Linfor3D User Manual

IDL version 7.3.0

Matthias Steffen, Hans-Günter Ludwig,

Sven Wedemeyer & Andrew J. Gallagher

F90 version 0.9.0

Andrew J. Gallagher & Matthias Steffen

November 1, 2023

Contents

List of Figures

List of Tables

1 Introduction

Linfor3D is based on the old code LINFOR (short for **LIN**e **FOR**mation code). There are two versions of the code that are currently being developed side-by-side.

The first version of the code has been developed and tested from conception in IDL. This can also run on certain versions of GDL (see Sect. 4). The bulk of the heavy computations are done inside Fortran routines, which are called by the IDL code's program flow.

The second version of the code is based on the first version, but is completely written in Fortran. This version has the advantage of being faster overall for a given compiler (e.g. Intel© Fortran or GNU Fortran) as it is capable of running in parallel with MPI (Message Passing Interface), which is discussed later.

Like the codes themselves this user manual is under construction, too. Nevertheless, it will be of substantial help while installing and running Linfor3D, creating new command and spectral line data files for Linfor3D in either IDL or Fortran, and interpreting the output data.

To get a brief overview of how to install and run both versions of Linfor3D in their simplest forms, you should read the "Getting started" section (Sect. 2) after this brief overview of both codes.

1.1 The IDL version

Here is a short break down of the limitations of the IDL/GDL version of the code:

- **Geometry:** This version is limited to spectrum synthesis from **local hydrodynamical models** (solar-type convective atmospheres).
- **Efficiency:** No effort was yet taken to really improve the execution speed of this IDL/Fortran code. In fact there are still some parts in the code which are unnecessary for the current operation. Their execution should be controlled by additional flags in a future release.
- **Parallel processing:** The newer versions of Linfor3D (version 6.0.0 onwards) have been ported so that it runs on GDL as well as IDL. This means that parallel processing – as an embarassingly parallel job – is possible. See Sect. 4.2 for further information and updates. However, no effort has been made to add parallel processing in to the internal program flow of Linfor3D.

1.2 The F90 version

The Fortran version is newer, and as such has been developed to remove some of the IDL versions' limitations:

- **Geometry:** Like its IDL counterpart, this version is limited to spectrum synthesis from **local hydrodynamical models** (solar-type convective atmospheres).
- **Effeciency:** This version of the code is domain decomposed. This means that it can be effectively ran on many CPUs without large memory consumptions, meaning that one can scale a job according to the number of CPUs available and the grid size of the job. Naturally, one must be smart when running the code in parallel on their own machine. Running this with too many CPUs can lead to unintended effects.
- **Parallel processing:** The Fortran version of the code has been developed with parallelization in mind. As such, the code is parallelized over several dimensions that are automatically sorted according to a priory decided by the code (this is discussed later).

1.3 The differences

The F90 version of Linfor3D was written so that it closely follows its IDL counterpart. This was so that both could continue to be developed together. As such, the more computationally expensive parts of Linfor3D are still called by both versions. One reason behind this is so that bugs can be more easily found and fixed. Another reason is that it is important to have a working code and a reference while efforts are made to include new state-of-the-art physics, which is always in consideration.

At the moment there are some small differences in the behaviour of the two versions:

1. **Installing:** The methods of installing the two codes differ slightly. The F90 version of the code as a script - `install` that will deal with most machine-dependent issues this version may face. The IDL version is installed from a more traditional Makefile (see Sect 2).

2. **Parallelization:** While it is possible to run the IDL version of Linfor3D using an "*embarrasingly parallel*" approach (although mostly through GDL because of licensing concerns, see Sect. 4), this version of Linfor3D remains an inherently serial code. The F90 version was written to use with MPI in mind (see Sect. 3).

3. **Program flow information:** You will notice as you experiment with Linfor3D that information that is printed to screen is quite dynamic, especially in the new F90 verion of the code. This is so that the screen or a file is not overwhelmed with trivial information, or in the case of the F90 version, the same print outs on every CPU. This version can also print much more detailed information about the input data when CO^{5}BOLD models are provided.

4. **Plotting:** The IDL version of Linfor3D can be used to plot output in several ways depending on how one sets up the parameter file (see Sect. 7). The F90 version does not do this. In all likelyhood, this will not be a feature of this code.

1.4 Final word

It is important to stress that while the two codes will eventually deviate so much that the IDL version is deprecated, the main objective to any and all development has always been user friendlyness. All post-process libraries available to the community should continue to work in the same way, and there are no plans to ever change this.

2 Getting started

This section deals with installing and executing the simplest form of Linfor3D.

Linfor3D should compile and run on any machine running Debian or Red Hat distribution Linux with Intel© or AMD© chips. Linfor3D can also compile and run on MacOS powered by either Intel© or Apple Silicon© chips.

2.1 Running the IDL version

First make sure that you have all files which are listed in Tables 2 and 4. These files should be put together in a directory which can be accessed under IDL (or GDL using versions 6.0.0 onwards).

The routines responsible for the most computationally expensive parts of Linfor3D are called as external Fortran routines by `linfor_3D_ionopa.pro` and `linfor_dort.pro`. In order to properly run Linfor3D, the Linux path variable `$IDL_SO` should be defined. After unpacking Linfor3D, create a new sub-directory within the directory tree called `bin`, then define `IDL_SO`:

```
> export IDL_SO=<LINFOR_DIRECTORY>/bin/
```

This needs to be added to your Linux login script to be consistently defined by your Linux OS.

Next you need to compile the Fortran libraries `dort_idl.so` and `ionopa2_idl.so`. The source code and Makefiles are located inside the `xmono` directory. By default, the libraries are compiled using the Intel© Fortran, `IFORT`, compilers as the Makefile is linked to `Makefile_ifort`. One can change the link to any other Makefile in the directory. To compile simply enter the following command:

```
> make
```

If the shell variable `$IDL_SO` is properly defined, the two libraries should compile and be moved to the `bin` sub-directory. If the compilation fails, please check this variable and that the directory it points to exists and the compiler selected is available on your machine.

Now two files have to be edited and provided in order to run Linfor3D:

- `linfor_setcmd.pro`:
  This file, which is an IDL script, defines the data structure `cmd`. This structure contains all necessary information (except for spectral line data) like, e.g., paths and names of model file(s). See Sect. 7 for more details.

- `line.dat`:
  This file contains all data for spectral line such as, e.g., oscillator strength and broadening parameters. The usual file name is `line.dat` but it might be given another name which then has be to entered in `linfor_setcmd.pro`. For a quick execution, examples of these line files in every file format (see Sect. 8 for details on these formats) are supplied within the directory tree under the `Data` subdirectory.

Finally, all IDL/GDL versions of Linfor3D require the "Universal Input Output" (UIO) IDL routine library (written by B. Freytag) for I/O related to CO^{5}BOLD files, and version 6.0.0 onwards requires them for ALL I/O done during its execution.

This directory must be defined in the `$IDL_PATH` either in one's `BASH` script or an `IDL` starup script. After doing so, you can run Linfor3D by starting IDL or GDL (see Sect. 4) and type:

```
IDL> .r linfor_3D.pro
```

Several output files are created. It is possible to load these files in IDL or GDL. See Sect. 10 for more details.

Normally, one uses a bespoke `linfor_setcmd.pro` file in a directory of their choosing. If this is the case then one must start IDL or GDL in the proper sub-diretory and then type the following:

```
IDL> common linfordata, info, cmd, const, atom, abu, line, gas, eos, result
IDL> .com bespoke_linfor_setcmd.pro
% Compiled module: LINFOR_SETCMD.
IDL> .r linfor_3D.pro
```

One of the most efficient ways to run Linfor3D is to create an IDL/GDL wrapper around one's bespoke `setcmd` program, as the following example depicts:

```
common linfordata, info, cmd, const, atom, abu, line, gas, eos, result

.com ./linfor_setcmd.pro
.run linfor_3D

save_flag = 0
if (keyword_set(cmd.cc3d_flag)) then save_flag = save_flag + 1
if (abs(cmd.cog) gt 1)      then save_flag = save_flag + 2

save_file = 'linfor3D.uiosave'

if (save_flag eq 0) then $
   uio_save, filename=save_file, /verbose, $
   info, cmd, const, atom, abu, line, result, maps, imuphi, contf
if (save_flag eq 1) then $
   uio_save, filename=save_file, /verbose, $
   info, cmd, const, atom, abu, line, result, maps, imuphi, contf, contf3d
if (save_flag eq 2) then $
   uio_save, filename=save_file, /verbose, $
   info, cmd, const, atom, abu, line, result, maps, imuphi, contf, cgout
if (save_flag eq 3) then $
   uio_save, filename=save_file, /verbose, $
   info, cmd, const, atom, abu, line, result, maps, imuphi, contf, contf3d,
       cgout

exit
```

This runs the bespoke Linfor3D script and saves the entire output as a tailored save file, as well as the default uiosave files usually output by Linfor3D that splits the contents of the structures mostly defined in `linfordata`.

A brief description of the changes made for all Linfor3D releases (up to the version you are running) is given in `version_history_IDL.md` given in the top directory of your Linfor3D installation.

2.2 Running the F90 version

The `f90/` subdirectory contains all modules necessary to compile Linfor3D. Currently, the code will compile using the GNU OpenMPI and Intel© parellel studios X MPI Fortran compilers; `mpifort,mpif90,mpiifort`, respectively. The Intel© compiler must be newer than the 2017 version. If one wishes to use the Intel© compiler suite, we highly recommend that the user install the latest version,

which is available to most linux and UNIX-based systems via Intel© oneAPI. The instructions for downloading and installing Intel© oneAPI can be found in this link. The install file – `install` – will expect one of these compilers on your machine so that it can compile Linfor3D. By default it will look for the faster Intel© compilers first. One can also force the install script to use a particular compiler using the `install -c <compiler name>` option. If it cannot find any compiler the script will exit.

Installation is similar to installing CO^{5}BOLD:

```
> install
```

The `install` file is used to create a bespoke `Makefile` then that is used to compile the code using all available CPUs. Several architecture-defined `IO` related flags[1] that should be defined by the computer Linfor3D is installed on.

Once properly compiled create a new working directory, copy or link the executable to this directory and copy `linfor3d.setcmd` from `Data/`. Edit this file so that all input files and directories can be found. This file is extremely similar to `linfor_setcmd.pro` discussed above in Sect. 2.1.

The code can be executed with the following command:

```
> ./linfor3d.x
```

Unlike its IDL companion, the F90 version of Linfor3D is capable of executing on multiple CPUs. To run the same set up on multiple CPUs one must execute the code with the following command:

```
> mpirun -np <NCPU> linfor3d.x
```

where `NCPU` is the number of CPUs requested by the user at execution. The parallelization module will then use this to parallelize the job over the number of CPUs allocated by the user.

If you wish to compile a fresh copy of Linfor3D type:

```
> make fresh
```

To remove all compiled modules and objects while leaving the module links and Linfor3D executable type:

```
> make clean
```

To uninstall Linfor3D simply type:

```
> make uninstall
```

This will completely remove all compiled objects and modules, and also remove the Makefile itself. One can the install once again as before using the install file.

2.2.1 Getting a custom installation

In addition to the basic installation illustrated in Sect. 2.2, `install` can also install a more custom version of Linfor3D using the options now tabulated.

Command	Description
`install -h`	Returns the complete help guide to terminal. No code is compiled.
`install -d`	Compiles Linfor3D with some useful debug options. Will stop all optimization. Not recommended for normal use.
`install -m`	Compiles the code without using the F90-only `dort_module` module flags, essentially compiling the same version of `dort.f90` as is compiled for use with the IDL version of the code. **Not** recommened.

[1] The `IO` framework `UIO` – written and developed by B. Freytag – requires bespoke compiler flags to properly extract information from the binary `UIO` formatted input files.

`install -D`	Compiles Linfor3D with a large amount of debug options, including all warnings. For use only for development. Not recommended for normal use.
`install -c <ARG>`	Compiles Linfor3D with a user specified compiler.
`install -s`	Will compile the Makefile on a single CPU. Normally install looks at the machine architecture and makes the install on as many CPUs as possible to speed up the compilation time.
`install -l <ARG>`	The default Makefile created by install links the default location of `xmono` directory to the Linfor3D directory tree. This option uses the directory expressed in <ARG> instead.
`install -x <ARG>`	The Makefile created calls the executable `linfor3d.x` as standard. This option changes the executable name to what is expressed inside <ARG>.

Table 1: A list of options that can be included in install to create a custom Linfor3D compilation.

3 Linfor3D and MPI

This section will briefly detail what aspects of the Linfor3D program flow runs in parallel and what aspects do not. Reasons as to why are also given.

Considerable effort has been put into writing the F90 version of Linfor3D so that it can be executed in parallel. It was written and developed using the GNU OpenMP MPI libraries. Measures have been taken to make sure that the code will compile with Intel© MPI compilers and execute without issues. It is therefore possible to run Linfor3D using the Intel© oneAPI compilers on machines that have them, or with the older Intel© parallel studio X on HPC centres that still run it.

There are two stages when the program flow parallelizes the execution. Once when the continuous opacities for **3D model atmospheres** are computed with the IONDIS opacity package, and again when all radiative transfer is computed.

3.1 Reading input data

At present all input data is read in to Linfor3D sequentially, and only to the MASTER CPU. The reason for this is to maintain parity with the IDL version of the code, i.e. use of the IONDIS package and the radiative transfer modules inside `dort.F90` in their present condition. Almost all data stored during execution is only available on the MASTER CPU. This also saves massive amounts of memory-per-CPU. As the code deviates from its IDL counterpart, it may be necessary to improve this inefficiency and further parallelize the code. This is a low priority.

3.2 Writing data to file

Only the MASTER CPU writes data to file using the UIO package. Until input data is read in parallel there is no need for this to change. This is a **very** low priority.

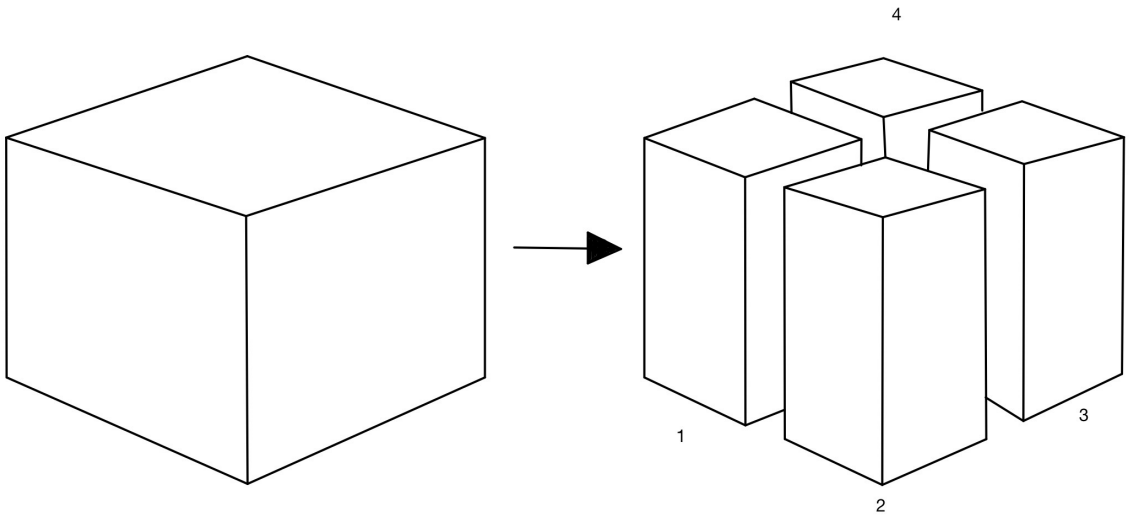
3.3 Computing 1D continuous opacities

During execution of a standard setup, Linfor3D uses the IONDIS package to compute continuous opacities for two 1D models; the ⟨3D⟩ model and the 1D external model. As this is done so quickly, it was not even considered to be written in a non-sequential way. If input data becomes CPU dependent, this will most likely have to change. This is a **very** low priority.

3.4 Computing 3D continuous opacities

The execution of the IONDIS package for a 3D atmosphere can be time consuming in Linfor3D. It depends on the parameters of the input 3D atmosphere(s). Consequently, Linfor3D executes the IONDIS package in a parallel way when it is made available. The code creates smaller domains – or subdomains – within each 3D cube, dividing over horizontal x and y dimensions according to the number of CPUs that have been assigned to the execution. Therefore, the number of subdomains is quivalent to the number of CPUs. The vertical z dimension is not considered as opacities are computed from the bottom of the 3D cube to the top in 1D columns.

The following diagram depicts how the parallelization scheme creates subdomains when four CPUs are assigned to a single snapshot. This is known as 2D domain decomposition.



The size of the x and y indexes have each decreased by a factor of 2. As a result each CPU only performs a quarter of the number of the computations a lone CPU would have to during a sequential (or IDL) execution.

As a priority, Linfor3D divides the number of snapshots over the available number of CPUs as equally as possible. This is because they are the most time consuming "dimension" dealt with by the IONDIS package. The remaining CPUs are then divided across the x and y dimensions of every 3D cube as was just explained. To summerize, Linfor3D parallelizes over three dimensions; the number of snapshots (`nsnap`), the x (`nx`) and y (`ny`) horizontal dimensions.

The following example demonstrates how the parallelization scheme behaves when it is given a domain consisting of a single snapshot – or 3D cube – which consists of 28×28 grid points. There are 8 CPUs over which the parallelization scheme has to divide the domain in to subdomains:

```
============================================================
              Subdomain index scheme for IONOPA:
============================================================
    n_CPU = 8, n_snap = 1, x-grid = 28, y-grid = 28
------------------------------------------------------------
    CPU |    snapshots    |      x-grid     |      y-grid
     ID |  ss1  ss2   ds |   x1   x2   dx |   y1   y2   dy
------------------------------------------------------------
      0 |    1    1    1 |    1   14   14 |    1    7    7
      1 |    1    1    1 |    1   14   14 |    8   14    7
      2 |    1    1    1 |    1   14   14 |   15   21    7
      3 |    1    1    1 |    1   14   14 |   22   28    7
      4 |    1    1    1 |   15   28   14 |    1    7    7
      5 |    1    1    1 |   15   28   14 |    8   14    7
      6 |    1    1    1 |   15   28   14 |   15   21    7
      7 |    1    1    1 |   15   28   14 |   22   28    7
============================================================
```

As you can see, the parallelization scheme has divided the horizontal dimensions as equally as it can. The next example represents a more typical run of Linfor3D whose domain now consists of 20 snapshots, and 28×28 horizontal grid points. The parallelization scheme was given 16 CPUs to create 16 subdomains:

```
============================================================
```

Subdomain index scheme for IONOPA:

n_CPU = 16, n_snap = 20, x-grid = 28, y-grid = 28

CPU ID	snapshots ss1	ss2	ds	x-grid x1	x2	dx	y-grid y1	y2	dy
0	1	5	5	1	14	14	1	14	14
1	6	10	5	1	14	14	1	14	14
2	11	15	5	1	14	14	1	14	14
3	16	20	5	1	14	14	1	14	14
4	1	5	5	1	14	14	15	28	14
5	6	10	5	1	14	14	15	28	14
6	11	15	5	1	14	14	15	28	14
7	16	20	5	1	14	14	15	28	14
8	1	5	5	15	28	14	1	14	14
9	6	10	5	15	28	14	1	14	14
10	11	15	5	15	28	14	1	14	14
11	16	20	5	15	28	14	1	14	14
12	1	5	5	15	28	14	15	28	14
13	6	10	5	15	28	14	15	28	14
14	11	15	5	15	28	14	15	28	14
15	16	20	5	15	28	14	15	28	14

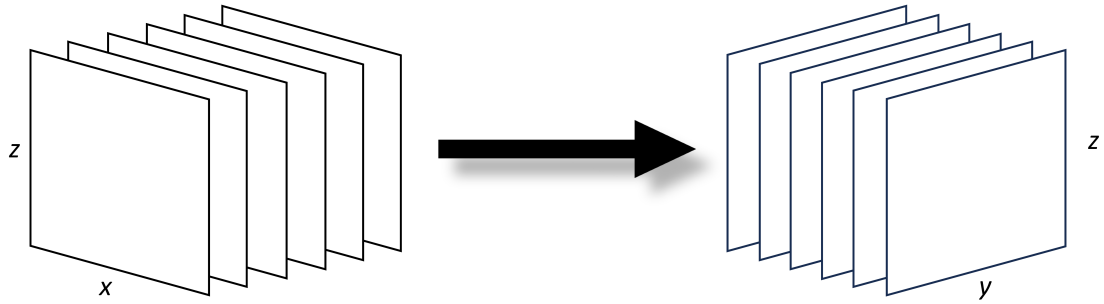
As expected, the parallelization scheme has divided the snapshots as equally as possible into four subdomains. The remaining CPUs are assigned to the horizontal grid points for a total of 16 unique subdomains over which the IONDIS package computes in parallel.

3.5 Computing all radiative transfer

The primary reason that Linfor3D was ported to Fortran was so that the "heavy-duty" computations could take advantage of the MPI modules. The most time consuming parts of a typical Linfor3D run is the execution of `dort.F90` for the 3D model atmospheres and for the 1D model atmospheres when `dort.F90` is expected to compute curves–of–growth. However, because `dort.F90` performs several different operations depending on the type of atmosphere and the set up requirements requested by the user, it was necessary to write an equally adaptive parallization scheme for each atmosphere type. Appropriately, the execution of this parallelization scheme differs enough that each one is described below in separate sections.

3.5.1 DoRT

The execution of `dort.F90` is distributed over six dimensions; the number of 3D model snapshots (`nsnap`), the number line lists (`kline`), the number of wavelength points over which to perform the radiative transfer computations (`nlam`), the number of *xz* and *yz* slices (as shown below), the number of Curve–of–Growth computations (`icg`), the number of microturbulence computations to compute (`imt`).



The parallelization scheme sets priorities on how these dimensions are divided into adequate subdomains. At present, the number of grid points are equally distributed over the number of CPUs. The number of subdomains should always be equal to the number of allocated CPUs. If one were to run Linfor3D on 10 CPUs, the 3D parallelisation could look like:

```
================================================================================
                    Subdomain index scheme for  3D  DoRT:
================================================================================
         nsnap = 20, xz-plane = 50, yz-plane = 50, kline = 1, nlam = 401
--------------------------------------------------------------------------------
  CPU |    nsnap     |   xz-plane   |   yz-plane   |  kline   |         nlam
  ID  | ss1 ss2  ds  |  ys  ye  dy  |  xs  xe  dx  | ks ke dk |    l1     l2     dl
--------------------------------------------------------------------------------
    0 |   1   2   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    1 |   3   4   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    2 |   5   6   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    3 |   7   8   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    4 |   9  10   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    5 |  11  12   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    6 |  13  14   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    7 |  15  16   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    8 |  17  18   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
    9 |  19  20   2  |   1  50  50  |   1  50  50  |  1  1  1 |     1    401    401
================================================================================
```

3.5.2 DoRT in 1D with Curve-of-growth computations

Naturally, the two 1D models have only a single x and y gridpoint. Therefore there is no parallelisation done for nx and ny. The 1D external and ⟨3D⟩ models are parallelised over the number of Curve–of–Growth computations and the microturbulence grid computations. If the 1D models are run with the same number of CPUs, the parallelisation scheme may look like this for the ⟨3D⟩ model:

```
================================================================================
                    Subdomain index scheme for <3D> DoRT:
================================================================================
           nsnap = 20, icg = 51, imt = 32, kline = 1, nlam = 401
--------------------------------------------------------------------------------
  CPU |    nsnap     |     icg      |     imt      |  kline   |         nlam
  ID  | ss1 ss2  ds  | cg1 cg2  dc  | mt1 mt2  dm  | ks ke dk |    l1     l2     dl
--------------------------------------------------------------------------------
    0 |   1   2   2  |   1  51  51  |   1  32  32  |  1  1  1 |     1    401    401
    1 |   3   4   2  |   1  51  51  |   1  32  32  |  1  1  1 |     1    401    401
```

```
    2 |   5   6   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
    3 |   7   8   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
    4 |   9  10   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
    5 |  11  12   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
    6 |  13  14   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
    7 |  15  16   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
    8 |  17  18   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
    9 |  19  20   2 |   1  51  51 |   1  32  32 |  1  1  1 |     1    401    401
================================================================================
```

For the 1D external model atmosphere, the parallelisation scheme might look like this:

```
================================================================================
                    Subdomain index scheme for 1Dx  DoRT:
================================================================================
            nsnap = 1, icg = 51, imt = 32, kline = 1, nlam = 401
--------------------------------------------------------------------------------
  CPU |    nsnap    |     icg     |     imt     |  kline   |         nlam
  ID  | ss1 ss2  ds | cg1 cg2  dc | mt1 mt2  dm | ks ke dk |    l1     l2     dl
--------------------------------------------------------------------------------
    0 |   1   1   1 |   1   5   5 |   1  32  32 |  1  1  1 |     1    401    401
    1 |   1   1   1 |   6  10   5 |   1  32  32 |  1  1  1 |     1    401    401
    2 |   1   1   1 |  11  15   5 |   1  32  32 |  1  1  1 |     1    401    401
    3 |   1   1   1 |  16  20   5 |   1  32  32 |  1  1  1 |     1    401    401
    4 |   1   1   1 |  21  25   5 |   1  32  32 |  1  1  1 |     1    401    401
    5 |   1   1   1 |  26  30   5 |   1  32  32 |  1  1  1 |     1    401    401
    6 |   1   1   1 |  31  35   5 |   1  32  32 |  1  1  1 |     1    401    401
    7 |   1   1   1 |  36  40   5 |   1  32  32 |  1  1  1 |     1    401    401
    8 |   1   1   1 |  41  45   5 |   1  32  32 |  1  1  1 |     1    401    401
    9 |   1   1   1 |  46  51   6 |   1  32  32 |  1  1  1 |     1    401    401
================================================================================
```

Naturally, when the Curve-of-Growth is switched off, there is no parallelisation done over the arrays responsible for them. Therefore, it is important to make sure that the appropriate number of CPUs are selected. Linfor3D will assign all CPUs to `nlam`, but it is important to make sure that there are enough grid points to compute data over. In the examples above, all 10 CPUs will be assigned to compute `nlam`. Having too many domains in `nlam` could lead to unexpected consequences. It is therefore recommended to have at least 5 wavelength points per domain, especially when `dclam` is set, to make sure that each domain can compute the continuum over the expected $\lambda \pm$ dclam.

3.6 Scaling Linfor3D

The latest versions of Linfor3D opt to limit the amount of memory used by each CPU by only passing data that is required for computations when it is required. Naturally, that communication between the Master and Slave CPUs reduces the scalability of Linfor3D. While it is technically possible to improve on this, testing revealed that even HPC facilities struggled with the memory requirements. Therefore, the current version will remain in service until such time as MPI IO is introduced into Linfor3D and CO^{5}BOLD.

Figure 1 demonstrates how Linfor3D scales with increasing numbers of CPUs. All jobs were computed using the CO^{5}BOLD model `d3gt57g44msc600` using a horizontal skipping of either 5 (`50x50`) or 25 (`10x10`). Between one and 160 CPUs were used. The lithium 6707 Å line was used for both `kline=1` and `kline=2`. The dotted line indicates the theoretical "ideal" speedup in both cases.

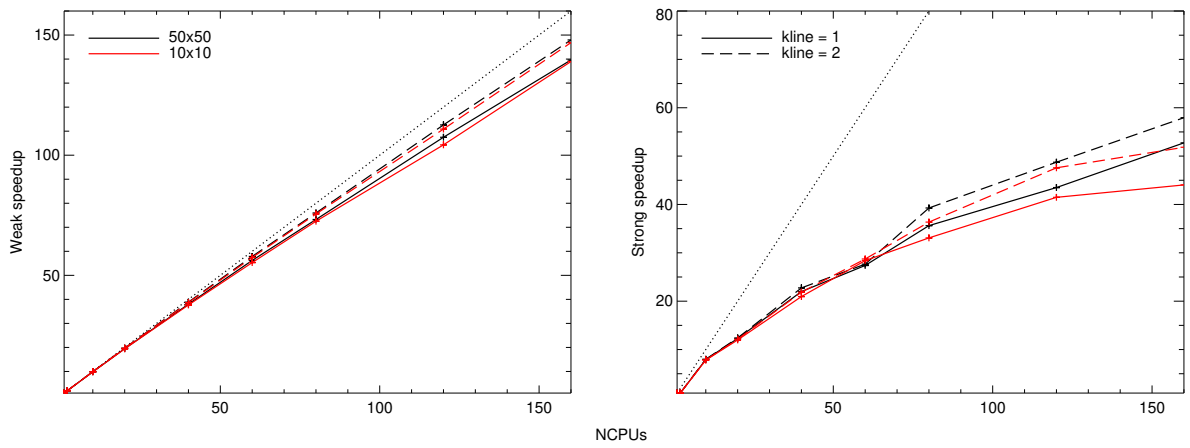


Figure 1: Strong and weak scaling relationships

The weak scaling relationship follows Gustafson's law. Strong scaling is computed as speedup = $t(1)/t(N)$, where $t(1)$ is the compute time taken for a seqential run and $t(N)$ is the time taken for a run using N CPUs.

The weak scaling relationship shows a very good speedup in the `DoRT` and `iondis` modules, that take advantage of the parallel schemes included inside Linfor3D. However, the strong scaling relationship shows a poorer speedup. However, until the gradient in these curves plateaus or becomes negative, there are still gains to be had by adding further CPUs. Naturally, because adding more lines to `kline` effectively increases the work by a factor of `kline`, the strong scaling relation improves with increasing `kline`. However, speedup is not the same as compute time, which will increase with increasing `kline`.

3.7 Limitations of Linfor3D with MPI

One must be careful and choose the number of CPUs that can be appropriately distributed across the 3D ionopa grid, the 3D DoRT grid, the ⟨3D⟩ DoRT grid, and the 1D external DoRT grid.

This version of F90 Linfor3D was written to mostly follow the program flow of the IDL counterpart. This was so that the two most computationally expensive parts of Linfor3D could be used in the same manner in both versions, as was previously mentioned.

A massive effort has been put into making sure that the heavy-duty computations (DoRT and IONOPA) are run so that memory is not needlessly consumed. This means that that one can run a typical Linfor3D job on a small computer and utilise most of the available CPUs without consuming all resources or crashing the operating system. Nevertheless, one must be cautiuous. A very small machine will not handle a 20+ snapshot model, even in sequential mode. That is because the entire flowfield must be loaded on to the master CPU before any computations can be done. Conversely, the IDL version loads the flowfield and computes data sequentially, so that only a single flowfield exists at any one time. The F90 version of Linfor3D was designed for larger jobs on larger computing facilities.

4 Installing GDL and running Linfor3D

GDL offers a free way to run the IDL version of Linfor3D so that one may run several jobs in parallel, or gain access to Linfor3D and its output when IDL licenses are not available. **Linfor3D has been successfully tested on versions of GDL 0.9.4 and later**. One must compile the shared objects, `ionopa2_idl.so` and `dort_idl.so` using gfortran rather than Intel Fortran (ifort). There appears to be some difficulties for GDL to properly interpret how array are optimised by Intel's proprietary compilers.

Since this manual was written, GDL has been improved in many ways, and has changed in many ways too. Installation of GDL used to be quite complicated. It required root privileges and a basic understanding of the Linux kernel. However, there is now a script that can take care of everything for the user. It still requires root privileges to download the required libraries from your kernel repository, but it is faster and more user friendly than before.

The code can be downloaded or cloned from GitHub https://github.com/gnudatalanguage/gdl. To clone it on the terminal, type

```
git clone git@github.com:gnudatalanguage/gdl.git
```

Once downloaded, change directories to the scripts sub-directory and run `build_gdl.sh` several times in the manner explained at https://github.com/gnudatalanguage/gdl/wiki, once you navigate to the appropriate installation guide. We cannot offer more than the following

```
$> git clone git@github.com:gnudatalanguage/gdl.git
$> cd gdl/scripts
$> sudo ./build_gdl.sh prep
$> ./build_gdl.sh configure
$> ./build_gdl.sh build
$> ./build_gdl.sh install
```

As of version 1.0.3, GDL will also run on Apple Silicon chips (ARM64), but that has never been tested with Linfor3D.

No help can or will be offered on installing GDL. We ask that you contact the developers of this code to deal with any problems you may have.

4.1 Running Linfor3D with GDL

Installing and running Linfor3D under GDL does not differ from running under IDL. However, for those who wish to exploit its new ability of running on GDL (e.g. use with HPC centres, etc.) a small change must be made in the routine `monocubic.pro`. Line 165 contains the following:

```
iout=(0 > long(interpol(findgen(n)+1.0,xin,xout))) < n
```

This must be replaced with the following more formal syntax, because of the minute differences in which GDL and IDL handle array information:

```
iout=(0 > long(interpol(findgen(n)+1.0,xin,[xout]))) < n
```

This change will not effect any part of the IDL version of Linfor3D, but prevents a fatal error when running Linfor3D under GDL. To make sure this condition is always upheld, and to make sure that user changes do not affect the output from Linfor3D, `linfor_monocubic.pro` was added to the Routines list in March 2022. Linfor3D will compile this version of monocubic during normal execution.

Finally, copy your `IDL_PATH` and `IDL_STARTUP` to `GDL_PATH` and `GDL_STARTUP`, and add in the PRO library from the GDL install to the start of the `GDL_PATH`. If this is properly done, Linfor3D will run without error by using the start guide in Sect. 2.

4.2 Running Linfor3D in parallel

The new feature of this version of Linfor3D is that it now runs on GDL, is its ability to run in parallel without the concerns of IDL licenses. This means that completion times for jobs run sequentially can be split into much quicker jobs by, e.g. snapshot or wavelength interval (for large wavelength ranges), which can later be combined. Therefore, for the first time, one can compute large wavelength ranges or complex molecules in hours, not days or weeks. This requires you to create elaborate BASH or TCSH scripts that use EOFs to edit `linfor_setcmd.pro`.

4.3 Known issues

Since the release of the Fortran 90 version, very little effort has been put into maintaining Linfor3D with GDL. It is known that there are some issues with the standard version of Linfor3D and GDL. Here is a small breakdown.

1. To avoid nasty segmentation faults when running the DoRT routines through the C interface, one must compile the GNU Fortran version of DoRT instead of the default Intel Fortran version. Array indexing optimisations in the Intel compilers are not properly dealt with by GDL.

2. GDL versions 1.0.2 and 1.0.3 are known not to work properly. A bug report was issued to the developers of GDL and a fix was put in place by version 1.0.4.

5 Basic Equations of Radiative Transfer

5.1 Transfer equation for the continuum intensity

$$\frac{\mathrm{d}\, I^c_\lambda}{\mathrm{d}\, s} = -\kappa^c_\lambda\, I^c_\lambda + \kappa^c_\lambda\, S^c_\lambda \tag{1}$$

together with the definition of the optical depth along the ray

$$\mathrm{d}\,\tau^c_\lambda = -\kappa^c_\lambda\, \mathrm{d}\, s\,, \tag{2}$$

reads

$$\frac{\mathrm{d}\, I^c_\lambda}{\mathrm{d}\,\tau^c_\lambda} = I^c_\lambda - S^c_\lambda\,. \tag{3}$$

The solution of Eq. (3) is

$$I^c_\lambda(\tau^c_\lambda) = \int_{\tau^c_\lambda}^{\tau^b_\lambda} S^c_\lambda(\tau')\, \exp\{-(\tau' - \tau^c_\lambda)\}\, \mathrm{d}\,\tau' \;+\; I^c_\lambda(\tau^b_\lambda)\, \exp\{-(\tau^b_\lambda - \tau^c_\lambda)\} \tag{4}$$

where τ^b_λ is the continuum optical depth at the lower boundary. The **emergent** continuum intensity is:

$$I^c_\lambda(\tau^c_\lambda = 0) = \int_0^{\tau^b_\lambda} S^c_\lambda(\tau')\, \exp\{-\tau'\}\, \mathrm{d}\,\tau' \;+\; I^c_\lambda(\tau^b_\lambda)\, \exp\{-\tau^b_\lambda\}\,. \tag{5}$$

Defining

$$u^c_\lambda = I^c_\lambda - S^c_\lambda \tag{6}$$

we have the transport equation

$$\frac{\mathrm{d}\, u^c_\lambda}{\mathrm{d}\,\tau^c_\lambda} = u^c_\lambda - \frac{\mathrm{d}\, S^c_\lambda}{\mathrm{d}\,\tau^c_\lambda}\,. \tag{7}$$

The solution for u^c_λ is found by replacing S^c_λ by $\mathrm{d}\, S^c_\lambda/\mathrm{d}\,\tau^c_\lambda$ in Eq.(4):

$$u^c_\lambda(\tau^c_\lambda) = \int_{\tau^c_\lambda}^{\tau^b_\lambda} \frac{\mathrm{d}\, S^c_\lambda(\tau')}{\mathrm{d}\,\tau^c_\lambda}\, \exp\{-(\tau' - \tau^c_\lambda)\}\, \mathrm{d}\,\tau' \;+\; u^c_\lambda(\tau^b_\lambda)\, \exp\{-(\tau^b_\lambda - \tau^c_\lambda)\} \tag{8}$$

The emergent intensity can also be obtained from Eq.(8):

$$I^c_\lambda(\tau^c_\lambda = 0) = S^c_\lambda(\tau^c_\lambda = 0) + \int_0^{\tau^b_\lambda} \frac{\mathrm{d}\, S^c_\lambda(\tau')}{\mathrm{d}\,\tau^c_\lambda}\, \exp\{-\tau'\}\, \mathrm{d}\,\tau' \;+\; u^c_\lambda(\tau^b_\lambda)\, \exp\{-\tau^b_\lambda\}\,. \tag{9}$$

Now we define a fixed central wavelength, λ_0, with the corresponding **fixed (universal) optical depth scale** τ_0, which is equidistant in $\log\tau_0$ and may used alternatively for all integrations. On this optical depth scale, Eq.(4) becomes

$$I^c_\lambda(\tau_0) = \int_{\tau_0}^{\tau^b_0} \frac{\kappa^c_\lambda}{\kappa^c_0}\, S^c_\lambda(\tau'_0)\, \exp\{-\left(\tau^c_\lambda(\tau'_0) - \tau^c_\lambda(\tau_0)\right)\}\, \mathrm{d}\,\tau'_0 \;+\; I^c_\lambda(\tau^b_0)\, \exp\{-\left(\tau^c_\lambda(\tau^b_0) - \tau^c_\lambda(\tau_0)\right)\}\,, \tag{10}$$

giving the continuum intensity at wavelength λ as a function of optical depth τ_0. Note the factor $\kappa^c_\lambda/\kappa^c_0$ under the integral. The intensity at the lower boundary, $I^c_\lambda(\tau^b_0)$, can be computed from the diffusion approximation,

$$I^c_\lambda(\tau^b_0) = S^c_\lambda(\tau^b_0) + \frac{\kappa^c_0}{\kappa^c_\lambda}\, \frac{\mathrm{d}\, S^c_\lambda}{\mathrm{d}\,\tau_0}(\tau^b_0)\,, \tag{11}$$

but the boundary term may also be neglected, at least for the emergent intensity, because the exponential factor is usually very small. For the **emergent** intensity we have from Eq.(5):

$$I^c_\lambda(\tau_0 = 0) = \int_0^{\tau^b_0} \frac{\kappa^c_\lambda}{\kappa^c_0}\, S^c_\lambda(\tau'_0)\, \exp\{-\tau^c_\lambda(\tau'_0)\}\, \mathrm{d}\,\tau'_0 \;+\; I^c_\lambda(\tau^b_0)\, \exp\{-\tau^c_\lambda(\tau^b_0)\}\,. \tag{12}$$

Similarly, Eq.(8) becomes

$$u_\lambda^c(\tau_0) = \int_{\tau_0}^{\tau_0^b} \frac{\mathrm{d}\, S_\lambda^c}{\mathrm{d}\,\tau_0}(\tau_0') \, \exp\{-\tau_\lambda^c(\tau_0') - \tau_\lambda^c(\tau_0)\} \, \mathrm{d}\,\tau_0' \; + \; u_\lambda^c(\tau_0^b) \, \exp\{-\tau_\lambda^c(\tau_0^b) - \tau_\lambda^c(\tau_0)\} \, . \tag{13}$$

Note the absence of the factor $\kappa_\lambda^c/\kappa_0^c$ under the integral in this case. $u_\lambda^c(\tau_0^b)$ is obtained from the diffusion approximation,

$$u_\lambda^c(\tau_0^b) = \frac{\kappa_0^c}{\kappa_\lambda^c} \frac{\mathrm{d}\, S_\lambda^c}{\mathrm{d}\,\tau_0}(\tau_0^b). \tag{14}$$

The emergent intensity can be computed from Eq.(13) as:

$$I_\lambda^c(\tau_0 = 0) = S_\lambda^c(\tau_0 = 0) + \int_0^{\tau_0^b} \frac{\mathrm{d}\, S_\lambda^c}{\mathrm{d}\,\tau_0}(\tau_0') \, \exp\{-\tau_\lambda^c(\tau_0')\} \, \mathrm{d}\,\tau_0' \; + \; u_\lambda^c(\tau_0^b) \, \exp\{-\tau_\lambda^c(\tau_0^b)\} \, . \tag{15}$$

In the latest version of Linfor3D, the continuum intensity is calculated from Eqs.(8) and (9), at 3 different wavelengths: $\lambda_0 - \Delta\lambda$, λ_0, and $\lambda_0 + \Delta\lambda$, where $\Delta\lambda$ is specified by the parameter `dclam`. We ensure that the derivative $\mathrm{d}\, S_\lambda^c/\mathrm{d}\,\tau_0$ fulfills the condition

$$\int_{\tau_1}^{\tau_2} \frac{\mathrm{d}\, S_\lambda^c}{\mathrm{d}\,\tau_\lambda}(\tau_\lambda') \, \mathrm{d}\,\tau_\lambda' = S_\lambda^c(\tau_2) - S_\lambda^c(\tau_1) \, . \tag{16}$$

The reason for using Eqs.(8) and (9) instead of Eq.(5) is that the quantity $u_\lambda^c(\tau)$ is needed to compute the line depression source function (see Sect. 5.3). We have checked that the usual transfer equation, Eq.(5), gives numerically very closely the same results for the emergent continuum intensity as Eq.(9).

5.2 Transfer equation for the line intensity

In the presence of lines, the transfer equation at wavelength λ reads

$$\frac{\mathrm{d}\, I_\lambda^\ell}{\mathrm{d}\, s} = -\left(\kappa_\lambda^c + \sum_\ell \kappa_\lambda^\ell\right) I_\lambda^\ell + \kappa_\lambda^c \, S_\lambda^c + \sum_\ell \kappa_\lambda^\ell \, S_\lambda^\ell \, . \tag{17}$$

The line source functions S_λ^ℓ may be different from the LTE continuum source function S_λ^c. With the definition of the total optical depth

$$\mathrm{d}\,\tau_\lambda = -\left(\kappa_\lambda^c + \sum_\ell \kappa_\lambda^\ell\right) \mathrm{d}\, s \equiv \mathrm{d}\,\tau_\lambda^c + \mathrm{d}\,\tau_\lambda^\ell \, , \tag{18}$$

and the total source function

$$S_\lambda = \frac{\kappa_\lambda^c \, S_\lambda^c}{\kappa_\lambda^c + \sum_\ell \kappa_\lambda^\ell} + \frac{\sum_\ell \kappa_\lambda^\ell S_\lambda^\ell}{\kappa_\lambda^c + \sum_\ell \kappa_\lambda^\ell} = \frac{S_\lambda^c + \eta\, \overline{S_\lambda^\ell}}{1+\eta} = \frac{1+\beta}{1+\eta} \, S_\lambda^c \, , \tag{19}$$

where

$$\overline{S_\lambda^\ell} = \frac{\sum_\ell \kappa_\lambda^\ell S_\lambda^\ell}{\sum_\ell \kappa_\lambda^\ell} \, , \quad \eta = \frac{\sum_\ell \kappa_\lambda^\ell}{\kappa_\lambda^c} \, , \quad \beta = \frac{\sum_\ell \kappa_\lambda^\ell S_\lambda^\ell}{\kappa_\lambda^c \, S_\lambda^c} \, , \tag{20}$$

we can write

$$\frac{\mathrm{d}\, I_\lambda^\ell}{\mathrm{d}\,\tau_\lambda} = I_\lambda^\ell - S_\lambda \, . \tag{21}$$

In LTE, $S_\lambda = S_\lambda^c$. The solution of Eq.(21) is

$$I_\lambda^\ell(\tau_\lambda = 0) = \int_0^{\tau_\lambda^b} S_\lambda(\tau_\lambda') \, \exp\{-\tau_\lambda'\} \, \mathrm{d}\,\tau_\lambda' \; + \; I_\lambda^\ell(\tau_\lambda^b) \, \exp\{-\tau_\lambda^b\} \, . \tag{22}$$

In analogy to Eq.(12), we can also obtain the emergent line intensity by integration on the universal optical depth scale τ_0:

$$I_\lambda^\ell(\tau_0 = 0) = \int_0^{\tau_0^b} \frac{\kappa_\lambda^c}{\kappa_0^c} (1+\eta)\, S_\lambda(\tau_0')\, \exp\{-\tau_\lambda(\tau_0')\}\, \mathrm{d}\,\tau_0' \;+\; I_\lambda^\ell(\tau_0^b)\, \exp\{-\tau_\lambda(\tau_0^b)\}\,, \tag{23}$$

or, substituting S_λ from Eq.(19),

$$I_\lambda^\ell(\tau_0 = 0) = \int_0^{\tau_0^b} \frac{\kappa_\lambda^c}{\kappa_0^c} (1+\beta)\, S_\lambda^c(\tau_0')\, \exp\{-\tau_\lambda(\tau_0')\}\, \mathrm{d}\,\tau_0' \;+\; I_\lambda^\ell(\tau_0^b)\, \exp\{-\tau_\lambda(\tau_0^b)\}\,. \tag{24}$$

Integration on the $\log \tau_0$ scale ($z_0 \equiv \log \tau_0$) gives:

$$I_\lambda^\ell(z_0^a) = \int_{z_0^a}^{z_0^b} \ln(10)\, \tau_0(z_0') \frac{\kappa_\lambda^c}{\kappa_0^c} (1+\beta)\, S_\lambda^c(z_0')\, \exp\{-\tau_\lambda(z_0')\}\, \mathrm{d}\, z_0' \;+\; I_\lambda^\ell(z_0^b)\, \exp\{-\tau_\lambda(z_0^b)\}\,, \tag{25}$$

where z_0^a is the minimum log optical depth. Alternatively, in analogy to Eq.(9) we obtain:

$$I_\lambda^\ell(\tau_\lambda = 0) = S_\lambda(\tau_\lambda = 0) + \int_0^{\tau_\lambda^b} \frac{\mathrm{d}\, S_\lambda}{\mathrm{d}\,\tau_\lambda}(\tau_\lambda')\, \exp\{-\tau_\lambda'\}\, \mathrm{d}\,\tau_\lambda' \;+\; u_\lambda^\ell(\tau_\lambda^b)\, \exp\{-\tau_\lambda^b\}\,, \tag{26}$$

where we have defined

$$u_\lambda^\ell = I_\lambda^\ell - S_\lambda\,, \tag{27}$$

which in the diffusion approximation may be written as

$$u_\lambda^\ell(\tau_\lambda^b) = \frac{\mathrm{d}\, S_\lambda}{\mathrm{d}\,\tau_\lambda}(\tau_\lambda^b) \quad \text{or} \quad u_\lambda^\ell(\tau_0^b) = \frac{\kappa_0^c/\kappa_\lambda^c}{1+\eta} \frac{\mathrm{d}\, S_\lambda}{\mathrm{d}\,\tau_0}(\tau_0^b). \tag{28}$$

On the universal optical depth scale τ_0 we obtain from Eq.(26:

$$I_\lambda^\ell(\tau_0 = 0) = S_\lambda(\tau_0 = 0) + \int_0^{\tau_0^b} \frac{\mathrm{d}\, S_\lambda}{\mathrm{d}\,\tau_0}(\tau_0')\, \exp\{-\tau_\lambda(\tau_0')\}\, \mathrm{d}\,\tau_0' \;+\; u_\lambda^\ell(\tau_0^b)\, \exp\{-\tau_\lambda(\tau_0^b)\}\,, \tag{29}$$

In LTE, where $S_\lambda = S_\lambda^c$, the integral in Eq.(29) differs from the integral in Eq.(15) only by the exponential factor which involves the total optical depth τ_λ instead of the continuum optical depth τ_λ^c. The absolute line depression is then calculated as

$$D_\lambda = I_\lambda^c(\tau = 0) - I_\lambda^\ell(\tau = 0)\,. \tag{30}$$

In the current version of Linfor3D, Eq.(25) is used if the parameter `intline` is set to -1, and Eq.(26) is used if `intline` is set to -2.

5.3 Transfer equation for the line depression

We may analyse the transfer equation for the absolute line depression defined in Eq.(30):

$$\frac{\mathrm{d}\, D_\lambda}{\mathrm{d}\, s} = \frac{\mathrm{d}\, I_\lambda^c}{\mathrm{d}\, s} - \frac{\mathrm{d}\, I_\lambda^\ell}{\mathrm{d}\, s} = -\kappa_\lambda^c I_\lambda^c + \left(\kappa_\lambda^c + \sum_\ell \kappa_\lambda^\ell\right) I_\lambda^\ell - \sum_\ell \kappa_\lambda^\ell S_\lambda^\ell \tag{31}$$

or

$$\frac{\mathrm{d}\, D_\lambda}{\mathrm{d}\, s} = -\left(\kappa_\lambda^c + \sum_\ell \kappa_\lambda^\ell\right) D_\lambda + \left(I_\lambda^c \sum_\ell \kappa_\lambda^\ell - \sum_\ell \kappa_\lambda^\ell S_\lambda^\ell\right) \tag{32}$$

or

$$\frac{\mathrm{d}\, D_\lambda}{\mathrm{d}\,\tau_\lambda} = D_\lambda - S_\lambda^D\,, \tag{33}$$

where the line depression source function is

$$S_\lambda^D = \frac{\eta}{1+\eta}\left(I_\lambda^c - \overline{S_\lambda^\ell}\right) = \frac{\eta}{1+\eta}\left((I_\lambda^c - S_\lambda^c) + (S_\lambda^c - \overline{S_\lambda^\ell})\right). \tag{34}$$

In LTE, $\overline{S_\lambda^\ell} = S_\lambda^c$, and

$$S_\lambda^D = \frac{\eta}{1+\eta}(I_\lambda^c - S_\lambda^c). \tag{35}$$

The solution of Eq.(33) is

$$D_\lambda(\tau_\lambda = 0) = \int_0^{\tau_\lambda^b} S_\lambda^D(\tau'_\lambda)\, \exp\{-\tau'_\lambda\}\, \mathrm{d}\,\tau'_\lambda\,, \tag{36}$$

neglecting the boundary term. Integration on the fixed τ_0 scale:

$$D_\lambda(\tau_0 = 0) = \int_0^{\tau_0^b} \frac{\kappa_\lambda^c}{\kappa_0^c}(1+\eta)\, S_\lambda^D(\tau'_0)\, \exp\{-\tau_\lambda(\tau'_0)\}\, \mathrm{d}\,\tau'_0\,. \tag{37}$$

Substituting S_λ^D from Eq.(34) gives

$$D_\lambda(\tau_0 = 0) = \int_0^{\tau_0^b} \frac{\kappa_\lambda^c}{\kappa_0^c}\,\eta\left(I_\lambda^c - \overline{S_\lambda^\ell}\right) \exp\{-\tau_\lambda(\tau'_0)\}\, \mathrm{d}\,\tau'_0\,, \tag{38}$$

where κ_λ^c, κ_0^c, η, I_λ^c, $\overline{S_\lambda^\ell}$, and τ_λ are defined as a function of τ_0. We can also write

$$D_\lambda(\tau_0 = 0) = \int_0^{\tau_0^b} \frac{\kappa_\lambda^c}{\kappa_0^c}\,\eta\left(u_\lambda^c + (S_\lambda^c - \overline{S_\lambda^\ell})\right) \exp\{-\tau_\lambda(\tau'_0)\}\, \mathrm{d}\,\tau'_0\,, \tag{39}$$

where

$$\frac{\eta}{1+\eta}(S_\lambda^c - \overline{S_\lambda^\ell}) = S_\lambda^c\,\frac{(\eta-\beta)}{1+\eta} \tag{40}$$

is the NLTE correction to the line depression source function. Integration on the $\log\tau_0$ scale ($z_0 \equiv \log\tau_0$) gives:

$$D_\lambda(z_0^a) = \int_{z_0^a}^{z_0^b} \ln(10)\,\tau_0(z'_0)\,\frac{\kappa_\lambda^c}{\kappa_0^c}\,\eta\left(u_\lambda^c + (S_\lambda^c - \overline{S_\lambda^\ell})\right) \exp\{-\tau_\lambda(z'_0)\}\, \mathrm{d}\,z'_0\,. \tag{41}$$

In the current version of Linfor3D, Eq.(41) is used to compute the line depression if the parameter `intline` is set to 1, while Eq.(36) is used if `intline` = 2.

5.4 Contribution functions

The Continuum Intensity Contribution Function for a ray with inclination angle $\mu = \cos\theta$, azimuthal angle ϕ, and wavelength λ is simply the horizontal average of the integrand of Eq.(12)

$$C_I^c(\tau_0, \mu, \phi, \lambda) = \frac{1}{\mu}\left\langle \frac{\kappa_\lambda^c}{\kappa_0^c}\, S_\lambda^c(\tau_0/\mu)\, \exp\{-\tau_\lambda^c(\tau_0/\mu)\}\right\rangle_{x,y}, \tag{42}$$

such that

$$I_\lambda^c(\tau_0 = 0, \mu, \phi, \lambda) = \int_0^{\tau_0^b} C_I^c(\tau'_0, \mu, \phi, \lambda)\, \mathrm{d}\,\tau'_0 = \int_0^{z_0^b} \ln(10)\,\tau_0(z'_0)\, C_I^c(\tau_0(z'_0), \mu, \phi, \lambda)\, \mathrm{d}\,z'_0\,. \tag{43}$$

Note that now (τ_0/μ) is the optical depth along the line-of-sight, and τ_0 is the corresponding vertical optical depth (a formal quantity in the presence of horizontal inhomogeneities).

The Continuum Flux Contribution Function at wavelength λ is consequently

$$C_F^c(\tau_0, \lambda) = \int_0^{2\pi}\int_0^1 \mu'\, C_I^c(\tau_0, \mu', \phi', \lambda)\, \mathrm{d}\,\mu'\, \mathrm{d}\,\phi'\,, \tag{44}$$

such that

$$F^c_\lambda(\tau_0 = 0, \lambda) = \int_0^{\tau_0^b} C^c_F(\tau'_0, \lambda)\, \mathrm{d}\,\tau'_0 \;=\; \int_0^{z_0^b} \ln(10)\, \tau_0(z'_0)\, C^c_F(\tau_0(z'_0), \lambda)\, \mathrm{d}\, z'_0\,. \tag{45}$$

Note that the horizontal averaging in Eq.(42) works only because the transfer equation is integrated on the fixed universal optical depth scale, τ_0. The contribution functions $C^c_I(\tau_0, \mu_0, \phi_0, \lambda_0)$ and $C^c_F(\tau_0, \lambda_0)$ are saved in `contf.cfc3i` and `contf.cfc3f`, respectively. Corresponding contribution functions are also computed for the $\langle 3D \rangle$ model and saved in `contf.cfc1i` and `contf.cfc1f`, respectively, and for the external 1D reference atmosphere (`contf.cfcxi` and `contf.cfcxf`).

Similarly, we can also write down the Line Intensity Contribution Function as the horizontal average of the integrand of Eq.(24):

$$C^\ell_I(\tau_0, \mu, \phi, \lambda) = \frac{1}{\mu} \left\langle \frac{\kappa^c_\lambda}{\kappa^c_0} (1+\beta)\, S^c_\lambda(\tau_0/\mu)\, \exp\{-\tau_\lambda(\tau_0/\mu)\} \right\rangle_{x,y}\,, \tag{46}$$

such that the intensity at a given wavelength in the line profile is

$$I^\ell_\lambda(\tau_0 = 0, \mu, \phi, \lambda) = \int_0^{\tau_0^b} C^\ell_I(\tau'_0, \mu, \phi, \lambda)\, \mathrm{d}\,\tau'_0 \;=\; \int_0^{z_0^b} \ln(10)\, \tau_0(z'_0)\, C^\ell_I(\tau_0(z'_0), \mu, \phi, \lambda)\, \mathrm{d}\, z'_0\,. \tag{47}$$

The Line Flux Contribution Function at wavelength λ is

$$C^\ell_F(\tau_0, \lambda) = \int_0^{2\pi} \int_0^1 \mu'\, C^\ell_I(\tau_0, \mu', \phi', \lambda)\, \mathrm{d}\,\mu'\, \mathrm{d}\,\phi'\,, \tag{48}$$

such that

$$F^\ell_\lambda(\tau_0 = 0, \lambda) = \int_0^{\tau_0^b} C^\ell_F(\tau'_0, \lambda)\, \mathrm{d}\,\tau'_0 \;=\; \int_0^{z_0^b} \ln(10)\, \tau_0(z'_0)\, C^\ell_F(\tau_0(z'_0), \lambda)\, \mathrm{d}\, z'_0\,. \tag{49}$$

$C^\ell_I(\tau_0, \mu_0, \phi_0, \lambda_0)$ and $C^\ell_F(\tau_0, \lambda_0)$ are stored in `contf.cfl3i` and `contf.cfl3f`, respectively, and similarly for the 1D atmospheres in `contf.cfl1i`, `contf.cfl1f`, `contf.cflxi`, and `contf.cflxf`.

Formally, a Line Depression Contribution Function could be defined as

$$\tilde{C}^D_I = C^c_I - C^\ell_I = \frac{1}{\mu} \left\langle \frac{\kappa^c_\lambda}{\kappa^c_0} S^c_\lambda(\tau_0/\mu)\, \exp\{-\tau^c_\lambda(\tau_0/\mu)\} \left(1 - (1+\beta)\, \exp\{-\tau^\ell_\lambda(\tau_0/\mu)\}\right) \right\rangle_{x,y}\,, \tag{50}$$

such that the absolute line depression at any wavelength in the line profile is

$$D_I(\tau_0 = 0, \mu, \phi, \lambda) = \int_0^{\tau_0^b} \tilde{C}^D_I(\tau'_0, \mu, \phi, \lambda)\, \mathrm{d}\,\tau'_0 \;=\; \int_0^{z_0^b} \ln(10)\, \tau_0(z'_0)\, \tilde{C}^D_I(\tau_0(z'_0), \mu, \phi, \lambda)\, \mathrm{d}\, z'_0\,. \tag{51}$$

Note however, that $\tilde{C}^D_I$ does not have the desired physical meaning, because the factor $(1 - (1+\beta)\, \exp\{-\tau^\ell_\lambda\})$ (i) becomes negative when τ^ℓ_λ is small (τ^ℓ_λ is the optical depth due to the line opacity only), and (ii) it is non-zero also in layers where the line opacity vanishes. For this reason, $\tilde{C}^D_I$ is not considered useful and hence is not computed in the current version of Linfor3D.

A much better way to define the Line Depression Contribution Function is to consider Eq.(39) and to define it as

$$C^D_I(\tau_0, \mu, \phi, \lambda) = \frac{1}{\mu} \left\langle \frac{\kappa^c_\lambda}{\kappa^c_0} \left(\eta\, u^c_\lambda(\tau_0/\mu) + (\eta - \beta)\, S^c_\lambda(\tau_0/\mu)\right) \exp\{-\tau_\lambda(\tau_0/\mu)\} \right\rangle_{x,y}\,. \tag{52}$$

Note that this contribution function vanisches whereever the line opacity (η, β) is zero. For the flux spectrum we define, as before,

$$C^D_F(\tau_0, \lambda) = \int_0^{2\pi} \int_0^1 \mu'\, C^D_I(\tau_0, \mu', \phi', \lambda)\, \mathrm{d}\,\mu'\, \mathrm{d}\,\phi'\,. \tag{53}$$

Then the absolute line depression at any wavelength in the line profile is

$$D_I(\tau_0 = 0, \mu, \phi, \lambda) = \int_0^{\tau_0^b} C_I^D(\tau_0', \mu, \phi, \lambda)\,\mathrm{d}\,\tau_0' = \int_0^{z_0^b} \ln(10)\,\tau_0(z_0')\,C_I^D(\tau_0(z_0'), \mu, \phi, \lambda)\,\mathrm{d}\,z_0'\,, \tag{54}$$

and

$$D_F(\tau_0 = 0, \lambda) = \int_0^{\tau_0^b} C_F^D(\tau_0', \lambda)\,\mathrm{d}\,\tau_0' = \int_0^{z_0^b} \ln(10)\,\tau_0(z_0')\,C_F^D(\tau_0(z_0'), \lambda)\,\mathrm{d}\,z_0'\,, \tag{55}$$

for the intensity and flux spectrum, respectively.

The Equivalent Width Contribution Function is computed as

$$C_I^W(\tau_0, \mu, \phi) = \int_\lambda C_I^D(\tau_0, \mu, \phi, \lambda')\mathrm{d}\,\lambda'\,, \tag{56}$$

and

$$\begin{aligned} W_I(\mu, \phi) &= \frac{1}{\langle I_\lambda^c(\mu, \phi, \lambda)\rangle} \int_0^{\tau_0^b} C_I^W(\tau_0', \mu, \phi)\,\mathrm{d}\,\tau_0' \\ &= \frac{1}{\langle I_\lambda^c(\mu, \phi, \lambda)\rangle} \int_0^{z_0^b} \ln(10)\,\tau_0(z_0')\,C_I^W(\tau_0(z_0'), \mu, \phi)\,\mathrm{d}\,z_0'\,, \end{aligned} \tag{57}$$

where

$$\langle I_\lambda^c(\mu, \phi, \lambda)\rangle = \frac{\int_\lambda D_I(\mu, \phi, \lambda')\,\mathrm{d}\,\lambda'}{\int_\lambda D_I(\mu, \phi, \lambda')/I_\lambda^c(\mu, \phi, \lambda')\,\mathrm{d}\,\lambda'}\,. \tag{58}$$

For the flux spectrum we have

$$C_F^W(\tau_0) = \int_\lambda C_F^D(\tau_0, \lambda')\mathrm{d}\,\lambda'\,, \tag{59}$$

and

$$\begin{aligned} W_F &= \frac{1}{\langle F_\lambda^c(\lambda)\rangle} \int_0^{\tau_0^b} C_F^W(\tau_0')\,\mathrm{d}\,\tau_0' \\ &= \frac{1}{\langle F_\lambda^c(\lambda)\rangle} \int_0^{z_0^b} \ln(10)\,\tau_0(z_0')\,C_F^W(\tau_0(z_0'))\,\mathrm{d}\,z_0'\,, \end{aligned} \tag{60}$$

with

$$\langle F_\lambda^c(\lambda)\rangle = \frac{\int_\lambda D_F(\lambda')\,\mathrm{d}\,\lambda'}{\int_\lambda D_F(\lambda')/F_\lambda^c(\lambda')\,\mathrm{d}\,\lambda'}\,. \tag{61}$$

Irrespective of the parameter `intline`, the structures `contf.cfd3i` and `contf.cfd3f` hold the contribution functions $C_I^D(\tau_0, \mu 0, \phi_0, \lambda 0)$ and $C_F^D(\tau_0, \lambda 0)$, while $C_I^W(\tau_0, \mu 0, \phi_0)$ and $C_F^W(\tau_0)$ are stored in `contf.cfw3i` and `contf.cfw3f`, respectively.

5.5 Grey test case

If `cmd.context` is set to 'grey', a 3D ($n_x = n_y = 10$) hydrostatic atmosphere is constructed, instead of reading a 3D model. The temperature stratification on the Rosseland optical depth scale is given by

$$T(\tau_{\mathrm{Ross}}) = T_{\mathrm{eff}} \left(\frac{1}{2} + \frac{3}{4}\,\tau_{\mathrm{Ross}}\right)^{1/4} \tag{62}$$

and the source function is linear in τ_{Ross}:

$$S(\tau_{\mathrm{Ross}}) = \frac{\sigma}{\pi}\,T_{\mathrm{eff}}^4 \left(\frac{1}{2} + \frac{3}{4}\,\tau_{\mathrm{Ross}}\right). \tag{63}$$

The Eddington-Barbier relation is strictly correct in this case. For any inclination $\mu = \cos\,\theta$, the emergent continuum intensity is given by

$$I_c(\mu) = \frac{\sigma}{\pi}\,T_{\rm eff}^4\left(\frac{1}{2}+\frac{3}{4}\,\mu\right). \tag{64}$$

In particular, at disk-center ($\mu = 1$) the continuum intensity is

$$I_c(\mu=1) = \frac{5}{4}\,\frac{\sigma}{\pi}\,T_{\rm eff}^4, \tag{65}$$

and the flux is

$$F_c = 2\,\pi\int_0^1 \mu\,I_c(\mu)\,{\rm d}\,\mu = \sigma\,T_{\rm eff}^4. \tag{66}$$

Comparison of the results obtained from Linfor3D for continuum intensity and flux for

```
TEFF = 5000.00, GRAV = 316.200
LUTAU1 = -8.0000000, LUTAU2 = 2.0000000, DLUTAU = 0.0800000
OPAFILE = 't5000g250mm30_marcs_idmean3xRT3.opta',
GASFILE = 'gas_cifist2006_m30_a04_l5.eos',
EOSFILE = 'eos_cifist2006_m30_a04_l5.eos'
```

with the above theoretical results yields (Linfor3D 3.1.3):

ratio	ntheta				
numerical / analytical	1	2	3	−3	4
I_c(linfor3D)/I_c(Eq.(65)	1.0005573	1.0005573	1.0005573	1.0005573	1.0005573
F_c(linfor3D)/F_c(Eq.(66)	1.0004105	1.0148776	1.0079553	1.0004507	1.0050481

If the ratio η of line opacity, κ_ℓ and continuum opacity, κ_c is constant with optical depth ($\eta = \kappa_\ell/\kappa_c$), the intensity in the line is simply

$$I_\ell(\mu) = \frac{\sigma}{\pi}\,T_{\rm eff}^4\left(\frac{1}{2}+\frac{3}{4}\,\frac{\mu}{1+\eta}\right), \tag{67}$$

the **absolute** line depression is

$$D_I(\mu) = I_c(\mu) - I_\ell(\mu) = \frac{\sigma}{\pi}\,T_{\rm eff}^4\,\frac{3}{4}\,\mu\,\frac{\eta}{1+\eta}, \tag{68}$$

and the **relative** line depression at disk-center is

$$D_I(\mu=1)/I_c(\mu=1) = \frac{3}{5}\,\frac{\eta}{1+\eta}. \tag{69}$$

The absolute line depression for flux is

$$D_F = F_c - F_\ell = 2\,\pi\int_0^1 \mu\,D_I(\mu)\,{\rm d}\,\mu = \sigma\,T_{\rm eff}^4\,\frac{1}{2}\,\frac{\eta}{1+\eta}, \tag{70}$$

and the relative line depression for flux is

$$D_F/F_c = \frac{1}{2}\,\frac{\eta}{1+\eta}. \tag{71}$$

The ratio between the relative line depression in flux and at disk-center is therfore 5/6, and the same ratio holds for the equivalent widths.

The local absorption line profile is now defined by

$$\eta(\alpha, v) = \eta_0\,H(\alpha, v), \tag{72}$$

where $v = (\lambda - \lambda_0)/\Delta\lambda_D$, and $\alpha = \Delta\lambda_N/2/\Delta\lambda_D$ ($\Delta\lambda_D$: Doppler width, $\Delta\lambda_N$: full width at half maximum of the Lorentzian damping profile). The 'Voigt function' $H(\alpha, v)$ is normalized such that (for $\alpha \ll 1$), $H(\alpha, v = 0) \approx 1$. Assuming that η_0, α, and $\Delta\lambda_D$ are constant, we can compute the emergent line profile from Eq. (69) or (71). At disk-center, we have

$$D_I(\mu = 1)/I_c(\mu = 1) = R_I = \frac{3}{5} \frac{\eta_0 H(\alpha, v)}{\eta_0 H(\alpha, v) + 1}, \tag{73}$$

and for flux

$$D_F/F_c = R_F = \frac{1}{2} \frac{\eta_0 H(\alpha, v)}{\eta_0 H(\alpha, v) + 1}. \tag{74}$$

Clearly, the emergent line profiles are no longer Voigt profiles due to saturation effects.

The (reduced) disk-center equivalent width is obtained from numerical integration of the emergent line profile:

$$\tilde{W}_I = \int_{-\infty}^{+\infty} R_I(v, \eta_0, \alpha)\, \mathrm{d}\, v, \tag{75}$$

and $\tilde{W}_F = 5/6\, \tilde{W}_I$. An 'analytical' Curve-of-Growth, $\tilde{W}(\eta_0; \alpha = 0.01)$ is shown in Fig. 2.

The equivalent width in [mA] is obtained from the reduced equivalent width by

$$W_\lambda[\mathrm{m\AA}] = 1000\, \lambda_0[\mathrm{\AA}]\, \frac{\Delta v_D}{c}\, \tilde{W}. \tag{76}$$

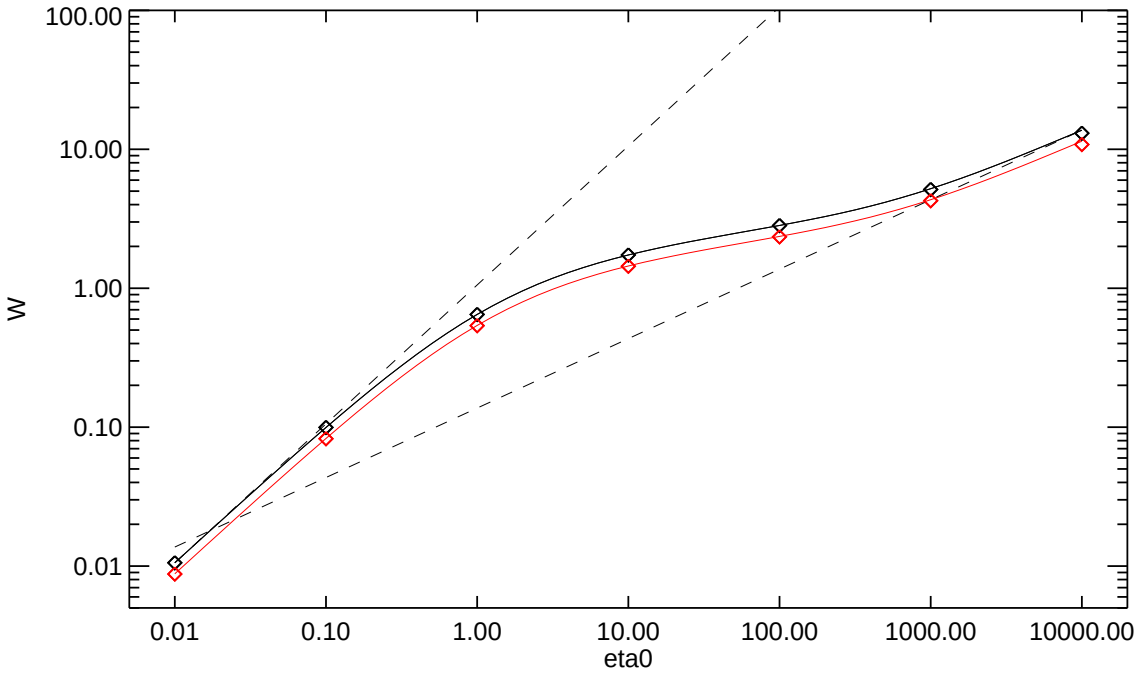


Figure 2: Analytical Curve-of-Growth showing the (reduced!) equivalent width (integrated from $v = -100$ to $v = +100$) as a function of η_0, assuming $\alpha = 0.01$. **Black**: disk-center, **red**: flux. The dashed lines have slopes 0.5 and 1.0. Diamonds show the numerical results obtained with Linfor3D (integration from $v = -50$ to $v = +50$).

The results of a number of test calculations are listed below. The wavelength resolution was chosen to be 1/10 of the Doppler width: $\delta\lambda = 0.1\, \lambda_0\, \Delta v_D/c$. The wavelength range was set to ±50 Doppler widths; $\Delta v_D = 6$ km/s, $\alpha = 0.01$. The line file used for the test calculations is shown below.

```
alam     Vdop     eta0    avgt    dlam    ddlam
7    7
Test    grey sf  Vdop=2.D-5, eta0=1.0D-2, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D-2   1.0D-2  4.00D0  0.80D-2
Test    grey sf  Vdop=2.D-5, eta0=1.0D-1, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D-1   1.0D-2  4.00D0  0.80D-2
Test    grey sf  Vdop=2.D-5, eta0=1.0D0, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D0    1.0D-2  4.00D0  0.80D-2
Test    grey sf  Vdop=2.D-5, eta0=1.0D1, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D1    1.0D-2  4.00D0  0.80D-2
Test    grey sf  Vdop=2.D-5, eta0=1.0D2, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D2    1.0D-2  4.00D0  0.80D-2
Test    grey sf  Vdop=2.D-5, eta0=1.0D3, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D3    1.0D-2  4.00D0  0.80D-2
Test    grey sf  Vdop=2.D-5, eta0=1.0D4, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D4   1.0D-2  4.00D0  0.80D-2
clam        gfscale
-4000.000   1.0
```

For the following tabulations we have defined

$$\Delta W_I = \log_{10} W_I(\text{linfor3D}) - \log_{10} W_I(\text{Eq.(75)}\,, \tag{77}$$

and

$$\Delta W_F = \log_{10} W_F(\text{linfor3D}) - \log_{10} \frac{5}{6} W_I(\text{Eq.(75)}\,, \tag{78}$$

These results are obtained with `intline=1`:

η_0	ΔW_I	ΔW_F		
	[dex]	ntheta=−3	ntheta=3	ntheta=4
1.0E-02	+0.000342	+0.000336	−0.002949	−0.001690
1.0E-01	+0.000331	+0.000327	−0.002958	−0.001698
1.0E+00	+0.000269	+0.000271	−0.003014	−0.001755
1.0E+01	+0.000072	+0.000081	−0.003203	−0.001942
1.0E+02	−0.000820	−0.000807	−0.004088	−0.002831
1.0E+03	−0.005441	−0.005432	−0.008714	−0.007456
1.0E+04	−0.021428	−0.021421	−0.024704	−0.023446

These results are obtained with `intline=-2`:

η_0	ΔW_I	ΔW_F		
	[dex]	ntheta=−3	ntheta=3	ntheta=4
1.0E-02	+0.000334	+0.000328	−0.002957	−0.001698
1.0E-01	+0.000324	+0.000320	−0.002965	−0.001706
1.0E+00	+0.000261	+0.000263	−0.003022	−0.001762
1.0E+01	−0.000063	+0.000074	−0.003211	−0.001950
1.0E+02	−0.000825	−0.000814	−0.004097	−0.002838
1.0E+03	−0.005447	−0.005438	−0.008720	−0.007463
1.0E+04	−0.021432	−0.021425	−0.024708	−0.023450

6 Program Files

In this section all the program files making up the two versions of Linfor3D are listed. While the two codes work independently, and the user is free to choose which version of the code they wish to run, it should be noted that there are some Fortran-based routines and modules that both versions of the code share. All of those routines are found in the subdirectory <LINFOR_DIRECTORY>/xmono/:

File name	Type	Description
dort.F90	M/S	Does all radiative transfer
cubint_module.f90	M	Performs a cubic interpolation/ integration
ha_convol_module.f90	M	Used by cubint_module.f90
ionopa2.f90	S	Control routine for the IONDIS opacity package
iondis.f90	S	Computes pressures and densities
opalam.f90	S	Computes opacities
abuini.f90	S	Initializes the IONDIS package

Table 2: List of routines shared by F90 and IDL/GDL versions of Linfor3D: the table shows the file name, the type (**S**ubroutine or **M**odule, and its description.

In the first three subsections below, an overview of the program flow, the structures in common block linfordata, and a list of its programs are provided for the IDL/GDL version of Linfor3D. The final two subsections describe the program flow in the F90 version of Linfor3D, followed by a list of its modules with a brief description.

6.1 IDL program flow

Basically, the calling sequence is as follows (incomplete listing of linfor_3D.pro):

- Read input parameters (linfor_setcmd.pro)
- Initialize atomic data (linfor_atom.pro)
- Read line data: (linfor_rdline.pro)
- Initialize ionopa abundances, opacity tables and EOS tables
- Set constants (linfor_init)
- Define ff, type linfor_flowfield (linfor_flowfield__define.pro)
- Define f1, type linfor_flowfield: (linfor_flowfield__define.pro)
- Define fx, type linfor_flowfield: (linfor_flowfield__define.pro)
- Define ss, type linfor_spectrum: (linfor_spectrum__define.pro)
- Define s1, type linfor_spectrum: (linfor_spectrum__define.pro)
- Define sx, type linfor_spectrum: (linfor_spectrum__define.pro)
- Read model data into ff structure (linfor_rduio.pro)
- Recompute model on refined z-grid (linfor_regrid.pro)

- Compute ionopa quantities (pe, kappa, zeta) and monochromatic tau for 3D model (`linfor_ionopa_3d.pro`)
- Construct 1D reference atmosphere from `ff`, store in `f1`: (`linfor_refatm.pro`)
- Compute ionopa quantities (pe, kappa, zeta) and monochromatic tau for 1D reference atmosphere (`linfor_ionopa_3d`)
- Do radiative transfer calculations for 3D model (`linfor_dort.pro`)
- Do radiative transfer calculations for averaged 3D atmosphere (`linfor_dort.pro`)
- Store results for later evaluation (`linfor_eval, ss, s1, nf, kl`)
- Make Plots of line profiles and bisectors (`linfor_plot1.pro`)
- Do radiative transfer calculations for 1D reference atmosphere (`linfor_dort.pro`)
- Store results for later evaluation (`linfor_evalx.pro`)
- Create postcript file(s) (`linfor_plot2.pro`)
- Generate output files `linfor_3D_1.uiosave` and `linfor_3D_2.uiosave` (`uio_save.pro`).
- (Generate `linfor_3D_3.uiosave` if `cc3d_flag=1`.)
- (Generate `linfor_3D_1X.uiosave` if `run_flag=-3`.)
- Free pointers to structures `ff`, `f1`, `fx`, `ss`, `s1`, and `sx` if `free_flag` = 1 (see Sect. 7.2) (`linfor_flowfield_free.pro`)

6.2 Structures in Common Block linfordata

Table 3 shows a list of the structrues in common block 'linfordata' used by the linfor_3D package.

Structure	Defined in	Description
`atom`	`linfor_atom.pro`	Atomic weights & ionization potentials
`const`	`linfor_init.pro`	Physical & model constants
`cmd`	`linfor_setcmd.pro`	Input parameters controlling program execution
`line`	`linfor_rdline.pro`	Line data derived from 'line.dat'
`gas`	`linfor_init.pro`	GAS tables initialized by 'tabinter_rdcoeff'
`eos`	`linfor_init.pro`	EOS tables initialized by 'tabinter_rdcoeff'
`result`	`linfor_init.pro`	Basic results for computing abundance corrections

Table 3: List of all structures in common block 'linfordata': the table shows the name of the structure, the routine where it is defined, and a description. A brief description of the arrays/sub-structures contained within each structure is given in Sect. 10.

6.3 IDL/GDL Files

Table 4 shows a list of all source files necessary to run Linfor3D.
All **IDL/GDL** versions of Linfor3D require the UIO library to handle the I/O of the CO^{5}BOLD files, and version 6.0.0 and above requires them for ALL I/O done during the program flow.

File name	Type	Description
linfor_3D.pro	S	main program
linfor_flowfield__define.pro	S	Definition of flow field structure
linfor_spectrum__define.pro	S	Definition of spectrum structure
linfor_raysys__define.pro	S	Definition of ray system structure
linfor_atom.pro	S	Defines atomic data
linfor_setwts.pro	S	Defines weights for angle quadrature (deprecated after version 6.2.7)
linfor_setwts_lobatto.pro	S	Replacement for linfor_setwts.pro. Further explanations given in Sect. 7.11
linfor_setwts_dblgaus.pro	S	Additional definition for angle quadrature. See Sect. 7.11
linfor_setwts_dblrdau.pro	S	Additional definition for angle quadrature. See Sect. 7.11
linfor_setwts_special.pro	S	Special definition for angle quadrature. See Sect. 7.11
linfor_setcmd.pro	S	Command file, parameter input
linfor_rdxatm.pro	S	Reads 1D reference atmosphere, calling linfor_rdatmos, linfor_rdatlas9, linfor_rdmarcs, rdl50 or linfor_rdfalmod
linfor_rdatlas9.pro	S	Reads ATLAS9 1D atmosphere (atm.dat)
linfor_rdmarcs.pro	S	Reads MARCS 1D atmosphere (atm.dat)
linfor_rdfalmod.pro	S	Reads FAL 1D atmosphere (atm.dat)
linfor_rdatmos.pro	S	Reads ATMOS 1D atmosphere (atm.dat)
linfor_rdf15.pro	S	Reads a sequence of FOR15 snapshots from 2D Kiel hydro simulations (FOR15)
linfor_rdsav.pro	S	Reads 3D snapshot from Copenhagen code (savfs)
linfor_rduio.pro	S	Reads 3D snapshot from CO^{5}BOLD uio output files
linfor_rdvog.pro	S	Reads 3D snapshot from Voegler MHD code
linfor_findff.pro	S	Finds cached flow fields
linfor_rdline.pro	S	Reads line data (line.dat)
linfor_init.pro	S	Initializes ionopa, EOS, Opacities, several constants
linfor_bisector.pro	S	Computes line bisector positions called by linfor_plot1 and linfor_plot2
linfor_convol.pro	S	Convolves line profile with Gauss kernel called by linfor_plot1 and linfor_plot2
linfor_dort.pro	S	Computes spectrum from flow field (main RT module calling several lower level routines)
linfor_eval.pro	S	Evaluates mean spectrum, “abundance corrections”
linfor_evalx.pro	S	Evaluates reference spectrum, “abundance corrections”
linfor_incline.pro	S	Inclines 3D flow field, called by linfor_ztau
linfor_ionopa_3d.pro	S	Calculates electron pressure, ionization fractions, and monochromatic optical depth for given flow field
linfor_rad3.pro	S	Integration of RT equation
linfor_refatm.pro	S	Define 1D reference atmosphere from 3D flow field
linfor_regrid.pro	S	Cut out surface layers from original model, re-define grid
linfor_tauinfo.pro	S	Prints information about optical depth scales
linfor_ztau.pro	S	Prepares bundle of (inclined) rays on monochromatic tau
linfor_monocubic.pro	F	Performs monotonic piecewise cubic interpolation.

rdl50.pro	S	Reads the LHD l50 file format.
linfor_plot0.pro	S	Plots flow field
linfor_plot1.pro	S	Plots spatially resolved line profiles
linfor_plot2.pro	S	Plots averaged line profiles
(linfor_plot3.pro)	S	Plots monochromatic granulation images
alpha_line.pro	F	Computes α-parameter for VOIGT function
eta0.pro	F	Computes η_0, the opacity at line center of metal lines
rrca.pro	F	Computes mean square orbital radius of electron (Unsöld)
vdop.pro	F	Computes (thermal+turbulent) Doppler velocity [c_s]
linfor_timing.pro	S	Prepares and gathers timing statistics
linfor_timing_print.pro	S	Print timing statistics
uio_save.pro	S	UIO formatted save procedure
uio_restore.pro	S	UIO formatted restore procedure

Table 4: List of all IDL modules: the table shows the file name, the type (**S**ubroutine or **F**unction, and its description.

6.4 F90 program flow

The F90 call makes use of a series of modules, each pertaining to a specific area of the run time of Linfor3D.

- Initialize MPI (linfor_parallel.f90)
- Initialize the timing features (linfor_timing.f90)
- Initialize UIO (uio_base_module.f90)
- Read input parameters (linfor_input.f90)
- Initialize atomic data (linfor_input.f90)
- Read line data (linfor_input.f90)
- Initialize ionopa2, atmosphere parameters, opacity tables and EOS tables (linfor_io.F90)
- Set constants (linfor_input.f90)
- Set up the angle quadratures (linfor_raybase.f90)
- Read 1DX model data (linfor_rd1dx.f90)
- Define and populate the 1DX flowfield (fx) type (linfor_rd1dx.f90)
- Read 1DX departure coefficients (linfor_nlte.f90)
- Read 3D model data (linfor_rd3d.f90)
- Recompute model on refined z-grid (linfor_flowfield.f90)
- Define and populate 3D flowfield (f3) type (linfor_flowfield.f90)
- Compute ⟨3D⟩ models based on the 3D model data (linfor_flowfield.f90)
- Define and populate ⟨3D⟩ flowfield (f1) type (linfor_flowfield.f90)
- Read 3D departure coefficients (linfor_nlte.f90)

- Define and populate global ⟨3D⟩ flowfield (fa) type (linfor_flowfield.f90)
- Compute ionopa quantities (pe, kappa, zeta) and monochromatic τ for 1DX input model (linfor_ionopa.f90)
- Compute ionopa quantities (pe, kappa, zeta) and monochromatic τ for ⟨3D⟩ models (linfor_ionopa.f90)
- Compute ionopa quantities (pe, kappa, zeta) and monochromatic τ for 3D models (linfor_ionopa.f90)
- Do radiative transfer calculations for 3D models (linfor_dort.f90)
- Do radiative transfer calculations for ⟨3D⟩ atmospheres (linfor_dort.f90)
- Do radiative transfer calculations for 1DX atmosphere (linfor_dort.f90)
- Store results for in output data types (linfor_eval.f90)
- Generate output files (linfor_wrdata.f90).

6.5 F90 Files

Table 5 shows a list of all source files necessary to run Linfor3D.
All **F90** versions of Linfor3D require the UIO library to handle the I/O of the CO^{5}BOLD files, and versions 6.0.0 require them for ALL I/O done during the program flow.

File name	Type	Description
base:		
linfor_main.f90	P	main program.
var_input.f90	M	contains input data types.
var_charlen.f90	M	sets character string lengths throughout Linfor3D.
debug:		
linfor_debug.f90	P	developer module used to output UIO-formatted data for use with the IDL environment.
eos:		
gasinter_routines.F90	M	responsible for reading in the interpolating the equation-of-state to extract hydrodynamical properties such as temperature and pressure. Taken verbatim from CO^{5}BOLD.
gfx:		
linfor_gfx.f90	M	Deals with most print-to-screen statements.
io:		
linfor_flowfield.f90	M	deals with the flowfield allocation and array population for most atmosphere types. In particular for CO^{5}BOLD models.
linfor_input.F90	M	deals with almost all input data (line, cmd, atom, abu-file, etc.).
linfor_io.F90	M	responsible for most of the generic IO done by Linfor3D.
linfor_rd1dx.f90	M	reads in all 1D external model atmospheres.
linfor_rd3d.f90	M	control module that calls the appropriate module to read the particular 3D model.
linfor_rdcobold.F90	M	reads in CO^{5}BOLD model atmospheres.
linfor_rdstagger.f90	M	reads in STAGGER model atmospheres.
linfor_wrdata.f90	M	writes all output to UIO-formatted files.
var_const.f90	M	contains all constant information and the const data type.
var_output.f90	M	contains the output data types.
var_snaps.F90	M	contains data type necessary to read in CO^{5}BOLD 3D full files.
math:		
linfor_box.f90	M	contains several routines to manipulate various properties of the flowfield.
linfor_functions.f90	M	contains several functions and subroutines to deal with general mathematical procedures and array manipulation.
linfor_raybase.f90	M	sets up the angle quadratures.
mpi:		
linfor_parallel.f90	M	sets up and defines the parallelization schemes used by Linfor3D.
var_mpi.f90	M	contains data pertaining to the global parameters and information that is passed on to linfor_parallel.f90.
nlte		
linfor_nlte.f90	M	controls the departure coefficient modules.
linfor_rdxbc.f90	M	reads the xbc and xbc2 formatted departure coefficient modules.

opta		
`linfor_ionopa.f90`	M	controls every call to `ionopa2.f90`.
`var_xkaroslin.f90`	M	contains older versions of the xkaros series of subroutines that use a linear interpolation scheme.
`opta_par_module.f90`	M	contains arrays pertaining to `opta_routines.F90`. Taken verbatim from CO^{5}BOLD.
`opta_routines.F90`	M	control modules for handling and manipulating data from CO^{5}BOLD opacity tables. Taken verbatim from CO^{5}BOLD.
`var_ionopa.f90`	M	contains data types and local and global arrays used by `linfor_ionopa.f90`.
rad		
`linfor_dort.F90`	M	control module that sets up all calls to `dort.F90`.
`linfor_eval.f90`	M	evaluates the output data from `dort.F90` and allocates and populations output data types.
`var_dort.f90`	M	Contains a data type used by `linfor_dort.f90`.
time		
`linfor_timing.f90`	M	deals with all timing aspects of Linfor3D.
uio		
`uio_base_module.f90`	M	These modules are taken verbatim from CO^{5}BOLD and are responcible for
`uio_filedef_module.f90`	M	almost all IO throughout a typical Linfor3D execution.
`uio_bulk_module.f90`	M	
`uio_mac_module.F90`	M	

Table 5: List of all F90 modules: the table shows the file name, the type (**P**rogram or **M**odule, and its description.

7 Parameter Input: linfor_setcmd.pro (IDL) linfor3d.setcmd (F90)

The input parameters (except for those defined in `line.dat`, see Sect. 8) are basically specified by editing the routine `linfor_setcmd.pro` in IDL or `linfor3d.sectcmd` when using the F90 version of Linfor3D. In this way, the user defines the structure `cmd` (see Table 3). The order of entries is irrelevant. Parameters which are not required may be omitted and set by default in both codes.
A detailed explanation of the various input parameters and their possible values is given in the following sections. An example of the IDL setcmd file follows in Sect. 7.13 and an example of the F90 setcmd file follows in Sect. 7.14.

7.1 F90 specific flags and settings

While the information below can be pertinant to both versions of the code, and is defined in an identical fashion, the following **optional** input is only considered in the F90 execution when set within `linfor3d.setcmd`:

7.1.1 outfile

function	:	file allocation
required	:	optional
type	:	character
values	:	'`output.uiosave`' (default '`Linfor3D.uiosave`')

One sets this parameter to save the entire output to file. By default it is saved as `Linfor3D.uiosave`. This contains all input and output structures that can be loaded into IDL, Python and Fortran.

7.1.2 printcobold

function	:	execution flag
required	:	optional
type	:	integer
values	:	0 (default), 1

When set to 1 Linfor3D will print extra information concerning the EOS, opta tables and full files to screen. By default this is set to 0, as this information is usually superfluous to a typical Linfor3D run.

7.1.3 debug

function	:	execution flag
required	:	optional
type	:	integer
values	:	0 (default), 1

This is only used to determine how the parallelization scheme executes on the allocated number of CPUs. When `debug=1` no calls to `ionopa.f90` or `dort.f90` are made. Rather, the parallelization scheme outputs further information on how `mpirun -np <NCPU>` has been allocated by the parallelization scheme, given the number of CPUs, `<NCPU>`. This is mostly used for development purposes.

Therefore, it can be ignored in most normal scenarios.

7.1.4 debug_save

function	:	execution flag
required	:	optional
type	:	integer
values	:	0 (default), 1

If Linfor3D has been compiled using the debug flags, then one can save raw DoRT outputs to file. This is used by the developers to confirm the parallelisation scheme is correctly interpreted by DoRT when local outputs are gathered to the master CPU. This can also be ignored in standard scenarios.

7.1.5 wr3x3

function	:	execution flag
required	:	optional
type	:	integer
values	:	0, 1 (default)

This switches off writing the 3x3 model atmospheres.

7.1.6 d1_flag

function	:	execution flag
required	:	optional
type	:	integer
values	:	0, 1 (default)

Tells Linfor3D whether it should compute the ⟨3D⟩ models for spectrum synthesis.

7.2 IDL specific flags and settings

There are certain features available in the IDL version that is not available or deprecated in the F90 version, such as the plotting ability.

7.2.1 plt_flag

function	:	plotting of bisectors
required	:	always
type	:	integer
values	:	-1, 0, 1

The parameter `plt_flag` controls if line bisctors should be plotted or not (0: no, 1: yes). If `plt_flag` is set to -1, all plotting is suppressed.

7.2.2 free_flag

function	:	free pointers in structures at end of program
required	:	always
type	:	integer
values	:	0, 1

If `free_flag = 1`, then each run of Linfor3D allocates fresh memory for the structures `ff`, `f1`, `fx`, `ss`, `s1`, and `sx`. In this case the corresponding pointers are removed at the end. If you want to examine the

structures after the end of execution, you must have `free_flag = 0`. If you want to run the program several times in a row with different input parameters, you should set `free_flag = 0` in order to avoid additional memory allocation for each run.

7.2.3 ff_path

function	:	directory to be used for reading and writing cached flow fields
required	:	always
type	:	string
values	:	e.g. '`/data/mst/ffcache/`'

7.3 Program execution flags (IDL/F90)

The user can control the program execution by setting the flags `run_flag`, `nlte_flag`, `cv1_flag`, `cv2_flag`, `cv3_flag`, `maps_flag`, `cc3d_flag`, `rdbb_flag`, which are explained in more detail below.

7.3.1 run_flag

function	:	program mode
required	:	always
type	:	integer
values	:	−3, −2, −1, 0, 1, 2, 3 (usually 3)

This parameter determines the general function of Linfor3D:

IMPORTANT: The F90 version of the code currently only runs using flags = -3, -2, and 3. Flags = -1, 1 or 2 will most likely never be included inside the F90 version.
Setting `run_flag = -3` allows you to compute the external 1D atmosphere only. While a snapshot is still required to run Linfor3D correctly, no 3D or ⟨3D⟩ data is computed or written to file. The results are stored in the structure '`linfor_1X.uiosave`'. N.B. mode is only available from version 6.1.0 onwards.
Setting `run_flag = -2` allows you to compute 3x3 file for the external reference model only.
Setting `run_flag = -1` allows you to restore old results, and replace the results of the previous 1D external atmosphere with those of a different 1D external atmosphere.
Setting `run_flag = 0` (similar to run_flag = -1) allows you to quickly compare the 3D spectra with another external 1D reference atmosphere. Finally, the results are saved in files '`linfor_3D_1.uiosave`' and '`linfor_3D_2.uiosave`'. Rarely used setting.
Setting `run_flag = 1` is used for plotting the structure of the input model on the original grid. No radiative transfer calculations are done.
Setting `run_flag = 2` is used for plotting the structure of the input model on the reduced (refined) grid. No radiative transfer calculations are done.
Setting `run_flag = 3` is the usual case. After construction of the 3D atmosphere on the reduced (refined) grid and of the 1D mean atmosphere, the line formation calculations are done, and the results are plotted ('`linfor_plot1`': spatially and temporally resolved line profiles and bisectors, '`linfor_plot2`': surface and time averaged line profiles and bisectors). Finally, the results are saved in files '`linfor_3D_1.uiosave`' and '`linfor_3D_2.uiosave`'.

run_flag value	version	control of program flow
-3	F90, IDL	load 3D models, (compute 1D ref. spectrum), save results
-2	F90, IDL	compute 1D 3x3 external atmosphere
-1	IDL	restore results, (compute 1D ref. atmosphere & spectrum), save results
0	IDL	restore results, (compute 1D ref. atmosphere & spectrum), plot2, save results
1	IDL	compute 3D, 1D atmospheres (1), plot01, stop
2	IDL	compute 3D, 1D atmospheres (1,2), plot02, stop
3	F90, IDL	compute 3D, 1D atmospheres (1,2), line formation, plot1, plot2, save results

7.3.2 cv1_flag

function : enforce $\langle \rho\, u_x \rangle = 0$
required : always
type : integer
values : 0, 1

The parameter cv1_flag controls whether or not the x-component of the velocity field is adjusted to ensure zero mass flux in x-direction. (0: no, 1: yes). Default 0

7.3.3 cv2_flag

function : enforce $\langle \rho\, u_y \rangle = 0$
required : always
type : integer
values : 0, 1

The parameter cv2_flag controls whether or not the y-component of the velocity field is adjusted to ensure zero mass flux in y-direction. (0: no, 1: yes). Default 0

7.3.4 cv3_flag

function : enforce $\langle \rho\, u_z \rangle = 0$
required : always
type : integer
values : 0, 1

The parameter cv3_flag controls whether or not the z-component of the velocity field is adjusted to ensure zero mass flux in z-direction. (0: no, 1: yes). Default 0

7.3.5 maps_flag

function : controls output of intensity maps
required : always
type : integer
values : 0, 1, 2

The parameter maps_flag controls the output of intensity maps which are provided in the IDL structure MAPS:

value		meaning
0	:	Continuum images only. Create map `ICLAM0`.
1	:	Continuum images (`ICLAM0`) plus images at the centre of the wavelength window (`ICLAM1`), all at wavelength λ =`clam`;
2	:	Continuum images (`ICLAM0`) plus images (`ICLAM2`) at all wavelengths within the wavelength window of width 2 · `dlam` around the central wavelength `clam`: λ_i = `clam` − `dlam` + i · `ddlam` (see Sect. 8);

7.3.6 cc3d_flag

function	:	output of 3D contribution function
required	:	always
type	:	integer
values	:	0, 1

The parameter `cc3d_flag` controls whether the 3D continuum intensity contribution function should be saved in structure contf3d or not (0: no, 1: yes).

7.3.7 nlte_flag

function	:	output of 3D contribution function
required	:	always
type	:	integer
values	:	0, 1, 2, 3

The parameter `nlte_flag` controls whether the line transfer is performed in LTE (`nlte_flag`=0) or in NLTE (`nlte_flag`=1, 2, 3). The NLTE options work only for lines with available departure coefficients, which are read from a separate data file (see below).

value		meaning
0	:	Continuum and lines in LTE.
1	:	Continuum in LTE, line source function in LTE, line opacity in NLTE
2	:	Continuum in LTE, line opacity in LTE, line source function in NLTE,
3	:	Continuum in LTE, line opacity and source function in NLTE

7.4 General paths

7.4.1 abupath

function	:	directory where '`.abu`' files and '`atom.dat`' are located
required	:	always
type	:	string
values	:	e.g. '`/home/mst/ABU/`'

If `abupath` is not specified in the command file, the path is taken from environment variable '`$LINFOR3D_ABU`'.

7.4.2 opapath

function	:	directory with opacity tables (.opta files)
required	:	always
type	:	string
values	:	e.g. '/home/mst/RHD/opa/dat/'

We recommend setting the environment variable $OPTABLES

7.4.3 gaspath

function	:	directory with GAS tables (gas_*.eos files)
required	:	always
type	:	string
values	:	e.g. '/home/mst/RHD/eos/dat/'

7.4.4 eospath

function	:	directory with EOS tables (eos_*.eos files)
required	:	always
type	:	string
values	:	e.g. '/home/mst/RHD/eos/dat/'

We recommend setting the environment variable $EOSTABLES for these two paths.

7.5 Model data

7.5.1 context

function	:	source of input model
required	:	always
type	:	string
values	:	e.g. 'cobold'

value		meaning
'cobold'	:	3D CO5BOLD
'copenhagen'	:	N&S 3D code (IDL only)
'kiel'	:	Kiel 2D HDW-Code (IDL only)
'muram'	:	MURAM 3D MHD Code (IDL only)
'grey'	:	construct grey 3D ($n_x = n_y = 10$)
	:	hydrostatic atmosphere for test purposes
	:	(IDL only)
'stagger'	:	3D STAGGER models

The τ_{Ross} grid of the grey atmosphere is defined by the parameters cmd.lutau1, cmd.lutau2, cmd.dlutau. The atmospheric parameters must be specified as cmd.Teff and cmd.grav. The opacity table must be specified as cmd.opafile, and the Equation-of-State as cmd.eosfile and cmd.gasfile.

For STAGGER models, the input Equation-of-State in cmd.eosfile is defined by the original STAGGER Equation-of-State, while the Equation-of-State in cmd.gasfile should be an equivalent CO5BOLD gas file [2] . The latter is only used to redefine the surface boundary when topbox is executed. Additionally, an

[2]Example: If a solar STAGGER model is input (e.g. t5777g44m0005_00000.dat and EOS EOSrhoe_AGSS09+0.00.tab), then

equivalent CO5BOLD opacity file should be included for the same reason.

7.5.2 rhdpath

function	:	directory with 2D/3D model atmospheres (.end, .full files)
required	:	always
type	:	string
values	:	e.g. '`/data/mst/model/`'

7.5.3 modelid

function	:	name of 2D/3D model file
required	:	always
type	:	string
values	:	e.g. '`gt57g44n66_3Dgz.end`'

Note: A list of files can be specified by using wildcards, e.g. '`chro3D04*.full`'.

7.5.4 parfs

function	:	full path to parameter file (rhd.par)
required	:	CO5BOLD only
type	:	string
values	:	e.g. '`/data/mst/model/par/gt57g44n66.par`'

7.5.5 xbcpath3

function	:	full path to 3D xbc or xbc2 files (*.xbc/*.xbc2)
required	:	optional
type	:	string
values	:	e.g. '`/data/mst/NLTE3D_data/model/`'

7.5.6 xbcpathx

function	:	full path to 1DX(external) xbc or xbc2 files (*.xbc/*.xbc2)
required	:	optional
type	:	string
values	:	e.g. '`/data/mst/NLTE3D_data/model/`'

7.5.7 xbcpath

function	:	full path to 3D and 1DX xbc or xbc2 files (*.xbc/*.xbc2)
required	:	optional
type	:	string
values	:	e.g. '`/data/mst/NLTE3D_data/model/`'

Note: departure files are necessary for NLTE line formation calculations. `xbcpath` is superseeded by `xbcpath3` and `xbcpathx`.

a solar gas file should be used (e.g. `gas_cifist2006_m00_a00_l5.eos`).

7.5.8 abuid

function	:	Model abundance mixture to be used in the `ionopa` (or `ionopa2`) routine
required	:	always
type	:	string
values	:	'kiel', 'cifist2006', 'special'

`abuid` identifies the solar abundance mix which is then modified according to `dmetal` and `dalpha` (see below). The corresponding tables, `kiel.abu`, `cifist2006.abu`, or `special.abu` must be located in directory `abupath`. Version 6.2.2 onwards use `ionopa2`, which requires two abundance mixture files: The model abundance mixture, `abuid` (above); and the spectrum abundance mixture, `abuidx` (below).

7.5.9 abuidx

function	:	Spectrum abundance mixture to be used in `ionopa2` routine
required	:	always (version 6.2.2 onwards)
type	:	string
values	:	'kiel', 'cifist2006', 'special'

Linfor3D has an additional way it computes `ionopa` quantities (pe, kappa, zeta) and the monochromatic tau scale. When `abuid=abuidx` (or if `abuidx` is **not** defined), these quantities are computed as they were in previous versions of Linfor3D. When `abuid` contains the solar abundance mixture of the CO^{5}BOLD model and `abuidx` contains the desired abundance mixture of the spectrum synthesis then `ionopa2` computes the quantities twice to compensate for the change in abundance.

7.5.10 dmetal

function	:	metallicity [M/H] ($\log_{10}$) to be used in `ionopa`-routines
required	:	always
type	:	float
values	:	e.g. 0.0, −0.5, −2.0

The logarithmic abundance of all elements beyond Li ($N > 3$) is changed by `dmetal`.

7.5.11 dalpha

function	:	alpha enhancement to be used in `ionopa`-routines
required	:	always
type	:	float
values	:	e.g. 0.0, +0.4

The logarithmic enhancement factor to be applied to all α-elements.
Linfor3D considers O, Ne, Mg, Si, S, Ar, Ca, and Ti as α-elements.

7.5.12 nx_skip

function	:	sampling of model in x-direction
required	:	if `context`='`cobold`','`kiel`','`muram`'
type	:	integer
values	:	1, 4, 10; −1

If both `nx_skip` and `ny_skip` (see Sect. 7.5) are negative, the original data are re-binned from (nx,ny) to `(nx/abs(nx_skip),ny/abs(ny_skip))`. In the usual case that both `nx_skip` and `ny_skip` are positive,

the original data are re-sampled, skipping by `nx_skip` in x, and by `ny_skip` in y-direction (`nx/nx_skip`, and `ny/ny_skip` should preferably be an integer). If `nx_skip` and `ny_skip` have different signs, an error message is printed and the program is stopped. The value 1 has no effect.

7.5.13 ny_skip

function	:	sampling of model in x-direction
required	:	if `context='cobold','kiel','muram'`
type	:	integer
values	:	1, 4, 10; −1

For details see description of `nx_skip` (Sect. 7.5).

7.6 More model information (MOST read from parameter file for CO5BOLD data)

The ***majority*** of parameters in this section are ignored in the case of CO5BOLD data and instead read from the specified CO5BOLD parameter file. Please read this section carefully to avoid errors in your synthesis.

7.6.1 opafile

function	:	name of opacity file (binned opacity tables)
required	:	not needed if `context='cobold'`
type	:	string
values	:	e.g. '`g2v.opta`'

7.6.2 gasfile

function	:	name of GAS file ($P, T \rightarrow \rho, e, \ldots$)
required	:	not needed if `context='cobold'`
type	:	string
values	:	e.g. '`gas_mm00_l.eos`'

7.6.3 eosfile

function	:	name of Equation-of-State file ($\rho, e \rightarrow P, T, \ldots$)
required	:	not needed if `context='cobold'`
type	:	string
values	:	e.g. '`eos_mm00_l.eos`'

7.6.4 htau0

function	:	opacity scale height [cm] at top of 3D model
required	:	always
type	:	float
values	:	e.g. $60.0E5$; **default = 0.0**

A **default value of 0.0** tells Linfor3D to take this parameter from the parameter file (set at sect. 7.5 –

parfs – e.g. rhd.par).

7.6.5 qmol

function	:	mean molecular weight of neutral gas
required	:	not needed if context='cobold'
type	:	float
values	:	e.g. 1.301855

Important Note: This parameter has been deprecated in Linfor3D version 6.2.6 onwards.

7.6.6 Teff

function	:	effective temperature of 3D model
required	:	not needed if context='cobold'
type	:	float
values	:	e.g. 5770.0

7.6.7 grav

function	:	surface gravity [cm/s^2] of 3D model
required	:	not needed if context='cobold'
type	:	float
values	:	e.g. 27500.0

7.6.8 tsurffac

function	:	surface temperature ($\tau = 0$) of 3D model is tsurffac$\cdot T_{\rm eff}$
required	:	always
type	:	float
values	:	e.g. 0.727903; **default = 0.0**

A **default value of 0.0** tells Linfor3D to take this parameter from the parameter file (set at sect. 7.5 – parfs – e.g. rhd.par). This only affects ***Linfor3D version 6.2.6*** onwards. Versions of Linfor3D older than this do not read this parameter from setcmd if context='cobold'.

7.7 Model data - reading of 'full' files (CO5BOLD only)

The parameters in this section are only needed for reading snapshot from CO5BOLD data files.
These three parameters are only needed when there is a possibility that there is more than one dataset stored in a .full file. By default these are set such that Linfor3D assumes a single dataset is stored in each model file. The .end data type is always assumed to contain one dataset per file.

7.7.1 isnap_full_1

function	:	first snapshot to be read from full file(s)
required	:	only needed if context='cobold'
type	:	integer
values	:	$1 - n_{\rm dataset}$; **default = 1**

7.7.2 isnap_full_2

function	:	last snapshot to be read from full file(s)
required	:	only needed if `context='cobold'`
type	:	integer
values	:	`isnap_full_1`$-n_{\text{dataset}}$; **default = 1**

7.7.3 istep_full

function	:	step for reading snapshots from full file(s)
required	:	only needed if `context='cobold'`
type	:	integer
values	:	> 0; **default = 1**

7.8 ⟨3D⟩ mean model

7.8.1 mavg

function	:	mode of averaging 3D T-structure on $\tau_{\rm Ross}$
required	:	always
type	:	integer
values	:	1, 4

value		meaning
1	:	$T_{\langle 3D\rangle}(\tau_{\rm Ross}) = \langle T_{\rm 3D}(\tau_{\rm Ross})\rangle$
4	:	$T_{\langle 3D\rangle}(\tau_{\rm Ross}) = \langle T^4_{\rm 3D}(\tau_{\rm Ross})\rangle^{1/4}$

7.9 External 1D reference model

7.9.1 atmpath

function	:	directory with 1D model atmospheres
required	:	always
type	:	string
values	:	e.g. '`/home/mst/atm/`'

7.9.2 atmfile

function	:	name of 1D reference model
required	:	always
type	:	string
values	:	e.g. '`NONE`', '`dxgt57g44n59.l50`', '`falc.at9`', '`falc.mod`', '`<3D>`'

Note: No external 1D reference atmosphere will be used if `atmfile='NONE'`. In this case the parameter `atmpath` has no meaning. If `atmfile='<3D>'` the external model atmosphere is replaced by a global ⟨3D⟩ model atmosphere constructed by averaging the individual ⟨3D⟩ snapshots.
Linfor3D has the capability to read in several types of external 1D model atmospheres. The way it determines the type of model atmosphere is with the file extension. For example, '`.l50`' is determined as an `LHD` model atmosphere. This is determined at the `linfor_rdxatm.pro` routine level. The result of that will invoke one of several routines to properly read the model atmosphere. Here is a list of 1D model atmospheres accepted by Linfor3D, the routine name that reads the model, and what the external model atmosphere file extension should be:

Model atmosphere		Invoked routine		File extension
`LHD`	:	`rdl50.pro`	:	'`.l50`'
Kiel ATMOS	:	`linfor_rdatmos.pro`	:	'`.atm`'
ATLAS9	:	`linfor_rdatlas9.pro`	:	'`.at9`' or '`a12`'
MARCS	:	`linfor_rdmarcs.pro`	:	'`.mod`'
FAL*	:	`linfor_rdfalmod.pro`	:	'`.fal`'

*Fontenla, Avrett, Loeser models (1993, ApJ 406, 319)

Finally, should you wish to compute a ⟨3D⟩ model again, for different parameters, such as microturbulence, the routine `d3a21dx.pro` will convert a standard ⟨3D⟩ model (which is saved as an `idlsave`) to a properly formatted ATLAS9 model accepted by Linfor3D. However, this model is only compatable with Linfor3D and the routine is only available with Linfor3D version 6.2.6 onwards.

7.10 Line data and radiative transfer

7.10.1 linfs

function	:	name of line data file
required	:	always
type	:	string
values	:	e.g. '`Li67.line`'

Note: If `linfs` is not specified, the default value '`line.dat`' is assumed.

7.10.2 lutau1

function	:	smallest $\log \tau_{\rm Ross}$ covered by sub-model (refined z-grid)
required	:	always
type	:	float
values	:	e.g. $-7.0D0$

7.10.3 lutau2

function	:	largest $\log \tau_{\rm Ross}$ covered by sub-model (refined z-grid
required	:	always
type	:	float
values	:	e.g. $2.0D0$

7.10.4 dlutau

function	:	z-spacing of sub-model corresponds roughly to $\Delta \log \tau_{\rm Ross}$ =`dlutau`
required	:	always
type	:	float
values	:	e.g. $8.0D-2$

7.10.5 lctau1

function	:	smallest $\log \tau_{\rm cont}$ used for RT integration
required	:	always
type	:	float
values	:	e.g. $-7.0D0$, $\geq$ `lutau1`

7.10.6 lctau2

function	:	largest $\log \tau_{\rm cont}$ used for RT integration
required	:	always
type	:	float
values	:	e.g. $2.0D0$, $\leq$ `lutau2`

7.10.7 dlctau

function	:	resolution in $\log \tau_{\rm cont}$ used for RT integration
required	:	always
type	:	float
values	:	e.g. $8.0D-2$

7.10.8 Hbrd

function : controls broadening of hydrogen lines
required : always
type : integer
values : 0, 1, 2, 3, 4

value		meaning
0	:	Cayrel & Traving (1960), default
1	:	Resonance broadening: AG , Stark broadening: G
2	:	Resonance broadening: BPO, Stark broadening: G
3	:	Resonance broadening: A08 , Stark broadening: G
4	:	Resonance broadening: A08 , Stark broadening: SH

AG : Ali & Griem (1966, Phys. Rev. 144, 366),
BPO: Barklem, Piskunov and O'Mara (2000, A&A 363, 1091),
A08 : Allard et al. (2008, A&A 480, 581),
G : Griem (1960, ApJ 132, 883), with corrections to approximate the Vidal, Cooper & Smith (1973, ApJS 25, 37) profiles.
SH : Stehlé & Hutcheon (1999 A&AS, 140, 93)

Note 1: option `Hbrd = 2` has an effect **only on Hα, Hβ, and Hγ**, and `Hbrd = 3` affects **only Hα**; all other hydrogen lines are treated according to option `Hbrd = 1`, unless `Hbrd = 0`.
Note 2: option `Hbrd = 4` is not currently working (as of version 6.2.4 – check the readme file in the later versions of Linfor3D for updates on this).

7.10.9 vsini

function : Sets $v \sin i$ value for all spectra in `linfor_plot2.pro` only
required : always
type : float
values : e.g. 1.0

7.10.10 ximicx

function : isotropic Gaussian microturbulence velocity $[km/s]$ for external 1D reference model (added quadratically to thermal velocity)
required : always
type : float
values : e.g. 1.0

7.10.11 ximic1

function : isotropic Gaussian microturbulence velocity $[km/s]$ for $\langle$3D$\rangle$ mean models (added quadratically to thermal velocity)
required : always
type : float
values : e.g. 1.0

7.10.12 ximic3

function	:	isotropic Gaussian microturbulence velocity [km/s] for 2D/3D models (added quadratically to thermal flow velocity)
required	:	always
type	:	float
values	:	e.g. 1.0

7.10.13 ximacx

function	:	Isotropic Gaussian macroturbulence velocity [km/s] for external 1D reference model (additional line broadening after line formation)
required	:	always
type	:	float
values	:	e.g. 1.6

7.10.14 ximac1

function	:	Isoptropic Gaussian macroturbulence velocity [km/s] for $\langle$3D$\rangle$ mean models (additional line broadening after line formation)
required	:	always
type	:	float
values	:	e.g. 1.6

7.10.15 ximac3

function	:	Isotropic Gaussian macroturbulence velocity [km/s] for 2D/3D models (additional line broadening after line formation)
required	:	always
type	:	float
values	:	e.g. 1.6

7.10.16 vfacx

function	:	the x-component of the hydrodynamical velocity field of the 2D/3D models is multiplied by this factor
required	:	always
type	:	float
values	:	e.g. 0.0, 1.0

7.10.17 vfacy

function	:	the y-component of the hydrodynamical velocity field of the 2D/3D models is multiplied by this factor
required	:	always
type	:	float
values	:	e.g. 0.0, 1.0

7.10.18 vfacz

function	:	the z-component of the hydrodynamical velocity field of the 2D/3D models is multiplied by this factor
required	:	always
type	:	float
values	:	e.g. 0.0, 1.0

7.10.19 micro

function	:	controls microturbulence in 1D Curve-of-Growth
required	:	always
type	:	integer
values	:	0, 1

Determines whether or not different microturbulence values should be used when computing the 1D Curve-of-Growth. 0: only one value, given by `ximicx` and `ximic1`, respectively; 1: sequence of microturbulence values defined by parameters `xi_a`, `xi_b`, `xi_d` (see below).

7.10.20 xi_a

function	:	determines start value for microturbulence sequence
required	:	always
type	:	float
values	:	e.g. 0.0, default: 0.5

7.10.21 xi_b

function	:	determines end value for microturbulence sequence
required	:	always
type	:	float
values	:	e.g. 2.0, default: 1.5

7.10.22 xi_d

function	:	determines intervals of microturbulence sequence
required	:	always
type	:	float
values	:	e.g. 0.1, default: 0.125

The microturbulence sequence is computed as xi(i) = xi0 * (`xi_a` + i * `xi_d`), i=0 .. im, where xi0 is `ximicx` and `ximic1`, respectively, and im= (`xi_b` - `xi_a`)/ `xi_d`.

7.10.23 dclam

function	:	determines the variation of the continuum
required	:	always
type	:	float
values	:	e.g. 20.0, default: 0

If `dclam`=0, the continuum is treated as constant (default). Otherwise, the continuum is computed at 3 wavelength points, `clam`-`dclam`, `clam`, `clam`+`dclam`, where `clam` is the central wavelength (in Å) of the computed spectral range (see Sect. 8), and `dclam` is half the width of the specified spectral range (in Å).

The continuum is computed by parabolic interpolation inside the spectral window.

If the spectral range of the specified synthetic spectrum (which is defined by the parameters of the line file (see Sect. 8) exceeds a few Å, `dclam` should be set to match half the total spectral range.

When `dclam` is set in the Fortran version of the code, the parabolic interpolation is done over descreate wavelength ranges set according to wavelength parallelisation; λ-`dclam`, λ, λ+`dclam`, where λ is the central wavelength (in Å) of the smaller wavelength range set by the parallelisation scheme.

7.10.24 intmode

function	:	mode of integration in routines `ms_int_tau` and `ms_int_exp`
required	:	always
type	:	integer
values	:	0, 1

Determines the mode of integration in routines `ms_int_tau` and `ms_int_exp`, which can be linear (0) or monotonic and cubic (1, standard).

7.10.25 intline

function	:	mode of integrating the line transfer equation
required	:	always
type	:	integer
values	:	1, 2, −1, −2

Determines the method of integrating the line transfer equation (see Section 5 for details). Default value is `intline`=1.

value		meaning
1	:	Line depression on fixed log τ scale (Eq. 41)
2	:	Line depression on monochromatic τ scale (Eq. 36)
-1	:	Line intensity on fixed log τ scale (Eq. 25)
-2	:	Line intensity on monochromatic τ scale (Eq. 26)

7.11 Angle quadrature schemes

By default, Linfor3D requires the following information to compute the transfer equation over several ray angles.

7.11.1 ntheta

function	:	number of *inclined* θ-angles for which spectrum is computed
required	:	always
type	:	integer
values	:	0, 1, 2, 3, (−3), 4, 6, 8

0: Intensity spectrum ('vertical' ray only), > 0: Intensity and flux spectrum;

7.11.2 nphi

function	:	number of ϕ-angles for integration of flux spectrum
required	:	always
type	:	integer
values	:	2, or 4 (typically 4)

2: $\phi = 0, \pi$; 4: $\phi = 0, \pi/2, \pi, 3\pi/2$;

7.11.3 mu0

function	:	view angle $\mu = \cos\theta$
required	:	when `ntheta=0`
type	:	float
values	:	$0.0 < \cos\theta \leq 1.0$

If the parameter `ntheta`=0, then the spectrum and intensity maps are computed for inclination angle `mu0` ($= \cos\theta_0$).
`mu0`= 1.0 corresponds to vertical rays, i.e. disk center view.
`mu0`= 0.0 corresponds to the very limb, but a value of mu0 < 0.1 are not recomended.

7.11.4 kphi

function	:	view angle
required	:	when `ntheta=0`
type	:	integer
values	:	0, 1, 2, 3

The parameter `kphi` determines the direction from which the model is viewed (if `ntheta=0`):

value		meaning
0	:	rays emerge parallel to the x-axis, i.e. the model is viewed somewhere on the 'equator' between the left limb and disk center.
1	:	rays emerge parallel to the y-axis, i.e. the model is viewed somewhere on the 'meridian' between the lower limb and disk center.
2	:	rays emerge anti-parallel to the x-axis, i.e. the model is viewed somewhere on the 'equator' between the right limb and disk center.
3	:	rays emerge anti-parallel to the y-axis, i.e. the model is viewed somewhere on the 'meridian' between the upper limb and disk center.

Since the release of Linfor3D version 6.3.0, several new quadrature schemes have been introduced for the user to select:

Name		meaning
Lobatto	:	quadrature through routine `linfor_setwts_lobatto.pro`
Double Gauss	:	quadrature through routine `linfor_setwts_dblgaus.pro`
Double Gauss-Radau	:	quadrature through routine `linfor_setwts_dblrdau.pro`
Custom-made	:	quadrature through routine `linfor_setwts_special.pro`

These new quadrature schemes can be used by selecting them in the `setcmd`:

7.11.5 raybase

function	:	quadrature scheme
required	:	always
type	:	string
values	:	'lobatto', 'dblgaus', 'dblrdau', and 'special'

If the `raybase` option is missing from the `setcmd` then the default `lobatto` is selected. For the user, little has changed. One must still select the number of μ and ϕ angles to use, like before. The only exception to this is when the user elects to use a custom angle quadrature scheme (`special`).

To use the `special` case, a file called 'special.xmu' that contains the list of mu-angles and corresponding weights must be made available in the working directory. The following was taken from Allende Prieto et al. (2004, A&A, 423, 1109) and serve as an example of the expected format for these files:

```
---  mu values observed by Allende Prieto (2004)
 7   ::  nrays
 1.00D0          0.97D0          0.87D0          0.71D0          0.50D0
 0.26D0          0.17D0                                                  :: xmu
 0.014800391D0   0.061464714D0   0.11869823D0    0.16704234D0    0.24705583D0
 0.152487360D0   0.238451150D0                                           :: wts
```

The strict formatting is given on four lines. The first line is a header, which is ignored by Linfor3D. The second line must contain the number of rays. The third must contain the corresponding number of μ-angles, and the final line the weights associated to each μ-angle. Further examples of this special file file can be found in the `Data` subdirectory of the Linfor3D directory tree.

7.12 Curve-of-Growth computations

As standard, Linfor3D computes a Curve-of-Growth (CoG) for the 1D external and ⟨3D⟩ model atmospheres. The range in abundance, and the sampling of the range were fixed within Linfor3D. Version 6.2.5 onwards now includes the option to tailor the Curve-of-Growth, or deactivate the computations.

7.12.1 cog

function	:	Tailors the Curve-of-Growth computations
required	:	always
type	:	integer
values	:	$-2, -1, 0, 1, 2$

value		meaning
-2	:	Computes Curve–of–Growth computations for user-defined $\log gf$ range, as set by `icg`, `gflgmin` and `gflgmax`. The entire set of output line profiles are saved in `linfor_3D_4.uiosave`.
-1	:	Computes Curve–of–Growth computations for user-defined $\log gf$ range, as set by `icg`, `gflgmin` and `gflgmax`.
0	:	Curve–of–Growth computations are deactivated. Highly recommended if the line list contains blends.
1	:	Computes a standard Curve–of–Growth computations for a set range of $\log gf$ values.
2	:	Computes a standard Curve–of–Growth computations for a set range of $\log gf$ values. The entire set of output line profiles are saved in `linfor_3D_4.uiosave`.

7.12.2 icg

function	:	sets index from default to user defined
required	:	yes, if `cog`=−1, −2
type	:	integer
values	:	e.g. 51, 101

7.12.3 gflgmin

function	:	sets minimum $\Delta \log gf$ value over which to perform CoG computations
required	:	yes, if `cog`=−1, −2
type	:	float
values	:	e.g. −3.0

7.12.4 gflgmax

function	:	sets maximum $\Delta \log gf$ value over which to perform CoG computations
required	:	yes, if `cog`=−1, −2
type	:	float
values	:	e.g. +2.5

If these properties are not included in `linfor_setcmd`, the default settings are invoked; `icg`= 51, `gflgmin`= −1.0, `gflgmax`= +1.5.

All of the options defined in Sect. 7 are checked by `linfor_checkcmd.pro` and are set to default values in the event that they are missing from `setcmd`.

7.13 IDL Example

```
pro linfor_setcmd
common linfordata

cmd = {$
;-------------------------------------------------------------------------
; Program execution flag:
run_flag: 3, $       ; execution mode: -3, -2, -1, 0, 1, 2, 3
              ; -3 - Compute the external 1D model atmosphere only
              ; -2 - Generate a 3x3 file for the external model atmosphere only
              ; -1 - Restore old results and compare with new 1D external model
              ;      (IDL-version only -- PLOTTING)
              ;  0 - Restore old results, replace 1D external model
              ;      (IDL-version only)
              ;  1 - Plotting input model structure on original vertical grid
              ;      (IDL-version only -- PLOTTING)
              ;  2 - Plotting input model structure on refined  vertical grid
              ;      (IDL-version only -- PLOTTING)
              ;  3 - Compute RT for 3D, <3D>, and 1DX model atmospheres
              ;      (Default)
;
;-------------------------------------------------------------------------
; NLTE execution flag:
nlte_flag: 0, $     ; 0 / 1 / 2 / 3
              ; 0 - No XBC files read. Everything is computed in LTE
              ;     (Default)
              ; 1 - Continuum & source function are LTE, line opacities are NLTE
              ; 2 - Continuum & line opacities are LTE, source function  is NLTE
              ; 3 - Continuum, source function & line opacities are NLTE
;
;-------------------------------------------------------------------------
; Other execution flags:
 cv1_flag: 1, $               ; 0 / 1: enforce <rho*v1>(z)=0 off / on
 cv2_flag: 1, $               ; 0 / 1: enforce <rho*v2>(z)=0 off / on
 cv3_flag: 1, $               ; 0 / 1: enforce <rho*v3>(z)=0 off / on
 plt_flag: 1, $               ; -1 / 0 / 1: plotting off / bisectors off / on
maps_flag: 1, $               ; create maps ICLAM0 .. ICLAMm, m=map_flag
cc3d_flag: 1, $               ; 0 / 1: output of CC3(nx,ny,nx) off / on
rdbb_flag: 0, $               ; 0/1/2: Read magnetic field, write SIR output
free_flag: 0, $               ; free pointers in structures at end of program
;
;-------------------------------------------------------------------------
; General paths:
abupath: getenv('LINFOR3D_ABU'), $
         ; Path to abu files and atom.dat
         ; if not set, abupath is read from environment variable LINFOR3D_ABU
ff_path: 'NONE', $                  ; directory with cached flow fields
;                                    'NONE': do not use cached flow fields
opapath: getenv('OPTABLES'),  $   ; directory with opacity tables
gaspath: getenv('EOSTABLES'), $   ; directory with GAS tables
eospath: getenv('EOSTABLES'), $   ; directory with EOS tables
;
;-------------------------------------------------------------------------
; Model data:
context:  'cobold', $
rhdpath:  '/data/models/d3gt57g44n59/bigsel/', $      ; directory with model data
modelid:  'd3gt57g44n59.*.full', $                    ; data file name
parfs:    '/data/models/d3gt57g44n59/rhd.par', $      ; parameter file
```

```
;
xbcpath:  '/data/models/d3gt57g44n59/NLTE3D/', $ ; directory of general
;                                                   departure coefficients
xbcpath3: '/data/models/d3gt57g44n59/NLTE3D/', $      ; directory of 3D
;                                                   departure coefficients
xbcpathx: '/data/models/d3gt57g44n59/NLTE3D/', $      ; directory of 1DX
;                                                   departure coefficients
abuid:    'cifist2006', $     ; model    abundance mixture, e.g. 'cifist2006'
abuidx:   'special',    $     ; spectrum abundance mixture, e.g. 'special'
;
dmetal: 0.0, $                ; log10 scaling for metal abundances (Z>3)
dalpha: 0.0, $                ; log10 scaling for alpha elements
;                               (O, Ne, Mg, Si, S, Ar, Ca, Ti)
nx_skip: 4, ny_skip: 4, $     ; sampling in x, y (kiel, cobold only)
;
; more information (all read from parameter file for CO5BOLD)
opafile: 'undefined', $
gasfile: 'undefined', $
eosfile: 'undefined', $
teff: 5770.0, grav: 27500.0, $     ; grey, copenhagen, muram only
;-------------------------------------------------------------------------------
htau0:    0.0E0, $            ; tau scale height; special 0 and negative
tsurffac: 0.0E0, $            ; Surface temperature = tsurffac*Teff
                              ; Read from parameter file when context='cobold'
;-------------------------------------------------------------------------------
; Reading of 'full' files (CO5BOLD only):
isnap_full_1: 1, $             ;  first snapshot to be read from full file(s)
isnap_full_2: 9, $             ;   last snapshot to be read from full file(s)
istep_full:   2, $             ; step for reading snapshots from full file(s)
;-------------------------------------------------------------------------------
; <3D> mean model:
mavg: 4, $      ; 1: T-average, 4: T^4-average for defining <3D> atmosphere
;-------------------------------------------------------------------------------
; External 1D reference model:
atmpath: '/data/models/d3gt57g44n59/lhdmodels/', $ ; directory of 1D
;                                                    reference model
atmfile:  't5780g44mm00ml3a10ob12_marcs.l50', $    ; name of reference model
;                                                    'NONE': no reference model
;-------------------------------------------------------------------------------
; Line data / radiative transfer:
linfs:   'line.dat', $            ; File with line data
;
lutau1: -7.0D0, lutau2: 2.0D0, dlutau: 8.0D-2, $ ; tau scale defining vertical
;                                                  model extent and resolution
lctau1: -7.0D0, lctau2: 2.0D0, dlctau: 8.0D-2, $ ; universal tau scale for
;                                                  integration of RT equation
ntheta: 3, nphi: 4, $           ; number of theta and phi angles
;
raybase: 'dblrdau', $     ; mu-quadrature method
;
mu0: 0.40, kphi: 0, $     ; view angle if ntheta=0 (cos theta,kphi*pi/2)
;-------------------------------------------------------------------------------
; Curve-of-Growth control
cog:        1, $  ; (-2,-1 / 0 / 1,2) Custom CoG / CoG off / default CoG
                  ; -2, 2: saves cg_out to large file linfor_3D_4.uiosave
                  ; -1, 1: no outputs saved
icg:       51, $  ; number of points to compute COG over (used when cog = -1,-2)
gflgmin: -1.0, $  ; minimum delta log(gf) (used when cog = -1,-2)
gflgmax: +1.5, $  ; maximum delta log(gf) (used when cog = -1,-2)
```

```
;
micro:  1,    $                  ; compute microturbulence sequence (0/1)
xi_a: 0.0,    $                  ; microturbulence sequence start [km/s]
xi_b: 2.0,    $                  ; microturbulence sequence stop  [km/s]
xi_d: 0.1,    $                  ; microturbulence delta sequence [km/s]
;-------------------------------------------------------------------------
; Balmer line computation control
Hbrd:     3 , $                 ; option for H line broadening
                                ; 0 - (default)
                                ; Cayrel&Traving (self res.) + Griem  (Stark)
                                ; 1 - Ali-Griem  (self res.) + Griem  (Stark)
                                ; 2 - BPO        (self res.) + Griem  (Stark)
                                ; 3 - Allard 08  (self res.) + Griem  (Stark)
                                ; 4 - Allard 08  (self res.) + Stehle (Stark)
                                ;     using properly convolved tabulated
                                ;     profiles
;-------------------------------------------------------------------------
; Plotting controls
vsini:  1.80, $           ; v sini (plot2); same for all spectra
;
ximacx: 1.60, $           ; macroturbulence [km/s], 1D-REF atmosphere
ximac1: 1.60, $                 ; macroturbulence [km/s], 1D-AVG atmosphere
ximac3: 0.00, $                 ; macroturbulence [km/s], 3D-RHD atmosphere
;-------------------------------------------------------------------------
; RT controls
ximicx: 1.00, $           ; microturbulence [km/s], 1D-REF atmosphere
ximic1: 1.00, $                 ; microturbulence [km/s], 1D-AVG atmosphere
ximic3: 0.00, $                 ; microturbulence [km/s], 3D-RHD atmosphere
;
vfacx:  1.00, $                 ; fudge factor for 3D x-velocity
vfacy:  1.00, $                 ; fudge factor for 3D y-velocity
vfacz:  1.00, $                 ; fudge factor for 3D z-velocity
;
intmode:  1,  $                 ; integration mode (linfor_msint)
intline:  1   $                 ; line integration: depth (1,2) / I (-1,-2)
;
dclam:  0.0,  $     ; variation of continuum from clam-dclam .. clam+dclam [A]
;
}
;
end
```

7.14 F90 Example

```
;-------------------------------------------------------------------------------
; Linfor3D Fortran 90 only variables
; ==================================
;
; Output filename (optional)
outfile     = 'linfor3D.uiosave'
;
; Printing options for UIO print outputs
printcobold = 0              ; 0 UIO print out is supressed
                             ; 1 UIO if fully output
;
;-------------------------------------------------------------------------------
; Program execution flag
;=======================
;
run_flag = 3 ; -3 - Compute the external 1D model atmosphere only
             ; -2 - Generate a 3x3 file for the external model atmosphere only
             ; -1 - Restore old results and compare with new 1D external model
             ;      (IDL-version only -- PLOTTING)
             ;  0 - Restore old results, replace 1D external model
             ;      (IDL-version only)
             ;  1 - Plotting input model structure on original vertical grid
             ;      (IDL-version only -- PLOTTING)
             ;  2 - Plotting input model structure on refined  vertical grid
             ;      (IDL-version only -- PLOTTING)
             ;  3 - Compute RT for 3D, <3D>, and 1DX model atmospheres
             ;      (Default)
;
;-------------------------------------------------------------------------------
; NLTE execution flag
;====================
;
nlte_flag = 0 ; read XBC file departure coefficients
              ; 0 - No XBC files read. Everything is computed in LTE
              ;     (Default)
              ; 1 - Continuum & source function are LTE, line opacities are NLTE
              ; 2 - Continuum & line opacities are LTE, source function  is NLTE
              ; 3 - Continuum, source function & line opacities are NLTE
;
;-------------------------------------------------------------------------------
; Other program execution flags
;==============================
;
wr3x3_flag = 1  ; (0/1) write the 1DX and global <3D> 3x3 models
wrtst_flag = 1  ; (0/1) write linfor_timing.txt
   d1_flag = 1  ; (0/1) create <3D> & global <3D> models for dort
  cv1_flag = 1  ; (0/1) enforce <rho*v1>(z)=0 off / on
  cv2_flag = 1  ; (0/1) enforce <rho*v2>(z)=0 off / on
  cv3_flag = 1  ; (0/1) enforce <rho*v3>(z)=0 off / on
 maps_flag = 1  ;         create maps ICLAM0 .. ICLAMm, m=map_flag
 cc3d_flag = 1  ; (0/1) Output of CC3(nx,ny,nx) off / on
 rdbb_flag = 0  ; (0/1) Read magnetic field, write SIR output
;
;-------------------------------------------------------------------------------
; General paths
;==============
;
```

```
; Path to abu files and atom.dat
abupath = 'LINFOR3D_ABU'  ; directory containing the .abu file
                          ; if 'LINFOR3D_ABU' then Linfor3D looks for the shell
                          ; variable $LINFOR3D_ABU
opapath = 'OPTABLES'      ; directory with opacity tables
gaspath = 'EOSTABLES'     ; directory with GAS tables
eospath = 'EOSTABLES'     ; directory with EOS tables
;
; === IMPORTANT ===
; if the above paths are not set, they are read from system environments.
; Not set means missing or
; = 'OPTABLES', = 'EOSTABLES', = 'LINFOR3D_ABU', = 'NONE', = ''
;
;-------------------------------------------------------------------------------
; Input data
;===========
;
context  = 'cobold'
rhdpath  = '~/snaps/d3gt57g44n59/bigsel/'          ; directory with model data
modelid  = 'd3gt57g44n59.*.full'                   ; data file name(s)
parfs    = '~/snaps/d3gt57g44n59/scripts/rhd.par'  ; parameter file
;
xbcpath  = '~/snaps/d3gt57g44n59/xbc/'             ; directory of matching
                                                   ; departure coefficients
xbcpathx = '~/snaps/d3gt57g44n59/xbc/'             ; directory of matching 1DX
                                                   ; departure coefficients
   ; (overrides xbcpath)
xbcpath3 = '~/snaps/d3gt57g44n59/xbc/'             ; directory of matching 3D
                                                   ; departure coefficients
   ; (overrides xbcpath)
;
abuid  = 'cifist2006'     ; model abundance mixture 1 (Atmosphere abundances)
abuidx = 'special'        ; model abundance mixture 2 (Synthesis  abundances)
;
dmetal = 0.0              ; log10 scaling for metal abundances (Z>3)
dalpha = 0.0              ; log10 scaling for alpha elements
                          ; (O, Ne, Mg, Si, S, Ar, Ca, Ti)
;
nx_skip = 4               ; sampling in x (kiel, cobold only)
ny_skip = 4           ; sampling in y (kiel, cobold only)
                          ; > 1 samples, < -1 rebins
                          ; = -1,0,1 does nothing
;
; more information (all read from parameter file for CO5BOLD)
opafile = 'undefined'
gasfile = 'undefined'
eosfile = 'undefined'
;
teff = 5770.0             ; context = grey, copenhagen, muram only
grav = 27500.0             ; context = grey, copenhagen, muram only
;
htau0    = 0.0E0            ; tau scale height; special 0 and negative
tsurffac = 0.0E0            ; Surface temperature = tsurffac*Teff
;
;-------------------------------------------------------------------------------
; <3D> mean model
;===============
;
mavg = 4         ; 1: T-average, 4: T^4-average for defining <3D> atmosphere
```

```
;
;-------------------------------------------------------------------------------
; External 1D reference model
;============================
;
atmpath = '~/snaps/d3gt57g44n59/lhdmodels/'     ; directory of 1D
                                                ; reference model
atmfile =  't5780g44mm00ml3a10ob12_marcs.l50'   ; name of reference model
                                                ; 'NONE' - no reference model
;
;-------------------------------------------------------------------------------
; Line data / radiative transfer
;===============================
;
linfs  = 'line.dat'   ; Line file: looks in working directory
;
lutau1 = -7.0D+0
lutau2 =  2.0D+0
dlutau =  8.0D-2      ; tau scale defining vertical model extent and resolution
;
; model extent and resolution
;
lctau1 = -7.0D+0
lctau2 =  2.0D+0
dlctau =  8.0D-2      ; universal tau scale for integration of RT equation
;
ntheta  = 3           ; number of theta (hence mu) angles
nphi    = 4           ; number of phi              angles
;
raybase = 'dblrdau'   ; mu-quadrature method:
                      ; 'lobatto', 'dblgauss', 'dblrdau', 'special'
;
mu0     = 1.00        ; mu angle if ntheta=0 (also used for writing SIR data)
kphi    = 0           ; view angle if ntheta=0 (cos theta,kphi*pi/2)
;
;-------------------------------------------------------------------------------
; Curve-of-Growth control
;========================
;
cog     =    1   ; (-2,-1 / 0 / 1,2) Custom CoG / CoG off / default CoG
                 ; -2, 2: saves cg_out to large file linfor_3D_4.uiosave
                 ; -1, 1: no outputs saved
icg     =   51   ; number of points to compute COG over (used when cog = -1,-2)
gflgmin = -1.0   ; minimum delta log(gf) (used when cog = -1,-2)
gflgmax = +1.5   ; maximum delta log(gf) (used when cog = -1,-2)
;
micro   = 1      ; compute microturbulence sequence (0/1)
xi_a    = 0.0    ; microturbulence sequence start [km/s]
xi_b    = 2.0    ; microturbulence sequence stop  [km/s]
xi_d    = 0.1    ; microturbulence delta sequence [km/s]
;
;-------------------------------------------------------------------------------
; Balmer line computation control
;================================
;
Hbrd = 3   ; Option for H line broadening
           ; 0 - (default) Cayrel & Traving (self res.) + Griem  (Stark)
           ; 1 -           Ali-Griem        (self res.) + Griem  (Stark)
           ; 2 -           BPO              (self res.) + Griem  (Stark)
```

```
           ; 3 -           Allard 08        (self res.) + Griem  (Stark)
           ; 4 -           Allard 08        (self res.) + Stehle (Stark)
           ;                                using properly convolved tabulated
           ;                                profiles
;
;-------------------------------------------------------------------------------
; RT controls
;============
;
ximicx = 1.00      ; microturbulence [km/s], 1D-REF atmosphere
ximic1 = 1.00      ; microturbulence [km/s], 1D-AVG atmosphere
ximic3 = 0.00      ; microturbulence [km/s], 3D-RHD atmosphere
;
vfacx =  1.00      ; fudge factor for 3D x-velocity
vfacy =  1.00      ; fudge factor for 3D y-velocity
vfacz =  1.00      ; fudge factor for 3D z-velocity
;
intmode =  1       ; integration mode (linfor_msint)
intline =  2       ; line integration - depth (1,2) / I (-1,-2)
;
dclam = 0.0d0      ; variation of continuum from clam-dclam .. clam+dclam [A]
```

8 Line Data File: line.dat

There are several different formats (for historical reasons) to specify line data which are described in Sect. 8.3.
Note that all formats were extended in version 1.5.0 and now do have to contain the two lines

```
clam     gfscale
2000.0   1.0
```

at the end. These parameters are explained in Sect. 8.1. Some helpful remarks concerning the conversion of line broadening parameters are given in Sect. 8.4.

A basic IDL program for creating a properly formatted line can be found in `linfor_wrline.pro`, within the `Routines` sub-directory within the Linfor3D directory tree.

8.1 Parameters in Line Data File

8.1.1 clam

function	:	continuum wavelength in Å, also center of wavelength window
required	:	always
type	:	float
values	:	e.g., 2000.0

`clam` defines the wavelength where the continuum opacities are computed, and also defines the center of the window for which spectrum synthesis is done. The window extends from λ =`clam-dlam` to λ =`clam+dlam`, depending on the value of `dlam` specified for the particular line.
From Version 3.1.2, a **negative** `clam` indicates that the continuum source function is to be set to the wavelength-integrated Planck-Function, $S = \sigma T^4/\pi$, and the continuum opacity is set to the Rosseland mean opacity, $\kappa_{\rm cont} = \kappa_{\rm Ross}$..

8.1.2 gfscale

function	:	global scaling factor for oscillator strengths
required	:	always
type	:	float
values	:	e.g., 1.0

Note: The value 1 has no effect. *Useful when* `line.dat` *contains more than one transition.*

8.2 Setting equivalent widths – `Wlam0` – in line files

It is possible to enter an **equivalent width** (W_0 in [mÅ]) in every line file, regardless of format. For this purpose, *nbl* (explained below) must be negative, with |*nbl*| being the number of blend components. The $\log gf$ value producing this equivalent width W_0 for the ***average 3D atmosphere*** is returned in `result.gflg01` and in `result.gflg0x` for the ***1D reference atmosphere***.

It uses the Curve-of-Growth computations (Sect. 7.12) to determine the corrections the user must make to the ⟨3D⟩ and 1D external line's $\log gf$ value. If this parameter is set, then the **Curve-of-Growth** is turned on, should it be off in the **`cmd`** file.

8.3 Line Data Formats

8.3.1 Continuum only

It is possible to do pure continuum calculations. In this case, the `line.dat` file looks like this.

Example:

```
Some text header
1   1
Continuum, 2000 A
1   -1
clam     gfscale
2000.0   1.0
```

Description of entries:

Row 1: Header (identifies the meaning of the columns for data in row 5)
Row 2: Two integers, *kline* and *ktotal*; both of them must be 1
Row 3: String, identifier of the continuum calculation
Row 4: Two integers, $nbl = 1$, $incode = -1$
Row 5: Description for data in row 6
Row 6: `clam` and `gfscale` (see Sect.8.1)

All the line parameters remain undefined.

8.3.2 Single line calculations, line data format '0'

For a **single unblended line**, the simplest form of the 'line.dat' file looks like this.

Example:

```
Mult   namj      ei      alam     gflg     dlgC6   drrca1  dlam   ddlam
1   1
Fe I, 5500 A, 0.00 eV
1   0
0000   2600      0.000   5500.0   -6.000   1.0     10.0    5.5D-1 5.5D-3
clam     gfscale
2000.0   1.0
```

Description of entries:

Row 1: Header (identifies the meaning of the columns for data in row 5)
Row 2: Two integers, *kline* and *ktotal*
  kline: number of line calculations requested in this file
  ktotal: is the total number of spectral lines including blends
  in this case $kline = 1$, $ktotal = 1$
Row 3: String, identifier of the (first) line calculation
Row 4: Integer *nbl*, integer array *incode*(*nbl*)
  nb: number of blend components for this line calculation (= 1)
  incode: integer array identifying the input format for each of the blend components (= 0)
Row 5: Line data in format '0' (7 + 2 columns):
  C1: Multiplet number (for information only)
  C2: Identifier of atom or ion (e.g. 2601 mean FeII)
  C3: Excitation potential of lower level in [eV]

C4: Central wavelength of blend component
C5: $\log gf$ value of blend component
C6: $\Delta \log C_6$: Enhancement factor for van der Waals line broadening
C7: $\Delta\overline{r^2}/a_0^2$: Difference of mean square electron orbital radii
C8: $\Delta\lambda$ [Å]: Line profile is computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$
C9: $\delta\lambda$ [Å]: Spacing of wavelength points for spectrum synthesis
(C10: W_0 [mÅ]: total equivalent width of this blend, see Sect 8.2)
Row 6: Description for data in row 6
Row 7: `clam` and `gfscale` (see Sect.8.1)

In this case, the Stark broadening (due to collisions with electrons) is neglected ($C_4 = 0$). Radiative damping ($\gamma_{\rm rad}$) is treated in the classical approximation.

In the case of a **single blended line** the 'line.dat' file looks as follows:

Example:

```
Mult   namj      ei      alam     gflg     dlgC6   drrca1  dlam   ddlam
1   2
Fe I, 0.00 eV + Fe II, 3.00 eV, 2000 A
2   0 0
9999   2600      0.000   2000.0   -6.441   1.0     10.0
9999   2601      3.000   2000.0   -4.550   1.0     10.0    1.5D-1 1.5D-3
clam     gfscale
2000.0   1.0
```

Note that it is not necessary that the blend components belong to the same ion. Here $kline = 1$, $ktotal = 2$, $nbl = 2$, $incode = [0, 0]$. Note that only the last of the rows describing the blend need entries $C8$ and $C9$.

With a slight modification, it is possible to enter an **equivalent width** (W_0 in [mÅ]) in column $C10$. For this purpose, nbl must be negative, with $|nbl|$ being the number of blend components. The gf value producing this equivalent width W_0 is returned in `result.gflg01` (average 3D atmosphere) and `result.gflg0x` (1D reference atmosphere).

Example unblended line:

```
Mult   namj  chik    alam      gflg    dlgC6  drrca1  dlam     ddlam      W0
  1  1
  N I Fictitious Line  1: /   0.000   5500.0  -7.6914  1.00   10.00   75.00 /
 -1  0
9999   700   0.000   5500.0  -7.6914  1.00   10.00  3.00E-01  3.00E-03   75.00
clam     gfscale
2000.0   1.0
```

Example blended line:

```
Mult   namj  chik    alam      gflg    dlgC6  drrca1  dlam     ddlam      W0
  1  2
Fe I, 0.00 eV + Fe II, 3.00 eV, 2000 A
 -2  0 0
9999   2600  0.000   2000.0   -6.441   1.0    10.0
9999   2601  3.000   2000.0   -4.550   1.0    10.0  1.50D-1   1.50D-03  100.00
clam     gfscale
2000.0   1.0
```

8.3.3 Single line calculations, line data format '1'

For a **single unblended line**, the this form of the 'line.dat' file looks like this.

Example:

```
Mult   namj   ei      alam     gflg     dlgC6 lu  diu  lo  dio  dlam   ddlam
1   1
O I ApJ Line 2: 92  6300.30  0.000 -9.773
1   1
  92   800     0.000  6300.30  -9.773   1.0   1   0.0  2   0.0  4.D-1  4.D-3
clam     gfscale
2000.0   1.0
```

Description of entries:

Row 1: Header (identifies the meaning of the columns for data in row 5)
Row 2: Two integers, *kline* and *ktotal*
kline: number of line calculations requested in this file
ktotal: is the total number of spectral lines including blends
in this case *kline* = 1, *ktotal* = 1
Row 3: String, identifier of the (first) line calculation
Row 4: Integer *nbl*, integer array *incode*(*nbl*)
nb: number of blend components for this line calculation (= 1)
incode: integer array identifying the input format for each of the blend components (= 1)
Row 5: Line data in format '1' (10 + 2 columns):
C1: Multiplet number (for information only)
C2: Identifier of atom or ion (e.g. 2601 mean FeII)
C3: Excitation potential of lower level in [eV]
C4: Central wavelength of blend component
C5: $\log gf$ value of blend component
C6: $\Delta \log C_6$: Enhancement factor for van der Waals line broadening
C7: *LU*: Orbital quantum number of valence electron of lower level
C8: *DIU*: excitation energy [*eV*] of parent term for lower level
C9: *LO*: Orbital quantum number of valence electron of upper level
C10: *DIO*: excitation energy [*eV*] of parent term for upper level
C11: $\Delta\lambda$ [Å]: Line profile is computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$
C12: $\delta\lambda$ [Å]: Spacing of wavelength points for spectrum synthesis
(C13: W_0 [mÅ]: total equivalent width of this blend, see Sect 8.2)
Row 6: Description for data in row 6
Row 7: `clam` and `gfscale` (see Sect.8.1)

In this case, $\Delta\overline{r^2}/a_0^2$ is computed from *LU*, *DIU*, *LO*, *DIO* (Function *rrca*). As before, the Stark broadening (due to collisions with electrons) is neglected ($C_4 = 0$). Radiative damping ($\gamma_{\rm rad}$) is treated in the classical approximation.

In the case of a **single blended line** the 'line.dat' file looks as follows:

Example:

```
Mult   namj   ei      alam     gflg     dlgC6  lu  diu  lo  dio  dlam   ddlam
1   3
O I ApJ Line 1: 67  6158.17 10.741 -1.140
```

```
3   1 1 1
  67   800    10.741  6158.15  -1.985   1.0    1   0.0  2   0.0
  67   800    10.741  6158.17  -1.140   1.0    1   0.0  2   0.0
  67   800    10.741  6158.19  -0.553   1.0    1   0.0  2   0.0  4.D-1  4.D-3
clam     gfscale
2000.0   1.0
```

Here *kline* = 1, *ktotal* = 3, *nbl* = 3, *incode* = [1, 1, 1]. Note that only the last of the rows describing the blend need entries $C11$ and $C12$.

As in the case of format '0', it is possible to enter an **equivalent width** (W_0 in [mÅ]) in column $C13$. For this purpose, *nbl* must be negative, with |*nbl*| being the number of blend components. The gf value producing this equivalent width W_0 is returned in `result.gflg01` (average 3D atmosphere) and `result.gflg0x` (1D reference atmosphere).

Example unblended line:

```
Mult   namj  ei     alam     gflg   dlgC6 lu  diu  lo  dio  dlam  ddlam  W0
 1  1
O I ApJ Line 2: 92  6300.30  0.000 -9.773
-1  1
  92   800   0.000  6300.30  -9.773 1.0   1   0.0  2   0.0  4.D-1 4.D-3  7.00
clam     gfscale
2000.0   1.0
```

Example blended line:

```
Mult   namj   ei    alam     gflg   dlgC6 lu  diu  lo  dio  dlam  ddlam  W0
 1  3
O I ApJ Line 1: 67  6158.17 10.741 -1.140
-3  1 1 1
  67   800  10.741  6158.15  -1.985 1.0   1   0.0  2   0.0
  67   800  10.741  6158.17  -1.140 1.0   1   0.0  2   0.0
  67   800  10.741  6158.19  -0.553 1.0   1   0.0  2   0.0  4.D-1 4.D-3 10.00
clam     gfscale
2000.0   1.0
```

8.3.4 Single line calculations, complete line data format '2'

For a **single unblended line**, the this form of the 'line.dat' file looks like this.

Example:

```
Mult   namj   ei      alam     gflg   dlgC6  drrca1  dlgC4 C4lg   dlggr  Crad  dlam   ddlam
1   1
Si I AA Line 5: 16   5948.540  5.0823  -1.130  390.03   11.80  -1  86
1   2
   16  1400   5.0823  5948.540 -1.130  1.0   390.03  0.0   11.80  0.0    -1.0  5.D-1  5.D-3
clam     gfscale
2000.0   1.0
```

Description of entries:

Row 1: Header (identifies the meaning of the columns for data in row 5)
Row 2: Two integers, *kline* and *ktotal*
kline: number of line calculations requested in this file
ktotal: is the total number of spectral lines including blends

in this case *kline* = 1, *ktotal* = 1
Row 3: String, identifier of the (first) line calculation
Row 4: Integer *nbl*, integer array *incode*(*nbl*)
nb: number of blend components for this line calculation (= 1)
incode: integer array identifying the input format for each of the blend components (= 2)
Row 5: Line data in format '2' (11 + 2 columns):
C1: Multiplet number (for information only)
C2: Identifier of atom or ion (e.g. 2601 mean FeII)
C3: Excitation potential of lower level in [eV]
C4: Central wavelength of blend component
C5: $\log gf$ value of blend component
C6: $\Delta \log C_6$: Enhancement factor for van der Waals line broadening
C7: $\Delta\overline{r^2}/a_0^2$: Difference of mean square electron orbital radii
C8: $\Delta \log C_4$: Enhancement factor for Stark line broadening
C9: $-\log C_4$: Stark broadening constant.
if $-\log C_4 = 0$, then use Griem (Phys. Rev. 165, 258, 1968)
and Cowley (Obs. 91, 139, 1971) approximation
if $-\log C_4 < 0$ (typically $-\log C_4 = -1.0$), then $C_4 = 0$
C10: $\Delta \log \gamma_{\rm rad}$: Enhancement factor for natural line broadening
C11: $C_{\rm rad}$: Natural line broadening ($10^{-8}\gamma_{\rm rad}$)
if $C_{\rm rad} < 0$, use classical formula ($\gamma_{\rm rad} = 2.22 \cdot 10^{15}/\lambda^2$) [rad/s], where λ is in Å.
C12: $\Delta\lambda$ [Å]: Line profile is computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$
C13: $\delta\lambda$ [Å]: Spacing of wavelength points for spectrum synthesis
(C14: W_0 [mÅ]: total equivalent width of this blend, see Sect 8.2)
Row 6: Description for data in row 6
Row 7: `clam` and `gfscale` (see Sect.8.1)

In the case of a **single blended line** the 'line.dat' file looks as follows:

Example:

```
Mult   namj   ei      alam     gflg   dlgC6  drrca1  dlgC4 C4lg   dlggr  Crad  dlam   ddlam
1   2
Si I / Si II blend: 16   5948.540  5.0823  -1.130  390.03   11.80  -1  86
2   2 2
   16  1400   5.0823  5948.540 -1.130  1.0   390.03  0.0   11.80  0.0    -1.0
   16  1401   0.0823  5948.530 -3.130  1.0    90.00  0.0   13.80  0.0    -1.0  5.D-1  5.D-3
clam     gfscale
2000.0   1.0
```

Here *kline* = 1, *ktotal* = 2, *nbl* = 2, *incode* = [2, 2, 2]. Note that only the last of the rows describing the blend need entries *C*12 and *C*13.

As in the cases of format '0' and '1', it is possible to enter an **equivalent width** (W_0 in [mÅ]) in column *C*14. For this purpose, *nbl* must be negative, with |*nbl*| being the number of blend components. The gf value producing this equivalent width W_0 is returned in `result.gflg01` (average 3D atmosphere) and `result.gflg0x` (1D reference atmosphere). No examples are given since the necessary modification the the data format should be obvious.

8.3.5 Single line calculations, complete line data format '3'

This data format has a maximum of 17 columns. It differs from format '2' only in the way the van der Waals broadening parameters are specified. Columns *C*7 with $\Delta\overline{r^2}/a_0^2$ is replaced by the four columns:

C7: LU: Orbital quantum number of valence electron of lower level
C8: DIU: excitation energy [eV] of parent term for lower level
C9: LO: Orbital quantum number of valence electron of upper level
C10: DIO: excitation energy [eV] of parent term for upper level

as in format '1'. The remaining columns are as in format '2', but shifted by +3:

C11: $\Delta \log C_4$: Enhancement factor for Stark line broadening
C12: $-\log C_4$: Stark broadening constant.
if $\log C_4 = 0$, then use Griem (Phys. Rev. 165, 258, 1968)
and Cowley (Obs. 91, 139, 1971) approximation
if $-\log C_4 < 0$ (typically $-\log C_4 = -1.0$), then $C_4 = 0$
C13: $\Delta \log \gamma_{\rm rad}$: Enhancement factor for natural line broadening
C14: $C_{\rm rad}$: Natural line broadening ($10^{-8}\gamma_{\rm rad}$)
if $C_{\rm rad} < 0$, use classical formula ($\gamma_{\rm rad} = 2.22 \cdot 10^{15}/\lambda^2$) [rad/s], where λ is in Å.
C15: $\Delta\lambda$ [Å]: Line profile is computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$
C16: $\delta\lambda$ [Å]: Spacing of wavelength points for spectrum synthesis
(C17: W_0 [mÅ]: total equivalent width of this blend, see Sect 8.2)

8.3.6 Single line calculations, complete line data format '4'

This data format has a maximum of 14 columns. It differs from format '2' only in the way the van der Waals broadening parameter is specified. Column $C7$ with $\Delta\overline{r^2}/a_0^2$ is replaced by the parameter $-\log C_6$.

C7: $-\log C_6$: negative logarithmic van der Waals broadening parameter C_6

The remaining columns are as in format '2'.

C8: $\Delta \log C_4$: Enhancement factor for Stark line broadening
C9: $-\log C_4$: Stark broadening constant.
if $\log C_4 = 0$, then use Griem (Phys. Rev. 165, 258, 1968)
and Cowley (Obs. 91, 139, 1971) approximation
if $-\log C_4 < 0$ (typically $-\log C_4 = -1.0$), then $C_4 = 0$ (no Stark broadening)
C10: $\Delta \log \gamma_{\rm rad}$: Enhancement factor for natural line broadening
C11: $C_{\rm rad}$: Natural line broadening ($10^{-8}\gamma_{\rm rad}$)
if $C_{\rm rad} < 0$, use classical formula ($\gamma_{\rm rad} = 2.22 \cdot 10^{15}/\lambda^2$) [rad/s], where λ is in Å.
C12: $\Delta\lambda$ [Å]: Line profile is computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$
C13: $\delta\lambda$ [Å]: Spacing of wavelength points for spectrum synthesis
(C14: W_0 [mÅ]: total equivalent width of this blend, see Sect 8.2)

Example:

```
Mult   namj    ei     alam        gflg      dlgC6  C6log    dlgC4  C4log    dlggr Crad  dlam    ddl
1  12
Li7, 6 components Li6, 6 components: glog=-0.427905 .. -0.831310
12  4 4 4 4 4 4 4 4 4 4 4 4
9999   0300.7  0.00   6707.7560  -0.427905  0.84   31.3843  0.0    14.1505  0.0   -1.0
9999   0300.7  0.00   6707.7680  -0.206158  0.84   31.3843  0.0    14.1505  0.0   -1.0
9999   0300.7  0.00   6707.9070  -0.808148  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.7  0.00   6707.9080  -1.507150  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.7  0.00   6707.9190  -0.808148  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.7  0.00   6707.9200  -0.808148  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.6  0.00   6707.9200  -0.478953  0.84   31.3844  0.0    14.1505  0.0   -1.0
```

```
9999   0300.6  0.00   6707.9230  -0.178176  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.6  0.00   6708.0690  -0.831310  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.6  0.00   6708.0700  -1.734310  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.6  0.00   6708.0740  -0.734310  0.84   31.3844  0.0    14.1505  0.0   -1.0
9999   0300.6  0.00   6708.0750  -0.831310  0.84   31.3844  0.0    14.1505  0.0   -1.0  10.D-1  5.D-3
clam     gfscale
6707.840  5.0119
```

8.3.7 Single line calculations, complete line data format '5'

This data format has a maximum of 15 columns. It differs substantially form format '4': (i) an extra column is inserted that allows the specification of $\log gf$ offsets; (ii) the van der Waals broadening is specified by $\log \gamma_6$ instead of $-\log C_6$; (iii) the Stark broadening is specified by $\log \gamma_4$ instead of $-\log C_4$; (iv) the natural broadening is specified by $\log \gamma_{\rm rad}$ instead of $\gamma_{\rm rad}/10^8$. More precisely, column $C5$–$C15$ have the following meaning in format '5':

C5: $\Delta \log gf$: Correction factor for the line's $\log gf$ value
C6: $\log gf$: the line's logarithmic gf value
C7: $\Delta \log \gamma_6$: Enhancement factor for van der Waals γ parameter
C8: $\log\left(\frac{\gamma_6(T=10^4)}{N_{\rm H}}\right)$: logarithmic van der Waals broadening parameter $\gamma_6/N_{\rm H}$ at $T = 10^4$ K.
C9: $\Delta \log \gamma_4$: Enhancement factor for Stark γ parameter
C10: $\log\left(\frac{\gamma_4(T=10^4)}{N_{\rm e}}\right)$: logarithmic Stark broadening parameter $\gamma_4/N_{\rm e}$ at $T = 10^4$ K.
if $\log \gamma_4/N_{\rm e} \geq 0$, then use Griem (Phys. Rev. 165, 258, 1968)
and Cowley (Obs. 91, 139, 1971) approximation
C11: $\Delta \log \gamma_{\rm rad}$: Enhancement factor for the natural line broadening
C12: $\log \gamma_{\rm rad}$: Natural line broadening ($\log \gamma_{\rm rad}$ [rad/s])
if $\log \gamma_{\rm rad} \geq 99.0$, use classical formula ($\gamma_{\rm rad} = 2.22 \cdot 10^{15}/\lambda^2$) [rad/s], where λ is in Å.
C13: $\Delta\lambda$ [Å]: Line profile is computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$
C14: $\delta\lambda$ [Å]: Spacing of wavelength points for spectrum synthesis
(C15: W_0 [mÅ]: total equivalent width of this blend, see Sect 8.2)

Example:

```
Mult   namj    ei     alam       dlggf  gflg      dlgg6   g6log   dlgg4   g4log    dlggr grlog dlam    ddlam
1  12
Li7, 6 components Li6, 6 components: glog=-0.427905 .. -0.831310
12  5 5 5 5 5 5 5 5 5 5 5 5
9999   0300.7  0.00   6707.7560  0.00  -0.427905  2.10  -7.94973  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.7680  0.00  -0.206158  2.10  -7.94974  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9070  0.00  -0.808148  2.10  -7.94975  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9080  0.00  -1.507150  2.10  -7.94975  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9190  0.00  -0.808148  2.10  -7.94975  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9200  0.00  -0.808148  2.10  -7.94975  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6707.9200  0.00  -0.478953  2.10  -7.94975  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6707.9230  0.00  -0.178176  2.10  -7.94975  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0690  0.00  -0.831310  2.10  -7.94976  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0700  0.00  -1.734310  2.10  -7.94976  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0740  0.00  -0.734310  2.10  -7.94976  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0750  0.00  -0.831310  2.10  -7.94976  0.00   -5.7800   0.0   99.0  10.D-1  5.D-3
clam     gfscale
6707.840  5.0119
```

8.3.8 Single line calculations, complete line data format '6'

This data format also has a maximum of 15 columns. It differs from format '5' only in the way the van der Waals broadening parameters are specified. In format '6', column $C7$–$C8$ have the following meaning:

C7: $\sigma_{\rm ABO}$: van der Waals broadening cross section in atomic units at $v_0 = 10\,$km/s according ABO theory
C8: $\alpha_{\rm ABO}$: α parameter of ABO theory defining the velocity (temperature) dependence of the cross section σ.

The remaining columns $C9$–$C15$ are as in format '5'. Note that:

- no enhancement factor for van der Waals broadening is foreseen in this line data format.
- the temperature dependence of the broadening cross section is correctly taken into account according to the ABO theory when this line data format is used.

Example:

```
Mult   namj    ei     alam       dlggf  gflg      s_abo     a_abo    dlgg4   g4log    dlggr grlog dlam    ddla
1  12
Li7, 6 components Li6, 6 components: glog=-0.427905 .. -0.831310
12  6 6 6 6 6 6 6 6 6 6 6 6
9999   0300.7  0.00   6707.7560  0.00  -0.427905  355.909   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.7680  0.00  -0.206158  355.900   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9070  0.00  -0.808148  355.892   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9080  0.00  -1.507150  355.892   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9190  0.00  -0.808148  355.892   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.7  0.00   6707.9200  0.00  -0.808148  355.892   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6707.9200  0.00  -0.478953  355.892   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6707.9230  0.00  -0.178176  355.892   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0690  0.00  -0.831310  355.894   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0700  0.00  -1.734310  355.894   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0740  0.00  -0.734310  355.894   0.40000  0.00   -5.7800   0.0   99.0
9999   0300.6  0.00   6708.0750  0.00  -0.831310  355.894   0.40000  0.00   -5.7800   0.0   99.0  10.D-1  5.D-
clam     gfscale
6707.840  5.0119
```

8.3.9 Single line calculations, complete line data format '7'

This data format was designed for simple test calculations where the line profile is fixed, i.e. the line parameters are depth-independent (see also Sect. 5.5). This format has a maximum of 7 columns:

Description of entries:

Row 1: Header (identifies the meaning of the columns for data in row 5)
Row 2: Two integers, *kline* and *ktotal*
kline: number of line calculations requested in this file
ktotal: is the total number of spectral lines including blends
in this case *kline* = 1, *ktotal* = 1
Row 3: String, identifier of the line calculation
Row 4: Integer *nbl*, integer array *incode*(*nbl*)
nb: number of blend components for this line calculation (= 1)
incode: integer array identifying the input format for each of the blend components (= 7)
Row 5: C1: Central wavelength of blend component [Å]
C2: Doppler broadening in units of c, $v_{\rm D}/c$
C3: $\eta_0 = \kappa_{\rm line}/\kappa_{\rm cont}$ at line center
C4: α-parameter for Voigt profile, $\alpha = \gamma/2/\Delta\omega_{\rm D}$
(ratio of **half** half width of dispersion profile and Doppler widths of Gaussian).
C5: $\Delta\lambda$ [Å]: Line profile is computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$

C6: $\delta\lambda$ [Å]: Spacing of wavelength points for spectrum synthesis
(C7: W_0 [mÅ]: total equivalent width of this blend, see Sect 8.2)
Row 6: Description for data in row 6
Row 7: `clam` and `gfscale` (see Sect.8.1)

Example:

```
alam     Vdop     eta0    avgt    dlam    ddlam
1    1
Test    grey sf  Vdop=2.D-5, eta0=1.0D0, avgt=1.D-2
1   7
4000.000 2.0D-5   1.0D0   1.0D-2  0.90D0  0.90D-2
clam        gfscale
-4000.000   1.0
```

8.3.10 Multiple Line Calculations

It is also possible to process a whole set of lines in a single run. The requirement is, however, that all lines have the same central wavelength (continuum wavelength). This mode was designed for parameter studies, e.g. investigating the "granulation abundance corrections" as a function of line excitation potential.

Example, 8 unblended N I lines of different excitation potential:

```
Mult   namj  chik    alam      gflg    dlgC6  drrca1  dlam     ddlam      W0
  8  8
  N I Fictitious Line  1: /   0.000   5500.0  -7.6914  1.00   10.00   75.00 /
 -1  0
9999   700   0.000   5500.0  -7.6914  1.00   10.00  3.00E-01  3.00E-03   75.00
  N I Fictitious Line  2: /   2.000   5500.0  -5.7282  1.00   10.00   75.00 /
 -1  0
9999   700   2.000   5500.0  -5.7282  1.00   10.00  3.00E-01  3.00E-03   75.00
  N I Fictitious Line  3: /   4.000   5500.0  -3.8298  1.00   10.00   75.00 /
 -1  0
9999   700   4.000   5500.0  -3.8298  1.00   10.00  3.00E-01  3.00E-03   75.00
  N I Fictitious Line  4: /   6.000   5500.0  -1.9876  1.00   10.00   75.00 /
 -1  0
9999   700   6.000   5500.0  -1.9876  1.00   10.00  3.00E-01  3.00E-03   75.00
  N I Fictitious Line  5: /   8.000   5500.0  -0.1961  1.00   10.00   75.00 /
 -1  0
9999   700   8.000   5500.0  -0.1961  1.00   10.00  3.00E-01  3.00E-03   75.00
  N I Fictitious Line  6: /  10.000   5500.0   1.5485  1.00   10.00   75.00 /
 -1  0
9999   700  10.000   5500.0   1.5485  1.00   10.00  3.00E-01  3.00E-03   75.00
  N I Fictitious Line  7: /  11.000   5500.0   2.4046  1.00   10.00   75.00 /
 -1  0
9999   700  11.000   5500.0   2.4046  1.00   10.00  3.00E-01  3.00E-03   75.00
  N I Fictitious Line  8: /  12.000   5500.0   3.2510  1.00   10.00   75.00 /
 -1  0
9999   700  12.000   5500.0   3.2510  1.00   10.00  3.00E-01  3.00E-03   75.00
clam     gfscale
2000.0   1.0
```

Note that now *kline* = 8, and *ktotal* = 8, since all lines have one blend component only.

8.4 Conversion of line broadening parameters

The line broadening can be specified in different ways, e.g. as $-\log C_4$ for quadratic Stark broadening. The required data is, however, not always available and must be converted from other broadening parameters, e.g. γ_4. In the particular case of the *Vienna Atomic Line Database* the broadening is provided as $\log(\gamma_4/N_e)$ for a temperature of $T = 10^4$ K.

Please note that here and in Linfor3D in general, the parameters C_n ($n = 4, 6$) are defined via

$$\Delta\omega = \frac{C_n}{r^n} \tag{79}$$

whereas the definition by Unsöld is

$$\Delta\omega = 2\,\pi\,\frac{C_n}{r^n}. \tag{80}$$

The Linfor parameters C_n are thus a factor 2π larger than in the definition by Unsöld.

Note that $\gamma_{\rm rad}, \gamma_4, \gamma_6$ measure the **full** width at half maximum of the Lorentzian profile in units of rad/s.

8.4.1 Quadratic Stark effect

The broadening parameter γ_4 for the quadratic Stark effect can be written as

$$\gamma_4 = 11.37\, C_4^{2/3}\, {v_{\rm rel}}^{1/3}\, N_{\rm e}\ , \tag{81}$$

where $v_{\rm rel}$ is the relative velocity between the regarded atom and the perturber, i.e. the colliding particle:

$${v_{\rm rel}}^2 = \frac{8\,k\,T}{\pi\,m_{\rm H}} \cdot \left(\frac{1}{A_1} + \frac{1}{A_2}\right)\ . \tag{82}$$

A_1 and A_2 are the atomic weights in atomic mass units, e.g., $A_2 = 1$ for a colliding hydrogen atom and $A_1 \approx 56$ for iron atoms and $A_2 = 1/1837 = m_{\rm e}/m_{\rm H}$ for electrons. For Stark broadening with electrons as perturbers the following good approximation can be made:

$$A_1 >> A_2 \Rightarrow \frac{1}{A_1} + \frac{1}{A_2} \approx \frac{1}{A_2} = 1837 = m_{\rm H}/m_{\rm e} \tag{83}$$

With this Eq. 81 can be written as

$$\begin{aligned}
\log\frac{\gamma_4}{N_{\rm e}} &= \log 11.37 + \log C_4^{2/3} + \log {v_{\rm rel}}^{1/3} && (84)\\
&= 1.056 + \frac{2}{3}\log C_4 + \frac{1}{6}\log\frac{8\,k\,T}{\pi\,m_{\rm e}} && (85)\\
&= 1.056 + \frac{2}{3}\log C_4 + 1.931 + \frac{1}{6}\log T && (86)\\
&= 1.056 + \frac{2}{3}\log C_4 + 1.931 + \frac{1}{6}\log 10^4 + \frac{1}{6}\log\frac{T}{10^4\,\rm K} && (87)\\
& && (88)
\end{aligned}$$

With $T = 10^4$ K, which is assumed for data in VALD, we derive

$$\log\frac{\gamma_4}{N_{\rm e}} = 3.654 + \frac{2}{3}\log C_4 \tag{89}$$

and finally the conversion formula:

$$\log C_4 = 1.5\log\frac{\gamma_4}{N_{\rm e}} - 5.4805 \tag{90}$$

For instance a value of -5.491 from VALD gives $\log C_4 = -13.717$. The parameter `C4lg` is thus set to `13.717`.

8.4.2 Van der Waals broadening

The broadening parameter γ_6 for the van der Waals effect can be written as

$$\gamma_6 = 8.08\, {C_6}^{2/5}\, {v_{\rm rel}}^{3/5}\, N_{\rm H} \quad . \tag{91}$$

The perturbing particles are mostly hydrogen atoms with $A_2 = 1$. We now make the approximation

$$A_1 > A_2 \Rightarrow \frac{1}{A_1} + \frac{1}{A_2} \approx \frac{1}{A_2} = 1 \tag{92}$$

With this the relative velocity of the particles (Eq. 82) reduces to

$${v_{\rm rel}}^2 = \frac{8\,k\,T}{\pi\, m_{\rm H}}. \tag{93}$$

We can thus rewrite Eq. 91:

$$\begin{aligned}
\log \frac{\gamma_6}{N_{\rm H}} &= \log 8.08 + \log C_6^{2/5} + \log {v_{\rm rel}}^{3/5} && (94)\\
&= 0.907 + \frac{2}{5}\log C_6 + \frac{3}{10}\log \frac{8\,k\,T}{\pi\, m_{\rm H}} && (95)\\
&= 0.907 + \frac{2}{5}\log C_6 + 2.497 + \frac{3}{10}\log T && (96)\\
&= 0.907 + \frac{2}{5}\log C_6 + 2.497 + \frac{3}{10}\log 10^4 + \frac{3}{10}\log \frac{T}{10^4\,\mathrm{K}} && (97)\\
& && (98)
\end{aligned}$$

With $T = 10^4$ K, which is assumed for data in VALD, we derive

$$\log \frac{\gamma_6}{N_{\rm H}} = 4.604 + \frac{2}{5}\log C_6 \tag{99}$$

and finally the conversion formula:

$$\log C_6 = 2.5 \log \frac{\gamma_6}{N_{\rm H}} - 11.510 \tag{100}$$

For instance a value of −7.619 from VALD gives $\log C_6 = -30.558$. Before Linfor3D Version 6.5.0, neither the parameter γ_6 nor the parameter `C6log`$= -\log C_6$ could be specified in the line data file directly. Instead the van der Waals broadening had to be specified via the difference of mean square electron orbital radii $\Delta\overline{r^2}/a_0^2$, where a_0 is the Bohr radius:

$$\log\left(\Delta\overline{r^2}/a_0^2\right) = \log C_6 + 32.3867 \quad . \tag{101}$$

The necessary relation for the conversion between $\left(\Delta\overline{r^2}/a_0^2\right)$ and γ_6 is:

$$\Delta\overline{r^2}/a_0^2 = 10^{20.877 + 2.5 \log \frac{\gamma_6}{N_{\rm H}}} \quad . \tag{102}$$

The exemplary value of −7.619 from VALD thus gives `67.437` for the parameter `drrca1`. In addition `dlgC6` should be set to 0 unless you want to apply an additional enhancement of the broadening.

Since Linfor3D Version 6.5.0, line data format '4' and '5' allows to enter directly the parameter `C6log` or $\gamma_6/N_{\rm H}$, respectively.

8.4.3 ABO van der Waals broadening formalism

In the van der Waals broadening formalism of Anstee, Barklem, and O'Mara, γ_6 is computed as

$$\frac{w}{N_{\rm H}} = \frac{\gamma_6}{2\,N_{\rm H}} = \sigma_{\rm ABO}\,a_0^2\left(\frac{4}{\pi}\right)^{\alpha_{\rm ABO}/2}\Gamma\left(2-\alpha_{\rm ABO}/2\right)\,v_0\left(\frac{v_{\rm rel}}{v_0}\right)^{1-\alpha_{\rm ABO}}\quad, \tag{103}$$

where w is the **half** half width in rad s^{-1}, $\sigma_{\rm ABO}$ and $\alpha_{\rm ABO}$ are the two tabulated quantities of the ABO line broadening theory, Γ denotes the mathematical Γ-function. The parameter $\sigma_{\rm ABO}$ is the broadening cross section at relative velocity $v_0 = 10$ km/s between the perturbing hydrogen atom and the atom of interest in atomic units. The factor a_0^2 (a_0 is the Bohr radius) converts the cross section to units of cm^2. $v_{\rm rel}$ is the mean relative velocity averaged over the Maxwellian velocity distribution as given by Eq. 82.

The parameter $\alpha_{\rm ABO}$ describes the velocity dependence of the broadening cross section

$$\sigma_{\rm ABO}(v) = \sigma_{\rm ABO}(v_0)\left(\frac{v}{v_0}\right)^{-\alpha_{\rm ABO}}\quad. \tag{104}$$

For details see, e.g., Barklem, Anstee and O'Mara, Publ. Astron. Soc. Aust., 1998, 15, 336–8.
Numerically, we obtain

$$\log\frac{\gamma_6}{N_{\rm H}} = \log\sigma_{\rm ABO} + 0.052455\alpha_{\rm ABO} + \log\Gamma\left(2-\alpha_{\rm ABO}/2\right) + (1-\alpha_{\rm ABO})\log\left(\frac{v_{\rm rel}}{v_0}\right) - 10.25177\,. \tag{105}$$

This relation may be compared to the classical van der Waals formula (Eq.94) which may be rewritten as

$$\log\frac{\gamma_6}{N_{\rm H}} = 0.4\,\log C_6 + 0.6\,\log\left(\frac{v_{\rm rel}}{v_0}\right) + 4.5074114\,. \tag{106}$$

We can convert the ABO parameters $\sigma_{\rm ABO}$ and $\alpha_{\rm ABO}$ to C_6 by requiring the two expressions (105) and (106) to yield identical results for $\gamma_6(v_{\rm rel} = v_0) = \gamma_6(T \approx 4760\,{\rm K})$:

$$\log C_6 = 2.5\,\log\sigma_{\rm ABO} + 0.1311376\,\alpha_{\rm ABO} + 2.5\,\log\Gamma\left(2-\alpha_{\rm ABO}/2\right) - 36.89795\,. \tag{107}$$

For $\sigma_{\rm ABO} = 530$, $\alpha_{\rm ABO} = 0.277$, we obtain $\log C_6 = -30.1076$.
If we choose a different reference velocity, v^*, for matching both expressions, we obtain

$$\log C_6 = 2.5\,\log\sigma_{\rm ABO} + 0.1311376\,\alpha_{\rm ABO} + 2.5\,\log\Gamma\left(2-\alpha_{\rm ABO}/2\right) + \left(1-\frac{5}{2}\,\alpha_{\rm ABO}\right)\log\frac{v^*}{v_0} - 36.89795\,. \tag{108}$$

This relation shows that, for $\alpha_{\rm ABO} = 2/5$, ABO and Linfor3D can be matched to give identical γ_6 for arbitrary temperatures. In Linfor3D we choose $v^* = 14.495$ km/s, corresponding to $T \approx 10^4$ K. Then $\log(v^*/v_0) = 0.1612$.

On the other hand, any C_6 can be uniquely converted to $\sigma_{\rm ABO}$ and $\alpha_{\rm ABO}$:

$$\log\sigma_{\rm ABO} = 0.4\,\log C_6 + 14.76906834\quad,\quad \alpha_{\rm ABO} = 2/5\,. \tag{109}$$

For example, $\log C_6 = -30.1076$ implies $\log\sigma_{\rm ABO} = 2.7260324$ or $\sigma_{\rm ABO} = 532.15$.

For use in Linfor3D, we rewrite Eq. (105) as

$$\log\frac{\gamma_6}{10^8} = \log\sigma_{\rm ABO} + \frac{1+\alpha_{\rm ABO}}{2}\,\log\theta + \log P_{\rm H} + F(\alpha_{\rm ABO})\,, \tag{110}$$

where $\theta = 5039.67/T$, $P_{\rm H} = N_{\rm H}\,k\,T$ is the partial pressure of neutral hydogen atoms, and

$$\begin{aligned} F(\alpha_{\rm ABO}) &= c_1\,\alpha_{\rm ABO} + \log\Gamma\left(2-\alpha_{\rm ABO}/2\right) - (1-\alpha_{\rm ABO})\log v_0 \\ &+ \frac{1-\alpha_{\rm ABO}}{2}\,c_2 - \frac{1+\alpha_{\rm ABO}}{2}\,c_3 + c_4\,, \end{aligned} \tag{111}$$

or

$$F(\alpha_{\rm ABO}) \;=\; \log\Gamma\,(2-\alpha_{\rm ABO}/2) + f_1\,\alpha_{\rm ABO} + f_2\,, \tag{112}$$

with the constants

$$v_0 \;=\; 10^6\ [\mathrm{cm/s}]\,, \tag{113}$$
$$a_0 \;=\; 5.2917725\,10^{-09}\,,\ \text{Bohr radius [cm]}\,, \tag{114}$$
$$c_1 \;=\; \frac{1}{2}\,\log\left(\frac{4}{\pi}\right) = 0.052455\,, \tag{115}$$
$$c_2 \;=\; \log\left(\frac{8}{\pi\,m_{\rm H}}\right) = 24.182288\,, \tag{116}$$
$$c_3 \;=\; \log\,(k\,5039.67) = -12.15750\,, \tag{117}$$
$$c_4 \;=\; \log\left(\frac{2\,v_0\,a_0^2}{10^8}\right) = -18.25177\,, \tag{118}$$
$$f_1 \;=\; c_1 - (c_2+c_3)/2 + \log(v_0) = 0.040060295\,, \tag{119}$$
$$f_2 \;=\; (c_2-c_3)/2 + c_4 - \log(v_0) = -6.0818740\,. \tag{120}$$

For $\alpha_{\rm ABO} = 2/5$ we obtain

$$\log\frac{\gamma_6}{10^8} = \log\sigma_{\rm ABO} + \frac{7}{10}\,\log\theta + \log P_{\rm H} - 6.0967212\,, \tag{121}$$

and with Eq. (109) we get

$$\log\frac{\gamma_6}{10^8} = \frac{2}{5}\,\log C_6 + \frac{7}{10}\,\log\theta + \log P_{\rm H} + 8.6723475\,, \tag{122}$$

which is the standard formula used in Linfor3D for decades.

8.4.4 Natural line broadening

The broadening parameter $\gamma_{\rm rad}$ can be converted like this:

$$C_{\rm rad} = 10^{\log\gamma_{\rm rad}-8.0} \tag{123}$$

For instance, $\log\gamma_{\rm rad} = 7.877$ would give $C_{\rm rad} = 0.753$. In line data formats '0' – '4', the parameter `Crad` is thus set to `0.753`, and `dlggr` is set to `0.0`. In line data formats '5' – '6', the parameter `grlog` is set to `7.887`.

9 Departure Coefficients

Linfor3D is able to read departure coefficients from statistical equilibrium computations and incorporate how they affect the source function, and hence the line depression and the over all line profile in a normalised spectrum.

Until early 2022, Linfor3D was only able to read departure coefficients from the 3D statistical equilibrium code, NLTE3D. These files are formatted according to the `UIO` standard, and are often called `xbc` files.

Since then, modifications were made to Linfor3D that made it possible for it to read a new type of file – an `xbc2` file. Like this file's predecessor, it is written using the `UIO` package, but the contents of these files differs. These files are created with the counterpart 1.5D statistical equilibrium wrapper, NLTE15D. This code is an MPI wrapper that can be adapted for most 1D statistical equilibrium codes, such as MULTI or DETAIL.

Linfor3D is capable – with some effort – of reading most formats containing departure coefficients. To add new formats, a new module should be added to `nlte/` and those considerations should be added to `linfor_nlte.f90`. In fact `linfor_nlte.f90` was designed so that one could easily add new modules that deal with NLTE departures.

The contents of the `xbc` and `xbc2` files are now explained.

9.1 The xbc file

Add a table of the inputs

Description of entries:		
Z-Structure:		
M10	:	First index of first horizontal dimension
N10	:	Last index of first horizontal dimension
M20	:	First index of second horizontal dimension
N20	:	Last index of second horizontal dimension
M30	:	First index of the vertical dimension
N30	:	Last index of the vertical dimension
K1SKIP	:	Skipping value of the first horizontal dimension
K2SKIP	:	Skipping value of the second horizontal dimension
K3BOT	:	Bottom boundary grid point considered for statistical equilibrium computations from NLTE3D
K3TOP	:	Top boundary grid point considered for statistical equilibrium computations from NLTE3D
NLEVEL	:	Number of levels stored in departure file
GSTLEV	:	Statistical weights of each level
EEVLEV	:	Energies of each level in electron volts
XC3	:	Vertical grid points in centimetres
TEMP	:	Temperature values at every grid point of the 3D cube in Kelvin
DNE0	:	Electron number density values at every grid point of the 3D cube
DNH0	:	Hydrogen number density values at every grid point of the 3D cube
XBCOEF	:	Departure coefficients, $n_{\mathrm{NLTE}}/n_{\mathrm{LTE}}$, for each level at every grid point of the 3D cube

9.2 The xbc2 file

Description of entries:

FILETYPE	:	File type. Set to "xbc2" in files generated by NLTE15D
Z-Structure:		
ID	:	Element ID
ANAM	:	Element code ID
M10	:	First index of first horizontal dimension
N10	:	Last index of first horizontal dimension
M20	:	First index of second horizontal dimension
N20	:	Last index of second horizontal dimension
M30	:	First index of the vertical dimension
N30	:	Last index of the vertical dimension
NLAM	:	Number of transition wavelengths stored in departure file
NLEVEL	:	Number of levels stored in departure file
K1SKIP	:	Skipping value of the first horizontal dimension
K2SKIP	:	Skipping value of the second horizontal dimension
ZSHIFT	:	First geometrical depth point over which departures are computed
ILEVEL	:	Corresponding departure indexes in xbcoef for each level
GSTLEV	:	Statistical weights of each level
EEVLEV	:	Energies of each level in electron volts
CLAM	:	Central wavelength of every transition considered in departure file
LEVIJ	:	Corresponding lower (first index) and upper (second index) departure indexes for each transition
XC1	:	First horizontal grid points in centimetres
XC2	:	Second horizontal grid points in centimetres
XC3	:	Vertical grid points in centimetres
TEMP	:	Temperature values at every grid point of the 3D cube in Kelvin
LTAUV	:	Monochromatic optical depth scale evaluated in base 10 logarithm
LTAUR	:	Rosseland optical depth scale evaluated in base 10 logarithm
XBCOEF	:	Departure coefficients, $n_{\mathrm{NLTE}}/n_{\mathrm{LTE}}$, for each level at every grid point of the 3D cube

10 Output files

Linfor3D generates the following output files in the Linfor3D working directory:

name		content
`linfor_3D_1.uiosave`	:	UIO formatted structures: `ABU, ATOM, CMD, CONST, INFO, LINE` (see Sect. 11.1 for details).
`linfor_3D_2.uiosave`	:	UIO formatted structures: `CONTF, IMUPHI, MAPS, RESULT` (see Sect. 11.2 for details).
`linfor_3D_3.uiosave`	:	UIO formatted structure: `CONTF3D` – written to file if `cc3d` flag is set to 1 in `CMD` (see Sect. 11.3 for details).
`linfor_3D_4.uiosave`	:	UIO formatted structure: `CGOUT` – written to file if `cog` flag is set to -2 or 2 in `CMD` (see Sect. 11.3 for details).
`linfor_1X.uiosave`	:	UIO formatted structures: `ABU, ATOM, CMD, CONST, INFO, LINE, CONTF, IMUPHI, RESULT` – writen to file if `run` flag is set to -3 in `CMD` (See Sect. 11.5 for details).
`linfor_timing.txt`	:	Timing statistics (see Sect. 13).
`linfor_3D_1.ps`	:	Postscript file: local line profiles plus average.
`linfor_3D_2.ps`	:	Postscript file: line profiles for 1D reference atmospheres and time-averaged 1D and 3D spectra; granulation abundance correction
`<LHD_model_name>_l50.3x3`	:	The 1D LHD model written as a 3X3 RHD box in UIO format
`<3D_model_name>_avg.3x3`	:	The $\langle$3D$\rangle$ model written as a 3X3 RHD box in UIO format

The latest versions of Linfor3D (version 6.0.0 onwards) are compatible with the CVS versions of GNU data language (GDL)[3]. To make this possible, two new routines were written to replace the intrinsic IDL I/O routines, SAVE/RESTORE, previously used by Linfor3D. Both these new routines, written by A. J. Gallagher, were written to exploit the Universal Input Output (UIO) routines, which were designed by B. Freytag for handling I/O in CO^{5}BOLD.

10.1 uio_save

The `uio_save.pro` routine is rather complex, but is nevertheless designed to work as a viable replacement to the intrinsic IDL routine, SAVE. Therefore its call is simple. At the current time, the maximum number of variables `uio_save` can save is 15. This can be extended when necessary by adding further variables into the routine, but for the purposes of Linfor3D it was not required.

It saves a binary file, which is commonly given the file format name `uiosave`. A typical call for this routine is as follows:

```
uio_save, FILE = '<filename>', variable1, variable2, variableN [, /verbose]
```

where `<filename>` is a string of the exact file name to be used; `variable1` – `variableN` are the variable names to be saved.

[3]The tarball can be downloaded at http://gnudatalanguage.cvs.sourceforge.net/ and the GDL manual can be found at http://gnudatalanguage.sourceforge.net

The uio_save.pro routine can save scalars, arrays and structures. However, at present, the UIO routines do not work with IDL pointers.

The switch verbose can be used to output several useful checks to screen, including the results of an error check, which is performed by the UIO routines throughout the save procedure. This is particularly useful for error checking one's own coding. As a simple example, the uio_save routine is used to save a scaler, two arrays and a structure and then uio_restore (see Sect. 10.2) is used to open the saved file below:

```
IDL> a = 45L & b = findgen(100) & c = dblarr(50, 100, /nozero)
IDL> d = {a:a, b:b, c:c}
IDL> uio_save, FILE = 'example.uiosave', a, b, c, d, /verbose
% UIO_SAVE: Writing A vector to file
% UIO_SAVE: Write of A successful
% UIO_SAVE: Writing B vector to file
% UIO_SAVE: Write of B successful
% UIO_SAVE: Writing C vector to file
% UIO_SAVE: Write of C successful
% UIO_SAVE: Writing D structure to file
% UIO_SAVE: Write of D successful
% UIO_SAVE: Closing file and checking...
% UIO_SAVE: Data has been successfully written to file
% UIO_SAVE: Write status: done
IDL> .reset ; reset the session and delete variable(s)
IDL> uio_restore, 'example.uiosave', /verbose
% UIO_RESTORE: Restoring structure A
% UIO_RESTORE: Restoring structure B
% UIO_RESTORE: Restoring structure C
% UIO_RESTORE: Restoring structure D
IDL> help
% At $MAIN$
A               LONG     =          45
B               FLOAT    = Array[100]
C               DOUBLE   = Array[50, 100]
D               STRUCT   = -> <Anonymous> Array[1]
```

The routine calls upon the following sub-routines from the UIO database directly:

Routine		Description
uio_filedefinc.pro	:	Parameter definitions for standard file descriptors and labels
uio_uionaminc.pro	:	Common block that contains parameters for UIO initialisation routines
uio_init.pro	:	Initialisation procedure for UIO routine package.
uio_wr.pro	:	Writes scalar or array data to file.
uio_wrlabl.pro	:	Writes a label for structures or datasets.
uio_openwr.pro	:	Opens a file for writing and writes the data block header.
uio_closwr.pro	:	Closes a file after writing.

Each of these sub-routines call on several other sub-routines within the UIO routine package.

A very simple example of how to use these sub-routines to write a basic structure to file in IDL or GDL (without using uio_save.pro) is given with step-by-step annotations:

Create a structure, C, with arrays A and B:

```
IDL> a = findgen(100) & b = fltarr(20, 50) & c = {a:a, b:b}
```

Initialise the UIO procedures and common blocks:

```
IDL> @uio_filedefinc
IDL> @uio_uionaminc
IDL> uio_init, progrm = 'example_save'
```

Open a binary file (`form = 'unformatted'`) called `test.uiosave` and use the default conversion type (`conv = 'ieee_4'`):

```
IDL> uio_openwr, nc, 'test.uiosave', outstr, ierr, $
IDL> form = 'unformatted', conv = 'ieee_4', prog = 'example_save'
```

Write the name of the dataset to file for the binary file header using special definition `dataset_ident`:

```
IDL> uio_wrlabl, nc, dataset_ident, outstr, ierr, date = 'now', $
IDL> name = 'test.uiosave'
```

Write the structure name, `C`, to file using special definition `box_ident`:

```
IDL> uio_wrlabl, nc, box_ident, outstr, ierr, date = 'now', name = 'c'
```

Begin the write of the C structure to file by declaring the box ID name as `C` using special definition `box_id_ident`:

```
IDL> uio_wr, nc, 'C', box_id_ident, name = 'C structure'
```

Write the contents of structure `C` to file:

```
IDL> uio_wr, nc, c.a, 'A', outstr, ierr, name = 'c.A'
IDL> uio_wr, nc, c.b, 'B', outstr, ierr, name = 'c.B'
```

Declare the end of the structure write using special definition `endbox_ident`:

```
IDL> uio_wrlabl, nc, endbox_ident, outstr, ierr
```

Declare the end of the dataset write using special definition `enddataset_ident`:

```
IDL> uio_wrlabl, nc, enddataset_ident, outstr, ierr
```

Close the file for writing

```
IDL> uio_closwr, nc, outstr, ierr
```

The `uio_save.pro` routine and other sub-routines within the Linfor3D routine list use this basic principle to write structures to file. A similar (though not as complex) set of procedures are used when writing arrays or scalars to file:

Create two arrays, A and B, and a scalar, C:

```
IDL> a = findgen(100) & b = fltarr(20, 50) & c = 55L
```

Initialise the UIO procedures and common blocks:

```
IDL> @uio_filedefinc
IDL> @uio_uionaminc
IDL> uio_init, progrm = 'example_save'
```

Open a binary file (`form = 'unformatted'`) called `test.uiosave` and use the default conversion type (`conv = 'ieee_4'`):

```
IDL> uio_openwr, nc, 'test.uiosave', outstr, ierr, $
IDL> form = 'unformatted', conv = 'ieee_4', prog = 'example_save'
```

Write the name of the dataset to file for the binary file header using special definition `dataset_ident`:

```
IDL> uio_wrlabl, nc, dataset_ident, outstr, ierr, date = 'now', $
IDL> name = 'test.uiosave'
```

Write the arrays/scalars to file:

```
IDL> uio_wr, nc, a, 'A', outstr, ierr, name = 'A'
IDL> uio_wr, nc, b, 'B', outstr, ierr, name = 'B'
IDL> uio_wr, nc, c, 'C', outstr, ierr, name = 'C'
```

Declare the end of the dataset write using special definition `enddataset_ident`:

```
IDL> uio_wrlabl, nc, enddataset_ident, outstr, ierr
```

Close the file for writing

```
IDL> uio_closwr, nc, outstr, ierr
```

10.2 uio_restore

The `uio_restore.pro` routine is a wrapper designed around the high level IDL function `uio_dataset_rd.pro` to read a UIO formatted binary or ASCII file and return the output to the call level within IDL or GDL. The call procedure for this wrapper is identical to that of the intrinsic RESTORE procedure in IDL, i.e.:

```
uio_restore, '<filename>' [, variable1, variable2, ..., variableN [, /verbose]]
```

where variables 1–N are optional, but useful where computer memory is limited. An example of its use:

```
IDL> uio_restore, 'linfor_3D_1.uiosave', /verbose
% UIO_RESTORE: Restoring structure ABU
% UIO_RESTORE: Restoring structure ATOM
% UIO_RESTORE: Restoring structure CMD
% UIO_RESTORE: Restoring structure CONST
% UIO_RESTORE: Restoring structure LINE
% UIO_RESTORE: Restoring structure INFO
IDL> help
% At $MAIN$
ABU             STRUCT    = -> <Anonymous> Array[1]
ATOM            STRUCT    = -> <Anonymous> Array[1]
CMD             STRUCT    = -> <Anonymous> Array[1]
CONST           STRUCT    = -> <Anonymous> Array[1]
INFO            STRUCT    = -> <Anonymous> Array[1]
LINE            STRUCT    = -> <Anonymous> Array[1]
```

The user can specify what data should be restored by adding additional command(s) to the call:

```
IDL> uio_restore, 'linfor_3D_1.uiosave', CMD, LINE, /verbose
% UIO_RESTORE: Restoring structure CMD
% UIO_RESTORE: Restoring structure LINE
IDL> help
% At $MAIN$
CMD             STRUCT    = -> <Anonymous> Array[1]
LINE            STRUCT    = -> <Anonymous> Array[1]
```

Additionally, this routine will open all CO^{5}BOLD model atmospheres and is useful when a single piece of information (e.g. the model time) is required. It also means that for the first time, the user has a choice of computer languages (Fortran/IDL/GDL) to do their analysis without the need for any conversion of the output file. Further details of the routine's use can be found in the header of `uio_restore.pro`, which is located in the Routines sub-directory of Linfor3D.

10.3 Useful UIO information

The UIO routines allow the user to restore arrays with up to four dimensions, as modifying the UIO routines for use with Fortran so that more than four dimensions can be read is not a trivial matter. In its current form, the UIO routines will successfully save an array with more than four dimensions:

```
IDL> a = fltarr(10, 10, 10, 10, 10, 10, /nozero)
IDL> uio_save, file = 'example.uiosave', a, /verbose
% UIO_SAVE: Writing A vector to file
% UIO_SAVE: Write of A successful
% UIO_SAVE: Closing file and checking...
% UIO_SAVE: Data has been successfully written to file
% UIO_SAVE: Write status: done
```

however, the routines will not allow you to open the file afterwards:

```
IDL> uio_restore, 'example.uiosave'
% Attempt to subscript SARR with NDIM is out of range.
% Execution halted at: UIO_ST2DIM   86
   /data/Linfor/uio/uio_st2dim.pro
%                  UIO_RD         140
   /data/Linfor/uio/uio_rd.pro
%                  UIO_STRUCT_RD  333
   /data/Linfor/uio/uio_struct_rd.pro
%                  UIO_DATASET_RD 150
   /data/Linfor/uio/uio_dataset_rd.pro
%                  UIO_RESTORE    134
   /data/Linfor/Linfor_6_0_2/Routines/uio_restore.pro
%                  $MAIN$
```

It is shown that the restore procedure fails during the `uio_st2dim.pro` sub-routine call. If one only wishes to work in IDL or GDL, and has little interest in working under Fortran, there is a very simple modification that can be added to the UIO routines so that an array with more than four dimensions can be saved and successfully restored under the UIO convention. At line 69 in the routine `uio_st2dim.pro`, the following is seen `sarr=strarr(2,4)`, where 4 represents the maximum number of dimensions that the UIO routines (in IDL and GDL) can load. The user can simply replace 4 with a higher number so that larger dimension arrays can be successfully restored using the UIO routines. However, it must be stressed that any alteration to this routine will only affect any file opened in IDL and GDL, not in Fortran. Indeed, any attempt to open these larger dimension arrays in Fortran will result in a read failure.

For further details on the UIO repository, as well as some other examples, please consult the CO^{5}BOLD manual[4], (Sect. 4).

[4]Downloadable at http://www.astro.uu.se/~bf/co5bold_main.html.

11 Output file structures

The binary files saved by Linfor3D contain several structures. In this section, a brief description of each array in every output structure is given.

11.1 linfor_3D_1.uiosave

The UIO formatted output file `linfor_3D_1.uiosave` contains the following structures:

11.1.1 ABU

The ABU structure contains information on the input file, `<ABUFILE>.abu`, found in the `Data` sub-directory of Linfor3D, where `<ABUFILE>` is either `kiel`, `cifist2006` or `special`. It is created after the successful initialisation of the routine `ionopa.pro`.

Description of entries:		
NAMI	:	Column 1 from `abuid`.
ABUI	:	Column 2 from `abuid`.
NAMIX	:	Column 1 from `abuidx` (version 6.2.2 onwards). See Sect. 7.5
ABUIX	:	Column 2 from `abuidx` (version 6.2.2 onwards). See Sect. 7.5

11.1.2 ATOM

The ATOM structure contains information on the input file, `atom.dat`, found in the `Data` sub-directory of Linfor3D, e.g. `1201.24` corresponds to Mg II 24, `1201.25` corresponds to Mg II 25, etc.

Description of entries:		
NIONS	:	Number of species included in `atom.dat`
ANAM	:	Linfor3D formatted atoms, ions and molecules. (Column 1 of `atom.dat`)
WTJ	:	Corresponding baryon masses of ANAM. (Column 2 of `atom.dat`)
CHIJ	:	Corresponding χ (eV) energies of ANAM (Column 3 of `atom.dat`)
FISO	:	Corresponding isotope fractions of ANAM. Usually = 1.0, unless the isotopes are considered. (Column 4 of `atom.dat`)

This file can be edited before running Linfor3D to alter, for example, isotope fractions. However '`line.dat`' should be properly formatted to reflect the changes.

11.1.3 CMD

This structure contains inputs defined by the user in the `linfor_setcmd.pro` routine (plus additional parameters defined by `linfor_checkcmd.pro`). See Sect. 7 for details.

This input structure routine can be defined and compiled in IDL/GDL before running Linfor3D by writing a BASH/TCSH script to produce this file using the native EOS procedure. This is usually done to run Linfor3D when several sessions need to be computed at the same time.

11.1.4 CONST

This structure contains a set of constants used by Linfor3D throughout the synthesis.

Description of entries:

AVMEOS	:	Average mass of heavy particles in EOS.
AVMION	:	Average mass of heavy particles in IONDIS of the model.
EPSHE	:	Helium number density of the model.
FRACH	:	Hydrogen mass fraction of the model.
AVMIONX	:	Average mass of heavy particles in IONDIS of the spectrum.
EPSHEX	:	Helium number density of the spectrum.
FRACHX	:	Hydrogen mass fraction of the spectrum.
LUTAU1	:	Smallest $\log \tau_{\rm ROSS}$ covered by sub-model (refined z-grid) – set by user in `linfor_setcmd.pro`
LUTAU2	:	Largest $\log \tau_{\rm ROSS}$ covered by sub-model (refined z-grid) – set by user in `linfor_setcmd.pro`
LUTAU	:	Array of $\log \tau_{\rm ROSS}$ values covered by sub-model (refined z-grid)
UTAU	:	Array of $\tau_{\rm ROSS}$ values covered by sub-model (refined z-grid)
NUTAU	:	Number of points in `LUTAU`
LCTAU1	:	Smallest $\log \tau_0$ used for RT integration – set by user in `linfor_setcmd.pro`
LCTAU2	:	Largest $\log \tau_0$ used for RT integration – set by user in `linfor_setcmd.pro`
LCTAU	:	Array of $\log \tau_0$ points used for the RT integration
CTAU	:	Array of τ_0 points used for the RT integration
NCTAU	:	Number of points in `LCTAU`
ICG	:	Index number of points over which the Curve-of-Growth is computed. Default: 51. Can be set in CMD structure by user.
DLGF_CG	:	$\Delta \log gf$ index used to compute the Curve-of-Growth. By default this is set at 51 points between $-0.5 \leq \Delta \log gf \leq +1.0$, but the user can extend this if the CoG control parameters are set in the CMD structure (see CMD.COG).
IMT	:	The index number of microturbulence values over which to compute the Curve-of-Growth. If CMD.MICRO = 0 then IMT = 1.
XIMC_MTX	:	The range of microturbulence values over which to compute the Curve-of-Growth using the 1D external atmosphere.
XIMC_MT1	:	The range of microturbulence values over which to compute the Curve-of-Growth using the average 3D atmosphere.
MLIST	:	String array of CO5BOLD full files used during the spectrum synthesis run.
NFILE	:	Number of model files for which the spectrum synthesis was done
NDATA	:	Number of snapshots for which spectrum synthesis was done
WALFA0	:	Switch for the `ALFA` parameter, which is related to the upper boundary condition (0/1). `ALFA` $= H_{\rm P_0}/H_{\tau_0} - 1$, where $H_{\rm P_0}$ and H_{τ_0} is the pressure scale height and the optical depth scale height, respectively, at the upper boundary. If `WALFA0` = 1, $H_{\tau_0} = f \times H_{\rm P_0}$, where f is $-H_{\tau_0}$ (if $-10 < H_{\tau_0} < 0$).
WALFA1	:	Switch for the `ALFA` parameter, which is related to the upper boundary condition (0/1). `ALFA` $= H_{\rm P_0}/H_{\tau_0} - 1$, where $H_{\rm P_0}$ and H_{τ_0} is the pressure scale height and the optical depth scale height, respectively, at the upper boundary. If `WALFA1` = 1, $H_{\tau_0} = f \times H_{\rm P_0}$, where f is $-H_{\tau_0}$ (if $H_{\tau_0} > 0$).
WALFA2	:	Switch for the `ALFA` parameter, which is related to the upper boundary condition (0/1). `ALFA` $= H_{\rm P_0}/H_{\tau_0} - 1$, where $H_{\rm P_0}$ and H_{τ_0} is the pressure scale height and the optical depth scale height, respectively, at the upper boundary. If `WALFA2` = 1, $H_{\tau_0} = f \times H_{\rm P_0}$, where f is $-H_{\tau_0}$ (if $H_{\tau_0} \leq -10$).
YR1	:	Plot ranges used by `linfor_plot0.pro`.(IDL version only)
YR2	:	Plot ranges used by `linfor_plot0.pro`.(IDL version only)
YR3	:	Plot ranges used by `linfor_plot0.pro`.(IDL version only)
YR4	:	Plot ranges used by `linfor_plot0.pro`.(IDL version only)
YR5	:	Plot ranges used by `linfor_plot0.pro`.(IDL version only)
YR6	:	Plot ranges used by `linfor_plot0.pro`.(IDL version only)

NLTE_TYPE3	:	2D array [NDATA,KTOTAL] of identifier strings giving the type of departure coefficient used for each component of the input lines for every 3D snapshot.
NLTE_TYPEX	:	1D array [KTOTAL] of identifier strings giving the type of departure coefficient used for each component of the input lines for the external 1D model atmosphere.

11.1.5 INFO

The INFO structure contains information about the machine that Linfor3D was run on.

Description of entries:

VERSION	:	Version of Linfor3D used for synthesis run
DATE	:	Date of synthesis run
MACHINE	:	Machine used for synthesis run (Sometimes missing from INFO structure)

11.1.6 LINE

The LINE structure contains wavelength information and line parameters on the synthesis. This structure is typically used to reconstruct the equivalent wavelength array for use with the RESULT and IMUPHI structures.

Description of entries:

KLINE	:	Number of line calculations done by the synthesis
KTOTAL	:	Total number of spectral lines including blends
K1	:	Array containing corresponding index numbers of each line or blend in kline
NBLEND	:	Array containing number of blends for each line synthesis
CLAM	:	Central wavelength, λ_0, and continuum wavelength in Å, set in line file
DLAM	:	Line profile, $\Delta\lambda$, in Å computed from $\lambda_0 - \Delta\lambda$ to $\lambda_0 + \Delta\lambda$
DDLAM	:	Spacing of wavelength points for spectrum synthesis
WLAM0	:	Requested equivalent width from line file. (set to '`0`' unless user includes `W0` in `line.dat`, see Sect. 8.3.2).
LINEID	:	Header from the line file
LFLAG	:	Control string set to 'cont' (continuum synthesis) or 'line' (line synthesis) by contents of line file.
MULT	:	Integer array identifying the multiplet number of the lines synthesised
ANAM	:	The atomic number of lines included in the synthesis (multiplied by 100)
WTJ	:	Array of baryon masses for every transition considered during spectrum synthesis (taken from `atom.dat`).
CHIJ	:	Array of χ values of lower energy in eV for every transition considered during synthesis (taken from `[cifist2006, special, kiel].abu` input file).
FISO	:	Array of isotope fractions for every transition considered during spectrum synthesis (taken from `atom.dat`).
CHIK	:	Array containing χ values of upper energy in eV
ALAM	:	Central wavelength of line or blend component
GFLG	:	Array containing $\log gf$ values of lines synthesised
C6LOG	:	Array containing $\log C_6$ values of lines synthesised
DLGC6	:	Array containing $\Delta \log C_6$ values of lines synthesised
DRRCA1	:	Difference of mean square electron orbital radii, $\Delta\overline{r^2}/a_0^2$

C4LOG	:	Array containing $\log C_4$ values of lines synthesised
DLGC4	:	Array containing $\Delta \log C_4$ values of lines synthesised
GRAD8	:	Natural line broadening parameter, $\gamma_{\rm rad}$ of `KLINE`.
DLGGR	:	Array of mean square electron orbital radii differences ($\Delta\overline{r^2}/a_0^2$)
VDOP	:	Doppler width in units of the speed of light.
ETA0	:	$\eta_0 = \kappa_l/\kappa_c$. See Sect. 5.5 and Fig. 2
AVGT	:	Damping parameter "a" for Voigt profile.
ILOWER[3,X]	:	Lower level index from NLTE departure files (XBC)
IUPPER[3,X]	:	Upper level index from NLTE departure files (XBC)
XBCFIL3	:	String array containing XBC information for the 3D synthesis (set to "LTE" if no XBC is used)
XBCFILX	:	String array containing XBC information for the 1D synthesis (set to "LTE" if no XBC is used)
XCFLAG	:	Set to 'grey': Continuum source funtion was set to wavelength-integrated Planck-Function, $S = \sigma T^4/\pi$ and continuum opacity is set to Rosseland mean opacitiy, $\kappa_0 = \kappa_{\rm ROSS}$. Set to 'mono': spectrum synthesis was computed as normal.

11.2 linfor_3D_2.uiosave

The UIO formatted output file `linfor_3D_2.uiosave` contains the following:

11.2.1 CONTF

The arrays found in this structure relate to the contribution functions calculated by Linfor3D. See Sect. 5.4 for the formal derivations.

Description of entries:

NZX	:	Array containing resultant sampling considered during external 1D model synthesis, redefined by `lctau1` and `lctau2` set in `linfor_setcmd.pro`
ZZX	:	Vertical geometrical ray scale for the 1D external model.
CCX	:	Continuum intensity contribution functions of the external 1D model for "vertical" rays on the geometrical scale, `ZZX`.
NZ1	:	Array containing resultant sampling considered during $\langle 3D \rangle \log \tau_{\rm ROSS}$ synthesis, redefined by `lctau1` and `lctau2` set in `linfor_setcmd.pro`
ZZ1	:	Vertical geometrical ray scale for the $\langle 3D \rangle$ model.
CC1	:	Continuum intensity contribution functions of the $\langle 3D \rangle$ model for "vertical" rays on the geometrical scale, `ZZ1`.
NZ3	:	Array containing resultant sampling considered during 3D synthesis, redefined by `lctau1` and `lctau2` set in `linfor_setcmd.pro`
ZZ3	:	Vertical geometrical ray scale for the 3D model.
CC3	:	Continuum intensity contribution functions of the 3D model for "vertical" rays on the geometrical scale, `ZZ3`.
LTAUC	:	Array of $\log \tau_0$ (continuum optical depth) points
LTAURX	:	Array of 1D $\log \tau_{\rm ROSS}$ (Rosseland optical depth) points
DTRTCX	:	$d \log \tau_{\rm ROSS}/d \log \tau_0$ for the external 1D model used by `linfor_cf2cr.pro`.
LTAUR1	:	Array of $\langle 3D \rangle \log \tau_{\rm ROSS}$ points.
DTRTC1	:	$d \log \tau_{\rm ROSS}/d \log \tau_0$ for the $\langle 3D \rangle$ model used by `linfor_cf2cr.pro`.
LTAUR3	:	Array of 3D $\log \tau_{\rm ROSS}$ points
DTRTC3	:	$d \log \tau_{\rm ROSS}/d \log \tau_0$ for the 3D model used by `linfor_cf2cr.pro`.
CFCXI	:	Array containing the 1D *Continuum Intensity Contribution Function*, C_I^c, evaluated over a $\log \tau_0$ scale

CFC1I : Array containing the ⟨3D⟩ *Continuum Intensity Contribution Function*, C_I^c, evaluated over a log τ_0 scale

CFC3I : Array containing the 3D *Continuum Intensity Contribution Function*, C_I^c, evaluated over a log τ_0 scale, see Eq. (42).

CFLXI : Array containing the 1D *Line Intensity Contribution Function*, C_I^l, evaluated over a log τ_0 scale

CFL1I : Array containing the ⟨3D⟩ *Line Intensity Contribution Function*, C_I^l, evaluated over a log τ_0 scale

CFL3I : Array containing the 3D *Line Intensity Contribution Function*, C_I^l, evaluated over a log τ_0 scale, see Eq. (46).

CFDXI : Array containing the 1D *Line Intensity Depression Contribution Function*, $\tilde{C}_I^D = C_I^c - C_I^l$, evaluated over a log τ_0 scale

CFD1I : Array containing the ⟨3D⟩ *Line Intensity Depression Contribution Function*, $\tilde{C}_I^D = C_I^c - C_I^l$, evaluated over a log τ_0 scale

CFD3I : Array containing the 3D *Line Intensity Depression Contribution Function*, $\tilde{C}_I^D = C_I^c - C_I^l$, evaluated over a log τ_0 scale, see Eq. (50).

CFWXI : Array containing the 1D *Equivalent Width Intensity Contribution Function*, C_I^W, evaluated over a log τ_0 scale

CFW1I : Array containing the ⟨3D⟩ *Equivalent Width Intensity Contribution Function*, C_I^W, evaluated over a log τ_0 scale

CFW3I : Array containing the 3D *Equivalent Width Intensity Contribution Function*, C_I^W, evaluated over a log $\tau_{\rm cont}$ scale, see Eq. (56).

CFCXF : Array containing the 1D *Continuum Flux Contribution Function*, C_F^c, evaluated over a log τ_0 scale

CFC1F : Array containing the ⟨3D⟩ *Continuum Flux Contribution Function*, C_F^c, evaluated over a log τ_0 scale

CFC3F : Array containing the 3D *Continuum Flux Contribution Function*, C_F^c, evaluated over a log τ_0 scale, see Eq. (44).

CFLXF : Array containing the 1D *Line Flux Contribution Function*, C_F^l, evaluated over a log τ_0 scale

CFL1F : Array containing the ⟨3D⟩ *Line Flux Contribution Function*, C_F^l, evaluated over a log τ_0 scale

CFL3F : Array containing the 3D *Line Flux Contribution Function*, C_F^l, evaluated over a log τ_0 scale, see Eq. (48).

CFDXF : Array containing the 1D *Line Flux Depression Contribution Function*, C_F^D, evaluated over a log τ_0 scale

CFD1F : Array containing the ⟨3D⟩ *Line Flux Depression Contribution Function*, C_F^D, evaluated over a log τ_0 scale

CFD3F : Array containing the 3D *Line Flux Depression Contribution Function*, C_F^D, evaluated over a log τ_0 scale, see Eq. (53).

CFWXF : Array containing the 1D *Equivalent Width Flux Contribution Function*, C_F^W, evaluated over a log τ_0 scale

CFW1F : Array containing the ⟨3D⟩ *Equivalent Width Flux Contribution Function*, C_F^W, evaluated over a log τ_0 scale

CFW3F : Array containing the 3D *Equivalent Width Flux Contribution Function*, C_F^W, evaluated over a log $\tau_{\rm cont}$ scale, see Eq. (59).

CRCXI : Array containing the 1D *Continuum Intensity Contribution Function*, C_I^c, evaluated over a log $\tau_{\rm ROSS}$ scale

CRC1I : Array containing the ⟨3D⟩ *Continuum Intensity Contribution Function*, C_I^c, evaluated over a log $\tau_{\rm ROSS}$ scale

CRC3I : Array containing the 3D *Continuum Intensity Contribution Function*, C_I^c, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (42).

CRLXI	:	Array containing the 1D *Line Intensity Contribution Function*, C_I^l, evaluated over a log $\tau_{\rm ROSS}$ scale
CRL1I	:	Array containing the ⟨3D⟩ *Line Intensity Contribution Function*, C_I^l, evaluated over a log $\tau_{\rm ROSS}$ scale
CRL3I	:	Array containing the 3D *Line Intensity Contribution Function*, C_I^l, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (46).
CRDXI	:	Array containing the 1D *Line Intensity Depression Contribution Function*, $\tilde{C}_I^D = C_I^c - C_I^l$, evaluated over a log $\tau_{\rm ROSS}$ scale
CRD1I	:	Array containing the ⟨3D⟩ *Line Intensity Depression Contribution Function*, $\tilde{C}_I^D = C_I^c - C_I^l$, evaluated over a log $\tau_{\rm ROSS}$ scale
CRD3I	:	Array containing the 3D *Line Intensity Depression Contribution Function*, $\tilde{C}_I^D = C_I^c - C_I^l$, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (50).
CRWXI	:	Array containing the 1D *Equivalent Width Intensity Contribution Function*, C_I^W, evaluated over a log $\tau_{\rm ROSS}$ scale
CRW1I	:	Array containing the ⟨3D⟩ *Equivalent Width Intensity Contribution Function*, C_I^W, evaluated over a log $\tau_{\rm ROSS}$ scale
CRW3I	:	Array containing the 3D *Equivalent Width Intensity Contribution Function*, C_I^W, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (56).
CRCXF	:	Array containing the 1D *Continuum Flux Contribution Function*, C_F^c, evaluated over a log $\tau_{\rm ROSS}$ scale
CRC1F	:	Array containing the ⟨3D⟩ *Continuum Flux Contribution Function*, C_F^c, evaluated over a log $\tau_{\rm ROSS}$ scale
CRC3F	:	Array containing the 3D *Continuum Flux Contribution Function*, C_F^c, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (44).
CRLXF	:	Array containing the 1D *Line Flux Contribution Function*, C_F^l, evaluated over a log $\tau_{\rm ROSS}$ scale
CRL1F	:	Array containing the ⟨3D⟩ *Line Flux Contribution Function*, C_F^l, evaluated over a log $\tau_{\rm ROSS}$ scale
CRL3F	:	Array containing the 3D *Line Flux Contribution Function*, C_F^l, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (48).
CRDXF	:	Array containing the 1D *Line Flux Depression Contribution Function*, C_F^D, evaluated over a log $\tau_{\rm ROSS}$ scale
CRD1F	:	Array containing the ⟨3D⟩ *Line Flux Depression Contribution Function*, C_F^D, evaluated over a log $\tau_{\rm ROSS}$ scale
CRD3F	:	Array containing the 3D *Line Flux Depression Contribution Function*, C_F^D, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (53).
CRWXF	:	Array containing the 1D *Equivalent Width Flux Contribution Function*, C_F^W, evaluated over a log $\tau_{\rm ROSS}$ scale
CRW1F	:	Array containing the ⟨3D⟩ *Equivalent Width Flux Contribution Function*, C_F^W, evaluated over a log $\tau_{\rm ROSS}$ scale
CRW3F	:	Array containing the 3D *Equivalent Width Flux Contribution Function*, C_F^W, evaluated over a log $\tau_{\rm ROSS}$ scale, see Eq. (59).

11.2.2 IMUPHI

This structure contains selective information from the RESULT structure (as well as information on ray angles). This structure is used in conjunction with several post-processing routines, such as `linfor_rotate.pro`.

Description of entries:

NDATA	:	Number of snapshots for which spectrum synthesis was done
KLINE	:	Number of lines for which spectrum synthesis was done
NLAMX	:	Total number of wavelength and flux points in calculated in the synthesis
NMUPHI	:	Number of μ and ϕ angles used in the synthesis
MODELIDX	:	Name of the external 1D model atmosphere
MODELID3	:	String array containing the name of snapshot, the x and y sampling and snapshot time in seconds
MODELID1	:	String array containing the name of average model snapshot and snapshot time in seconds
DV3	:	Array containing a velocity-spaced wavelengths, $\left(\frac{\lambda-\lambda_0}{\lambda_0}\right) c_0$
MU	:	Array containing ray inclination angles, $\mu = \cos\theta$
PHI	:	Array containing azimuthal angles, ϕ
XMU	:	An extended array of inclination angles, conformal with `NMUPHI` dimension of other arrays – allows for a simple way to perform the flux integration; $\int \mathtt{I3}\mu\, d\mu = \sum(\mathtt{I3} * \mathtt{XMU} * \mathtt{WTS})$
XPHI	:	An extended array of azimuthal angles, conformal with `NMUPHI` dimension of other arrays – allows for a simple way to perform the flux integration; $\int \mathtt{I3}\phi\, d\phi = \sum(\mathtt{I3} * \mathtt{XPHI} * \mathtt{WTS})$
WTS	:	Weightings used for μ and ϕ angle quadratures
I1	:	Array of ⟨3D⟩ fluxes evaluated over NMUPHI angles: `[NLAMX, NMUPHI, NDATA, KLINE]`
D1	:	Array of ⟨3D⟩ line depression fluxes evaluated over NMUPHI angles: `[NLAMX, NMUPHI, NDATA, KLINE]`
I3	:	Array of 3D fluxes evaluated over NMUPHI angles: `[NLAMX, NMUPHI, NDATA, KLINE]`
D3	:	Array of 3D line depression fluxes evaluated over NMUPHI angles: `[NLAMX, NMUPHI, NDATA, KLINE]`
IX	:	Array of 1D fluxes evaluated over NMUPHI angles: `[NLAMX, NMUPHI, NDATA, KLINE]`
DX	:	Array of 1D line depression fluxes evaluated over NMUPHI angles: `[NLAMX, NMUPHI, NDATA, KLINE]`

11.2.3 MAPS

The output file `linfor_3D_2.uiosave` contains a structure `MAPS`. An example of this structure is:

```
** Structure <83027f4>, 11 tags, length=11289820, data length=11289820, refs=1:
   NX              LONG                140
   NY              LONG                140
   NDATA           INT              12
   KLINE           INT               1
   NLAM            LONG                 11
   MODELID         STRING    Array[12]
   MU0             FLOAT          1.00000
   PHI0            FLOAT          0.00000
   CLAM            FLOAT          3966.34
   LINEID          STRING    Array[1]
   DV3             FLOAT     Array[11]
   ICLAM0          FLOAT     Array[140, 140, 12]
```

```
ICLAM2          FLOAT     Array[140, 140, 11, 1, 12]
```

Depending on the value of the control parameter `maps_flag` (see Sect. 7.3), there might be a tag named `ICLAM1`

```
ICLAM1          FLOAT     Array[140, 140, 12, 1]
```

instead of `ICLAM2` or even both might be missing if `maps_flag` = 0.

Description of entries:		
`nx`, `ny`	:	x,y dimensions of the 2D images
`ndata`	:	number of models for which spectrum synthesis was done
`kline`	:	number of lines for which spectrum synthesis was done
`clam`	:	central continuum wavelength for all maps
`modelid`	:	model identifier (0:ndata-1)
`lineid`	:	line identifier (0:kline-1)
`ICLAM0`	:	continuum intensity maps for all models (at clam), dimensions: `nx`, `ny` (see Sect. 8)
`ICLAM1`	:	emergent intensity maps for all models and lines, including line absorption at clam (window center); dimensions: `nx`, `ny`, `ndata`, `kline`; only present if `maps_flag` = 1 (see Sect. 8)
`ICLAM2`	:	emergent intensity maps for all models and lines, including line absorption at all wavelengths within the wavelength window of width 2 · `dlam` around the central wavelength `clam`: λ_i = `clam` − `dlam` + i · `ddlam`, where `clam` = `alam` + `dclam`, `alam` is the wavelength of the main blend component as defined in `line.dat`; dimensions: `nx`, `ny`, `nlam`, `kline`, `ndata`; only present if `maps_flag` = 2 (see Sect. 8)
`W3LAM`	:	equivalent width maps for all models and lines

Note:

- Intensities are given in units of [erg cm^{-2} s^{-1} sr^{-1} Å^{-1}].
- The file formerly called ”linfor_3D.idlsave” was renamed to ”linfor_3D_1.idlsave”.

Note that the maps include foreshortening effects. A model with a quadratic cross section becomes a rectangle when viewed off-center.

If `ntheta`≠ 0, the flux spectrum is computed as before, and the intensity maps show the vertical view, as before.

Keyword `view` added to plotting routine `linfor_plot3`. If given, the intensity and equivalent width maps show the foreshortened view.

11.2.4 RESULT

This structure contains all results from the radiative transfer done by Linfor3D, as well as some other useful information.

Description of entries:

NDATA	:	Number of snapshots for which spectrum synthesis was done
KLINE	:	Number of lines for which spectrum synthesis was done
KTOTAL	:	The total number of spectral lines including blends
NLAMX	:	Total number of wavelength and flux points in calculated in the synthesis
MU0	:	Scalar containing first μ angle
PHI0	:	Scalar containing first ϕ angle
STRMU0	:	String of `MU0`
MODELIDX	:	Name of the `LHD` model atmosphere
MODELID3	:	String array containing the name of snapshot, the x and y sampling and snapshot time in seconds
MODELID1	:	String array containing the name of average model snapshot and snapshot time in seconds
GRIDID	:	String array containing the sampling size of the synthesis
LINEID	:	String array containing the headers in the line file
LFLAG	:	Control string set to 'cont' (continuum synthesis) or 'line' (line synthesis) by contents of line file.
VFACX	:	The x-component of the hydrodynamical velocity field of the 2D/3D models is multiplied by this factor. Set in `CMD`.
VFACY	:	The y-component of the hydrodynamical velocity field of the 2D/3D models is multiplied by this factor. Set in `CMD`.
VFACZ	:	The z-component of the hydrodynamical velocity field of the 2D/3D models is multiplied by this factor. Set in `CMD`.
NL3	:	Number of wavelength and flux points used for every line synthesised
DV3	:	Array containing a velocity-spaced wavelengths
XIMICX	:	Array containing the microturbulences of the 1D synthesis
XIMIC1	:	Array containing the microturbulences of the $\langle$3D$\rangle$ synthesis
XIMIC3	:	Array containing the microturbulences of the 3D synthesis
GFLG0X	:	If the user sets an equivalent width '`W0`' (stored in LINE.WLAM0) in the line file (see Sects. 8.3.2 & 8.3.3) then this array will contain the resulting $\log gf$ value(s) necessary to compute the line or blend of that set strength from the given external 1D model atmosphere.
GFLG01	:	If the user sets an equivalent width value in the line file (see Sects. 8.3.2 & 8.3.3) then this array will contain the resulting $\log gf$ value(s) necessary to compute the line or blend of that set strength from the $\langle$3D$\rangle$ model atmosphere.
FX	:	Structure containing arrays of 1D fluxes fluxes (F) and intensities (I)
DX	:	Structure containing arrays of 1D line depression fluxes (F) and intensities (I)
WX	:	Structure containing arrays of 1D equivalent widths for the absolute line depression (D) and intensity (I)
F1	:	Structure containing arrays of $\langle$3D$\rangle$ fluxes fluxes (F) and intensities (I)
D1	:	Structure containing arrays of $\langle$3D$\rangle$ line depression fluxes (F) and intensities (I)
W1	:	Structure containing arrays of $\langle$3D$\rangle$ equivalent widths for the absolute line depression (D) and intensity (I)
F3	:	Structure containing arrays of 3D fluxes fluxes (F) and intensities (I)
D3	:	Structure containing arrays of 3D line depression fluxes (F) and intensities (I)
W3	:	Structure containing arrays of 3D equivalent widths for the absolute line depression (D) and intensity (I)
AC1	:	Structure containing arrays of $\langle$3D$\rangle$ abundance corrections required to replicate the equivalent 3D profiles for absolute line depression (D) and intensity (I)
ACX	:	Structure containing arrays of 1D abundance corrections required to replicate the equivalent 3D profiles for absolute line depression (D) and intensity (I)

FCG1	:	Contains the ⟨3D⟩ continuum intensity (I) and continuum flux (F). They are constant, not changing with $\log gf$ or the line along the CoG.
WCG1	:	Structure containing arrays of ⟨3D⟩ Curve-of-Growth equivalent width fluxes (F) and intensities (I).
FCGX	:	Contains the external 1D continuum intensity (I) and continuum flux (F). They are constant, not changing with $\log gf$ or the line along the CoG.
WCGX	:	Structure containing arrays of Curve-of-Growth equivalent width fluxes (F) and intensities (I) computed from the 1D external model atmospheres.

11.3 linfor_3D_3.uiosave

The UIO formatted output file `linfor_3D_3.uiosave` contains the following:

11.3.1 CONTF3D

The CONTF structure contains information relating to the 3D contribution functions. When the `cc3d` flag is set, this structure is saved and contains extended information from that stored in the CONTF structure.

Description of entries:

NX3	:	Resultant sampling points considered in synthesis, redefined by `nx_skip` set in `linfor_setcmd.pro`
NY3	:	Resultant sampling points considered in synthesis, redefined by `ny_skip` set in `linfor_setcmd.pro`
NZ3	:	Array containing resultant sampling considered during synthesis, redefined by `lctau1` and `lctau2` set in `linfor_setcmd.pro`
ZZ3	:	Vertical geometrical ray scale for the 3D model.
CC3	:	Continuum intensity contribution functions of the 3D model on the geometrical scale, `ZZ3`.

11.4 linfor_3D_4.uiosave

11.4.1 CGOUT

The CGOUT structure contains the 1D (external 1D and ⟨3D⟩) model Curve–of–Growth profiles for every `icg` and `imt`. This is invoked when `cog` is set to either 2 or -2 inside setcmd. The structure contains the following information:

Description of entries:

CGX_LPLAM_F	:	The 1D external model Curve–of–Growth continuum normalised line depression fluxes.
CGX_LPLAM_I	:	The 1D external model Curve–of–Growth continuum normalised line depression intensities.
CG1_LPLAM_F	:	The ⟨3D⟩ model Curve–of–Growth continuum normalised line depression fluxes.
CG1_LPLAM_I	:	The ⟨3D⟩ model Curve–of–Growth continuum normalised line depression intensities.

11.5 linfor_1X.uiosave

This is a special output file, only written when Linfor3D performs synthesis under `run_flag` = −3. While most of the structures given in this file contain most of the same sub-structures and arrays that are found in `linfor_3D_1.uiosave` and `linfor_3D_2.uiosave`, the `MAPS` structure is not written and several sub-structures or arrays pertaining to the 3D or ⟨3D⟩ synthesis. The only exception to this is the inclusion of the `I3, D3, I1` and `D1` arrays in the structure `IMUPHI`. This is so that certain post-synthesis routines, such as `linfor_rotate.pro`, still work without error. While these arrays exist, they only contain zeros.

12 Plotting output

In this section we briefly present examples of how you can manipulate the output detailed in Sect. 11 and plot them in IDL or GDL.

12.1 Plotting the synthesis

Linfor3D has several routines that can quickly process the raw data from the uiosaves output after the synthesis has completed. The first of these is `linfor_rotate.pro`. The call procedure for this routine is:

```
A = linfor_rotate(IMUPHI, itime, kline, vsini [, /normalize, modid = modid])
```

where `IMUPHI` is the imuphi structure found in `linfor_3D_2.uiosave`; `itime` is the snapshot number (0–N-1) or the averaged time (-1); `i_kline` is the `kline` index; and `vsini` is the $v \sin i$ value of star in km s^{-1}. The switch, `/normalize`, is used to normalise the spectrum and keyword `modid` is used to select which synthesis to process (1 = 1D, 2 = ⟨3D⟩ and 3 = 3D).

The other useful routine is `linfor_convol.pro`. This routine is used to convolve a Gaussian profile with the synthesis. The call for this routine is:

```
linfor_convol, lambda, input_flux, output_flux, xi
```

where `lambda` is a 1D array containing the wavelength points; `input_flux` is a 1D array containing the corresponding unbroadened flux; `output_flux` is the 1D output array containing the broadened flux; and `xi` is a float/double scalar turbulence parameter in absolute units – $\xi = \sigma\sqrt{2} = \mathrm{FWHM}/\left(\sigma\sqrt{2}\right)$. Using both these routines will produce an array of flux points that can be plotted. The following step-by-step procedures can be used to successfully load and plot the synthesis.

After loading the two uiosaves, `linfor_3D_1.uiosave` and `linfor_3D_2.uiosave`, create a wavelength array for the number of lines synthesised (`kline`) and data points (`nlamx`):

```
IDL> lambda = fltarr(imuphi.nlamx, imuphi.kline)

IDL> for i = 0, imuphi.kline - 1 do begin &$

IDL> lambda[*, i] = (line.clam - line.dlam[i]) + $
IDL> findgen(1. + 2. * line.dlam[i] / line.ddlam[i]) * line.ddlam[i] &$
IDL> endfor
```

Create the corresponding flux arrays for the 3D, ⟨3D⟩ and 1D fluxes:

```
IDL> flux3 = fltarr(imuphi.nlamx, imuphi.kline) ; 3D flux array
IDL> flux1 = fltarr(imuphi.nlamx, imuphi.kline) ; <3D> flux array
IDL> fluxx = fltarr(imuphi.nlamx, imuphi.kline) ; 1D flux array
```

Set a $v \sin i$ value. For this example, we will set $v \sin i = 5$ km/s:

```
IDL> vsini = 5.
```

Use the routine `linfor_rotate.pro` to produce the normalised flux for the 3D, ⟨3D⟩ (averaged over all snapshots):

```
IDL> for i = 0, imuphi.kline - 1 do begin &$
IDL> flux3[*, i] = linfor_rotate(imuphi, -1, i, vsini, /normalize, modid = 3) &$
IDL> flux1[*, i] = linfor_rotate(imuphi, -1, i, vsini, /normalize, modid = 2) &$
IDL> fluxx[*, i] = linfor_rotate(imuphi, -1, i, vsini, /normalize, modid = 1) &$
IDL> endfor
```

From this procedure, the 3D, ⟨3D⟩ and 1D synthesis can be plotted in IDL or GDL using the `plot` command. However, if one wishes to include an instrumental broadening term in the synthesis, the

following demonstrates how to do this using the `linfor_convol.pro`.

Set up three new arrays for the convolved 3D, ⟨3D⟩ and 1D flux profiles:

```
IDL> f3_inst = make_array([size(flux3, /dimensions)])
IDL> f1_inst = make_array([size(flux1, /dimensions)])
IDL> fx_inst = make_array([size(fluxx, /dimensions)])
```

Set the instrumental broadening in km/s:

```
IDL> v_inst = 10.0
```

and the speed of light in km/s:

```
IDL> c = 2.9979246D+5
```

Calculate the equivalent instrumental broadening value in absolute units:

```
IDL> inst = (v_inst * line.clam / c) / (2 * sqrt(alog(2)))
```

Using `linfor_convol.pro` to broaden `flux3`, `flux1` and `fluxx` with a Gaussian of FWHM `v_inst`:

```
IDL> for i = 0, imuphi.kline - 1 do begin &$
IDL> linfor_convol, lambda[*, i], flux3[*, i], f3_inst[*, i], inst &$
IDL> linfor_convol, lambda[*, i], flux1[*, i], f1_inst[*, i], inst &$
IDL> linfor_convol, lambda[*, i], fluxx[*, i], fx_inst[*, i], inst &$
IDL> endfor
```

12.2 Plotting contribution functions

Linfor3D also contains information on contribution functions. This section explains how to plot one type of contribution function, the equivalent width contribution functions (`crw[3,1,X]f` see Sect. 11.2.1), which are derived by integrating the line-depth contribution functions (Magain 1986, A&A, 135) over all wavelength points considered by Linfor3D during the synthesis run. In this example, we will average over all snapshots computed during the synthesis as well. The `crw[3,1,X]f` has the following dimensions: `[CONST.NCTAU, RESULT.NDATA, RESULT.KLINE]`. Create the arrays and set some variables:

```
IDL> crw3f = fltarr(const.nctau, result.kline)
IDL> crw1f = fltarr(const.nctau, result.kline)
IDL> crwxf = fltarr(const.nctau, result.kline)
IDL> f3   = fltarr(result.nlamx, result.kline)
IDL> f1   = fltarr(result.nlamx, result.kline)
IDL> fx   = fltarr(result.nlamx, result.kline)
IDL> ltauc = contf.ltauc
IDL> tauc = 10.0^ltauc
IDL> ln10 = alog(10)
```

Fill the flux arrays:

```
IDL> for i = 0, result.kline - 1 do begin &$
IDL> f3[*, i] = avg(result.f3.f[*, *, i], 1) &$
IDL> f1[*, i] = avg(result.f1.f[*, *, i], 1) &$
IDL> fx[*, i] = result.fx.f[*, i] &$
IDL> endfor
```

Convert the contribution functions from Eq. (59) to a log τ_{ROSS} scale:

```
IDL> for j = 0, result.kline - 1 do begin &$
IDL> crw3f[*, j] = ln10 * avg(contf.crw3f[*, *, j], 1) * tauc / avg(f3[*, j]) &$
IDL> crw1f[*, j] = ln10 * avg(contf.crw1f[*, *, j], 1) * tauc / avg(f1[*, j]) &$
```

```
IDL> crwxf[*, j] = ln10 * avg(contf.crwxf[*, *, j], 1) * tauc / avg(fx[*, j]) &$
IDL> endfor
```

Finally, plot the contribution functions:

```
IDL> plot, ltauc,crwxf[*, 0], linestyle = 5
IDL> oplot, ltauc,crw3f[*, 0]
IDL> oplot, ltauc,crw1f[*, 0], linestyle = 4
```

The conversion just performed means that the plot depicts $dW/d\log\tau_{\rm ROSS}$ (in mÅ) as a function of $\log\tau_{\rm ROSS}$, where W is the equivalent width and $\log\tau_{\rm ROSS}$ is the logarithm of the optical depth evaluated over a Rosseland scale. As such, $\int (dW/d\log\tau_{\rm ROSS})\, d\log\tau_{\rm ROSS}$ reproduces the equivalent width in mÅ.

12.3 Plotting the Curve-of-Growth

The Curve-of-Growth (CoG) information is contained in three arrays within the RESULT, ABU and CONST structures `CONST.dlgf_cg`, `RESULT.wcgx`, `RESULT.wcg1`, `ABU.abui` and `ABU.abuix`, see Sect. 11 for information. The `wcgx` and `wcg1` are four dimensional arrays formatted according to the number of snapshots considered during the spectrum synthesis, `RESULT.ndata`; the number of lines synthesised, `RESULT/LINE.kline`; the number of microturbulences to be evaluated, `CONST.imt`; and the number of index points to compute the CoG, `CMD/CONST.icg`.

To plot a traditional Curve-of-Growth (i.e. $\log(W)$ as a function of A(X)) the correct abundance should be known. This is usually given in `ABU.abui` (or `ABU.abuix` in version 6.2.2 onwards), if their corresponding abundance files are edited and input into Linfor3D. Otherwise, this value should be input manually. In this example, we assume the former is accurate. First, let's define A(X), in this case we will work with lithium, A(Li):

```
IDL> N = 3
IDL> logA = abu.abui[N] + const.dlgf_cg
```

Next, let's define $\log(W)$, and average out the snapshot information:

```
IDL> logWX = fltarr(line.kline, const.imt, const.icg)
IDL> logW1 = fltarr(line.kline, const.imt, const.icg)
IDL> for k = 0, line.kline - 1 do begin &$
IDL> for i = 0, const.imt - 1 do begin &$
IDL> logWX[k, i, *] = alog10(avg(result.wcgx.f[*, k, i, *], 0)) &$
IDL> logW1[k, i, *] = alog10(avg(result.wcg1.f[*, k, i, *], 0)) &$
IDL> endfor &$
IDL> endfor
```

Finally, let's plot the first line for all microturbulence values:

```
IDL> plot, logA, logWX[0, 0, *]
IDL> for i = 1, const.imt - 1 do $
IDL> oplot, logA, logWX[0, i, *], linestyle = i
IDL> for i = 0, const.imt - 1 do $
IDL> oplot, logA, logW1[0, i, *], linestyle = i, color = 255
```

13 Timing statistics

At the end of a run of Linfor3D timing statistics are presented which are also saved to the file `linfor_timing.txt` in the current working directory. The file will look like this:

```
-------------------------------------------------------------------------------
  T I M I N G   S T A T I S T I C S
-------------------------------------------------------------------------------
  Routine linfor_find_ff.....(total   ) :        10.36 s  (   0.00 % )
  Restoring structure FF.....           :          ---
  Reading 3D model...........(total   ) :       122.68 s  (   0.02 % )
                             (average ) :         6.13 s  (   0.00 % )
                             (       0) :        24.30 s  (   0.00 % )
                             (       1) :         5.17 s  (   0.00 % )
                             (       2) :         5.57 s  (   0.00 % )
                             (       3) :         5.05 s  (   0.00 % )
                             (       4) :         5.74 s  (   0.00 % )
                             (       5) :         5.76 s  (   0.00 % )
                             (       6) :         4.87 s  (   0.00 % )
                             (       7) :         5.14 s  (   0.00 % )
                             (       8) :         5.41 s  (   0.00 % )
                             (       9) :         5.13 s  (   0.00 % )
                             (      10) :         5.41 s  (   0.00 % )
                             (      11) :         5.85 s  (   0.00 % )
                             (      12) :         5.05 s  (   0.00 % )
                             (      13) :         4.76 s  (   0.00 % )
                             (      14) :         5.06 s  (   0.00 % )
                             (      15) :         4.74 s  (   0.00 % )
                             (      16) :         4.72 s  (   0.00 % )
                             (      17) :         5.10 s  (   0.00 % )
                             (      18) :         4.93 s  (   0.00 % )
                             (      19) :         4.91 s  (   0.00 % )
  Routine linfor_ionopa_3d...(total   ) :       736.01 s  (   0.12 % )
                             (average ) :        36.80 s  (   0.01 % )
                             (       0) :        38.12 s  (   0.01 % )
                             (       1) :        37.65 s  (   0.01 % )
                             (       2) :        35.50 s  (   0.01 % )
                             (       3) :        40.03 s  (   0.01 % )
                             (       4) :        38.03 s  (   0.01 % )
                             (       5) :        38.49 s  (   0.01 % )
                             (       6) :        37.90 s  (   0.01 % )
                             (       7) :        36.29 s  (   0.01 % )
                             (       8) :        36.88 s  (   0.01 % )
                             (       9) :        37.29 s  (   0.01 % )
                             (      10) :        37.04 s  (   0.01 % )
                             (      11) :        37.64 s  (   0.01 % )
                             (      12) :        36.05 s  (   0.01 % )
                             (      13) :        33.74 s  (   0.01 % )
                             (      14) :        35.52 s  (   0.01 % )
                             (      15) :        35.83 s  (   0.01 % )
                             (      16) :        35.51 s  (   0.01 % )
                             (      17) :        36.46 s  (   0.01 % )
                             (      18) :        36.14 s  (   0.01 % )
                             (      19) :        35.90 s  (   0.01 % )
  Saving structure FF........           :          ---
  Rad. transfer for  3D  model....(total   ) :    594678.33 s  (  97.40 % )
                             (average ) :     29733.92 s  (   4.87 % )
                             (       0) :     31170.96 s  (   5.11 % )
```

```
                                (       1) :     31021.06 s  (   5.08 % )
                                (       2) :     29445.84 s  (   4.82 % )
                                (       3) :     30393.73 s  (   4.98 % )
                                (       4) :     30423.26 s  (   4.98 % )
                                (       5) :     29654.40 s  (   4.86 % )
                                (       6) :     29151.41 s  (   4.77 % )
                                (       7) :     29403.00 s  (   4.82 % )
                                (       8) :     29536.25 s  (   4.84 % )
                                (       9) :     29749.70 s  (   4.87 % )
                                (      10) :     33397.79 s  (   5.47 % )
                                (      11) :     30029.57 s  (   4.92 % )
                                (      12) :     29955.88 s  (   4.91 % )
                                (      13) :     28652.31 s  (   4.69 % )
                                (      14) :     28731.89 s  (   4.71 % )
                                (      15) :     29166.00 s  (   4.78 % )
                                (      16) :     28685.49 s  (   4.70 % )
                                (      17) :     28832.02 s  (   4.72 % )
                                (      18) :     28688.46 s  (   4.70 % )
                                (      19) :     28589.31 s  (   4.68 % )
  Rad. transfer for <3D> model....(total   ) :     14236.35 s  (   2.33 % )
                                (average ) :       711.82 s  (   0.12 % )
                                (       0) :       755.84 s  (   0.12 % )
                                (       1) :       716.50 s  (   0.12 % )
                                (       2) :       715.40 s  (   0.12 % )
                                (       3) :       721.79 s  (   0.12 % )
                                (       4) :       717.25 s  (   0.12 % )
                                (       5) :       715.77 s  (   0.12 % )
                                (       6) :       702.42 s  (   0.12 % )
                                (       7) :       717.40 s  (   0.12 % )
                                (       8) :       707.86 s  (   0.12 % )
                                (       9) :       703.97 s  (   0.12 % )
                                (      10) :       746.35 s  (   0.12 % )
                                (      11) :       707.23 s  (   0.12 % )
                                (      12) :       695.03 s  (   0.11 % )
                                (      13) :       691.42 s  (   0.11 % )
                                (      14) :       690.57 s  (   0.11 % )
                                (      15) :       736.33 s  (   0.12 % )
                                (      16) :       698.71 s  (   0.11 % )
                                (      17) :       698.63 s  (   0.11 % )
                                (      18) :       701.31 s  (   0.11 % )
                                (      19) :       696.56 s  (   0.11 % )
  Rad. transfer for  1D  model....(total   ) :       577.25 s  (   0.09 % )
  Total..........................(total   ) :    610549.27 s  ( 100.00 % )
-------------------------------------------------------------------------------
```

The file is also saved during a running Linfor3D process. Thus, time statistics are available even after aborting the process. The statistics show the system time needed for individual computation steps/routines of Linfor3D and their contribution to the total time in percent. For the case that the same operation is performed several times, e.g., doing the radiative transfer for more than one model snap shot, the total of all calls, the average time, and the duration for each individual step is given (see example above).

14 IONDIS

IONDIS is responsible for computing all information on requested atomic and molecular species requested in the `line.dat` file. As stated in Sect. 2, IONDIS is run under Fortran. In this section, we will briefly outline the considerations made by IONDIS, and a full list of the limited number of atomic and molecular species IONDIS currently takes into account.

14.1 Atoms

Linfor3D does not at present include the **complete** atomic data information used by CO^{5}BOLD. Rather, a number of selected atoms are properly treated by IONDIS. Additions to IONDIS.f are welcome and will be integrated, after proper testing. However, we ask that **FULL** considerations are taken to the entire program flow of Linfor3D before submitting them to us.

At present (version 6.2.5) there are 61 atomic species considered by IONDIS. You can change the atomic abundances considered during spectrum synthesis by changing the `abuid` (or `abuidx` in versions 6.2.2 onwards) and putting your changes in `special.abu`. Depending on how you want to run Linfor3D (see Sect. 7.5), `abuid` and `abuidx` can be equal or different. If they are different, `cifist2006.abu` is treated as the model abundance file (and is loaded into `abuid`) and `special.abu` is treated as the spectrum abundance file (and is loaded into `abuidx`). A full list of the atoms (and ionisation states, isotopes) are given below.

Table 20: List of atomic species currently considered by IONDIS

Species	Considered ionisation states	Considered isotopes
H	I	...
He	I, II	...
Li	I, II	6, 7
Be	I, II	...
C	I, II	...
N	I, II	...
O	I, II	...
F	I, II	...
Ne	I, II	...
Na	I, II	...
Mg	I, II	24, 25, 26
Al	I, II	...
Si	I, II	...
P	I, II	...
S	I, II	...
Cl	I, II	35, 37
Ar	I, II	...
K	I, II	...
Ca	I, II	...
Sc	I, II	...
Ti	I, II	...
V	I, II	...
Cr	I, II	...
Mn	I, II	...
Fe	I, II	...
Co	I, II	...

Ni	I, II	...
Cu	I, II	63, 65
Zn	I, II	...
As	I, II	...
Rb	I, II, III	...
Sr	I, II	...
Y	I, II	...
Zr	I, II	...
Nb	I, II	...
Mo	I, II	...
Ru	I, II	...
Rh	I, II	...
Pd	I, II	...
Ag	I, II	...
Ba	I, II	134, 135, 136, 137, 138
La	I, II	...
Ce	I, II	...
Pr	I, II	...
Nd	I, II	...
Sm	I, II	...
Eu	I, II	...
Gd	I, II	152, 154, 155, 156, 157, 158, 160
Dy	I, II	...
Er	I, II	162, 164, 166, 167, 168, 170
Tm	I, II	...
Yb	I, II	168, 170, 171, 172, 173, 174, 176
Lu	I, II	175, 176
Hf	I, II	...
Ta	I, II	...
W	I, II	...
Os	I, II	...
Pb	I, II	...
Th	I, II	...
U	I, II	...

14.2 Molecules

Linfor3D also considers a *limited* number of molecules. At present they are only diatomic/bimetallic molecules. We welcome new integrations into IONDIS, and will include them into the general realise, after they are properly tested. However, we ask that **FULL** considerations are taken to the entire program flow of Linfor3D before submitting them to us.

Table 21: Small molecular network: 5 atoms, 8 molecules

	H	C	N	O	Mg
H	H_2	CH	NH	OH	MgH
C	CH	C_2	CN	CO	—
N	NH	CN	–	–	—
O	OH	CO	–	–	—
Mg	MgH	–	–	–	—

Table 22: Large molecular network:10 atoms, 14 molecules

	H	Li	C	N	O	F	Mg	Ti	Cr	Fe
H	H_2	LiH	CH	NH	OH	FH	MgH	—	CrH	FeH
Li	LiH	—	–	–	LiO	–	—	—	—	—
C	CH	—	C_2	CN	CO	–	—	—	—	—
N	NH	—	CN	–	–	–	—	—	—	—
O	OH	LiO	CO	–	–	–	—	TiO	—	—
F	FH	—	–	–	–	–	—	—	—	—
Mg	MgH	—	–	–	–	–	—	—	—	—
Ti	—	—	–	–	TiO	–	—	—	—	—
Cr	CrH	—	–	–	–	–	—	—	—	—
Fe	FeH	—	–	–	–	–	—	—	—	—

14.2.1 Some definitions

N_i	Total number density of nuclei of element i, including nuclei bound in diatomic molecules
$\tilde{N}_i$	Number density of nuclei of element i not bound in diatomic molecules
$\tilde{N}_{i,0}$	Number density of neutral nuclei of element i not bound in diatomic molecules
$N_{i,k}$	Total number density of diatomic molecules made up of one nucleus of element i, and one nucleus of element k
N_e	Total electron number density.
$N_{\rm Kern} = \sum_i N_i$	Total number density of nuclei of all elements, diatomic molecules counting as 2 nuclei
$P_e = N_e\, kT$	Electron pressure.
$f_i = N_i/N_{\rm Kern}$	Fractional abundance of element i (constant)
$x_i = \tilde{N}_i/N_{\rm Kern}$	Fractional abundance of free nuclei of element i. $x_i = f_i = const.$ for elements not involved in molecule formation ($i \notin \{i_{mol}\}$). For elements forming molecules ($i \in \{i_{mol}\}$), x_i is the variable to be iterated.
$x_{i,k} = N_{i,k}/N_{\rm Kern}$	Fractional abundance of molecule (i,k).
$x_e = N_e/N_{\rm Kern}$	Fractional abundance of free electrons (iterated quantity).

14.2.2 Equations

The Saha equation provides the relation between $\tilde{N}_{i,0}$ and $\tilde{N}_i$:

$$\tilde{N}_{i,0} = S_{i,0}\, \tilde{N}_i \tag{124}$$

where the Saha factor $S_{i,0}$ depends on temperature and electron pressure (and on the ionization potentials and the partition functions of the different ionization stages).

Molecule partial pressures are given by the relation

$$P_{i,k} = \frac{P_i\, P_k}{K_{i,k}} \tag{125}$$

where $K_{i,k}$ is the dissociation constant for the (neutral) diatomic molecule (ik), composed of one nucleus of elements i and k each. P_i and P_k are the partial pressures of the **neutral** atoms of elements i and k, respectively. Since $P = N\, kT$, molecule densities are

$$N_{i,k} = \frac{kT\, \tilde{N}_{i,0}\, \tilde{N}_{k,0}}{K_{i,k}} \tag{126}$$

Dividing by $N_{\rm Kern}$, we obtain the fractional molecule abundance

$$x_{i,k} = \frac{N_{i,k}}{N_{\rm Kern}} = \frac{N_{\rm Kern}\, kT\, x_i S_{i,0}\, x_k S_{k,0}}{K_{i,k}} = \frac{P_e\, x_i S_{i,0}\, x_k S_{k,0}}{K_{i,k}\, x_e} = D_{i,k}\, \frac{x_i\, x_k}{x_e} \tag{127}$$

where we have defined

$$D_{i,k} = P_e\, \frac{S_{i,0}\, S_{k,0}}{K_{i,k}}. \tag{128}$$

We note that $D_{i,k}$ depends only on T and P_e, but not on absolute number densities, and hence is constant during the iteration.
For all elements i involved in molecule formation, $i \in \{i_{mol}\}$, we have the following conservation equation

$$f_i = x_i + \sum_k x_{i,k}\,(1 + \delta_{i,k}) \tag{129}$$

or

$$x_i + \sum_k D_{i,k}\, \frac{x_i\, x_k}{x_e}\,(1 + \delta_{i,k}) = f_i \tag{130}$$

where $\delta_{i,k}$ is the Kronecker symbol, accounting for the correct counting of atoms in molecules with two identical components.
The electron density is given by

$$N_e = \sum_i \tilde{N}_i\, \overline{Z}_i = \sum_{i \in \{i_{mol}\}} \tilde{N}_i\, \overline{Z}_i \; + \sum_{i \notin \{i_{mol}\}} \tilde{N}_i\, \overline{Z}_i \tag{131}$$

where the mean degree of ionization of element i, $\overline{Z}_i$ is defined as

$$\overline{Z}_i = \sum_{j=-1,3} j\, \tilde{N}_{i,j} \,/ \sum_{j=-1,3} \tilde{N}_{i,j} = \frac{1}{\tilde{N}_i} \sum_{j=-1,3} j\, \tilde{N}_{i,j} = \sum_{j=-1,3} j\, \tilde{S}_{i,j} = \sum_{j=1,3} j\, \tilde{S}_{i,j} - \tilde{S}_{i,-1}. \tag{132}$$

Here index j runs over the 4 ionization stages $j = 0 \ldots 3$ of element i. For elements forming negative ions, the sum includes the negative ion, $j = -1$. For such elements, $\overline{Z}_i$ may become negative! Note that $\overline{Z}_i$ depends only on T and P_e, but not on the degree of molecule formation. Dividing Eq.(131) by $N_{\rm Kern}$, we obtain

$$x_e = \sum_i x_i\, \overline{Z}_i = \sum_{i \in \{i_{mol}\}} x_i\, \overline{Z}_i \; + \sum_{i \notin \{i_{mol}\}} x_i\, \overline{Z}_i \tag{133}$$

Defining

$$f_e = \sum_{i \notin \{i_{mol}\}} x_i\, \overline{Z}_i = \sum_{i \notin \{i_{mol}\}} f_i\, \overline{Z}_i = const. \tag{134}$$

we have finally

$$x_e \; - \sum_{i \in \{i_{mol}\}} x_i\, \overline{Z}_i = f_e \tag{135}$$

Combining Eq.(130) and (135), we have the following vector equation

$$\vec{X} + \vec{F}(\vec{X}) = \vec{R} \tag{136}$$

where

$$\vec{X} = \{x_1, x_2, \ldots x_N, x_e\}, \tag{137}$$

is the vector of unknown number fractions, and

$$\vec{R} = \{f_1, f_2, \ldots f_N, f_e\} = const., \tag{138}$$

is the known (constant) right-hand side. N is the number of chemical elements included in the molecular network. Equation (136) is a system of $N + 1$ nonlinear algebraic equations which can be solved for $\vec{X}$ by Newton-Raphson iteration.

The first step it to find a suitable starting vector for the iteration, $\vec{X}_0$. This is done as described below. The correction $\delta\vec{X}$ giving the next improved estimate of $\vec{X}$ is computed as follows. Assume after n iterations we have

$$\vec{X}_n + \vec{F}(\vec{X}_n) = \vec{R}_n. \tag{139}$$

Then we require that

$$\vec{X}_n + \delta\vec{X} + \vec{F}(\vec{X}_n + \delta\vec{X}) = \vec{R} \tag{140}$$

or

$$\vec{X}_n + \delta\vec{X} + \vec{F}(\vec{X}_n) + \mathcal{J} \cdot \delta\vec{X} = \vec{R}, \tag{141}$$

hence

$$(\mathcal{J} + 1) \cdot \delta\vec{X} = \vec{R} - \vec{R}_n. \tag{142}$$

The elements of the Jacobian $\mathcal{J}$ are defined as

$$\mathcal{J}_{i,j} = \frac{\partial F_i}{\partial x_j}. \tag{143}$$

Since we know that

$$F_i(\vec{X}) = \frac{x_i}{x_e} \sum_{k=1,N} D_{i,k}\, x_k\, (1 + \delta_{i,k}) \qquad \text{for } i = 1, N \tag{144}$$

and

$$F_{N+1}(\vec{X}) = -\sum_{i=1,N} x_i\, \overline{Z}_i, \tag{145}$$

we can readily evaluate $\mathcal{J}_{i,j}$.
We find from Eqs.(144) and (145)

$$\mathcal{J}_{i,j} = \frac{\partial F_i}{\partial x_j} = D_{i,j}\, \frac{x_i}{x_e} \qquad \text{for } i = 1, N \text{ and } i \neq j \tag{146}$$

$$\mathcal{J}_{i,i} = \frac{\partial F_i}{\partial x_i} = \sum_{k \neq i} D_{i,k}\, \frac{x_k}{x_e} + 4\, D_{i,i}\, \frac{x_i}{x_e} = \sum_{k=1,N} D_{i,k}\, \frac{x_k}{x_e} + 3\, D_{i,i}\, \frac{x_i}{x_e} \qquad \text{for } i = 1, N \tag{147}$$

$$\mathcal{J}_{i,N+1} = \frac{\partial F_i}{\partial x_e} = -\frac{x_i}{x_e^2} \sum_{k=1,N} D_{i,k}\, x_k\, (1 + \delta_{i,k}) \qquad \text{for } i = 1, N \tag{148}$$

$$\mathcal{J}_{N+1,j} = \frac{\partial F_{N+1}}{\partial x_j} = -\overline{Z}_i \qquad \text{for } j = 1, N \tag{149}$$

$$\mathcal{J}_{N+1,N+1} = \frac{\partial F_{N+1}}{\partial x_e} = 0 \tag{150}$$

With this information , we can solve Eq.(142) for $\delta\vec{X}$, and obtain the next estimate

$$\vec{X}_{n+1} = \vec{X}_n + \delta\vec{X} \tag{151}$$

Once the iteration has converged, the molecule densities can be computed from Eq.(127).

14.2.3 Criterion for convergence

The criterion for convergence is currently:

$$|x_i^{(n+1)} - x_i^{(n)}| \leq 1 \cdot 10^{-4}\, f_i \tag{152}$$

and

$$|x_i^{(n)} + \sum_k D_{i,k}\, \frac{x_i^{(n)}\, x_k^{(n)}}{x_e^{(n)}}\, (1 + \delta_{i,k}) - f_i| \leq 1 \cdot 10^{-4}\, f_i \tag{153}$$

for all elements i. The maximum number of iterations is 15.

14.2.4 Initial guess

The initial concentrations of free atoms and ions of elements involved in molecule formation, x_i, are computed as follows.
First, we assume that no molecules are formed and so the initial x_i are set to f_i,

$$x_{i,0} = f_i \tag{154}$$

for all elements. From this, the electron fraction x_e is computed as

$$x_{e,0} = \max\left\{x_{e,\min},\ f_e + \sum_{i\in\{i_{mol}\}} x_i \overline{Z}_i\right\}, \tag{155}$$

where $x_{e,\min} = 1 \cdot 10^{-10}$. Using this value for x_e, we compute the molecule concentrations $x_{i,k}$ according to Eq.(127). If the resulting

$$x_{i,k} \leq 1 \cdot 10^{-5}\ \min\{x_i, x_k\}, \tag{156}$$

the formation of this molecule is considered negligible, and no correction of x_i, x_k and $x_{i,k}$ is necessary. If

$$1 \cdot 10^{-5} \min\{x_i, x_k\} < x_{i,k} \leq \min\{x_i, x_k\}, \tag{157}$$

molecule formation is no longer negligible, but also not exhaustive. In this case, the molecule concentrations must be iterated, but the initial guesses for $x_{i,k}$, x_i and x_k need not be changed. Finally, if

$$x_{i,k} > \min\{x_i, x_k\}, \tag{158}$$

then molecule formation is exhaustive, and the initial guesses for $x_{i,k}$, x_i and x_k are changed. We compute x_i and x_k as the equilibrium values that would result if only this particular molecule was present.
If the molecule consists of two atoms of the same element, the condition is (see Eq.(130))

$$2\,D_{i,i}\,\frac{x_i^2}{x_e} + x_i - f_i = 0 \tag{159}$$

which has the solution

$$x_{i,0} = \frac{2\,f_i}{1 + \sqrt{1 + 8\,f_i\,D_{i,i}/x_{e,0}}}. \tag{160}$$

If the molecule consists of two different atoms, we have two conditions, namely (see Eq.(130))

$$\begin{aligned} D_{i,k}\,\frac{x_i\,x_k}{x_e} + x_i - f_i &= 0 \\ D_{i,k}\,\frac{x_i\,x_k}{x_e} + x_k - f_k &= 0 \end{aligned} \tag{161}$$

From this we see that

$$x_i - x_k = f_i - f_k \equiv \Delta. \tag{162}$$

Then we have

$$D_{i,k}\,\frac{(x_k + \Delta)\,x_k}{x_e} + x_k - f_k = 0 \tag{163}$$

or

$$\frac{D_{i,k}}{x_e}\,x_k^2 + \left(1 + \Delta\,\frac{D_{i,k}}{x_e}\right) x_k - f_k = 0. \tag{164}$$

Assuming that $f_i \geq f_k$, and hence $\Delta \geq 0$, the solution for x_k can be written as

$$x_{k,0} = \frac{2\,f_k}{(1 + \Delta\,D_{i,k}/x_{e,0}) + \sqrt{(1 + \Delta\,D_{i,k}/x_{e,0})^2 + 4\,f_i\,D_{i,k}/x_{e,0}}}, \tag{165}$$

and for x_i we simply have

$$x_{i,0} = x_{k,0} + \Delta. \tag{166}$$

For each molecule, the values of $x_{i,0}$, $x_{k,0}$ obtained for the current molecule from Eq.(160) or Eqns.(165), (166) are compared to the previous values $x_{i,0}^{(n)}$, $x_{k,0}^{(n)}$ (obtained from the same conditions for a previous molecule). The new estimate of is then set to the minimum of previous and current estimate

$$\begin{aligned} x_{i,0}^{(n+1)} &= \min\left\{x_{i,0},\ x_{i,0}^{(n)}\right\} \\ x_{k,0}^{(n+1)} &= \min\left\{x_{k,0},\ x_{k,0}^{(n)}\right\} \end{aligned} \tag{167}$$

The initial guess for the current molecule is then computed as

$$x_{i,k} = D_{i,k}\,\frac{x_{i,0}^{(n+1)}\ x_{k,0}^{(n+1)}}{x_{e,0}}. \tag{168}$$

For simplicity (and stability), the initial guess for the electron fraction is not updated when changing the initial guesses x_i for the elements involved in molecule formation.

14.2.5 Variable names

`DAB`	$D_{i,k}/x_e$	
`DF00`	$\vec{R} - \vec{R}_n$	
`FRACEL`	x_e	
`FRACEL0`	f_e	
`FRACEL1`	$\sum_{i\in\{i_{mol}\}} f_i\,\overline{Z}_i$	
`FRACI`	f_i	
`FRACJ`	$x_i\,S_{i,j}$	(atoms and ions, $j = 1\ldots4$)
	$x_{i,k}$	(molecules)
`FREEI`	x_i	
`NATMOL`	N	
`PNUC`	$N_{\mathrm{Kern}}\,k\,T = P_e/x_e$	
`RIJSUM`	F_i	
`SAHAJ`	$S_{i,j}$	(atoms and ions, $j = 1\ldots4$)
	$K_{i,k}$	(molecules)
`ZEFF`	$\overline{Z}_i$	

15 ionopa

Linfor3D computes opacities, level populations, pressures and densities with the IDL/GDL routine `ionopa.pro`. It has evolved quite considerably over many versions of Linfor3D and this routine has now been replaced by `ionopa2.pro`. It calls on the IONDIS library described above, is fairly easy to use, and can be used separately from Linfor3D to compute various properties for 1D and 3D models. As this routine has developed, the call sequence has changed significantly. The call sequence described here is correct for the last version of `ionopa.pro`, which was running as of Linfor3D version 5.1.5 and deprecated after version 6.1.1:

```
ionopa, temp, pin, alam, pout, densnc, okappa, osigma, pgas=pgas, $
        namj=namj, fracj=fracj, zeta=zeta, init=init, $
        dm=dm, dalpha=dalpha, avm=avm, ehe=ehe, abupath=abupath, $
        nami=nami, abui=abui
```

A brief description of all entries into `ionopa` are now listed.

15.1 temp

description	:	gas temperature(s) in kelvins
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array, 2D array, 3D array: must have same dimensions as `pin`
values	:	`5777, [4000:6000]`

15.2 pin

description	:	input pressure(s) in dyn/cm^2
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array, 2D array, 3D array: must have same dimensions as `temp`
values	:	`1e4, [1e4:5e4]`

15.3 alam

description	:	wavelength range in angstroms
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array
values	:	`5500, [4000:6000]`

15.4 pout

description	:	output pressure(s) in dyn/cm^2
input/output	:	`output`
required	:	always
type	:	float
properties	:	same dimensions as `temp` and `pin`, i.e. scalar, 1D, 2D, 3D array
values	:	`1e4, [1e4:5e4]`

15.5 densnc

description	:	number densities of atomic nuclei
input/output	:	`output`
required	:	always
type	:	float
properties	:	same dimensions as `temp` and `pin`, i.e. scalar, 1D, 2D, 3D array
values	:	`1.0e14`

15.6 okappa

description	:	true absorption continuum opacity in cm^2 per nucleus
input/output	:	`output`
required	:	always
type	:	float
properties	:	[N `temp`, N `alam`] if `alam` is an array, otherwise [N `temp`]
values	:	`1.0e-22`

15.7 osigma

description	:	scattering continuum opacity in cm^2 per nucleus
input/output	:	`output`
required	:	always
type	:	float
properties	:	[N `temp`, N `alam`] if `alam` is an array, otherwise [N `temp`]
values	:	`1.0e-27`

Continuum opacity is given by `okappa` + `osigma`

15.8 pgas

description	:	toggles `pin` and `pout`
input/output	:	`input`
required	:	optional
type	:	switch
properties	:	scalar
values	:	`0 or 1`

`pgas = 0`: `ionopa` assumes that `pin` is the electron pressure and `pout` is the gas pressure (default).

`pgas = 1`: `ionopa` assumes that `pin` is the gas pressure and `pout` is the electron pressure (usual case).

15.9 namj

description	:	input ion identifier
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array
values	:	`2600.0, [2600.0, 2601.0, 5600.0]`

15.10 fracj

description	:	fractional number density n_j / `densnc`, where n_j is the number density of `namj` elements in ionization stage j
input/output	:	`output`
required	:	optional
type	:	float
properties	:	[N `temp`, N `namj`] if `namj` is an array, otherwise [N `temp`]
values	:	`3.30554e-12`

15.11 zeta

description	:	number densities such that `zeta` = `fracj` / $\mathrm{U}_j(T)$
input/output	:	`output`
required	:	optional
type	:	float
properties	:	[N `temp`, N `namj`] if `namj` is an array, otherwise [N `temp`]
values	:	`4.53707e-10`

15.12 init

description	:	type of abu file
input/output	:	`input`
required	:	for initialisation only
type	:	integer
properties	:	scalar
values	:	`1, 2, 3`

`init = 1`: abu file is defined as `kiel.abu`

`init = 2`: abu file is defined as `cifist2006.abu`

`init = 3`: abu file is defined as `special.abu`

15.13 dm

description	:	gas metallicity
input/output	:	`input`
required	:	for initialisation only
type	:	float
properties	:	scalar
values	:	`-1.0, 0.0`

15.14 dalpha

description	:	gas alpha enhancement, [α/Fe]
input/output	:	`input`
required	:	for initialisation only
type	:	float
properties	:	scalar
values	:	`0.0, 0.4`

Affects elements O, Ne, Mg, Si, S, Ar, Ca and Ti

15.15 avm

description	:	average mass of nucleus for chemical composition defined by `init`
input/output	:	`output`
required	:	optional
type	:	float
properties	:	scalar
values	:	`2.08985e-24`

15.16 ehe

description	:	ratio of helium to hydrogen number densities
input/output	:	`output`
required	:	optional
type	:	float
properties	:	scalar
values	:	`0.0851139`

15.17 abupath

description	:	path to directory where abu file defined by `init` can be found
input/output	:	`input`
required	:	always
type	:	string
properties	:	scalar
values	:	`getenv('LINFOR3D_ABU')`

15.18 nami

description	:	ion identifier from abu file defined by `init`
input/output	:	`output`
required	:	optional
type	:	integer
properties	:	1D array
values	:	`100, 200, 300, ..., 9200`

15.19 abui

description	:	corresponding abundances of ions in `nami`
input/output	:	`output`
required	:	optional
type	:	float
properties	:	1D array
values	:	`12.0000, 10.9300, 1.10000, ..., -0.470000`

15.20 Example

In order to use `ionopa` it first needs to be initialised. The initialisation tells ionopa properties of the model such as its metallicity, chemical abundances, alpha enhancement. An example of the initialisation may look as follows:

```
IDL> ionopa, 0.0, 0.0, 0.0, pout, densnc, okappa, osigma, init = 2, $
IDL> dm = -1.0, dalpha = 0.4, abupath = getenv('LINFOR3D_ABU'), $
IDL> nami = nami, abui = abui, ehe = ehe, avm = avm0
```

```
% Compiled module: IONOPA.
```

Not all of those options are required, however. Once initialised, ionopa can be executed. The input parameters should be used instead of the 0.0 values used to initialise.

```
IDL> ionopa, Tin, Pin, alam, pele, densnc, okappa, osigma, /pgas, $
IDL> namj = ions, fracj = fracj, zeta = zeta
```

16 ionopa2

`ionopa2` replaced `ionopa` in Linfor3D version 6.2.0 onwards. The inputs and outputs of `ionopa2` are fairly similar to `ionopa`, but there are certain differences:

```
ionopa2, temp, qin, alam, pe, pg, densnc, okappa, osigma, qflg=qflg, $
         namj=namj, fracj=fracj, zeta=zeta, init=init, $
         dm=dm, dalpha=dalpha, avm=avm, ehe=ehe, abupath=abupath, $
         nami=nami, abui=abui
```

A brief description of all entries into `ionopa2` are now listed.

16.1 temp

description	:	gas temperature(s) in kelvins
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array, 2D array, 3D array: must have same dimensions as `qin`
values	:	`5777, [4000:6000]`

16.2 qin

description	:	input quantities in cgs units
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array, 2D array, 3D array: must have same dimensions as `temp`
values	:	`1e4, [1e4:5e4]`

16.3 alam

description	:	wavelength range in angstroms
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array
values	:	`5500, [4000:6000]`

16.4 pe

description	:	electron pressure(s) in dyn/cm^2
input/output	:	`output`
required	:	always
type	:	float
properties	:	same dimensions as `temp` and `qin`, i.e. scalar, 1D, 2D, 3D array
values	:	`1e4, [1e4:5e4]`

16.5 pg

description	:	gas pressure(s) in dyn/cm^2
input/output	:	`output`
required	:	always
type	:	float
properties	:	same dimensions as `temp` and `qin`, i.e. scalar, 1D, 2D, 3D array
values	:	`1e4, [1e4:5e4]`

16.6 densnc

description	:	number densities of atomic nuclei
input/output	:	`output`
required	:	always
type	:	float
properties	:	same dimensions as `temp` and `qin`, i.e. scalar, 1D, 2D, 3D array
values	:	`1.0e14`

16.7 okappa

description	:	true absorption continuum opacity in cm^2 per nucleus
input/output	:	`output`
required	:	always
type	:	float
properties	:	[N `temp`, N `alam`] if `alam` is an array, otherwise [N `temp`]
values	:	`1.0e-22`

16.8 osigma

description	:	scattering continuum opacity in cm^2 per nucleus
input/output	:	`output`
required	:	always
type	:	float
properties	:	[N `temp`, N `alam`] if `alam` is an array, otherwise [N `temp`]
values	:	`1.0e-27`

Continuum opacity is given by `okappa` + `osigma`

16.9 qflg

description	:	toggles `qin`
input/output	:	`input`
required	:	optional. Set to 0 if ignored
type	:	switch
properties	:	scalar
values	:	`0, 1, 2, 3`

`qflg = 0`: `qin` is defined as the electron pressure, P_e (default).

`qflg = 1`: `qin` is defined as the gas pressure (usual case), $P_{\rm gas}$.

`qflg = 2`: `qin` is defined as $P_{\rm gas} - P_e$.

`qflg = 3`: `qin` is defined as continuum opacity per unit volume, `densnc * (okappa + osigma)`.

16.10 namj

description	:	input ion identifier
input/output	:	`input`
required	:	always
type	:	float
properties	:	scalar, 1D array
values	:	`2600.0, [2600.0, 2601.0, 5600.0]`

16.11 fracj

description	:	fractional number density n_j / `densnc`, where n_j is the number density of `namj` elements in ionization stage j
input/output	:	`output`
required	:	optional
type	:	float
properties	:	[N `temp`, N `namj`] if `namj` is an array, otherwise [N `temp`]
values	:	`3.30554e-12`

16.12 zeta

description	:	number densities such that `zeta` = `fracj` / $\mathrm{U}_j(T)$
input/output	:	`output`
required	:	optional
type	:	float
properties	:	[N `temp`, N `namj`] if `namj` is an array, otherwise [N `temp`]
values	:	`4.53707e-10`

16.13 init

description	:	type of abu file
input/output	:	input
required	:	on initalisation or when more than one abu file has been initialised
type	:	integer
properties	:	scalar
values	:	-3, -2, -1, 1, 2, 3

init = 1: abu file is defined as kiel.abu

init = 2: abu file is defined as cifist2006.abu

init = 3: abu file is defined as special.abu

init = -1: initialise composition 1 from memory*.

init = -2: initialise composition 2 from memory*.

init = -3: initialise composition 3 from memory*.

*Used if ionopa2 has been initialised for more than one abu file.

16.14 dm

description	:	gas metallicity
input/output	:	input
required	:	for initialisation only
type	:	float
properties	:	scalar
values	:	-1.0, 0.0

16.15 dalpha

description	:	gas alpha enhancement, [α/Fe]
input/output	:	input
required	:	for initialisation only
type	:	float
properties	:	scalar
values	:	0.0, 0.4

Affects elements O, Ne, Mg, Si, S, Ar, Ca and Ti

16.16 avm

description	:	average mass of nucleus for chemical composition defined by `init`
input/output	:	`output`
required	:	optional
type	:	float
properties	:	scalar
values	:	`2.08985e-24`

16.17 ehe

description	:	ratio of helium to hydrogen number densities
input/output	:	`output`
required	:	optional
type	:	float
properties	:	scalar
values	:	`0.0851139`

16.18 abupath

description	:	path to directory where abu file defined by `init` can be found
input/output	:	`input`
required	:	always
type	:	string
properties	:	scalar
values	:	`getenv('LINFOR3D_ABU')`

16.19 nami

description	:	ion identifier from abu file defined by `init`
input/output	:	`output`
required	:	optional
type	:	integer
properties	:	1D array
values	:	`100, 200, 300, ..., 9200`

16.20 abui

description	:	corresponding abundances of ions in `nami`
input/output	:	`output`
required	:	optional
type	:	float
properties	:	1D array
values	:	`12.0000, 10.9300, 1.10000, ..., -0.470000`

16.21 Example

Like ionopa, ionopa2 must first be initialised.

```
IDL> ionopa2, 0.0, 0.0, 0.0, pele, pgas, densnc, okappa, osigma, $
IDL> init = 2, dm = 0.0, dalpha = 0.0, avm=avm, ehe=ehe, $
IDL> abupath=getenv('LINFOR3D_ABU'), nami=nami, abui=abui
% Compiled module: IONOPA2.
```

Now that ionopa2 has been initialised, it can be executed.

```
IDL> ionopa2, tin, qin, alam, pele, pgas, densnc, okappa, osigma, $
IDL> qflg = 1, namj = ions, fracj = fracj, zeta = zeta
```

Additionally, ionopa2 can be initialised more than once for more than one abu file.

```
IDL> %----- initialise 1 -----
IDL> ionopa2, 0.0, 0.0, 0.0, pele, pgas, densnc, okappa, osigma, $
IDL> init = 2, dm = 0.0, dalpha = 0.0, avm = avm2, ehe = ehe2, $
IDL> abupath = getenv('LINFOR3D_ABU'), nami = nami2, abui = abui2
% Compiled module: IONOPA2.

IDL> %----- initialise 2 -----
IDL> ionopa2, 0.0, 0.0, 0.0, pele, pgas, densnc, okappa, osigma, $
IDL> init = 3, dm = 0.0, dalpha = 0.0, avm = avm3, ehe = ehe3, $
IDL> abupath = getenv('LINFOR3D_ABU'), nami = nami3, abui = abui3
```

Once it has been initialised for the requested parameters it can be executed without being reinitialised.

```
IDL> %----- execute 1 -----
IDL> ionopa2, tin2, qin2, alam, pele2, pgas2, densnc2, okappa2, osigma2, $
IDL> qflg = 1, namj=[ions], fracj = fracj2, zeta = zeta2, init = -2

IDL> %----- execute 2 -----
IDL> ionopa2, tin3, qin3, alam, pele3, pgas3, densnc3, okappa3, osigma3, $
IDL> qflg = 1, namj=[ions], fracj = fracj3, zeta = zeta3, init = -3
```

Note the use of init in these instances. As ionopa2 has been initialised for more than one abu file, init should be included from its memory.

Index