquist signal ('wiggles'); see §H.2.
tauheat_buffer [0.]	time scale for heating to target temperature (=*TTheat_buffer*); zero disables the buffer zone.
zheat_buffer [0.]	z coordinate of the thermal buffer zone. Buffering is active in $\|z\| >$*TTheat_buffer*.
dheat_buffer1 [0.]	Inverse thickness of transition to buffered layer.
TTheat_buffer [0.]	target temperature in thermal buffer zone (z direction only).
lhcond_global [F]	flag for calculating the heat conductivity K (and also $\boldsymbol{\nabla}\log K$) globally using the global arrays facility. Only valid when *iheatcond*='`K-profile`'.

Namelist *magnetic_run_pars*

B_ext [(0., 0., 0.)]	uniform background magnetic field (for fully periodic boundary conditions, uniform fields need to be explicitly added, since otherwise the vector potential $\boldsymbol{A}$ has a linear x-dependence which is incompatible with periodicity).
lignore_Bext_in_b2 [F] or *luse_Bext_in_b2* [T]	add uniform background magnetic field when computing $\boldsymbol{b}^2$ pencils
eta [0.]	magnetic diffusivity $\eta = 1/(\mu_0\sigma)$, where σ is the electric conductivity.
height_eta [0.], *eta_out* [0.]	used to add extra diffusivity in a halo region.
eta_int [0.]	used to add extra diffusivity inside sphere of radius *r_int*.
eta_ext [0.]	used to add extra diffusivity outside sphere of radius *r_ext*.
kinflow ['']	set type of flow fixed with '`nohydro`'. Currently the only recognized value is `'ABC'` for an ABC flow; all other values lead to $\boldsymbol{u} = \boldsymbol{0}$.
kx [1.], *ky* [1.], *kz* [1.]	wave numbers for ABC flow.
ABC_A [1.], *ABC_B* [1.], *ABC_C* [1.]	amplitudes A, B and C for ABC flow.

Namelist *pscalar_run_pars*	
pscalar_diff [0.]	diffusion for passive scalar concentration c.
tensor_pscalar_diff [0.]	coefficient for non-isotropic diffusion of passive scalar.

Namelist *forcing_run_pars*	
iforce [2]	select form of forcing in the equation of motion; currently valid choices are 'zero' (no forcing), 'irrotational' (irrotational forcing), 'helical' (helical forcing), 'fountain' (forcing of "fountain flow"; see Ref. [9]), 'horizontal-shear' (forcing localized horizontal sinusoidal shear). 'variable_gravz' (time-dependent vertical gravity for forcing internal waves),
iforce2 [0]	select form of additional forcing in the equation of motion; valid choices are as for *iforce*.
force [0.]	amplitude of forcing.
relhel [1.]	helicity of forcing. The parameter *relhel* corresponds to σ introduced in Sect. G.2. ($\sigma = \pm 1$ corresponds to maximum helicity of either sign).
height_ff [0.]	multiply forcing by z-dependent profile of width *height_ff* (if $\neq 0$) .
r_ff [0.]	if $\neq 0$, multiply forcing by spherical cutoff profile (of radius *r_ff*) and flip signs of helicity at equatorial plane.
width_ff [0.5]	width of vertical and radial profiles for modifying forcing.
kfountain [5]	horizontal wavenumber of the fountain flow.
fountain [1.]	amplitude of the fountain flow.
omega_ff [1.]	frequency of the cos or sin forcing [e.g. cos(omega_ff*t)].
ampl_ff [1.]	amplitude of forcing in front of cos or sin [e.g. ampl_ff*cos(omega_ff*t)].

Namelist *grav_run_pars*	
zref [start], *gravz* [start], *grav_profile* [start]	see p. 153.
nu_epicycle [start]	see Eq. (107) in Sect. C.4.

Namelist *viscosity_run_pars*	
nu [0.]	kinematic viscosity.
nu_hyper2 [0.]	kinematic hyperviscosity (with $\nabla^4 \boldsymbol{u}$).
nu_hyper3 [0.]	kinematic hyperviscosity (with $\nabla^6 \boldsymbol{u}$).
zeta [0.]	bulk viscosity.

ivisc ['nu-const']	select form of viscous term (see §6.2); currently valid choices are 'nu-const' – viscous force for ν = const, $\boldsymbol{F}_{\rm visc} = \nu(\nabla^2\boldsymbol{u} + \frac{1}{3}\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{u} + 2\mathsf{S}\cdot\boldsymbol{\nabla}\ln\rho)$ 'rho_nu-const' – viscous force for $\mu \equiv \rho\nu$ = const, $\boldsymbol{F}_{\rm visc} = (\mu/\rho)(\nabla^2\boldsymbol{u} + \frac{1}{3}\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\boldsymbol{u})$. With this option, the input parameter *nu* actually sets the value of μ/ρ_0 (*rho0*=ρ_0 is another input parameter, see pp. 152 and 158) 'simplified' – simplified viscous force $\boldsymbol{F}_{\rm visc} = \nu\nabla^2\boldsymbol{u}$

Namelist *shear_run_pars*	
qshear [start]	See p. 156.

J.3 List of parameters for 'print.in'

The following table lists all possible inputs to the file 'print.in' that are documented.

Variable	*Meaning*
	Module 'cdata.f90'
it	number of time step (since beginning of job only)
t	time t (since start.csh)
dt	time step δt
walltime	wall clock time since start of run.x, in seconds
Rmesh	$R_{\rm mesh}$
Rmesh3	$R^{(3)}_{\rm mesh}$
maxadvec	maxadvec
	Module 'hydro.f90'
u2tm	$\left\langle \boldsymbol{u}(t)\cdot\int_0^t \boldsymbol{u}(t')dt' \right\rangle$
uotm	$\left\langle \boldsymbol{u}(t)\cdot\int_0^t \boldsymbol{\omega}(t')dt' \right\rangle$
outm	$\left\langle \boldsymbol{\omega}(t)\cdot\int_0^t \boldsymbol{u}(t')dt' \right\rangle$
u2m	$\langle \boldsymbol{u}^2 \rangle$
uxpt	$u_x(x_1, y_1, z_1, t)$
uypt	$u_x(x_1, y_1, z_1, t)$
uzpt	$u_x(x_1, y_1, z_1, t)$
uxp2	$u_x(x_2, y_2, z_2, t)$
uyp2	$u_x(x_2, y_2, z_2, t)$
uzp2	$u_x(x_2, y_2, z_2, t)$
urms	$\langle \boldsymbol{u}^2 \rangle^{1/2}$
urmsx	$\langle \boldsymbol{u}^2 \rangle^{1/2}$ for the hydro_xaver_range
urmsz	$\langle \boldsymbol{u}^2 \rangle^{1/2}$ for the hydro_zaver_range
durms	$\langle \delta\boldsymbol{u}^2 \rangle^{1/2}$
umax	$\max(\lvert\boldsymbol{u}\rvert)$
uxrms	$\langle u_x^2 \rangle^{1/2}$

uzrms	$\langle u_z^2 \rangle^{1/2}$
uxmin	$\min(\lvert u_x \rvert)$
uymin	$\min(\lvert u_y \rvert)$
uzmin	$\min(\lvert u_z \rvert)$
uxmax	$\max(\lvert u_x \rvert)$
uymax	$\max(\lvert u_y \rvert)$
uzmax	$\max(\lvert u_z \rvert)$
uxm	$\langle u_x \rangle$
uym	$\langle u_y \rangle$
uzm	$\langle u_z \rangle$
ux2m	$\langle u_x^2 \rangle$
uy2m	$\langle u_y^2 \rangle$
uz2m	$\langle u_z^2 \rangle$
ux2ccm	$\langle u_x^2 \cos^2 kz \rangle$
ux2ssm	$\langle u_x^2 \sin^2 kz \rangle$
uy2ccm	$\langle u_y^2 \cos^2 kz \rangle$
uy2ssm	$\langle u_y^2 \sin^2 kz \rangle$
uxuycsm	$\langle u_x u_y \cos kz \sin kz \rangle$
uxuym	$\langle u_x u_y \rangle$
uxuzm	$\langle u_x u_z \rangle$
uyuzm	$\langle u_y u_z \rangle$
umx	$\langle u_x \rangle$
umy	$\langle u_y \rangle$
umz	$\langle u_z \rangle$
omumz	$\left\langle \langle \boldsymbol{W} \rangle_{xy} \cdot \langle \boldsymbol{U} \rangle_{xy} \right\rangle$ (xy-averaged mean cross helicity production)
umamz	$\left\langle \langle \boldsymbol{u} \rangle_{xy} \cdot \langle \boldsymbol{A} \rangle_{xy} \right\rangle$
umbmz	$\left\langle \langle \boldsymbol{U} \rangle_{xy} \cdot \langle \boldsymbol{B} \rangle_{xy} \right\rangle$ (xy-averaged mean cross helicity production)
umxbmz	$\left\langle \langle \boldsymbol{U} \rangle_{xy} \times \langle \boldsymbol{B} \rangle_{xy} \right\rangle_z$ (xy-averaged mean emf)
rux2m	$\langle \rho u_x^2 \rangle$
ruy2m	$\langle \rho u_y^2 \rangle$
ruz2m	$\langle \rho u_z^2 \rangle$
divum	$\langle \mathrm{div} \boldsymbol{u} \rangle$
rdivum	$\langle \varrho \mathrm{div} \boldsymbol{u} \rangle$
divu2m	$\langle (\mathrm{div} \boldsymbol{u})^2 \rangle$
gdivu2m	$\langle (\mathrm{grad}\, \mathrm{div} \boldsymbol{u})^2 \rangle$
u3u21m	$\langle u_3 u_{2,1} \rangle$
u1u32m	$\langle u_1 u_{3,2} \rangle$
u2u13m	$\langle u_2 u_{1,3} \rangle$
u2u31m	$\langle u_2 u_{3,1} \rangle$
u3u12m	$\langle u_3 u_{1,2} \rangle$
u1u23m	$\langle u_1 u_{2,3} \rangle$
ruxm	$\langle \varrho u_x \rangle$ (mean x-momentum density)
ruym	$\langle \varrho u_y \rangle$ (mean y-momentum density)
ruzm	$\langle \varrho u_z \rangle$ (mean z-momentum density)
ruxtot	$\langle \rho \lvert u \rvert \rangle$ (mean absolute x-momentum density)
rumax	$\max(\varrho \lvert \boldsymbol{u} \rvert)$ (maximum modulus of momentum)

ruxuym	$\langle \varrho u_x u_y \rangle$ (mean Reynolds stress)
ruxuzm	$\langle \varrho u_x u_z \rangle$ (mean Reynolds stress)
ruyuzm	$\langle \varrho u_y u_z \rangle$ (mean Reynolds stress)
divrhourms	$\left\vert \nabla \cdot (\varrho \boldsymbol{u}) \right\vert_{\rm rms}$
divrhoumax	$\left\vert \nabla \cdot (\varrho \boldsymbol{u}) \right\vert_{\max}$
rlxm	$\langle \rho y u_z - z u_y \rangle$
rlym	$\langle \rho z u_x - x u_z \rangle$
rlzm	$\langle \rho x u_y - y u_x \rangle$
rlx2m	$\langle (\rho y u_z - z u_y)^2 \rangle$
rly2m	$\langle (\rho z u_x - x u_z)^2 \rangle$
rlz2m	$\langle (\rho x u_y - y u_x)^2 \rangle$
tot_ang_mom	Total angular momentum in spherical coordinates about the axis.
dtu	$\delta t / [c_{\delta t}\, \delta x / \max \vert \mathbf{u} \vert]$ (time step relative to advective time step; see § 5.15)
oum	$\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle$
ou_int	$\int_V \boldsymbol{\omega} \cdot \boldsymbol{u}\, dV$
fum	$\langle \boldsymbol{f} \cdot \boldsymbol{u} \rangle$
odel2um	$\langle \boldsymbol{\omega} \nabla^2 \boldsymbol{u} \rangle$
o2m	$\langle \boldsymbol{\omega}^2 \rangle \equiv \langle (\nabla \times \boldsymbol{u})^2 \rangle$
orms	$\langle \boldsymbol{\omega}^2 \rangle^{1/2}$
omax	$\max(\vert \boldsymbol{\omega} \vert)$
ox2m	$\langle \omega_x^2 \rangle$
oy2m	$\langle \omega_y^2 \rangle$
oz2m	$\langle \omega_z^2 \rangle$
oxoym	$\langle \omega_x \omega_y \rangle$
oxozm	$\langle \omega_x \omega_z \rangle$
oyozm	$\langle \omega_y \omega_z \rangle$
qfm	$\langle \boldsymbol{q} \cdot \boldsymbol{f} \rangle$
q2m	$\langle \boldsymbol{q}^2 \rangle$
qrms	$\langle \boldsymbol{q}^2 \rangle^{1/2}$
qmax	$\max(\vert \boldsymbol{q} \vert)$
qom	$\langle \boldsymbol{q} \cdot \boldsymbol{\omega} \rangle$
quxom	$\langle \boldsymbol{q} \cdot (\boldsymbol{u} \times \boldsymbol{\omega}) \rangle$
pvzm	$\langle \omega_z + 2\Omega / \varrho \rangle$ (z component of potential vorticity)
oumphi	$\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle_\varphi$
ugurmsx	$\langle (\boldsymbol{u} \nabla \boldsymbol{u})^2 \rangle^{1/2}$ for the hydro_xaver_range
Marms	$\langle \boldsymbol{u}^2 / c_{\rm s}^2 \rangle$ (rms Mach number)
Mamax	$\max \vert \boldsymbol{u} \vert / c_{\rm s}$ (maximum Mach number)
ekin	$\langle \frac{1}{2} \varrho \boldsymbol{u}^2 \rangle$
ekintot	$\int_V \frac{1}{2} \varrho \boldsymbol{u}^2\, dV$
uxglnrym	$\langle u_x \partial_y \ln \varrho \rangle$
uyglnrxm	$\langle u_y \partial_x \ln \varrho \rangle$
uzdivum	$\langle u_z \nabla \cdot \boldsymbol{u} \rangle$
uxuydivum	$\langle u_x u_y \nabla \cdot \boldsymbol{u} \rangle$
divuHrms	$(\nabla_{\rm H} \cdot \boldsymbol{u}_{\rm H})^{\rm rms}$
uxxrms	$u_{x,x}^{\rm rms}$
uyyrms	$u_{y,y}^{\rm rms}$
uxzrms	$u_{x,z}^{\rm rms}$

uyzrms	$u^{\rm rms}_{y,z}$
uzyrms	$u^{\rm rms}_{z,y}$
udpxxm	components of symmetric tensor $\langle u_i\partial_j p + u_j\partial_i p\rangle$

Module 'density.f90'

rhom	$\langle\varrho\rangle$ (mean density)
rhomxmask	$\langle\varrho\rangle$ for the density_xaver_range
rhomzmask	$\langle\varrho\rangle$ for the density_zaver_range
rhomin	$\min(\rho)$
rhomax	$\max(\rho)$
ugrhom	$\langle\boldsymbol{u}\cdot\nabla\varrho\rangle$
totmass	$\int\varrho\,dV$
mass	$\int\varrho\,dV$
vol	$\int dV$ (volume)
grhomax	$\max(\lvert\nabla\varrho\rvert)$

Module 'entropy.f90'

dtc	$\delta t/[c_{\delta t}\,\delta_x/\max c_{\rm s}]$ (time step relative to acoustic time step; see § 5.15)
ethm	$\langle\varrho e\rangle$ (mean thermal [=internal] energy)
ssm	$\langle s/c_p\rangle$ (mean entropy)
ss2m	$\langle(s/c_p)^2\rangle$ (mean squared entropy)
eem	$\langle e\rangle$
ppm	$\langle p\rangle$
csm	$\langle c_{\rm s}\rangle$
pdivum	$\langle p\nabla\boldsymbol{u}\rangle$
fradbot	$\int F_{\rm bot}\cdot d\boldsymbol{S}$
fradtop	$\int F_{\rm top}\cdot d\boldsymbol{S}$
TTtop	$\int T_{\rm top} d\boldsymbol{S}$
ethtot	$\int_V\varrho e\,dV$ (total thermal [=internal] energy)
dtchi	$\delta t/[c_{\delta t,{\rm v}}\,\delta x^2/\chi_{\max}]$ (time step relative to time step based on heat conductivity; see § 5.15)
yHm	mean hydrogen ionization
yHmax	max of hydrogen ionization
TTm	$\langle T\rangle$
TTmax	$T_{\max}$
TTmin	$T_{\min}$
gTmax	$\max(\lvert\nabla T\rvert)$
ssmax	$s_{\max}$
ssmin	$s_{\min}$
gTrms	$(\nabla T)_{\rm rms}$
gsrms	$(\nabla s)_{\rm rms}$
gTxgsrms	$(\nabla T\times\nabla s)_{\rm rms}$
fconvm	$\langle c_p\varrho u_z T\rangle$
ufpresm	$\langle -u/\rho\nabla p\rangle$

Module 'magnetic.f90'

ab_int	$\int\boldsymbol{A}\cdot\boldsymbol{B}\,dV$
jb_int	$\int\boldsymbol{j}\cdot\boldsymbol{B}\,dV$
b2tm	$\left\langle\boldsymbol{b}(t)\cdot\int_0^t\boldsymbol{b}(t')dt'\right\rangle$

bjtm	$\left\langle \boldsymbol{b}(t) \cdot \int_0^t \boldsymbol{j}(t')dt' \right\rangle$
jbtm	$\left\langle \boldsymbol{j}(t) \cdot \int_0^t \boldsymbol{b}(t')dt' \right\rangle$
b2ruzm	$\left\langle \boldsymbol{B}^2 \rho u_z \right\rangle$
b2uzm	$\left\langle \boldsymbol{B}^2 u_z \right\rangle$
ubbzm	$\langle (\boldsymbol{u} \cdot \boldsymbol{B}) B_z \rangle$
b1m	$\langle \lvert \boldsymbol{B} \rvert \rangle$
b2m	$\left\langle \boldsymbol{B}^2 \right\rangle$
bm2	$\max(\boldsymbol{B}^2)$
j2m	$\left\langle \boldsymbol{j}^2 \right\rangle$
jm2	$\max(\boldsymbol{j}^2)$
abm	$\langle \boldsymbol{A} \cdot \boldsymbol{B} \rangle$
abumx	$\langle u_x \boldsymbol{A} \cdot \boldsymbol{B} \rangle$
abumy	$\langle u_y \boldsymbol{A} \cdot \boldsymbol{B} \rangle$
abumz	$\langle u_z \boldsymbol{A} \cdot \boldsymbol{B} \rangle$
abmh	$\langle \boldsymbol{A} \cdot \boldsymbol{B} \rangle$ (temp)
abmn	$\langle \boldsymbol{A} \cdot \boldsymbol{B} \rangle$ (north)
abms	$\langle \boldsymbol{A} \cdot \boldsymbol{B} \rangle$ (south)
abrms	$\langle (\boldsymbol{A} \cdot \boldsymbol{B})^2 \rangle^{1/2}$
jbrms	$\langle (\boldsymbol{j} \cdot \boldsymbol{B})^2 \rangle^{1/2}$
ajm	$\langle \boldsymbol{j} \cdot \boldsymbol{A} \rangle$
jbm	$\langle \boldsymbol{j} \cdot \boldsymbol{B} \rangle$
jbmh	$\langle \boldsymbol{J} \cdot \boldsymbol{B} \rangle$ (temp)
jbmn	$\langle \boldsymbol{J} \cdot \boldsymbol{B} \rangle$ (north)
jbms	$\langle \boldsymbol{J} \cdot \boldsymbol{B} \rangle$ (south)
ubm	$\langle \boldsymbol{u} \cdot \boldsymbol{B} \rangle$
dubrms	$\langle (\boldsymbol{u} - \boldsymbol{B})^2 \rangle^{1/2}$
dobrms	$\langle (\boldsymbol{\omega} - \boldsymbol{B})^2 \rangle^{1/2}$
uxbxm	$\langle u_x B_x \rangle$
uybxm	$\langle u_y B_x \rangle$
uzbxm	$\langle u_z B_x \rangle$
uxbym	$\langle u_x B_y \rangle$
uybym	$\langle u_y B_y \rangle$
uzbym	$\langle u_z B_y \rangle$
uxbzm	$\langle u_x B_z \rangle$
uybzm	$\langle u_y B_z \rangle$
uzbzm	$\langle u_z B_z \rangle$
cosubm	$\langle \boldsymbol{U} \cdot \boldsymbol{B} / (\lvert \boldsymbol{U} \rvert \, \lvert \boldsymbol{B} \rvert) \rangle$
uam	$\langle \boldsymbol{u} \cdot \boldsymbol{A} \rangle$
ujm	$\langle \boldsymbol{u} \cdot \boldsymbol{J} \rangle$
fbm	$\langle \boldsymbol{f} \cdot \boldsymbol{B} \rangle$
fxbxm	$\langle f_x B_x \rangle$
epsM	$\left\langle \eta \mu_0 \boldsymbol{j}^2 \right\rangle$
epsAD	$\langle \rho^{-1} t_{\rm AD} (\boldsymbol{J} \times \boldsymbol{B})^2 \rangle$ (heating by ion-neutrals friction)
bxpt	$B_x(x_1, y_1, z_1, t)$
bypt	$B_y(x_1, y_1, z_1, t)$
bzpt	$B_z(x_1, y_1, z_1, t)$
jxpt	$J_x(x_1, y_1, z_1, t)$
jypt	$J_y(x_1, y_1, z_1, t)$
jzpt	$J_z(x_1, y_1, z_1, t)$

Expt	$\mathcal{E}_x(x_1, y_1, z_1, t)$
Eypt	$\mathcal{E}_y(x_1, y_1, z_1, t)$
Ezpt	$\mathcal{E}_z(x_1, y_1, z_1, t)$
axpt	$A_x(x_1, y_1, z_1, t)$
aypt	$A_y(x_1, y_1, z_1, t)$
azpt	$A_z(x_1, y_1, z_1, t)$
bxp2	$B_x(x_2, y_2, z_2, t)$
byp2	$B_y(x_2, y_2, z_2, t)$
bzp2	$B_z(x_2, y_2, z_2, t)$
jxp2	$J_x(x_2, y_2, z_2, t)$
jyp2	$J_y(x_2, y_2, z_2, t)$
jzp2	$J_z(x_2, y_2, z_2, t)$
Exp2	$\mathcal{E}_x(x_2, y_2, z_2, t)$
Eyp2	$\mathcal{E}_y(x_2, y_2, z_2, t)$
Ezp2	$\mathcal{E}_z(x_2, y_2, z_2, t)$
axp2	$A_x(x_2, y_2, z_2, t)$
ayp2	$A_y(x_2, y_2, z_2, t)$
azp2	$A_z(x_2, y_2, z_2, t)$
exabot	$\int \boldsymbol{E} \times \boldsymbol{A}\, dS\vert_{\rm bot}$
exatop	$\int \boldsymbol{E} \times \boldsymbol{A}\, dS\vert_{\rm top}$
emag	$\int_V \frac{1}{2\mu_0} \boldsymbol{B}^2\, dV$
brms	$\left\langle \boldsymbol{B}^2 \right\rangle^{1/2}$
bfrms	$\left\langle \boldsymbol{B}'^2 \right\rangle^{1/2}$
bmax	$\max(\vert\boldsymbol{B}\vert)$
bxmin	$\min(\vert B_x\vert)$
bymin	$\min(\vert B_y\vert)$
bzmin	$\min(\vert B_z\vert)$
bxmax	$\max(\vert B_x\vert)$
bymax	$\max(\vert B_y\vert)$
bzmax	$\max(\vert B_z\vert)$
bbxmax	$\max(\vert B_x\vert) excluding Bv_{ext}$
bbymax	$\max(\vert B_y\vert) excluding Bv_{ext}$
bbzmax	$\max(\vert B_z\vert) excluding Bv_{ext}$
jxmax	$\max(\vert jv_x\vert)$
jymax	$\max(\vert jv_y\vert)$
jzmax	$\max(\vert jv_z\vert)$
jrms	$\left\langle \boldsymbol{j}^2 \right\rangle^{1/2}$
hjrms	$\left\langle \boldsymbol{j}^2 \right\rangle^{1/2}$
jmax	$\max(\vert\boldsymbol{j}\vert)$
vArms	$\left\langle \boldsymbol{B}^2/\varrho \right\rangle^{1/2}$
vAmax	$\max(\boldsymbol{B}^2/\varrho)^{1/2}$
dtb	$\delta t/[c_{\delta t}\, \delta x/v_{\rm A,max}]$ (time step relative to Alfvén time step; see § 5.15)
dteta	$\delta t/[c_{\delta t,{\rm v}}\, \delta x^2/\eta_{\rm max}]$ (time step relative to resistive time step; see § 5.15)
a2m	$\left\langle \boldsymbol{A}^2 \right\rangle$
arms	$\left\langle \boldsymbol{A}^2 \right\rangle^{1/2}$
amax	$\max(\vert\boldsymbol{A}\vert)$

divarms	$\langle(\nabla\cdot\boldsymbol{A})^2\rangle^{1/2}$
beta1m	$\langle\boldsymbol{B}^2/(2\mu_0 p)\rangle$ (mean inverse plasma beta)
beta1max	$\max[\boldsymbol{B}^2/(2\mu_0 p)]$ (maximum inverse plasma beta)
bxm	$\langle B_x\rangle$
bym	$\langle B_y\rangle$
bzm	$\langle B_z\rangle$
bxbym	$\langle B_x B_y\rangle$
bmx	$\left\langle\langle\boldsymbol{B}\rangle_{yz}^2\right\rangle^{1/2}$ (energy of yz-averaged mean field)
bmy	$\left\langle\langle\boldsymbol{B}\rangle_{xz}^2\right\rangle^{1/2}$ (energy of xz-averaged mean field)
bmz	$\left\langle\langle\boldsymbol{B}\rangle_{xy}^2\right\rangle^{1/2}$ (energy of xy-averaged mean field)
bmzS2	$\left\langle\langle\boldsymbol{B}_S\rangle_{xy}^2\right\rangle$
bmzA2	$\left\langle\langle\boldsymbol{B}_A\rangle_{xy}^2\right\rangle$
jmx	$\left\langle\langle\boldsymbol{J}\rangle_{yz}^2\right\rangle^{1/2}$ (energy of yz-averaged mean current density)
jmy	$\left\langle\langle\boldsymbol{J}\rangle_{xz}^2\right\rangle^{1/2}$ (energy of xz-averaged mean current density)
jmz	$\left\langle\langle\boldsymbol{J}\rangle_{xy}^2\right\rangle^{1/2}$ (energy of xy-averaged mean current density)
bmzph	Phase of a Beltrami field
bmzphe	Error of phase of a Beltrami field
bsinphz	sine of phase of a Beltrami field
bcosphz	cosine of phase of a Beltrami field
emxamz3	$\left\langle\langle\boldsymbol{E}\rangle_{xy}\times\langle\boldsymbol{A}\rangle_{xy}\right\rangle$ (xy-averaged mean field helicity flux)
embmz	$\left\langle\langle\boldsymbol{E}\rangle_{xy}\cdot\langle\boldsymbol{B}\rangle_{xy}\right\rangle$ (xy-averaged mean field helicity production)
ambmz	$\left\langle\langle\boldsymbol{A}\rangle_{xy}\cdot\langle\boldsymbol{B}\rangle_{xy}\right\rangle$ (magnetic helicity of xy-averaged mean field)
ambmzh	$\left\langle\langle\boldsymbol{A}\rangle_{xy}\cdot\langle\boldsymbol{B}\rangle_{xy}\right\rangle$ (magnetic helicity of xy-averaged mean field, temp)
ambmzn	$\left\langle\langle\boldsymbol{A}\rangle_{xy}\cdot\langle\boldsymbol{B}\rangle_{xy}\right\rangle$ (magnetic helicity of xy-averaged mean field, north)
ambmzs	$\left\langle\langle\boldsymbol{A}\rangle_{xy}\cdot\langle\boldsymbol{B}\rangle_{xy}\right\rangle$ (magnetic helicity of xy-averaged mean field, south)
jmbmz	$\left\langle\langle\boldsymbol{J}\rangle_{xy}\cdot\langle\boldsymbol{B}\rangle_{xy}\right\rangle$ (current helicity of xy-averaged mean field)
kx_aa	k_x
kmz	$\left\langle\langle\boldsymbol{J}\rangle_{xy}\cdot\langle\boldsymbol{B}\rangle_{xy}\right\rangle / \left\langle\langle\boldsymbol{B}\rangle_{xy}^2\right\rangle$
bx2m	$\langle B_x^2\rangle$
by2m	$\langle B_y^2\rangle$
bz2m	$\langle B_z^2\rangle$
uxbm	$\langle\boldsymbol{u}\times\boldsymbol{B}\rangle\cdot\boldsymbol{B}_0/B_0^2$
jxbm	$\langle\boldsymbol{j}\times\boldsymbol{B}\rangle\cdot\boldsymbol{B}_0/B_0^2$
magfricmax	Magneto-Frictional velocity $\langle\boldsymbol{j}\times\boldsymbol{B}\rangle\cdot\boldsymbol{B}^2$
b3b21m	$\langle B_3 B_{2,1}\rangle$
b3b12m	$\langle B_3 B_{1,2}\rangle$
b1b32m	$\langle B_1 B_{3,2}\rangle$

b1b23m	$\langle B_1 B_{2,3} \rangle$
b2b13m	$\langle B_2 B_{1,3} \rangle$
b2b31m	$\langle B_2 B_{3,1} \rangle$
uxbmx	$\langle (\boldsymbol{u} \times \boldsymbol{B})_x \rangle$
uxbmy	$\langle (\boldsymbol{u} \times \boldsymbol{B})_y \rangle$
uxbmz	$\langle (\boldsymbol{u} \times \boldsymbol{B})_z \rangle$
jxbmx	$\langle (\boldsymbol{j} \times \boldsymbol{B})_x \rangle$
jxbmy	$\langle (\boldsymbol{j} \times \boldsymbol{B})_y \rangle$
jxbmz	$\langle (\boldsymbol{j} \times \boldsymbol{B})_z \rangle$
examx	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle \vert_x$
examy	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle \vert_y$
examz	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle \vert_z$
exjmx	$\langle \boldsymbol{E} \times \boldsymbol{J} \rangle \vert_x$
exjmy	$\langle \boldsymbol{E} \times \boldsymbol{J} \rangle \vert_y$
exjmz	$\langle \boldsymbol{E} \times \boldsymbol{J} \rangle \vert_z$
dexbmx	$\langle \nabla \times \boldsymbol{E} \times \boldsymbol{B} \rangle \vert_x$
dexbmy	$\langle \nabla \times \boldsymbol{E} \times \boldsymbol{B} \rangle \vert_y$
dexbmz	$\langle \nabla \times \boldsymbol{E} \times \boldsymbol{B} \rangle \vert_z$
phibmx	$\langle \phi \boldsymbol{B} \rangle \vert_x$
phibmy	$\langle \phi \boldsymbol{B} \rangle \vert_y$
phibmz	$\langle \phi \boldsymbol{B} \rangle \vert_z$
b2divum	$\left\langle \boldsymbol{B}^2 \nabla \cdot \boldsymbol{u} \right\rangle$
ujxbm	$\langle \boldsymbol{u} \cdot (\boldsymbol{J} \times \boldsymbol{B}) \rangle$
jxbrmax	$\max(\vert \boldsymbol{J} \times \boldsymbol{B}/\rho \vert)$
jxbr2m	$\langle (\boldsymbol{J} \times \boldsymbol{B}/\rho)^2 \rangle$
bmxy_rms	$\sqrt{[\langle b_x \rangle_z (x,y)]^2 + [\langle b_y \rangle_z (x,y)]^2 + [\langle b_z \rangle_z (x,y)]^2}$
etasmagm	Mean of Smagorinsky resistivity
etasmagmin	Min of Smagorinsky resistivity
etasmagmax	Max of Smagorinsky resistivity
etavamax	Max of artificial resistivity $\eta \sim v_A$
etajmax	Max of artificial resistivity $\eta \sim J/\sqrt{\rho}$
etaj2max	Max of artificial resistivity $\eta \sim J^2/\rho$
etajrhomax	Max of artificial resistivity $\eta \sim J/\rho$
cosjbm	$\langle \boldsymbol{J} \cdot \boldsymbol{B}/(\vert \boldsymbol{J} \vert \, \vert \boldsymbol{B} \vert) \rangle$
jparallelm	Mean value of the component of J parallel to B
jperpm	Mean value of the component of J perpendicular to B
hjparallelm	Mean value of the component of $J_{\rm hyper}$ parallel to B
hjperpm	Mean value of the component of $J_{\rm hyper}$ perpendicular to B
brmsx	$\left\langle \boldsymbol{B}^2 \right\rangle^{1/2}$ for the magnetic_xaver_range
brmsz	$\left\langle \boldsymbol{B}^2 \right\rangle^{1/2}$ for the magnetic_zaver_range
Exmxy	$\langle \mathcal{E}_x \rangle_z$
Eymxy	$\langle \mathcal{E}_y \rangle_z$
Ezmxy	$\langle \mathcal{E}_z \rangle_z$

Module 'pscalar.f90'

rhoccm	$\langle \varrho c \rangle$
ccmax	$\max(c)$
ccglnrm	$\langle c \nabla_z \varrho \rangle$

Module '1D_loop.f90'

dtchi2	heatconduction
dtrad	radiative loss from RTV
dtspitzer	Spitzer heat conduction time step
qmax	max of heat flux vector
qrms	rms of heat flux vector

Module ‘`advective_gauge.f90`’

Lamm	$\langle \Lambda \rangle$
Lampt	$\Lambda(x1, y1, z1)$
Lamp2	$\Lambda(x2, y2, z2)$
Lamrms	$\langle \Lambda^2 \rangle^{1/2}$
Lambzm	$\langle \Lambda B_z \rangle$
Lambzmz	$\langle \Lambda B_z \rangle_{xy}$
gLambm	$\langle \Lambda \boldsymbol{B} \rangle$
apbrms	$\langle (\boldsymbol{A}'\boldsymbol{B})^2 \rangle^{1/2}$
jxarms	$\langle (\boldsymbol{J} \times \boldsymbol{A})^2 \rangle^{1/2}$
jxaprms	$\langle (\boldsymbol{J} \times \boldsymbol{A}')^2 \rangle^{1/2}$
jxgLamrms	$\langle (\boldsymbol{J} \times \nabla\Lambda)^2 \rangle^{1/2}$
gLamrms	$\langle (\nabla\Lambda)^2 \rangle^{1/2}$
divabrms	$\langle [(\nabla \cdot \boldsymbol{A})\boldsymbol{B}]^2 \rangle^{1/2}$
divapbrms	$\langle [(\nabla \cdot \boldsymbol{A}')\boldsymbol{B}]^2 \rangle^{1/2}$
d2Lambrms	$\langle [(\nabla^2\Lambda)\boldsymbol{B}]^2 \rangle^{1/2}$
d2Lamrms	$\langle [\nabla^2\Lambda]^2 \rangle^{1/2}$

Module ‘`anelastic.f90`’

rhom	$\langle \varrho \rangle$ (mean density)
ugrhom	$\langle \boldsymbol{u} \cdot \nabla\varrho \rangle$
mass	$\int \varrho \, dV$
divrhoum	$\langle \nabla \cdot (\varrho \boldsymbol{u}) \rangle$
divrhourms	$\lvert \nabla \cdot (\varrho \boldsymbol{u}) \rvert_{\rm rms}$
divrhoumax	$\lvert \nabla \cdot (\varrho \boldsymbol{u}) \rvert_{\max}$

Module ‘`bfield.f90`’

bmax	$\max B$
bmin	$\min B$
brms	$\langle B^2 \rangle^{1/2}$
bm	$\langle B \rangle$
b2m	$\langle B^2 \rangle$
bxmax	$\max \lvert B_x \rvert$
bymax	$\max \lvert B_y \rvert$
bzmax	$\max \lvert B_z \rvert$
bxm	$\langle B_x \rangle$
bym	$\langle B_y \rangle$
bzm	$\langle B_z \rangle$
bx2m	$\langle B_x^2 \rangle$
by2m	$\langle B_y^2 \rangle$
bz2m	$\langle B_z^2 \rangle$
bxbym	$\langle B_x B_y \rangle$
bxbzm	$\langle B_x B_z \rangle$

bybzm	$\langle B_y B_z \rangle$
dbxmax	$\max \lvert B_x - B_{\mathrm{ext},x} \rvert$
dbymax	$\max \lvert B_y - B_{\mathrm{ext},y} \rvert$
dbzmax	$\max \lvert B_z - B_{\mathrm{ext},z} \rvert$
dbxm	$\langle B_x - B_{\mathrm{ext},x} \rangle$
dbym	$\langle B_y - B_{\mathrm{ext},y} \rangle$
dbzm	$\langle B_z - B_{\mathrm{ext},z} \rangle$
dbx2m	$\langle (B_x - B_{\mathrm{ext},x})^2 \rangle$
dby2m	$\langle (B_y - B_{\mathrm{ext},y})^2 \rangle$
dbz2m	$\langle (B_z - B_{\mathrm{ext},z})^2 \rangle$
jmax	$\max J$
jmin	$\min J$
jrms	$\langle J^2 \rangle^{1/2}$
jm	$\langle J \rangle$
j2m	$\langle J^2 \rangle$
jxmax	$\max \lvert J_x \rvert$
jymax	$\max \lvert J_y \rvert$
jzmax	$\max \lvert J_z \rvert$
jxm	$\langle J_x \rangle$
jym	$\langle J_y \rangle$
jzm	$\langle J_z \rangle$
jx2m	$\langle J_x^2 \rangle$
jy2m	$\langle J_y^2 \rangle$
jz2m	$\langle J_z^2 \rangle$
divbmax	$\max \lvert \nabla \cdot \boldsymbol{B} \rvert$
divbrms	$\langle (\nabla \cdot \boldsymbol{B})^2 \rangle^{1/2}$
betamax	$\max \beta$
betamin	$\min \beta$
betam	$\langle \beta \rangle$
vAmax	$\max v_A$
vAmin	$\min v_A$
vAm	$\langle v_A \rangle$

Module 'chemistry.f90'

Y1m	$\langle Y_1 \rangle$
Y2m	$\langle Y_2 \rangle$
Y3m	$\langle Y_3 \rangle$
Y4m	$\langle Y_4 \rangle$
Y5m	$\langle Y_5 \rangle$
Y6m	$\langle Y_6 \rangle$
Y7m	$\langle Y_7 \rangle$
Y8m	$\langle Y_8 \rangle$
Y9m	$\langle Y_9 \rangle$
Y10m	$\langle Y_10 \rangle$
Y11m	$\langle Y_11 \rangle$
Y12m	$\langle Y_12 \rangle$
Y13m	$\langle Y_12 \rangle$
Y14m	$\langle Y_12 \rangle$
Y15m	$\langle Y_12 \rangle$
Y16m	$\langle Y_12 \rangle$

Y17m	$\langle Y_1 2 \rangle$
Y18m	$\langle Y_1 2 \rangle$
Y19m	$\langle Y_1 2 \rangle$
dY1m	$\langle dY_1 \rangle$
dY2m	$\langle dY_2 \rangle$
dY3m	$\langle dY_3 \rangle$
dY4m	$\langle dY_4 \rangle$
dY5m	$\langle dY_5 \rangle$
dY6m	$\langle dY_6 \rangle$
dY7m	$\langle dY_7 \rangle$
dY8m	$\langle dY_8 \rangle$
dY9m	$\langle dY_9 \rangle$
dY10m	$\langle dY_1 0 \rangle$
dY11m	$\langle dY_1 1 \rangle$
dY12m	$\langle dY_1 2 \rangle$
dY13m	$\langle dY_1 3 \rangle$
dY14m	$\langle dY_1 4 \rangle$
dY15m	$\langle dY_1 5 \rangle$
dY16m	$\langle dY_1 6 \rangle$
dY17m	$\langle dY_1 7 \rangle$
dY18m	$\langle dY_1 8 \rangle$
dY19m	$\langle dY_1 9 \rangle$
Y1max	$\langle Y_1 \rangle$
Y2max	$\langle Y_2 \rangle$
Y3max	$\langle Y_3 \rangle$
Y4max	$\langle Y_4 \rangle$
Y5max	$\langle Y_5 \rangle$
Y6max	$\langle Y_6 \rangle$
Y7max	$\langle Y_7 \rangle$
Y8max	$\langle Y_8 \rangle$
Y9max	$\langle Y_9 \rangle$
Y10max	$\langle Y_1 0 \rangle$
Y11max	$\langle Y_1 1 \rangle$
Y12max	$\langle Y_1 2 \rangle$
Y13max	$\langle Y_1 2 \rangle$
Y14max	$\langle Y_1 2 \rangle$
Y15max	$\langle Y_1 2 \rangle$
Y16max	$\langle Y_1 2 \rangle$
Y17max	$\langle Y_1 2 \rangle$
Y18max	$\langle Y_1 2 \rangle$
Y19max	$\langle Y_1 2 \rangle$
dY1max	$\langle dY_1 \rangle$
dY2max	$\langle dY_2 \rangle$
dY3max	$\langle dY_3 \rangle$
dY4max	$\langle dY_4 \rangle$
dY5max	$\langle dY_5 \rangle$
dY6max	$\langle dY_6 \rangle$
dY7max	$\langle dY_7 \rangle$
dY8max	$\langle dY_8 \rangle$
dY9max	$\langle dY_9 \rangle$

dY10max	$\langle dY_1 0\rangle$
dY11max	$\langle dY_1 1\rangle$
dY12max	$\langle dY_1 2\rangle$
dY13max	$\langle dY_1 3\rangle$
dY14max	$\langle dY_1 4\rangle$
dY15max	$\langle dY_1 5\rangle$
dY16max	$\langle dY_1 6\rangle$
dY17max	$\langle dY_1 7\rangle$
dY18max	$\langle dY_1 8\rangle$
dY19max	$\langle dY_1 9\rangle$
Y1mz	$\langle Y_1\rangle_{xy}(z)$
Y2mz	$\langle Y_2\rangle_{xy}(z)$
Y3mz	$\langle Y_3\rangle_{xy}(z)$
Y4mz	$\langle Y_4\rangle_{xy}(z)$
Y5mz	$\langle Y_5\rangle_{xy}(z)$
Y6mz	$\langle Y_6\rangle_{xy}(z)$
Y7mz	$\langle Y_7\rangle_{xy}(z)$
Y8mz	$\langle Y_8\rangle_{xy}(z)$
Y9mz	$\langle Y_9\rangle_{xy}(z)$
Y10mz	$\langle Y_1 0\rangle_{xy}(z)$
Y11mz	$\langle Y_1 1\rangle_{xy}(z)$
Y12mz	$\langle Y_1 2\rangle_{xy}(z)$
Y13mz	$\langle Y_1 3\rangle_{xy}(z)$
Y14mz	$\langle Y_1 4\rangle_{xy}(z)$
Y15mz	$\langle Y_1 5\rangle_{xy}(z)$
Y16mz	$\langle Y_1 6\rangle_{xy}(z)$
Y17mz	$\langle Y_1 7\rangle_{xy}(z)$
Y18mz	$\langle Y_1 8\rangle_{xy}(z)$
Y19mz	$\langle Y_1 9\rangle_{xy}(z)$

Module 'coronae.f90'

dtchi2	$\delta t/[c_{\delta t,\mathrm{v}}\,\delta x^2/\chi_{\max}]$ (time step relative to time step based on heat conductivity; see § 5.15)
dtspitzer	Spitzer heat conduction time step
dtrad	radiative loss from RTV

Module 'density_stratified.f90'

mass	$\int \rho\, d^3x$
rhomin	$\min \lvert\rho\rvert$
rhomax	$\max \lvert\rho\rvert$
drhom	$\langle\Delta\rho/\rho_0\rangle$
drho2m	$\langle(\Delta\rho/\rho_0)^2\rangle$
drhorms	$\langle\Delta\rho/\rho_0\rangle_{rms}$
drhomax	$\max \lvert\Delta\rho/\rho_0\rvert$

Module 'detonate.f90'

detn	Number of detonated sites (summed over time steps between adjacent outputs)
dettot	Total energy input (summed over time steps between adjacent outputs)

Module 'entropy_anelastic.f90'	
dtc	$\delta t/[c_{\delta t}\,\delta_x/\max c_{\rm s}]$ (time step relative to acoustic time step; see § 5.15)
ethm	$\langle\varrho e\rangle$ (mean thermal [=internal] energy)
ssm	$\langle s/c_p\rangle$ (mean entropy)
ss2m	$\langle(s/c_p)^2\rangle$ (mean squared entropy)
eem	$\langle e\rangle$
ppm	$\langle p\rangle$
csm	$\langle c_{\rm s}\rangle$
pdivum	$\langle p\nabla\boldsymbol{u}\rangle$
fradbot	$\int F_{\rm bot}\cdot d\boldsymbol{S}$
fradtop	$\int F_{\rm top}\cdot d\boldsymbol{S}$
TTtop	$\int T_{\rm top} d\boldsymbol{S}$
ethtot	$\int_V \varrho e\,dV$ (total thermal [=internal] energy)
dtchi	$\delta t/[c_{\delta t,{\rm v}}\,\delta x^2/\chi_{\max}]$ (time step relative to time step based on heat conductivity; see § 5.15)
ssmxy	$\langle s\rangle_z$
ssmxz	$\langle s\rangle_y$
Module 'gravitational_waves.f90'	
g22pt	$g_{22}(x_1,y_1,z_1,t)$
Module 'gravity_simple.f90'	
epot	$\langle\varrho\Phi_{\rm grav}\rangle$ (mean potential energy)
Module 'heatflux.f90'	
qmax	max of heat flux vector
qrms	rms of heat flux vector
Module 'lorenz_gauge.f90'	
phim	$\langle\phi\rangle$
phipt	$\phi(x1,y1,z1)$
phip2	$\phi(x2,y2,z2)$
phibzm	$\langle\phi B_z\rangle$
phibzmz	$\langle\phi B_z\rangle_{xy}$
Module 'meanfield.f90'	
qsm	$\left\langle q_p(\overline{B})\right\rangle$
qpm	$\left\langle q_p(\overline{B})\right\rangle$
qem	$\left\langle q_e(\overline{B})\right\rangle$, in the paper referred to as $\left\langle q_g(\overline{B})\right\rangle$
qam	$\left\langle q_a(\overline{B})\right\rangle$
alpm	$\langle\alpha\rangle$
etatm	$\langle\eta_{\rm t}\rangle$
EMFmz1	$\langle\mathcal{E}\rangle_{xy}\,\vert_x$
EMFmz2	$\langle\mathcal{E}\rangle_{xy}\,\vert_y$
EMFmz3	$\langle\mathcal{E}\rangle_{xy}\,\vert_z$
EMFdotBm	$\langle\mathcal{E}\cdot\boldsymbol{B}\rangle$
EMFdotB_int	$\int\mathcal{E}\cdot\boldsymbol{B}dV$
Module 'meanfield_demfdt.f90'	

EMFrms	$(\langle\mathcal{E}\rangle)_{\rm rms}$
EMFmax	$\max(\langle\mathcal{E}\rangle)$
EMFmin	$\min(\langle\mathcal{E}\rangle)$
Module 'noentropy.f90'	
dtc	$\delta t/[c_{\delta t}\,\delta_x/\max c_{\rm s}]$ (time step relative to acoustic time step; see § 5.15)
ethm	$\langle\varrho e\rangle$ (mean thermal [=internal] energy)
pdivum	$\langle p\nabla\boldsymbol{u}\rangle$
Module 'nolorenz_gauge.f90'	
phim	$\langle\phi\rangle$
phipt	$\phi(x1,y1,z1)>$
phip2	$\phi(x2,y2,z2)>$
phibzm	$\langle\phi B_z\rangle$
phibzmz	$\langle\phi B_z\rangle_{xy}$
Module 'nosolid_cells.f90'	
Module 'polymer.f90'	
polytrm	$\langle Tr[C_{ij}]\rangle$
frmax	$\max(f(r))$
Module 'shear.f90'	
dtshear	advec_shear/cdt
deltay	deltay
Module 'shock.f90'	
shockmax	Max shock number
Module 'shock_highorder.f90'	
gshockmax	$\max\lvert\nabla\nu_{shock}\rvert$
Module 'solar_corona.f90'	
dtvel	Velocity driver time step
dtnewt	Radiative cooling time step
dtradloss	Radiative losses time step
dtchi2	$\delta t/[c_{\delta t,{\rm v}}\,\delta x^2/\chi_{\max}]$ (time step relative to time step based on heat conductivity; see § 5.15)
dtspitzer	Spitzer heat conduction time step
mag_flux	Total vertical magnetic flux at
Module 'solid_cells.f90'	
Module 'temperature_idealgas.f90'	
TTmax	$\max(T)$
gTmax	$\max(\lvert\nabla T\rvert)$
TTmin	$\min(T)$
TTm	$\langle T\rangle$
TugTm	$\langle T\boldsymbol{u}\cdot\nabla T\rangle$
Trms	$\sqrt{\langle T^2\rangle}$

uxTm	$\langle u_x T \rangle$
uyTm	$\langle u_y T \rangle$
uzTm	$\langle u_z T \rangle$
gT2m	$\langle (\nabla T)^2 \rangle$
guxgTm	$\langle \nabla u_x \cdot \nabla T \rangle$
guygTm	$\langle \nabla u_y \cdot \nabla T \rangle$
guzgTm	$\langle \nabla u_z \cdot \nabla T \rangle$
Tugux_uxugTm	$\langle T\boldsymbol{u} \cdot \nabla u_x + u_x \boldsymbol{u} \cdot \nabla T \rangle = \langle \boldsymbol{u} \cdot \nabla (u_x T) \rangle$
Tuguy_uyugTm	$\langle T\boldsymbol{u} \cdot \nabla u_y + u_y \boldsymbol{u} \cdot \nabla T \rangle = \langle \boldsymbol{u} \cdot \nabla (u_y T) \rangle$
Tuguz_uzugTm	$\langle T\boldsymbol{u} \cdot \nabla u_z + u_z \boldsymbol{u} \cdot \nabla T \rangle = \langle \boldsymbol{u} \cdot \nabla (u_z T) \rangle$
Tdxpm	$\langle T dp/dx \rangle$
Tdypm	$\langle T dp/dy \rangle$
Tdzpm	$\langle T dp/dz \rangle$
fradtop	$< -K \frac{dT}{dz} >_{\mathrm{top}}$ (radiative flux at the top)
yHmax	DOCUMENT ME
yHmin	DOCUMENT ME
yHm	DOCUMENT ME
ethm	$\langle e_{\mathrm{th}} \rangle = \langle c_v \rho T \rangle$ (mean thermal energy)
eem	$\langle e \rangle = \langle c_v T \rangle$ (mean internal energy)
dtc	$\delta t / [c_{\delta t}\, \delta_x / \max c_{\mathrm{s}}]$ (time step relative to acoustic time step; see § 5.15)
dtchi	$\delta t / [c_{\delta t,\mathrm{v}}\, \delta x^2 / \chi_{\max}]$ (time step relative to time step based on heat conductivity; see § 5.15)

Module '`temperature_ionization.f90`'

TTmax	$\max(T)$
TTmin	$\min(T)$
TTm	$\langle T \rangle$
ethm	$\langle e_{\mathrm{th}} \rangle = \langle c_v \rho T \rangle$ (mean thermal energy)
eem	$\langle e \rangle = \langle c_v T \rangle$ (mean internal energy)

Module '`testfield_axisym.f90`'

alpPERP	$\alpha_\perp$
alpPARA	$\alpha_\perp$
gam	γ
betPERP	$\beta_\perp$
betPARA	$\beta_\perp$
del	δ
kapPERP	$\kappa_\perp$
kapPARA	$\kappa_\perp$
mu	μ
alpPERPz	$\alpha_\perp(z)$
alpPARAz	$\alpha_\perp(z)$
gamz	$\gamma(z)$
betPERPz	$\beta_\perp(z)$
betPARAz	$\beta_\perp(z)$
delz	$\delta(z)$
kapPERPz	$\kappa_\perp(z)$
kapPARAz	$\kappa_\perp(z)$
muz	$\mu(z)$

bx1pt	b_x^1
bx2pt	b_x^2
bx3pt	b_x^3
b1rms	$\langle b_1^2 \rangle^{1/2}$
b2rms	$\langle b_2^2 \rangle^{1/2}$
b3rms	$\langle b_3^2 \rangle^{1/2}$

Module 'testfield_axisym2.f90'

alpPERP	$\alpha_\perp$
alpPARA	$\alpha_\perp$
gam	γ
betPERP	$\beta_\perp$
betPARA	$\beta_\perp$
del	δ
kapPERP	$\kappa_\perp$
kapPARA	$\kappa_\perp$
mu	μ
bx1pt	b_x^1
bx2pt	b_x^2
bx3pt	b_x^3
b1rms	$\langle b_1^2 \rangle^{1/2}$
b2rms	$\langle b_2^2 \rangle^{1/2}$
b3rms	$\langle b_3^2 \rangle^{1/2}$

Module 'testfield_axisym4.f90'

alpPERP	$\alpha_\perp$
alpPARA	$\alpha_\perp$
gam	γ
betPERP	$\beta_\perp$
betPERP2	$\beta_\perp^{(2)}$
betPARA	$\beta_\perp$
del	δ
del2	$\delta^{(2)}$
kapPERP	$\kappa_\perp$
kapPERP2	$\kappa_\perp^{(2)}$
kapPARA	$\kappa_\perp$
mu	μ
mu2	$\mu^{(2)}$
alpPERPz	$\alpha_\perp(z)$
alpPARAz	$\alpha_\perp(z)$
gamz	$\gamma(z)$
betPERPz	$\beta_\perp(z)$
betPARAz	$\beta_\perp(z)$
delz	$\delta(z)$
kapPERPz	$\kappa_\perp(z)$
kapPARAz	$\kappa_\perp(z)$
muz	$\mu(z)$
bx1pt	b_x^1
bx2pt	b_x^2

bx3pt	b_x^3
b1rms	$\langle b_1^2 \rangle^{1/2}$
b2rms	$\langle b_2^2 \rangle^{1/2}$
b3rms	$\langle b_3^2 \rangle^{1/2}$

Module 'testfield_compress_z.f90'

alp11	α_{11}
alp21	α_{21}
alp31	α_{31}
alp12	α_{12}
alp22	α_{22}
alp32	α_{32}
eta11	$\eta_{11} k$
eta21	$\eta_{21} k$
eta12	$\eta_{12} k$
eta22	$\eta_{22} k$
alpK	α^K
alpM	α^M
alpMK	α^{MK}
phi11	ϕ_{11}
phi21	ϕ_{21}
phi12	ϕ_{12}
phi22	ϕ_{22}
phi32	ϕ_{32}
psi11	$\psi_{11} k$
psi21	$\psi_{21} k$
psi12	$\psi_{12} k$
psi22	$\psi_{22} k$
phiK	ϕ^K
phiM	ϕ^M
phiMK	ϕ^{MK}
alp11cc	$\alpha_{11} \cos^2 kz$
alp21sc	$\alpha_{21} \sin kz \cos kz$
alp12cs	$\alpha_{12} \cos kz \sin kz$
alp22ss	$\alpha_{22} \sin^2 kz$
eta11cc	$\eta_{11} \cos^2 kz$
eta21sc	$\eta_{21} \sin kz \cos kz$
eta12cs	$\eta_{12} \cos kz \sin kz$
eta22ss	$\eta_{22} \sin^2 kz$
s2kzDFm	$\langle \sin 2kz \nabla \cdot F \rangle$
M11	$\mathcal{M}_{11}$
M22	$\mathcal{M}_{22}$
M33	$\mathcal{M}_{33}$
M11cc	$\mathcal{M}_{11} \cos^2 kz$
M11ss	$\mathcal{M}_{11} \sin^2 kz$
M22cc	$\mathcal{M}_{22} \cos^2 kz$
M22ss	$\mathcal{M}_{22} \sin^2 kz$
M12cs	$\mathcal{M}_{12} \cos kz \sin kz$
bx11pt	b_x^{11}
bx21pt	b_x^{21}

bx12pt	b_x^{12}
bx22pt	b_x^{22}
bx0pt	b_x^{0}
by11pt	b_y^{11}
by21pt	b_y^{21}
by12pt	b_y^{12}
by22pt	b_y^{22}
by0pt	b_y^{0}
u11rms	$\langle u_{11}^2 \rangle^{1/2}$
u21rms	$\langle u_{21}^2 \rangle^{1/2}$
u12rms	$\langle u_{12}^2 \rangle^{1/2}$
u22rms	$\langle u_{22}^2 \rangle^{1/2}$
j11rms	$\langle j_{11}^2 \rangle^{1/2}$
b11rms	$\langle b_{11}^2 \rangle^{1/2}$
b21rms	$\langle b_{21}^2 \rangle^{1/2}$
b12rms	$\langle b_{12}^2 \rangle^{1/2}$
b22rms	$\langle b_{22}^2 \rangle^{1/2}$
ux0m	$\langle u_{0_x} \rangle$
uy0m	$\langle u_{0_y} \rangle$
ux11m	$\langle u_{11_x} \rangle$
uy11m	$\langle u_{11_y} \rangle$
u0rms	$\langle u_0^2 \rangle^{1/2}$
b0rms	$\langle b_0^2 \rangle^{1/2}$
jb0m	$\langle j b_0 \rangle$
E11rms	$\langle \mathcal{E}_{11}^2 \rangle^{1/2}$
E21rms	$\langle \mathcal{E}_{21}^2 \rangle^{1/2}$
E12rms	$\langle \mathcal{E}_{12}^2 \rangle^{1/2}$
E22rms	$\langle \mathcal{E}_{22}^2 \rangle^{1/2}$
E0rms	$\langle \mathcal{E}_0^2 \rangle^{1/2}$
Ex11pt	$\mathcal{E}_x^{11}$
Ex21pt	$\mathcal{E}_x^{21}$
Ex12pt	$\mathcal{E}_x^{12}$
Ex22pt	$\mathcal{E}_x^{22}$
Ex0pt	$\mathcal{E}_x^{0}$
Ey11pt	$\mathcal{E}_y^{11}$
Ey21pt	$\mathcal{E}_y^{21}$
Ey12pt	$\mathcal{E}_y^{12}$
Ey22pt	$\mathcal{E}_y^{22}$
Ey0pt	$\mathcal{E}_y^{0}$
bamp	bamp
E111z	$\mathcal{E}_1^{11}$
E211z	$\mathcal{E}_2^{11}$
E311z	$\mathcal{E}_3^{11}$
E121z	$\mathcal{E}_1^{21}$
E221z	$\mathcal{E}_2^{21}$
E321z	$\mathcal{E}_3^{21}$
E112z	$\mathcal{E}_1^{12}$
E212z	$\mathcal{E}_2^{12}$

E312z	$\mathcal{E}_3^{12}$
E122z	$\mathcal{E}_1^{22}$
E222z	$\mathcal{E}_2^{22}$
E322z	$\mathcal{E}_3^{22}$
E10z	$\mathcal{E}_1^{0}$
E20z	$\mathcal{E}_2^{0}$
E30z	$\mathcal{E}_3^{0}$
EBpq	$\mathcal{E} \cdot \boldsymbol{B}^{pq}$
E0Um	$\mathcal{E}^0 \cdot \boldsymbol{U}$
E0Wm	$\mathcal{E}^0 \cdot \boldsymbol{W}$
bx0mz	$\langle b_x \rangle_{xy}$
by0mz	$\langle b_y \rangle_{xy}$
bz0mz	$\langle b_z \rangle_{xy}$
M11z	$\langle \mathcal{M}_{11} \rangle_{xy}$
M22z	$\langle \mathcal{M}_{22} \rangle_{xy}$
M33z	$\langle \mathcal{M}_{33} \rangle_{xy}$

Module ‘`testfield_meri.f90`’

E11xy	E_{11xy}
E12xy	E_{12xy}
E13xy	E_{13xy}
E21xy	E_{21xy}
E22xy	E_{22xy}
E23xy	E_{23xy}
E31xy	E_{31xy}
E32xy	E_{32xy}
E33xy	E_{33xy}
E41xy	E_{41xy}
E42xy	E_{42xy}
E43xy	E_{43xy}
E51xy	E_{51xy}
E52xy	E_{52xy}
E53xy	E_{53xy}
E61xy	E_{61xy}
E62xy	E_{62xy}
E63xy	E_{63xy}
E71xy	E_{71xy}
E72xy	E_{72xy}
E73xy	E_{73xy}
E81xy	E_{81}
E82xy	E_{82}
E83xy	E_{83}
E91xy	E_{91}
E92xy	E_{92}
E93xy	E_{93}
a11xy	α_{11}
a12xy	α_{12}
a13xy	α_{13}
a21xy	α_{21}
a22xy	α_{22}

a23xy	α_{23}
a31xy	α_{31}
a32xy	α_{32}
a33xy	α_{33}
b111xy	$\_111$
b121xy	$\_121$
b131xy	$\_131$
b211xy	$\_211$
b221xy	$\_221$
b231xy	$\_231$
b311xy	$\_311$
b321xy	$\_321$
b331xy	$\_331$
b112xy	$\_112$
b122xy	$\_122$
b132xy	$\_132$
b212xy	$\_212$
b222xy	$\_222$
b232xy	$\_232$
b312xy	$\_312$
b322xy	$\_322$
b332xy	$\_332$

Module 'testfield_nonlin_z.f90'

alp11	α_{11}
alp21	α_{21}
alp31	α_{31}
alp12	α_{12}
alp22	α_{22}
alp32	α_{32}
eta11	$\eta_{11}k$
eta21	$\eta_{21}k$
eta12	$\eta_{12}k$
eta22	$\eta_{22}k$
alpK	α^K
alpM	α^M
alpMK	α^{MK}
phi11	ϕ_{11}
phi21	ϕ_{21}
phi12	ϕ_{12}
phi22	ϕ_{22}
phi32	ϕ_{32}
psi11	$\psi_{11}k$
psi21	$\psi_{21}k$
psi12	$\psi_{12}k$
psi22	$\psi_{22}k$
phiK	ϕ^K
phiM	ϕ^M
phiMK	ϕ^{MK}
alp11cc	$\alpha_{11}\cos^2 kz$

alp21sc	$\alpha_{21} \sin kz \cos kz$
alp12cs	$\alpha_{12} \cos kz \sin kz$
alp22ss	$\alpha_{22} \sin^2 kz$
eta11cc	$\eta_{11} \cos^2 kz$
eta21sc	$\eta_{21} \sin kz \cos kz$
eta12cs	$\eta_{12} \cos kz \sin kz$
eta22ss	$\eta_{22} \sin^2 kz$
s2kzDFm	$\langle \sin 2kz \nabla \cdot F \rangle$
M11	$\mathcal{M}_{11}$
M22	$\mathcal{M}_{22}$
M33	$\mathcal{M}_{33}$
M11cc	$\mathcal{M}_{11} \cos^2 kz$
M11ss	$\mathcal{M}_{11} \sin^2 kz$
M22cc	$\mathcal{M}_{22} \cos^2 kz$
M22ss	$\mathcal{M}_{22} \sin^2 kz$
M12cs	$\mathcal{M}_{12} \cos kz \sin kz$
bx11pt	b_x^{11}
bx21pt	b_x^{21}
bx12pt	b_x^{12}
bx22pt	b_x^{22}
bx0pt	b_x^{0}
by11pt	b_y^{11}
by21pt	b_y^{21}
by12pt	b_y^{12}
by22pt	b_y^{22}
by0pt	b_y^{0}
u11rms	$\langle u_{11}^2 \rangle^{1/2}$
u21rms	$\langle u_{21}^2 \rangle^{1/2}$
u12rms	$\langle u_{12}^2 \rangle^{1/2}$
u22rms	$\langle u_{22}^2 \rangle^{1/2}$
j11rms	$\langle j_{11}^2 \rangle^{1/2}$
b11rms	$\langle b_{11}^2 \rangle^{1/2}$
b21rms	$\langle b_{21}^2 \rangle^{1/2}$
b12rms	$\langle b_{12}^2 \rangle^{1/2}$
b22rms	$\langle b_{22}^2 \rangle^{1/2}$
ux0m	$\langle u_{0_x} \rangle$
uy0m	$\langle u_{0_y} \rangle$
ux11m	$\langle u_{11_x} \rangle$
uy11m	$\langle u_{11_y} \rangle$
u0rms	$\langle u_0^2 \rangle^{1/2}$
b0rms	$\langle b_0^2 \rangle^{1/2}$
jb0m	$\langle j b_0 \rangle$
E11rms	$\langle \mathcal{E}_{11}^2 \rangle^{1/2}$
E21rms	$\langle \mathcal{E}_{21}^2 \rangle^{1/2}$
E12rms	$\langle \mathcal{E}_{12}^2 \rangle^{1/2}$
E22rms	$\langle \mathcal{E}_{22}^2 \rangle^{1/2}$
E0rms	$\langle \mathcal{E}_0^2 \rangle^{1/2}$
Ex11pt	$\mathcal{E}_x^{11}$

Ex21pt	$\mathcal{E}_x^{21}$
Ex12pt	$\mathcal{E}_x^{12}$
Ex22pt	$\mathcal{E}_x^{22}$
Ex0pt	$\mathcal{E}_x^{0}$
Ey11pt	$\mathcal{E}_y^{11}$
Ey21pt	$\mathcal{E}_y^{21}$
Ey12pt	$\mathcal{E}_y^{12}$
Ey22pt	$\mathcal{E}_y^{22}$
Ey0pt	$\mathcal{E}_y^{0}$
bamp	bamp
E111z	$\mathcal{E}_1^{11}$
E211z	$\mathcal{E}_2^{11}$
E311z	$\mathcal{E}_3^{11}$
E121z	$\mathcal{E}_1^{21}$
E221z	$\mathcal{E}_2^{21}$
E321z	$\mathcal{E}_3^{21}$
E112z	$\mathcal{E}_1^{12}$
E212z	$\mathcal{E}_2^{12}$
E312z	$\mathcal{E}_3^{12}$
E122z	$\mathcal{E}_1^{22}$
E222z	$\mathcal{E}_2^{22}$
E322z	$\mathcal{E}_3^{22}$
E10z	$\mathcal{E}_1^{0}$
E20z	$\mathcal{E}_2^{0}$
E30z	$\mathcal{E}_3^{0}$
EBpq	$\mathcal{E} \cdot \boldsymbol{B}^{pq}$
E0Um	$\mathcal{E}^0 \cdot \boldsymbol{U}$
E0Wm	$\mathcal{E}^0 \cdot \boldsymbol{W}$
bx0mz	$\langle b_x \rangle_{xy}$
by0mz	$\langle b_y \rangle_{xy}$
bz0mz	$\langle b_z \rangle_{xy}$
M11z	$\langle \mathcal{M}_{11} \rangle_{xy}$
M22z	$\langle \mathcal{M}_{22} \rangle_{xy}$
M33z	$\langle \mathcal{M}_{33} \rangle_{xy}$

Module 'testfield_x.f90'

alp11	α_{11}
alp21	α_{21}
alp31	α_{31}
alp12	α_{12}
alp22	α_{22}
alp32	α_{32}
eta11	$\eta_{11} k$
eta21	$\eta_{21} k$
eta12	$\eta_{12} k$
eta22	$\eta_{22} k$
alp11cc	$\alpha_{11} \cos^2 kx$
alp21sc	$\alpha_{21} \sin kx \cos kx$
alp12cs	$\alpha_{12} \cos kx \sin kx$
alp22ss	$\alpha_{22} \sin^2 kx$

eta11cc	$\eta_{11}\cos^2 kx$
eta21sc	$\eta_{21}\sin kx\cos kx$
eta12cs	$\eta_{12}\cos kx\sin kx$
eta22ss	$\eta_{22}\sin^2 kx$
alp11_x	$\alpha_{11}x$
alp21_x	$\alpha_{21}x$
alp12_x	$\alpha_{12}x$
alp22_x	$\alpha_{22}x$
eta11_x	$\eta_{11}kx$
eta21_x	$\eta_{21}kx$
eta12_x	$\eta_{12}kx$
eta22_x	$\eta_{22}kx$
alp11_x2	$\alpha_{11}x^2$
alp21_x2	$\alpha_{21}x^2$
alp12_x2	$\alpha_{12}x^2$
alp22_x2	$\alpha_{22}x^2$
eta11_x2	$\eta_{11}kx^2$
eta21_x2	$\eta_{21}kx^2$
eta12_x2	$\eta_{12}kx^2$
eta22_x2	$\eta_{22}kx^2$
b11rms	$\langle b_{11}^2\rangle^{1/2}$
b21rms	$\langle b_{21}^2\rangle^{1/2}$
b12rms	$\langle b_{12}^2\rangle^{1/2}$
b22rms	$\langle b_{22}^2\rangle^{1/2}$
b0rms	$\langle b_0^2\rangle^{1/2}$
E11rms	$\langle \mathcal{E}_{11}^2\rangle^{1/2}$
E21rms	$\langle \mathcal{E}_{21}^2\rangle^{1/2}$
E12rms	$\langle \mathcal{E}_{12}^2\rangle^{1/2}$
E22rms	$\langle \mathcal{E}_{22}^2\rangle^{1/2}$
E0rms	$\langle \mathcal{E}_0^2\rangle^{1/2}$
E111z	$\mathcal{E}_1^{11}$
E211z	$\mathcal{E}_2^{11}$
E311z	$\mathcal{E}_3^{11}$
E121z	$\mathcal{E}_1^{21}$
E221z	$\mathcal{E}_2^{21}$
E321z	$\mathcal{E}_3^{21}$
E112z	$\mathcal{E}_1^{12}$
E212z	$\mathcal{E}_2^{12}$
E312z	$\mathcal{E}_3^{12}$
E122z	$\mathcal{E}_1^{22}$
E222z	$\mathcal{E}_2^{22}$
E322z	$\mathcal{E}_3^{22}$
E10z	$\mathcal{E}_1^{0}$
E20z	$\mathcal{E}_2^{0}$
E30z	$\mathcal{E}_3^{0}$
EBpq	$\mathcal{E}\cdot\boldsymbol{B}^{pq}$
bx0mz	$\langle b_x\rangle_{xy}$
by0mz	$\langle b_y\rangle_{xy}$

bz0mz	$\langle b_z \rangle_{xy}$
alp11x	$\alpha_{11}(x,t)$
alp21x	$\alpha_{21}(x,t)$
alp12x	$\alpha_{12}(x,t)$
alp22x	$\alpha_{22}(x,t)$
eta11x	$\eta_{11}(x,t)$
eta21x	$\eta_{21}(x,t)$
eta12x	$\eta_{12}(x,t)$
eta22x	$\eta_{22}(x,t)$

Module 'testfield_xz.f90'

E111z	$\mathcal{E}_1^{11}$
E211z	$\mathcal{E}_2^{11}$
E311z	$\mathcal{E}_3^{11}$
E121z	$\mathcal{E}_1^{21}$
E221z	$\mathcal{E}_2^{21}$
E321z	$\mathcal{E}_3^{21}$
alp11	α_{11}
alp21	α_{21}
eta11	$\eta_{113}k$
eta21	$\eta_{213}k$
b11rms	$\langle b_{11}^2 \rangle$
b21rms	$\langle b_{21}^2 \rangle$

Module 'testfield_z.f90'

alp11	α_{11}
alp21	α_{21}
alp31	α_{31}
alp12	α_{12}
alp22	α_{22}
alp32	α_{32}
alp13	α_{13}
alp23	α_{23}
eta11	$\eta_{113}k$ **or** $\eta_{11}k$ **if** leta_rank2=T
eta21	$\eta_{213}k$ **or** $\eta_{21}k$ **if** leta_rank2=T
eta31	$\eta_{313}k$
eta12	$\eta_{123}k$ **or** $\eta_{12}k$ **if** leta_rank2=T
eta22	$\eta_{223}k$ **or** $\eta_{22}k$ **if** leta_rank2=T
eta32	$\eta_{323}k$
alp11cc	$\alpha_{11}\cos^2 kz$
alp21sc	$\alpha_{21}\sin kz\cos kz$
alp12cs	$\alpha_{12}\cos kz\sin kz$
alp22ss	$\alpha_{22}\sin^2 kz$
eta11cc	$\eta_{11}\cos^2 kz$
eta21sc	$\eta_{21}\sin kz\cos kz$
eta12cs	$\eta_{12}\cos kz\sin kz$
eta22ss	$\eta_{22}\sin^2 kz$
s2kzDFm	$\langle \sin 2kz \nabla\cdot F \rangle$
M11	$\mathcal{M}_{11}$
M22	$\mathcal{M}_{22}$

M33	$\mathcal{M}_{33}$
M11cc	$\mathcal{M}_{11}\cos^2 kz$
M11ss	$\mathcal{M}_{11}\sin^2 kz$
M22cc	$\mathcal{M}_{22}\cos^2 kz$
M22ss	$\mathcal{M}_{22}\sin^2 kz$
M12cs	$\mathcal{M}_{12}\cos kz \sin kz$
bx11pt	b_x^{11}
bx21pt	b_x^{21}
bx12pt	b_x^{12}
bx22pt	b_x^{22}
bx0pt	b_x^{0}
by11pt	b_y^{11}
by21pt	b_y^{21}
by12pt	b_y^{12}
by22pt	b_y^{22}
by0pt	b_y^{0}
b11rms	$\langle b_{11}^2 \rangle^{1/2}$
b21rms	$\langle b_{21}^2 \rangle^{1/2}$
b12rms	$\langle b_{12}^2 \rangle^{1/2}$
b22rms	$\langle b_{22}^2 \rangle^{1/2}$
b0rms	$\langle b_{0}^2 \rangle^{1/2}$
jb0m	$\langle j b_0 \rangle$
E11rms	$\langle \mathcal{E}_{11}^2 \rangle^{1/2}$
E21rms	$\langle \mathcal{E}_{21}^2 \rangle^{1/2}$
E12rms	$\langle \mathcal{E}_{12}^2 \rangle^{1/2}$
E22rms	$\langle \mathcal{E}_{22}^2 \rangle^{1/2}$
E0rms	$\langle \mathcal{E}_{0}^2 \rangle^{1/2}$
Ex11pt	$\mathcal{E}_x^{11}$
Ex21pt	$\mathcal{E}_x^{21}$
Ex12pt	$\mathcal{E}_x^{12}$
Ex22pt	$\mathcal{E}_x^{22}$
Ex0pt	$\mathcal{E}_x^{0}$
Ey11pt	$\mathcal{E}_y^{11}$
Ey21pt	$\mathcal{E}_y^{21}$
Ey12pt	$\mathcal{E}_y^{12}$
Ey22pt	$\mathcal{E}_y^{22}$
Ey0pt	$\mathcal{E}_y^{0}$
bamp	bamp
alp11z	$\alpha_{11}(z,t)$
alp21z	$\alpha_{21}(z,t)$
alp12z	$\alpha_{12}(z,t)$
alp22z	$\alpha_{22}(z,t)$
alp13z	$\alpha_{13}(z,t)$
alp23z	$\alpha_{23}(z,t)$
eta11z	$\eta_{11}(z,t)$
eta21z	$\eta_{21}(z,t)$
eta12z	$\eta_{12}(z,t)$
eta22z	$\eta_{22}(z,t)$

E111z	$\mathcal{E}_1^{11}$
E211z	$\mathcal{E}_2^{11}$
E311z	$\mathcal{E}_3^{11}$
E121z	$\mathcal{E}_1^{21}$
E221z	$\mathcal{E}_2^{21}$
E321z	$\mathcal{E}_3^{21}$
E112z	$\mathcal{E}_1^{12}$
E212z	$\mathcal{E}_2^{12}$
E312z	$\mathcal{E}_3^{12}$
E122z	$\mathcal{E}_1^{22}$
E222z	$\mathcal{E}_2^{22}$
E322z	$\mathcal{E}_3^{22}$
E10z	$\mathcal{E}_1^{0}$
E20z	$\mathcal{E}_2^{0}$
E30z	$\mathcal{E}_3^{0}$
EBpq	$\mathcal{E} \cdot \boldsymbol{B}^{pq}$
E0Um	$\mathcal{E}^0 \cdot \boldsymbol{U}$
E0Wm	$\mathcal{E}^0 \cdot \boldsymbol{W}$
bx0mz	$\langle b_x \rangle_{xy}$
by0mz	$\langle b_y \rangle_{xy}$
bz0mz	$\langle b_z \rangle_{xy}$
M11z	$\langle \mathcal{M}_{11} \rangle_{xy}$
M22z	$\langle \mathcal{M}_{22} \rangle_{xy}$
M33z	$\langle \mathcal{M}_{33} \rangle_{xy}$

Module 'testflow_z.f90'

gal	GAL-coefficients, couple $\overline{F}$ and $\overline{U}$
aklam	AKA-λ-tensor, couples $\overline{F}$ and $\overline{W} = \nabla \times \overline{U}$
gamma	γ-vector, couples $\overline{F}$ and $\nabla \cdot \overline{U}$
nu	ν-tensor, couples $\overline{F}$ and $\partial^2 \overline{U}/\partial z^2$
zeta	ζ-vector, couples $\overline{F}$ and $\overline{G}_z = \nabla_z \overline{H}$
xi	ξ-vector, couples $\overline{F}$ and $\partial^2 \overline{H}/\partial z^2$
aklamQ	$aklam^Q$-vector, couples $\overline{Q}$ and $\overline{W}$
gammaQ	γ^Q-scalar, couples $\overline{Q}$ and $\nabla \cdot \overline{U} = dU_z/dz$
nuQ	ν^Q-vector, couples $\overline{Q}$ and $\partial^2 \overline{U}/\partial z^2$
zetaQ	ζ^Q-scalar, couples $\overline{Q}$ and $\overline{G}_z$
xiQ	ξ^Q-scalar, couples $\overline{Q}$ and $\partial^2 \overline{H}/\partial z^2$
	$\alpha_{K,ij}\ \gamma_i\ \nu_{ij}\ \zeta_i\ \xi_i\ \nu_i^Q\ aklam_i^Q\ \mathcal{F}_i^{pq}\ \mathcal{Q}^{pq}\ \langle u^{pq2} \rangle\ \langle h^{pq2} \rangle$
ux0mz	$\langle u_x \rangle_{xy}$
uy0mz	$\langle u_y \rangle_{xy}$
uz0mz	$\langle u_z \rangle_{xy}$

Module 'testperturb.f90'

alp11	α_{11}
alp21	α_{21}
alp31	α_{31}
alp12	α_{12}
alp22	α_{22}
alp32	α_{32}
eta11	$\eta_{113} k$

eta21	$\eta_{213}k$
eta31	$\eta_{313}k$
eta12	$\eta_{123}k$
eta22	$\eta_{223}k$
eta32	$\eta_{323}k$

Module '`testscalar.f90`'

gam11	$\gamma_1^{(1)}$
gam12	$\gamma_2^{(1)}$
gam13	$\gamma_3^{(1)}$
gam21	$\gamma_1^{(2)}$
gam22	$\gamma_2^{(2)}$
gam23	$\gamma_3^{(2)}$
gam31	$\gamma_1^{(3)}$
gam32	$\gamma_2^{(3)}$
gam33	$\gamma_3^{(3)}$
kap11	κ_{11}
kap21	κ_{21}
kap31	κ_{31}
kap12	κ_{12}
kap22	κ_{22}
kap32	κ_{32}
kap13	κ_{13}
kap23	κ_{23}
kap33	κ_{33}
gam11z	$\gamma_1^{(1)}(z,t)$
gam12z	$\gamma_2^{(1)}(z,t)$
gam13z	$\gamma_3^{(1)}(z,t)$
gam21z	$\gamma_1^{(2)}(z,t)$
gam22z	$\gamma_2^{(2)}(z,t)$
gam23z	$\gamma_3^{(2)}(z,t)$
gam31z	$\gamma_1^{(3)}(z,t)$
gam32z	$\gamma_2^{(3)}(z,t)$
gam33z	$\gamma_3^{(3)}(z,t)$
kap11z	$\kappa_{11}(z,t)$
kap21z	$\kappa_{21}(z,t)$
kap31z	$\kappa_{31}(z,t)$
kap12z	$\kappa_{12}(z,t)$
kap22z	$\kappa_{22}(z,t)$
kap32z	$\kappa_{32}(z,t)$
kap13z	$\kappa_{13}(z,t)$
kap23z	$\kappa_{23}(z,t)$
kap33z	$\kappa_{33}(z,t)$
mgam33	$\tilde{\gamma}_{33}$
mkap33	$\tilde{\kappa}_{33}$
ngam33	$\hat{\gamma}_{33}$
nkap33	$\hat{\kappa}_{33}$
c1rms	$\langle c_1^2 \rangle^{1/2}$

c2rms	$\langle c_2^2 \rangle^{1/2}$
c3rms	$\langle c_3^2 \rangle^{1/2}$
c4rms	$\langle c_4^2 \rangle^{1/2}$
c5rms	$\langle c_5^2 \rangle^{1/2}$
c6rms	$\langle c_6^2 \rangle^{1/2}$
c1pt	c^1
c2pt	c^2
c3pt	c^3
c4pt	c^4
c5pt	c^5
c6pt	c^6
F11z	$\mathcal{F}_1^1$
F21z	$\mathcal{F}_2^1$
F31z	$\mathcal{F}_3^1$
F12z	$\mathcal{F}_1^2$
F22z	$\mathcal{F}_2^2$
F32z	$\mathcal{F}_3^2$

Module 'testscalar_axisym.f90'

muc1	$\mu^{(c1)}$
muc2	$\mu^{(c2)}$
gamc	$\gamma^{(c)}$
kapcPERP1	$\kappa_\perp^{(1)}$
kapcPERP2	$\kappa_\perp^{(2)}$
kapcPARA	$\kappa_\parallel$
mucz	$\mu^{(c)}(z,t)$
gamcz	$\gamma^{(c)}(z,t)$
kapcPERPz	$\kappa_\perp(z,t)$
kapcPARAz	$\kappa_\parallel(z,t)$
gam11	$\gamma_1^{(1)}$
gam12	$\gamma_2^{(1)}$
gam13	$\gamma_3^{(1)}$
gam21	$\gamma_1^{(2)}$
gam22	$\gamma_2^{(2)}$
gam23	$\gamma_3^{(2)}$
gam31	$\gamma_1^{(3)}$
gam32	$\gamma_2^{(3)}$
gam33	$\gamma_3^{(3)}$
kap11	κ_{11}
kap21	κ_{21}
kap31	κ_{31}
kap12	κ_{12}
kap22	κ_{22}
kap32	κ_{32}
kap13	κ_{13}
kap23	κ_{23}
kap33	κ_{33}
gam11z	$\gamma_1^{(1)}(z,t)$

gam12z	$\gamma_2^{(1)}(z,t)$
gam13z	$\gamma_3^{(1)}(z,t)$
gam21z	$\gamma_1^{(2)}(z,t)$
gam22z	$\gamma_2^{(2)}(z,t)$
gam23z	$\gamma_3^{(2)}(z,t)$
gam31z	$\gamma_1^{(3)}(z,t)$
gam32z	$\gamma_2^{(3)}(z,t)$
gam33z	$\gamma_3^{(3)}(z,t)$
gam3z	$\gamma^{(c)}(z,t)$
kap11z	$\kappa_{11}(z,t)$
kap21z	$\kappa_{21}(z,t)$
kap31z	$\kappa_{31}(z,t)$
kap12z	$\kappa_{12}(z,t)$
kap22z	$\kappa_{22}(z,t)$
kap32z	$\kappa_{32}(z,t)$
kap13z	$\kappa_{13}(z,t)$
kap23z	$\kappa_{23}(z,t)$
kap33z	$\kappa_{33}(z,t)$
mgam33	$\tilde{\gamma}_{33}$
mkap33	$\tilde{\kappa}_{33}$
ngam33	$\hat{\gamma}_{33}$
nkap33	$\hat{\kappa}_{33}$
c1rms	$\langle c_1^2 \rangle^{1/2}$
c2rms	$\langle c_2^2 \rangle^{1/2}$
c3rms	$\langle c_3^2 \rangle^{1/2}$
c4rms	$\langle c_4^2 \rangle^{1/2}$
c5rms	$\langle c_5^2 \rangle^{1/2}$
c6rms	$\langle c_6^2 \rangle^{1/2}$
c1pt	c^1
c2pt	c^2
c3pt	c^3
c4pt	c^4
c5pt	c^5
c6pt	c^6
F11z	$\mathcal{F}_1^1$
F21z	$\mathcal{F}_2^1$
F31z	$\mathcal{F}_3^1$
F12z	$\mathcal{F}_1^2$
F22z	$\mathcal{F}_2^2$
F32z	$\mathcal{F}_3^2$

Module '`thermal_energy.f90`'

TTmax	$\max(T)$
TTmin	$\min(T)$
ppm	$\langle p \rangle$
TTm	$\langle T \rangle$
ethm	$\langle e_{\rm th} \rangle = \langle c_v \rho T \rangle$ (mean thermal energy)
ethmin	$\min e_{\rm th}$

ethmax	$\max e_{\rm th}$
eem	$\langle e\rangle = \langle c_v T\rangle$ (mean internal energy)
	Module 'viscosity.f90'
nu_tdep	time-dependent viscosity
fviscm	Mean value of viscous acceleration
fviscmin	Min value of viscous acceleration
fviscmax	Max value of viscous acceleration
fviscrmsx	Rms value of viscous acceleration for the vis_xaver_range
num	Mean value of viscosity
nusmagm	Mean value of Smagorinsky viscosity
nusmagmin	Min value of Smagorinsky viscosity
nusmagmax	Max value of Smagorinsky viscosity
visc_heatm	Mean value of viscous heating
qfviscm	$\langle \boldsymbol{q}\cdot\boldsymbol{f}_{\rm visc}\rangle$
Sij2m	$\langle \mathsf{S}^2\rangle$
epsK	$\langle 2\nu\varrho\mathsf{S}^2\rangle$
dtnu	$\delta t/[c_{\delta t,\rm v}\,\delta x^2/\nu_{\max}]$ (time step relative to viscous time step; see § 5.15)
meshRemax	Max mesh Reynolds number
Reshock	Mesh Reynolds number at shock

J.4 List of parameters for 'video.in'

The following table lists all (at the time of writing, May 16, 2014) possible inputs to the file 'video.in'.

Variable	*Meaning*
	Module 'hydro.f90'
uu	velocity vector $\boldsymbol{u}$; writes all three components separately to files 'u[xyz].{xz,yz,xy,xy2}'
u2	kinetic energy density $\boldsymbol{u}^2$; writes 'u2.{xz,yz,xy,xy2}'
oo	vorticity vector $\boldsymbol{\omega}=\nabla\times\boldsymbol{u}$; writes 'o[xyz].{xz,yz,xy,xy2}'
o2	enstrophy $\omega^2=\vert\nabla\times\boldsymbol{u}\vert^2$; writes 'o2.{xz,yz,xy,xy2}'
divu	$\boldsymbol{\nabla}\cdot\boldsymbol{u}$; writes 'divu.{xz,yz,xy,xy2}'
mach	Mach number squared Ma^2; writes 'mach.{xz,yz,xy,xy2}'
	Module 'density.f90'
lnrho	logarithmic density $\ln\rho$; writes 'lnrho.{xz,yz,xy,xy2}'
rho	density ρ; writes 'rho.{xz,yz,xy,xy2}'
	Module 'entropy.f90'
ss	entropy s; writes 'ss.{xz,yz,xy,xy2}'
pp	pressure p; writes 'pp.{xz,yz,xy,xy2}'
	Module 'temperature_idealgas.f90'
lnTT	logarithmic temperature $\ln T$; writes 'lnTT.{xz,yz,xy,xy2}'
TT	temperature T; writes 'TT.{xz,yz,xy,xy2}'

	Module 'shock.f90'
shock	shock viscosity $\nu_{\rm shock}$; writes 'shock.{xz,yz,xy,xy2}'
	Module 'eos_ionization.f90'
yH	ionization fraction $y_{\rm H}$; writes 'yH.{xz,yz,xy,xy2}'
	Module 'radiation_ray.f90'
Qrad	radiative heating rate $Q_{\rm rad}$; writes 'Qrad.{xz,yz,xy,xy2}'
Isurf	surface intensity $I_{\rm surf}$ (?); writes 'Isurf.xz'
	Module 'magnetic.f90'
aa	magnetic vector potential $\boldsymbol{A}$; writes 'aa[xyz].{xz,yz,xy,xy2}'
bb	magnetic flux density $\boldsymbol{B}$; writes 'bb[xyz].{xz,yz,xy,xy2}'
b2	magnetic energy density $\boldsymbol{B}^2$; writes 'b2.{xz,yz,xy,xy2}'
jj	current density $\boldsymbol{j}$; writes 'jj[xyz].{xz,yz,xy,xy2}'
j2	current density squared $\boldsymbol{j}^2$; writes 'j2.{xz,yz,xy,xy2}'
jb	$\boldsymbol{jB}$; writes 'jb.{xz,yz,xy,xy2}'
beta1	inverse plasma beta $\boldsymbol{B}^2/(2\mu_0 p)$; writes 'beta1.{xz,yz,xy,xy2}'
Poynting	Poynting vector $\eta\boldsymbol{j}\times\boldsymbol{B}-(\boldsymbol{u}\times\boldsymbol{B})\times\boldsymbol{B}/\mu_0$; writes 'Poynting[xyz].{xz,yz,xy,xy2}'
ab	magnetic helicity density $\boldsymbol{A}\cdot\boldsymbol{B}$; writes 'ab[xyz].{xz,yz,xy,xy2}'
	Module 'pscalar.f90'
lncc	logarithmic density of passive scalar $\ln c$; writes 'lncc.{xz,yz,xy,xy2}'
	Module 'cosmicray.f90'
ecr	energy $e_{\rm cr}$ of cosmic rays (?); writes 'ec.{xz,yz,xy,xy2}'

J.5 List of parameters for 'phiaver.in'

The following table lists all (at the time of writing, November 2003) possible inputs to the file 'phiaver.in'.

Variable	*Meaning*
	Module 'cdata.f90'
rcylmphi	cylindrical radius $\varpi=\sqrt{x^2+y^2}$ (useful for debugging azimuthal averages)
phimphi	azimuthal angle $\varphi=\arctan\frac{y}{x}$ (useful for debugging)
zmphi	z-coordinate (useful for debugging)
rmphi	spherical radius $r=\sqrt{\varpi^2+z^2}$ (useful for debugging)
	Module 'hydro.f90'
urmphi	$\langle u_\varpi\rangle_\varphi$ [cyl. polar coords (ϖ,φ,z)]
upmphi	$\langle u_\varphi\rangle_\varphi$

uzmphi	$\langle u_z \rangle_\varphi$
ursphmphi	$\langle u_r \rangle_\varphi$
uthmphi	$\langle u_\vartheta \rangle_\varphi$
uumphi	shorthand for *urmphi*, *upmphi* and *uzmphi* together
uusphmphi	shorthand for *ursphmphi*, *uthmphi* and *upmphi* together
u2mphi	$\langle \boldsymbol{u}^2 \rangle_\varphi$
	Module 'density.f90'
lnrhomphi	$\langle \ln \varrho \rangle_\varphi$
rhomphi	$\langle \varrho \rangle_\varphi$
	Module 'entropy.f90'
ssmphi	$\langle s \rangle_\varphi$
cs2mphi	$\langle c_s^2 \rangle_\varphi$
	Module 'magnetic.f90'
jbmphi	$\langle \boldsymbol{J} \cdot \boldsymbol{B} \rangle_\varphi$
brmphi	$\langle B_\varpi \rangle_\varphi$ [cyl. polar coords (ϖ, φ, z)]
bpmphi	$\langle B_\varphi \rangle_\varphi$
bzmphi	$\langle B_z \rangle_\varphi$
bbmphi	shorthand for *brmphi*, *bpmphi* and *bzmphi* together
bbsphmphi	shorthand for *brsphmphi*, *bthmphi* and *bpmphi* together
b2mphi	$\langle \boldsymbol{B}^2 \rangle_\varphi$
brsphmphi	$\langle B_r \rangle_\varphi$
bthmphi	$\langle B_\vartheta \rangle_\varphi$
	Module 'anelastic.f90'
lnrhomphi	$\langle \ln \varrho \rangle_\varphi$
rhomphi	$\langle \varrho \rangle_\varphi$
	Module 'entropy_anelastic.f90'
ssmphi	$\langle s \rangle_\varphi$
cs2mphi	$\langle c_s^2 \rangle_\varphi$

J.6 List of parameters for 'xyaver.in'

The following table lists possible inputs to the file 'xyaver.in'. This list is not complete and maybe outdated.

Variable	*Meaning*
	Module 'hydro.f90'
u2mz	$\langle \boldsymbol{u}^2 \rangle_{xy}$
o2mz	$\langle \boldsymbol{W}^2 \rangle_{xy}$
divu2mz	$\langle (\nabla \cdot \boldsymbol{u})^2 \rangle_{xy}$
curlru2mz	$\langle (\nabla \times \varrho \boldsymbol{U})^2 \rangle_{xy}$
divru2mz	$\langle (\nabla \cdot \varrho \boldsymbol{u})^2 \rangle_{xy}$
fmasszmz	$\langle \varrho u_z \rangle_{xy}$

fkinzmz	$\left\langle \frac{1}{2}\varrho \boldsymbol{u}^2 u_z \right\rangle_{xy}$
uxmz	$\langle u_x \rangle_{xy}$ (horiz. averaged x velocity)
uymz	$\langle u_y \rangle_{xy}$
uzmz	$\langle u_z \rangle_{xy}$
divumz	$\langle \mathrm{div} \boldsymbol{u} \rangle_{xy}$
uzdivumz	$\langle u_z \mathrm{div} \boldsymbol{u} \rangle_{xy}$
oxmz	$\langle \omega_x \rangle_{xy}$
oymz	$\langle \omega_y \rangle_{xy}$
ozmz	$\langle \omega_z \rangle_{xy}$
ux2mz	$\langle u_x^2 \rangle_{xy}$
uy2mz	$\langle u_y^2 \rangle_{xy}$
uz2mz	$\langle u_z^2 \rangle_{xy}$
ruxmz	$\langle \varrho u_x \rangle_{xy}$
ruymz	$\langle \varrho u_y \rangle_{xy}$
ruzmz	$\langle \varrho u_z \rangle_{xy}$
rux2mz	$\langle \varrho u_x^2 \rangle_{xy}$
ruy2mz	$\langle \varrho u_y^2 \rangle_{xy}$
ruz2mz	$\langle \varrho u_z^2 \rangle_{xy}$
uxuymz	$\langle u_x u_y \rangle_{xy}$
uxuzmz	$\langle u_x u_z \rangle_{xy}$
uyuzmz	$\langle u_y u_z \rangle_{xy}$
ruxuymz	$\langle \rho u_x u_y \rangle_{xy}$
ruxuzmz	$\langle \rho u_x u_z \rangle_{xy}$
ruyuzmz	$\langle \rho u_y u_z \rangle_{xy}$
oxuxxmz	$\langle \omega_x u_{x,x} \rangle_{xy}$
oyuxymz	$\langle \omega_y u_{x,y} \rangle_{xy}$
oxuyxmz	$\langle \omega_x u_{y,x} \rangle_{xy}$
oyuyymz	$\langle \omega_y u_{y,y} \rangle_{xy}$
oxuzxmz	$\langle \omega_x u_{z,x} \rangle_{xy}$
oyuzymz	$\langle \omega_y u_{z,y} \rangle_{xy}$
ekinmz	$\left\langle \frac{1}{2}\varrho \boldsymbol{u}^2 \right\rangle_{xy}$
oumz	$\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle_{xy}$
fkinxmx	$\left\langle \frac{1}{2}\varrho \boldsymbol{u}^2 u_x \right\rangle_{yz}$

Module ‘`density.f90`’

rhomz	$\langle \varrho \rangle_{xy}$
rho2mz	$\langle \varrho^2 \rangle_{xy}$
rho2mx	$\langle \varrho^2 \rangle_{yz}$

Module ‘`entropy.f90`’

fradz	F_{rad}
fconvz	$\langle c_p \varrho u_z T \rangle_{xy}$
ssmz	$\langle s \rangle_{xy}$
ppmz	$\langle p \rangle_{xy}$
TTmz	$\langle T \rangle_{xy}$
TT2mz	$\langle T^2 \rangle_{xy}$
uxTTmz	$\langle u_x T \rangle_{xy}$
uyTTmz	$\langle u_y T \rangle_{xy}$

uzTTmz	$\langle u_z T \rangle_{xy}$
fradz_kramers	$F_{\rm rad}$ (from Kramers' opacity)
fradz_Kprof	$F_{\rm rad}$ (from Kprof)
fradz_constchi	$F_{\rm rad}$ (from chi_const)
fturbz	$\langle \varrho T \chi_t \nabla_z s \rangle_{xy}$ (turbulent heat flux)
dcoolz	surface cooling flux

Module 'magnetic.f90'

axmz	$\langle \mathcal{A}_x \rangle_{xy}$
aymz	$\langle \mathcal{A}_y \rangle_{xy}$
azmz	$\langle \mathcal{A}_z \rangle_{xy}$
abuxmz	$\langle (\boldsymbol{A} \cdot \boldsymbol{B}) u_x \rangle_{xy}$
abuymz	$\langle (\boldsymbol{A} \cdot \boldsymbol{B}) u_y \rangle_{xy}$
abuzmz	$\langle (\boldsymbol{A} \cdot \boldsymbol{B}) u_z \rangle_{xy}$
uabxmz	$\langle (\boldsymbol{u} \cdot \boldsymbol{A}) B_x \rangle_{xy}$
uabymz	$\langle (\boldsymbol{u} \cdot \boldsymbol{A}) B_y \rangle_{xy}$
uabzmz	$\langle (\boldsymbol{u} \cdot \boldsymbol{A}) B_z \rangle_{xy}$
bbxmz	$\langle \mathcal{B}'_x \rangle_{xy}$
bbymz	$\langle \mathcal{B}'_y \rangle_{xy}$
bbzmz	$\langle \mathcal{B}'_z \rangle_{xy}$
bxmz	$\langle \mathcal{B}_x \rangle_{xy}$
bymz	$\langle \mathcal{B}_y \rangle_{xy}$
bzmz	$\langle \mathcal{B}_z \rangle_{xy}$
jxmz	$\langle \mathcal{J}_x \rangle_{xy}$
jymz	$\langle \mathcal{J}_y \rangle_{xy}$
jzmz	$\langle \mathcal{J}_z \rangle_{xy}$
Exmz	$\langle \mathcal{E}_x \rangle_{xy}$
Eymz	$\langle \mathcal{E}_y \rangle_{xy}$
Ezmz	$\langle \mathcal{E}_z \rangle_{xy}$
bx2mz	$\langle B_x^2 \rangle_{xy}$
by2mz	$\langle B_y^2 \rangle_{xy}$
bz2mz	$\langle B_z^2 \rangle_{xy}$
bx2rmz	$\langle B_x^2/\varrho \rangle_{xy}$
by2rmz	$\langle B_y^2/\varrho \rangle_{xy}$
bz2rmz	$\langle B_z^2/\varrho \rangle_{xy}$
beta1mz	$\langle (B^2/2\mu_0)/p \rangle_{xy}$
jbmz	$\langle \boldsymbol{J} \cdot \boldsymbol{B} \rangle \,|_{xy}$
d6abmz	$\langle \nabla^6 \boldsymbol{A} \cdot \boldsymbol{B} \rangle \,|_{xy}$
d6amz1	$\langle \nabla^6 \boldsymbol{A} \rangle_{xy} \,|_x$
d6amz2	$\langle \nabla^6 \boldsymbol{A} \rangle_{xy} \,|_y$
d6amz3	$\langle \nabla^6 \boldsymbol{A} \rangle_{xy} \,|_z$
abmz	$\langle \boldsymbol{A} \cdot \boldsymbol{B} \rangle \,|_{xy}$
ubmz	$\langle \boldsymbol{u} \cdot \boldsymbol{B} \rangle \,|_{xy}$
uamz	$\langle \boldsymbol{u} \cdot \boldsymbol{A} \rangle \,|_{xy}$
uxbxmz	$\langle u_x b_x \rangle \,|_{xy}$
uybxmz	$\langle u_y b_x \rangle \,|_{xy}$
uzbxmz	$\langle u_z b_x \rangle \,|_{xy}$
uxbymz	$\langle u_x b_y \rangle \,|_{xy}$

uybymz	$\langle u_y b_y \rangle\,|_{xy}$
uzbymz	$\langle u_z b_y \rangle\,|_{xy}$
uxbzmz	$\langle u_x b_z \rangle\,|_{xy}$
uybzmz	$\langle u_y b_z \rangle\,|_{xy}$
uzbzmz	$\langle u_z b_z \rangle\,|_{xy}$
examz1	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle_{xy}\,|_x$
examz2	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle_{xy}\,|_y$
examz3	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle_{xy}\,|_z$
e3xamz1	$\langle \boldsymbol{E}_{hyper3} \times \boldsymbol{A} \rangle_{xy}\,|_x$
e3xamz2	$\langle \boldsymbol{E}_{hyper3} \times \boldsymbol{A} \rangle_{xy}\,|_y$
e3xamz3	$\langle \boldsymbol{E}_{hyper3} \times \boldsymbol{A} \rangle_{xy}\,|_z$
etatotalmz	$\langle \eta \rangle_{xy}$
bxbymz	$\langle B_x B_y \rangle_{xy}$
bxbzmz	$\langle B_x B_z \rangle_{xy}$
bybzmz	$\langle B_y B_z \rangle_{xy}$
b2mz	$\langle \boldsymbol{B}^2 \rangle_{xy}$
bf2mz	$\langle \boldsymbol{B}'^2 \rangle_{xy}$
j2mz	$\langle \boldsymbol{j}^2 \rangle_{xy}$
poynzmz	Averaged poynting flux in z direction
epsMmz	$\langle \eta \mu_0 \boldsymbol{j}^2 \rangle_{xy}$

Module ‘`bfield.f90`’

bmz	$\langle B \rangle_{xy}$
b2mz	$\langle B^2 \rangle_{xy}$
bxmz	$\langle B_x \rangle_{xy}$
bymz	$\langle B_y \rangle_{xy}$
bzmz	$\langle B_z \rangle_{xy}$
bx2mz	$\langle B_x^2 \rangle_{xy}$
by2mz	$\langle B_y^2 \rangle_{xy}$
bz2mz	$\langle B_z^2 \rangle_{xy}$
bxbymz	$\langle B_x B_y \rangle_{xy}$
bxbzmz	$\langle B_x B_z \rangle_{xy}$
bybzmz	$\langle B_y B_z \rangle_{xy}$
betamz	$\langle \beta \rangle_{xy}$
beta2mz	$\langle \beta^2 \rangle_{xy}$

Module ‘`density_stratified.f90`’

drhomz	$\langle \Delta\rho/\rho_0 \rangle_{xy}$
drho2mz	$\langle (\Delta\rho/\rho_0)^2 \rangle_{xy}$

Module ‘`gravity_simple.f90`’

epotmz	$\langle \varrho \Phi_{\rm grav} \rangle_{xy}$
epotuzmz	$\langle \varrho \Phi_{\rm grav} u_z \rangle_{xy}$ (potential energy flux)

Module ‘`meanfield.f90`’

qpmz	$\langle q_p \rangle_{xy}$

Module ‘`shock_highorder.f90`’

Module ‘`temperature_idealgas.f90`’

ppmz	$\langle p \rangle_{xy}$
TTmz	$\langle T \rangle_{xy}$
ethmz	$\langle e_{\rm th} \rangle_{xy}$
fpresxmz	$\langle (\nabla p)_x \rangle_{xy}$
fpresymz	$\langle (\nabla p)_y \rangle_{xy}$
fpreszmz	$\langle (\nabla p)_z \rangle_{xy}$
TT2mz	$\langle T^2 \rangle_{xy}$
uxTmz	$\langle u_x T \rangle_{xy}$
uyTmz	$\langle u_y T \rangle_{xy}$
uzTmz	$\langle u_z T \rangle_{xy}$
fradz	$F_{\rm rad}$
Module 'temperature_ionization.f90'	
mumz	$\langle \mu \rangle_{xy}$
TTmz	$\langle T \rangle_{xy}$
Module 'thermal_energy.f90'	
ppmz	$\langle p \rangle_{xy}$
TTmz	$\langle T \rangle_{xy}$
Module 'viscosity.f90'	
fviscmz	$\langle 2\nu\varrho u_i \mathcal{S}_{iz} \rangle_{xy}$ (z-component of viscous flux)
epsKmz	$\langle 2\nu\varrho \mathbf{S}^2 \rangle_{xy}$

J.7 List of parameters for 'xzaver.in'

The following table lists possible inputs to the file 'xzaver.in'. This list is not complete and maybe outdated.

Variable	*Meaning*
Module 'hydro.f90'	
uxmy	$\langle u_x \rangle_{xz}$
uymy	$\langle u_y \rangle_{xz}$
uzmy	$\langle u_z \rangle_{xz}$
oumy	$\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle_{xz}$
Module 'density.f90'	
rhomy	$\langle \varrho \rangle_{xz}$
Module 'entropy.f90'	
ssmy	$\langle s \rangle_{xz}$
ppmy	$\langle p \rangle_{xz}$
TTmy	$\langle T \rangle_{xz}$
Module 'magnetic.f90'	
bxmy	$\langle B_x \rangle_{xz}$
bymy	$\langle B_y \rangle_{xz}$

bzmy	$\langle B_z \rangle_{xz}$
bx2my	$\langle B_x^2 \rangle_{xz}$
by2my	$\langle B_y^2 \rangle_{xz}$
bz2my	$\langle B_z^2 \rangle_{xz}$
bxbymy	$\langle B_x B_y \rangle_{xz}$
bxbzmy	$\langle B_x B_z \rangle_{xz}$
bybzmy	$\langle B_y B_z \rangle_{xz}$
	Module 'density_stratified.f90'
drhomy	$\langle \Delta\rho/\rho_0 \rangle_{xz}$
drho2my	$\langle (\Delta\rho/\rho_0)^2 \rangle_{xz}$
	Module 'gravity_simple.f90'
epotmy	$\langle \varrho \Phi_{\rm grav} \rangle_{xz}$
	Module 'shock_highorder.f90'
	Module 'temperature_idealgas.f90'
ppmy	$\langle p \rangle_{xz}$
TTmy	$\langle T \rangle_{xz}$
	Module 'thermal_energy.f90'
ppmy	$\langle p \rangle_{xz}$
TTmy	$\langle T \rangle_{xz}$

J.8 List of parameters for 'yzaver.in'

The following table lists possible inputs to the file 'yzaver.in'. This list is not complete and maybe outdated.

Variable	*Meaning*
	Module 'hydro.f90'
uxmx	$\langle u_x \rangle_{yz}$
uymx	$\langle u_y \rangle_{yz}$
uzmx	$\langle u_z \rangle_{yz}$
ruxmx	$\langle \varrho u_x \rangle_{yz}$
ruymx	$\langle \varrho u_y \rangle_{yz}$
ruzmx	$\langle \varrho u_z \rangle_{yz}$
ux2mx	$\langle u_x^2 \rangle_{yz}$
uy2mx	$\langle u_y^2 \rangle_{yz}$
uz2mx	$\langle u_z^2 \rangle_{yz}$
ox2mx	$\langle \omega_x^2 \rangle_{yz}$
oy2mx	$\langle \omega_y^2 \rangle_{yz}$
oz2mx	$\langle \omega_z^2 \rangle_{yz}$
oumx	$\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle_{yz}$
ekinmx	$\langle \frac{1}{2} \rho u^2 \rangle_{xy}$
	Module 'density.f90'

rhomx	$\langle \varrho \rangle_{yz}$
	Module 'entropy.f90'
ssmx	$\langle s \rangle_{yz}$
ppmx	$\langle p \rangle_{yz}$
TTmx	$\langle T \rangle_{yz}$
uxTTmx	$\langle u_x T \rangle_{yz}$
uyTTmx	$\langle u_y T \rangle_{yz}$
uzTTmx	$\langle u_z T \rangle_{yz}$
fconvxmx	$\langle c_p \varrho u_x T \rangle_{yz}$
fradmx	$\langle F_{\rm rad} \rangle_{yz}$
fturbmx	$\langle \varrho T \chi_t \nabla_x s \rangle_{yz}$ (turbulent heat flux)
dcoolx	surface cooling flux
	Module 'magnetic.f90'
bxmx	$\langle B_x \rangle_{yz}$
bymx	$\langle B_y \rangle_{yz}$
bzmx	$\langle B_z \rangle_{yz}$
bx2mx	$\langle B_x^2 \rangle_{yz}$
by2mx	$\langle B_y^2 \rangle_{yz}$
bz2mx	$\langle B_z^2 \rangle_{yz}$
bxbymx	$\langle B_x B_y \rangle_{yz}$
etatotalmx	$\langle \eta \rangle_{yz}$
	Module 'bfield.f90'
bmx	$\langle B \rangle_{yz}$
b2mx	$\langle B^2 \rangle_{yz}$
bxmx	$\langle B_x \rangle_{yz}$
bymx	$\langle B_y \rangle_{yz}$
bzmx	$\langle B_z \rangle_{yz}$
bx2mx	$\langle B_x^2 \rangle_{yz}$
by2mx	$\langle B_y^2 \rangle_{yz}$
bz2mx	$\langle B_z^2 \rangle_{yz}$
bxbymx	$\langle B_x B_y \rangle_{yz}$
bxbzmx	$\langle B_x B_z \rangle_{yz}$
bybzmx	$\langle B_y B_z \rangle_{yz}$
betamx	$\langle \beta \rangle_{yz}$
beta2mx	$\langle \beta^2 \rangle_{yz}$
	Module 'density_stratified.f90'
rux2mz	$\langle \rho u_x^2 \rangle_{xy}$
ruy2mz	$\langle \rho u_y^2 \rangle_{xy}$
ruz2mz	$\langle \rho u_z^2 \rangle_{xy}$
ruxuymz	$\langle \rho u_x u_y \rangle_{xy}$
ruxuzmz	$\langle \rho u_x u_z \rangle_{xy}$
ruyuzmz	$\langle \rho u_y u_z \rangle_{xy}$
drhomx	$\langle \Delta\rho/\rho_0 \rangle_{yz}$
drho2mx	$\langle (\Delta\rho/\rho_0)^2 \rangle_{yz}$
rux2mx	$\langle \rho u_x^2 \rangle_{yz}$

ruy2mx	$\langle \rho u_y^2 \rangle_{yz}$
ruz2mx	$\langle \rho u_z^2 \rangle_{yz}$
ruxuymx	$\langle \rho u_x u_y \rangle_{yz}$
ruxuzmx	$\langle \rho u_x u_z \rangle_{yz}$
ruyuzmx	$\langle \rho u_y u_z \rangle_{yz}$
	Module '`gravity_simple.f90`'
epotmx	$\langle \varrho \Phi_{\rm grav} \rangle_{yz}$
epotuxmx	$\langle \varrho \Phi_{\rm grav} u_x \rangle_{yz}$ (potential energy flux)
	Module '`shock_highorder.f90`'
	Module '`temperature_idealgas.f90`'
ppmx	$\langle p \rangle_{yz}$
TTmx	$\langle T \rangle_{yz}$
	Module '`thermal_energy.f90`'
ppmx	$\langle p \rangle_{yz}$
TTmx	$\langle T \rangle_{yz}$
	Module '`viscosity.f90`'
fviscmx	$\langle 2\nu\varrho u_i \mathcal{S}_{ix} \rangle_{yz}$ (x-component of viscous flux)
numx	$\langle \nu \rangle_{yz}$ (yz-averaged viscosity)

J.9 List of parameters for '`yaver.in`'

The following table lists possible inputs to the file '`yaver.in`'. This list is not complete and maybe outdated.

Variable	*Meaning*
	Module '`hydro.f90`'
uxmxz	$\langle u_x \rangle_y$
uymxz	$\langle u_y \rangle_y$
uzmxz	$\langle u_z \rangle_y$
ux2mxz	$\langle u_x^2 \rangle_y$
uy2mxz	$\langle u_y^2 \rangle_y$
uz2mxz	$\langle u_z^2 \rangle_y$
uxuymxz	$\langle u_x u_y \rangle_y$
uxuzmxz	$\langle u_x u_z \rangle_y$
uyuzmxz	$\langle u_y u_z \rangle_y$
oumxz	$\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle_y$
	Module '`density.f90`'
rhomxz	$\langle \varrho \rangle_y$
	Module '`entropy.f90`'
TTmxz	$\langle T \rangle_y$

ssmxz	$\langle s \rangle_y$
Module 'magnetic.f90'	
b2mxz	$\langle \boldsymbol{B}^2 \rangle_y$
axmxz	$\langle A_x \rangle_y$
aymxz	$\langle A_y \rangle_y$
azmxz	$\langle A_z \rangle_y$
bxmxz	$\langle B_x \rangle_y$
bymxz	$\langle B_y \rangle_y$
bzmxz	$\langle B_z \rangle_y$
bx2mxz	$\langle B_x^2 \rangle_y$
by2mxz	$\langle B_y^2 \rangle_y$
bz2mxz	$\langle B_z^2 \rangle_y$
bxbymxz	$\langle B_x B_y \rangle_y$
bxbzmxz	$\langle B_x B_z \rangle_y$
bybzmxz	$\langle B_y B_z \rangle_y$
Exmxz	$\langle \mathcal{E}_x \rangle_y$
Eymxz	$\langle \mathcal{E}_y \rangle_y$
Ezmxz	$\langle \mathcal{E}_z \rangle_y$
vAmxz	$\langle v_A^2 \rangle_y$
Module 'density_stratified.f90'	
drhomxz	$\langle \Delta\rho/\rho_0 \rangle_y$
drho2mxz	$\langle (\Delta\rho/\rho_0)^2 \rangle_y$
Module 'meanfield.f90'	
peffmxz	$\langle \mathcal{P}_{\rm eff} \rangle_y$
alpmxz	$\langle \alpha \rangle_y$
Module 'temperature_idealgas.f90'	
TTmxz	$\langle T \rangle_y$
Module 'thermal_energy.f90'	
TTmxz	$\langle T \rangle_y$

J.10 List of parameters for 'zaver.in'

The following table lists possible inputs to the file 'zaver.in'. This list is not complete and maybe outdated.

Variable	*Meaning*
Module 'hydro.f90'	
uxmxy	$\langle u_x \rangle_z$
uymxy	$\langle u_y \rangle_z$
uzmxy	$\langle u_z \rangle_z$
oxmxy	$\langle \omega_x \rangle_z$

oymxy	$\langle \omega_y \rangle_z$
ozmxy	$\langle \omega_z \rangle_z$
oumxy	$\langle \boldsymbol{\omega} \cdot \boldsymbol{u} \rangle_z$
pvzmxy	$\langle (\omega_z + 2\Omega)/\varrho \rangle_z$ (z component of potential vorticity)
uguxmxy	$\langle (\boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u})_x \rangle_z$
uguymxy	$\langle (\boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u})_y \rangle_z$
uguzmxy	$\langle (\boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u})_z \rangle_z$
ruxmxy	$\langle \rho u_x \rangle_z$
ruymxy	$\langle \rho u_y \rangle_z$
ruzmxy	$\langle \rho u_z \rangle_z$
ux2mxy	$\langle u_x^2 \rangle_z$
uy2mxy	$\langle u_y^2 \rangle_z$
uz2mxy	$\langle u_z^2 \rangle_z$
rux2mxy	$\langle \rho u_x^2 \rangle_z$
ruy2mxy	$\langle \rho u_y^2 \rangle_z$
ruz2mxy	$\langle \rho u_z^2 \rangle_z$
ruxuymxy	$\langle \rho u_x u_y \rangle_z$
ruxuzmxy	$\langle \rho u_x u_z \rangle_z$
ruyuzmxy	$\langle \rho u_y u_z \rangle_z$
fkinxmxy	$\left\langle \frac{1}{2} \varrho \boldsymbol{u}^2 u_x \right\rangle_z$
fkinymxy	$\left\langle \frac{1}{2} \varrho \boldsymbol{u}^2 u_y \right\rangle_z$

Module ‘`density.f90`’

rhomxy	$\langle \varrho \rangle_z$

Module ‘`entropy.f90`’

TTmxy	$\langle T \rangle_z$
ssmxy	$\langle s \rangle_z$
uxTTmxy	$\langle u_x T \rangle_z$
uyTTmxy	$\langle u_y T \rangle_z$
uzTTmxy	$\langle u_z T \rangle_z$
gTxmxy	$\langle \nabla_x T \rangle_z$
gTymxy	$\langle \nabla_y T \rangle_z$
gTzmxy	$\langle \nabla_z T \rangle_z$
gsxmxy	$\langle \nabla_x s \rangle_z$
gsymxy	$\langle \nabla_y s \rangle_z$
gszmxy	$\langle \nabla_z s \rangle_z$
gTxgsxmxy	$\langle (\nabla T \times \nabla s)_x \rangle_z$
gTxgsymxy	$\left\langle (\nabla T \times \nabla s)_y \right\rangle_z$
gTxgszmxy	$\langle (\nabla T \times \nabla s)_z \rangle_z$
fconvxy	$\langle c_p \varrho u_x T \rangle_z$
fconvyxy	$\langle c_p \varrho u_y T \rangle_z$
fconvzxy	$\langle c_p \varrho u_z T \rangle_z$
fradxy_Kprof	$F_x^{\rm rad}$ (x-component of radiative flux, z-averaged, from Kprof)
fradymxy_Kprof	$F_y^{\rm rad}$ (y-component of radiative flux, z-averaged, from Kprof)
fradxy_kramers	$F_{\rm rad}$ (z-averaged, from Kramers' opacity)
fturbxy	$\langle \varrho T \chi_t \nabla_x s \rangle_z$
fturbymxy	$\langle \varrho T \chi_t \nabla_y s \rangle_z$
fturbrxy	$\langle \varrho T \chi_{ri} \nabla_i s \rangle_z$ (radial part of anisotropic turbulent heat flux)

fturbthxy	$\langle \varrho T \chi_{\theta i} \nabla_i s \rangle_z$ (latitudinal part of anisotropic turbulent heat flux)
dcoolxy	surface cooling flux

Module 'magnetic.f90'

bxmxy	$\langle B_x \rangle_z$
bymxy	$\langle B_y \rangle_z$
bzmxy	$\langle B_z \rangle_z$
jxmxy	$\langle J_x \rangle_z$
jymxy	$\langle J_y \rangle_z$
jzmxy	$\langle J_z \rangle_z$
axmxy	$\langle A_x \rangle_z$
aymxy	$\langle A_y \rangle_z$
azmxy	$\langle A_z \rangle_z$
bx2mxy	$\langle B_x^2 \rangle_z$
by2mxy	$\langle B_y^2 \rangle_z$
bz2mxy	$\langle B_z^2 \rangle_z$
bxbymxy	$\langle B_x B_y \rangle_z$
bxbzmxy	$\langle B_x B_z \rangle_z$
bybzmxy	$\langle B_y B_z \rangle_z$
poynxmxy	$\langle \boldsymbol{E} \times \boldsymbol{B} \rangle_x$
poynymxy	$\langle \boldsymbol{E} \times \boldsymbol{B} \rangle_y$
poynzmxy	$\langle \boldsymbol{E} \times \boldsymbol{B} \rangle_z$
jbmxy	$\langle \boldsymbol{J} \cdot \boldsymbol{B} \rangle_z$
abmxy	$\langle \boldsymbol{A} \cdot \boldsymbol{B} \rangle_z$
examxy1	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle_z \vert_x$
examxy2	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle_z \vert_y$
examxy3	$\langle \boldsymbol{E} \times \boldsymbol{A} \rangle_z \vert_z$
StokesImxy	$\langle \epsilon_{B\perp} \rangle_z \vert_z$
StokesQmxy	$-\langle \epsilon_{B\perp} \cos 2\chi \rangle_z \vert_z$
StokesUmxy	$-\langle \epsilon_{B\perp} \sin 2\chi \rangle_z \vert_z$
StokesQ1mxy	$+\langle F \epsilon_{B\perp} \sin 2\chi \rangle_z \vert_z$
StokesU1mxy	$-\langle F \epsilon_{B\perp} \cos 2\chi \rangle_z \vert_z$
beta1mxy	$\langle \boldsymbol{B}^2/(2\mu_0 p) \rangle_z \vert_z$

Module 'density_stratified.f90'

drhomxy	$\langle \Delta\rho/\rho_0 \rangle_z$
drho2mxy	$\langle (\Delta\rho/\rho_0)^2 \rangle_z$

Module 'gravity_simple.f90'

epotmxy	$\langle \varrho \Phi_{\rm grav} \rangle_z$
epotuxmxy	$\langle \varrho \Phi_{\rm grav} u_x \rangle_z$ (potential energy flux)

Module 'temperature_idealgas.f90'

TTmxy	$\langle T \rangle_z$

Module 'thermal_energy.f90'

TTmxy	$\langle T \rangle_z$

Module 'viscosity.f90'

fviscmxy	$\langle 2\nu \varrho u_i \mathcal{S}_{ix} \rangle_z$ (x-xomponent of viscous flux)

fviscymxy	$\langle 2\nu\varrho u_i \mathcal{S}_{iy}\rangle_z$ (y-xomponent of viscous flux)

J.11 Boundary conditions

The following tables list all possible boundary condition labels that are documented.

J.11.1 Boundary condition bcx

Variable	*Meaning*
	Module 'boundcond.f90'
0	zero value in ghost zones, free value on boundary
p	periodic
s	symmetry, $f_{N+i} = f_{N-i}$; implies $f'(x_N) = f'''(x_0) = 0$
ss	symmetry, plus function value given
s0d	symmetry, function value such that df/dx=0
a	antisymmetry, $f_{N+i} = -f_{N-i}$; implies $f(x_N) = f''(x_0) = 0$
a2	antisymmetry relative to boundary value, $f_{N+i} = 2f_N - f_{N-i}$; implies $f''(x_0) = 0$
a2r	sets $d^2f/dr^2 + 2df/dr - 2f/r^2 = 0$ This is the replacement of zero second derivative in spherical coordinates, in radial direction.
cpc	cylindrical perfect conductor implies $f'' + f'/R = 0$
cpp	cylindrical perfect conductor implies $f'' + f'/R = 0$
cpz	cylindrical perfect conductor implies $f'' + f'/R = 0$
spr	spherical perfect conductor implies $f'' + 2f'/R = 0$ and $f(x_N) = 0$
v	vanishing third derivative
cop	copy value of last physical point to all ghost cells
1s	onesided
1so	onesided
cT	constant temperature (implemented as condition for entropy s or temperature T)
c1	constant temperature (or maybe rather constant conductive flux??)
Fgs	Fconv = - chi_t*rho*T*grad(s)
Fct	Fbot = - K*grad(T) - chi_t*rho*T*grad(s)
sT	symmetric temperature, $T_{N-i} = T_{N+i}$; implies $T'(x_N) = T'''(x_0) = 0$
asT	select entropy for uniform ghost temperature matching fluctuating boundary value, $T_{N-i} = T_N =$; implies $T'(x_N) = T'(x_0) = 0$
f	"freeze" value, i.e. maintain initial
fg	"freeze" value, i.e. maintain initial
1	$f = 1$ (for debugging)
set	set boundary value to *fbcx12*
der	set derivative on boundary to *fbcx12*
slo	set slope at the boundary = *fbcx12*
dr0	set boundary value [really??]
ovr	overshoot boundary condition ie $(d/dx - 1/\mathrm{dist})f = 0$.
out	allow outflow, but no inflow forces ghost cells and boundary to not point inwards
e1o	allow outflow, but no inflow uses the e1 extrapolation scheme

ant	stops and prompts for adding documentation
e1	extrapolation [describe]
e2	extrapolation [describe]
e3	extrapolation in log [maintain a power law]
hat	top hat jet profile in spherical coordinate.
jet	top hat jet profile in cartezian coordinate.
spd	sets $d(rA_\alpha)/dr =$ `fbcx12(j)`
sfr	stress-free boundary condition for spherical coordinate system.
nfr	Normal-field bc for spherical coordinate system. Some people call this the "(angry) hedgehog bc".
sa2	$(d/dr)(rB_\phi) = 0$ imposes boundary condition on 2nd derivative of rA_ϕ. Same applies to θ component.
pfc	perfect-conductor in spherical coordinate: $d/dr(A_r) + 2/r = 0$.
fix	set boundary value [really??]
fil	set boundary value from a file
g	set to given value(s) or function
nil	do nothing; assume that everything is set
ioc	inlet/outlet on western/eastern hemisphere in cylindrical coordinates do nothing; assume that everything is set
s	implies $f'(y_N) = f'''(y_0) = 0$

J.11.2 Boundary condition bcy

Variable	*Meaning*
	Module '`boundcond.f90`'
sds	symmetric-derivative-set
0	zero value in ghost zones, free value on boundary
p	periodic
pp	periodic across the pole
ap	anti-periodic across the pole
s	symmetry symmetry, $f_{N+i} = f_{N-i}$;
ss	symmetry, plus function value given
sds	symmetric-derivative-set
cds	complex symmetric-derivative-set
s0d	symmetry, function value such that df/dy=0
a	antisymmetry
a2	antisymmetry relative to boundary value
v	vanishing third derivative
v3	vanishing third derivative
out	allow outflow, but no inflow forces ghost cells and boundary to not point inwards
1s	onesided
cT	constant temp.
sT	symmetric temp.
asT	select entropy for uniform ghost temperature matching fluctuating boundary value, $T_{N-i} = T_N =$; implies $T'(x_N) = T'(x_0) = 0$
f	freeze value
s+f	freeze value
fg	"freeze" value, i.e. maintain initial

1	f=1 (for debugging)
set	set boundary value
sse	symmetry, set boundary value
sep	set boundary value
e1	extrapolation
e2	extrapolation
e3	extrapolation in log [maintain a power law]
der	set derivative on the boundary
cop	outflow: copy value of last physical point to all ghost cells
c+k	no-inflow: copy value of last physical point to all ghost cells, but suppressing any inflow
sfr	stress-free boundary condition for spherical coordinate system.
nfr	Normal-field bc for spherical coordinate system. Some people call this the "(angry) hedgehog bc".
spt	spherical perfect conducting boundary condition along θ boundary $f''+\cot\theta f'=0$ and $f(x_N)=0$
pfc	perfect conducting boundary condition along θ boundary
nil','	do nothing; assume that everything is set
sep	set boundary value

J.11.3 Boundary condition bcz

Variable	*Meaning*
	Module 'boundcond.f90'
cfb	radial centrifugal balance
fBs	frozen-in B-field (s)
fB	frozen-in B-field (a2)
0	zero value in ghost zones, free value on boundary
p	periodic
s	symmetry
sf	symmetry with respect to interface
s0d	symmetry, function value such that df/dz=0
0ds	symmetry, function value such that df/dz=0
a	antisymmetry
a2	antisymmetry relative to boundary value
af	antisymmetry with respect to interface
a0d	antisymmetry with zero derivative
v	vanishing third derivative
v3	vanishing third derivative
1s	one-sided
fg	"freeze" value, i.e. maintain initial
c1	complex
Fgs	Fconv = - chi_t*rho*T*grad(s)
c3	constant flux at the bottom with a variable hcond
pfe	potential field extrapolation
p1D	potential field extrapolation in 1D
pot	potential magnetic field
pwd	a variant of 'pot' for nprocx=1
hds	hydrostatic equilibrium with a high-frequency filter

cT	constant temp.
cT2	constant temp. (keep lnrho)
cT3	constant temp. (keep lnrho)
hs	hydrostatic equilibrium
hse	hydrostatic extrapolation rho or lnrho is extrapolated linearily and the temperature is calculated in hydrostatic equilibrium.
cp	constant pressure
sT	symmetric temp.
ctz	for interstellar runs copy T
cdz	for interstellar runs limit rho
asT	select entropy for uniform ghost temperature matching fluctuating boundary value, $T_{N-i} = T_N =$; implies $T'(x_N) = T'(x_0) = 0$
c2	complex
db	complex
ce	complex
e1	extrapolation
e2	extrapolation
ex	simple linear extrapolation in first order
exf	simple linear extrapolation in first order
exd	simple linear extrapolation in first order
exm	simple linear extrapolation in first order
b1	extrapolation with zero value (improved 'a')
b2	extrapolation with zero value (improved 'a')
b3	extrapolation with zero value (improved 'a')
f','fa	freeze value + antisymmetry
fs	freeze value + symmetry
fBs	frozen-in B-field (s)
fB	frozen-in B-field (a2)
g	set to given value(s) or function
1	f=1 (for debugging)
StS	solar surface boundary conditions
set	set boundary value
der	set derivative on the boundary
div	set the divergence of $\boldsymbol{u}$ to a given value use bc = 'div' for iuz
ovr	set boundary value
inf	allow inflow, but no outflow
ouf	allow outflow, but no inflow
in	allow inflow, but no outflow forces ghost cells and boundary to not point outwards
out	allow outflow, but no inflow forces ghost cells and boundary to not point inwards
in0	allow inflow, but no outflow forces ghost cells and boundary to not point outwards relaxes to vanishing 1st derivative at boundary
ou0	allow outflow, but no inflow forces ghost cells and boundary to not point inwards relaxes to vanishing 1st derivative at boundary
ind	allow inflow, but no outflow forces ghost cells and boundary to not point outwards creates inwards pointing or zero 1st derivative at boundary
oud	allow outflow, but no inflow forces ghost cells and boundary to not point inwards creates outwards pointing or zero 1st derivative at boundary
ubs	copy boundary outflow,

win	forces massflux given as $\Sigma\rho_i(u_i+u_0)=$ fbcz1/2(ρ)
cop	copy value of last physical point to all ghost cells
nil	do nothing; assume that everything is set

J.12 Initial condition parameter dependence

The following tables list which parameters from each Namelist are required (•), optional (◇) or irrelevant (blank). The distinction is made between required and optional where by a parameter requires a setting if the default value would give an invalid or degenerate case for the initial condition.

inituu	ampluu	widthuu	urand	uu_left	uu_right	uu_upper	uu_lower	uy_left	uy_right	kx_uu	ky_uu	kz_uu
zero												
gaussian-noise	•											
gaussian-noise-x	•											
xjump		◇		•	•			•	•			
Beltrami-x	•											
Beltrami-y	•											
Beltrami-z	•											
trilinear-x	•											
trilinear-y	•											
trilinear-z	•											
cos-cos-sin-uz	•											
tor_pert	•											
trilinear-x	•											
sound-wave	•									•		
shock-tube		◇		•	•							
bullets	•	◇										
Alfven-circ-x	•									◇		
const-ux	•											
const-uy	•											
tang-discont-z	◇	•				•	•					
Fourier-trunc	•	◇								•	•	
up-down	•	◇										

initss	ampl_ss	radius_ss	widthss	epsilon_ss	grads0	pertss	ss_left	ss_right	ss_const	mpoly0	mpoly1	mpoly2	isothtop	khor_ss	center1_x	center1_y	center1_z	center2_x	center2_y	center2_z	
zero																					
const_ss									•												
blob	•	•																			
isothermal																					
Ferrière																					
xjump			•				•	•													
hor-fluxtube	•	•		•																	
hor-tube	•	•		•																	
sedov		•													•	•	•				
sedov-dual		•													•	•	•	•	•	•	
isobaric																					
isentropic																					
linprof																					
piecew-poly																					
polytropic																					

References

[1] Abramowitz, A., Stegun, I. A., *Pocketbook of Mathematical Functions*, Harri Deutsch, Frankfurt (1984)

[2] Brandenburg, A., *Astrophys. J.* **550**, 824–840 (2001) "The inverse cascade and nonlinear alpha-effect in simulations of isotropic helical hydromagnetic turbulence"

[3] Brandenburg, A., in *Advances in non-linear dynamos*, ed. A. Ferriz-Mas & M. Núñez Jiménez, (The Fluid Mechanics of Astrophysics and Geophysics, Vol. **9**) Taylor & Francis, London and New York, pp. 269–344 (2003); `http://arXiv.org/abs/astro-ph/0109497`

[4] Brandenburg, A., Dobler, W., *Astron. Astrophys.* **369**, 329–338 (2001) "Large scale dynamos with helicity loss through boundaries"

[5] Brandenburg, A., & Hazlehurst, J., *Astron. Astrophys.* **370**, 1092–1102 (2001) "Evolution of highly buoyant thermals in a stratified layer"

[6] Brandenburg, A., & Sarson, G. R., *Phys. Rev. Lett.* **88**, 055003 (2002) "The effect of hyperdiffusivity on turbulent dynamos with helicity"

[7] Brandenburg, A., Dobler, W., & Subramanian, K., *Astron. Nachr.* **323**, 99–122 (2002) "Magnetic helicity in stellar dynamos: new numerical experiments"

[8] Brandenburg, A., Jennings, R. L., Nordlund, Å., Rieutord, M., Stein, R. F., & Tuominen, I., *J. Fluid Mech.* **306**, 325–352 (1996) "Magnetic structures in a dynamo simulation"

[9] Brandenburg, A., Moss, D., & Shukurov, A., *MNRAS* **276**, 651–662 (1995) "Galactic fountains as magnetic pumps"

[10] Brandenburg, A., Nordlund, Å., Stein, R. F., & Torkelsson, U., *Astrophys. J.* **446**, 741–754 (1995) "Dynamo-generated turbulence and large scale magnetic fields in a Keplerian shear flow"

[11] Collatz, L., *The numerical treatment of differential equations*, Springer-Verlag, New York, p. 164 (1966)

[12] Dobler, W., Stix, M., & Brandenburg, A.: 2006, "Convection and magnetic field generation in fully convective spheres," *Astrophys. J.* **638**, 336-347

[13] Gammie, C. F., *Astrophys. J.* **553**, 174–183 (2001) "Nonlinear outcome of gravitational instability in cooling, gaseous disks"

[14] Goodman, J., Narayan, R. & Goldreich, P., *Month. Not. Roy. Soc.* **225**, 695–711 (1987) "The stability of accretion tori – II. Nonlinear evolution to discrete planets"

[15] Haugen, N. E. L., & Brandenburg, A. *Phys. Rev. E* **70**, 026405 (2004) "Inertial range scaling in numerical turbulence with hyperviscosity"

[16] Hockney, R. W., & Eastwood, J. W., *Computer Simulation Using Particles*, McGraw-Hill, New York (1981)

[17] Hurlburt, N. E., Toomre, J., & Massaguer, J. M., *Astrophys. J.* **282**, 557–573 (1984) "Two-dimensional compressible convection extending over multiple scale heights"

[18] Kim, J., Moin, P. & Moser, R *J. of Fluid Mech.* **177**, 133 (1987) "Turbulence statistics in fully developed channel flow at low Reynolds number"

[19] Kippenhahn, R. & Weigert, A. *Stellar structure and evolution*, Springer: Berlin (1990)

[20] Krause, F., Rädler, K.-H., *Mean-Field Magnetohydrodynamics and Dynamo Theory*, Akademie-Verlag, Berlin; also Pergamon Press, Oxford (1980)

[21] Lele, S. K., *J. Comp. Phys.* **103**, 16–42 (1992) "Compact finite difference schemes with spectral-like resolution"

[22] Nordlund, Å., & Galsgaard, K., *A 3D MHD code for Parallel Computers*, `http://www.astro.ku.dk/~aake/NumericalAstro/papers/kg/mhd.ps.gz` (1995)

[23] Nordlund, Å., Stein, R. F., *Comput. Phys. Commun.* **59**, 119 (1990) "3-D simulations of solar and stellar convection and magnetoconvection"

[24] Press, W., Teukolsky, S., Vetterling, W., & Flannery, B., *Numerical Recipes in Fortran 90*, 2nd ed., Cambridge (1996)

[25] Snodin, A. P., Brandenburg, A., Mee, A. J., & Shukurov, A.: 2005, "Cosmic ray confinement in the diffusion approximation," *Monthly Notices Roy. Astron. Soc.* (submitted) (astro-ph/0507176)

[26] Stanescu, D., Habashi, W. G., *J. Comp. Phys.* **143**, 674 (1988) "$2N$-storage low dissipation and dispersion Runge–Kutta schemes for computational acoustics"

[27] Williamson, J. H., *J. Comp. Phys.* **35**, 48 (1980) "Low-storage Runge–Kutta schemes"

Part IV
Indexes

File Index

Variable Index

Index

This index contains options, names, definitions and commands. Files and variables have their own indexes.