NICOLE
Non-LTE Inversion COde using the Lorien Engine

Hector Socas-Navarro, Jaime de la Cruz R. & Andrés Asensio Ramos
e-mail: hsocas@iac.es; jaime@astro.su.se; aasensio@iac.es

Version 19.04

Contents

Chapter 1

Introduction

1.1 What is NICOLE?

NICOLE is a general-purpose synthesis and inversion code for the Stokes profiles emergent from solar/stellar atmospheres. Solar instrumentation is becoming more sophisticated every day and the data sets which are currently available (as well as those expected for the near future) demand the use of modern diagnostic tools in order to retrieve as much information as possible from the observations. The algorithm is described in Socas-Navarro et al (2012, in preparation; see also Socas-Navarro, Trujillo Bueno and Ruiz Cobo 2000). It seeks for the model atmosphere that provides the best fit to the profiles (in a least-squares sense) of an arbitrary number of simultaneously-observed spectral lines. The underlying hypotheses are:

- Atomic level populations in statistical equilibrium (NLTE), assuming complete angle and frequency redistribution.
- No sub-pixel atmospheric structure is considered, except for a filling factor and a prescribed external atmosphere.
- The observed Stokes profiles are induced by the Zeeman effect in transitions where L-S coupling is a valid approximation (with the exception of the infrared FeI line at 1565.28nm, which has an ad-hoc treatment).
- The hydrostatic equilibrium equation is used to calculate the gas density and the height scale in the atmosphere. This is optional in synthesis mode (in that case, the gas pressure/density would be read from the input model) and mandatory in inversion mode

- Undocumented feature for hyperfine structure calculations (please, check with the author for information)

The inversion core used for the development of NICOLE is the LORIEN engine (the Lovely Reusable Inversion ENgine, also publicly available for download from the C.I.C. web page), which combines the SVD technique with the Levenberg-Marquardt minimization method to solve the inverse problem (see Press *et al.* 1990).

I would like to hear comments about people using NICOLE, where you are, what your research is about and what your overall experience with the code is. Please, drop me a line at hsocas@iac.es. If you have complains, criticism or generally speaking have negative things to say, please include the word "cialis" in your message subject ;)

1.2 Requirements

1. A Fortran 90 compiler, or above (e.g., F95, 2000, 2003...). Strictly speaking, NICOLE uses F2003 but it uses only a very small subset of the 2003 standard which is supported by most F90 compilers. So your F90 compiler should work.

2. Python, recommended version 2.6.4 or above (although limited testing has shown that it works on 2.4 as well)

3. Some routines (see section 4.1) from the Numerical Recipes (Press *et al.* 1990.) book. You can type them directly from the book, obtain the distribution on a CD-ROM or download them from their website http://www.nr.com.

1.3 Features

Features of this code:

1. Written entirely in Fortran 90, taking advantage of the advanced capabilities provided by this programming language. The code is strictly compliant with the Fortran 2003 ANSI standard. It makes use of a very widely implemented extension to F90, namely allocatable arrays in derived data types. While this is not strictly standard in F90, most compilers are supporting it anyway. What all of this means in practice is that there is a slim chance that a particular F90 or F95 compiler will not compile NICOLE, but this would be a rare occurrence. Any compiler that is compatible with Fortran 2003 (or higher)

should be able to compile NICOLE without a problem since the code is strictly F03 compliant.

2. Dynamic memory management. NICOLE makes use of the heap storage capabilities of Fortran 90 to dynamically allocate and deallocate memory during the program execution. This means that you no longer have to worry about the array dimensions, as in old F77 code, where you had to recompile your code every time you changed the dimensionality of the problem. Instead, NICOLE will allocate and use the required memory during run time.

3. Modular, top-to-bottom design. There are no common blocks in NICOLE (almost) and the source code is clear and straightforward to understand and modify. Most of the program building blocks are encapsulated in modules so it is possible to take bits and pieces of NICOLE to use in your own code.

4. Easy to use. You don't even need to compile it. Pre-compiled executable files will be distributed for the most popular hardware platforms. Just download it and run it on your data.

5. Cross-platform. A Python script is included that analyzes the system it is running on and produces a suitable makefile for that platform (see compiling information below). Byte-endianness is taken into account to allow the code to use files written both in big-endian and little-endian machines (the code will always write little-endian files, regardless of the platform) in a way that is completely transparent to the user.

6. Distributed under the GPL license version 3 (except for the Numerical Recipes routines). The full license is at:

   `http://www.gnu.org/copyleft/gpl.html`

   Basically this means that NICOLE is free and open source. You can copy, edit, share or distribute it. We only ask that you give the authors proper credit. There is a paper currently in preparation. Please, cite us!

1.4 Credits

This code borrows significantly from previous efforts. Most of the actual code has been redesigned and rewritten, but not all. The inversion part of

NICOLE is based on the concept of the SIR code, written by Dr. B. Ruiz Cobo (Ruiz Cobo and del Toro Iniesta 1992).

The synthesis part involves the computation of NLTE populations. The package of routines that does this (in the forward/NLTE/ directory) employs a design that is similar to that of Carlsson's MULTI v2.2 (Scharmer & Carlsson 1985). By this I mean that the organization of loops and the order in which physical quantities are calculated and used is the same as in MULTI. Studying MULTI has been a great help since figuring out an efficient design is what took most of the actual work for this module. Most variable names have been maintained to facilitate code readability for users who are familiar with MULTI. Some routines (listed below) have been adapted directly from MULTI for compatibility with its model atom files. Examples are the routines for calculation of collisional rates. However, the core of the iterative algorithm is different from that implemented in MULTI. Instead of using the linearization scheme, NICOLE implements preconditioning with a local operator (Rybicki & Hummer 1991), as discussed in Socas-Navarro & Trujillo Bueno (1998). The solution of the radiative transfer equation is also different. NICOLE uses the short characteristics method.

Other routines that have been contributed to the project by various scientists around the world are the following:

- A few routines have been taken from the Numerical Recipes book (Press *et al.* 1986). These are detailed in section 4.1 below. Note that I am not allowed to include the source code of the Numerical Recipes routines in the NICOLE distribution.

- The Zeeman splitting is computed by an old routine that I inherited at some point. I'm not entirely sure of this but I believe that this routine was originally written by A. Wittmann. If someone could confirm this, I'd appreciate it.

- The following routines have been adapted from M. Carlsson's MULTI: CA2COL, GENCOL, ENEQ & NGFUNC (and their dependencies).

Disclaimer: This software is distributed "as is" and the authors take no responsibility for any consequence derived from its use.

Chapter 2

Note for users of previous versions

2.1 For users of versions prior to 18.06

A new mode has been introduced in this section where the chromospheric temperature is treated as a step-function temperature increase. This is useful in two situations: a) To simulate the effects of a shock in the chromosphere; b) To parameterize a hot chromosphere with only two free parameters in situations when we don't have enough information for a full depth-dependent inferrence with multiple nodes. This last scenario has become very frequent, typically with observations of CaII 8542 acquired with a Fabry-Perot instrument in low spectral resolution.

In the new mode we have two new free parameters, S_x and S_y, that parameterize the position of the temperature discontinuity (S_x, in units of $\log(tau_{5000})$) and the increase (S_y, in Kelvin). If S_y is not zero, then its value is added to the temperature stratification above S_x. This is true both in synthesis and inversion mode. In inversion mode, the nodes in temperature are placed below $\log(\tau_{5000}) = -3$. It may be somewhat confusing that the model temperature is not simply the stratification provided in the temperature array of the model. One needs to remember that S_y will be added to all grid points above S_x.

2.2 For users of versions prior to 2.6

If you have never used NICOLE before or if you plan to read this manual before working with the code, feel free to skip this chapter altogether. There are two important changes over previous versions. Starting with this version, the native (binary) model file format has changed to accomodate chemical abundances as part of the model. This has been implemented to allow for the new feature of abundance inversions. This means that file formats prior to 2.6 will be converted by the Python wrapper (run_nicole.py) into the new format, using the abundances specified in the NICOLE.input files (or selecting the default option of Grevesse & Sauval 1998). The other big change has to do with the code structure. The loop in inversion cycles has been moved to the outmost level of the main program and redefined to something much more general. Previously, inversion cycles were successive inversion runs varying the number of nodes. Now, the user can change not only the nodes but virtually every other parameter from one cycle to the next. This is done by defining several NICOLE.input files, each one with a suffix corresponding to a different cycle (e.g., NICOLE.input_1, NICOLE.input_2, ... etc). It is then possible to do things like keeping the intermediate inversion models resulting from each cycle, or using the synthetic profiles from a cycle as an input to the next, etc. This flexibility allows the user to do in one run of the Fortran code (thus needing only one job submission in a supercomputer) tasks that otherwise would require several separate runs. The nodes.dat file has been removed. The number of nodes is now specified in each one of the NICOLE.input file in a new section called (of course) [nodes].

Additionally, there are some other smaller changes as well. It is now possible to introduce in the input model certain variables such as electron pressure, gas pressure, density, Hydrogen number density (nH), other forms of Hydrogen (nHminus, nHplus, nH2, nH2plus) and tell the code to observe those values (the default behavior is to solve the ionization equilibrium and the chemical equilibrium and compute the variables from one of them, typically electron pressure, overwritting all other variables). Obviously this only makes sense in the synthesis or conversion modes but not in the inversion mode in which the model atmosphere is modified in succesive iterations.

2.3 For users of versions prior to 2.0

If you have never used NICOLE before or if you plan to read this manual before working with the code, feel free to skip this chapter altogether. Two

things work very differently in the current version of NICOLE with respect to those prior to 2.0. The first one has to do with the code compilation. In order to maximize portability and also to make life easier for the casual user, an automated tool written in Python has been included that works similarly to the popular autoconf tool in Linux. The user simply needs to run this Python program (create_makefile.py) and it will automatically scan the system for compilers, look for the suitable options (record length and byte-endianness for binary files) and even make some slight changes to the code to ensure compatibility of data written in multiple platforms. It will also produce either the serial version (default) or the MPI parallel version (when run with the –mpi flag). The tool recognizes the most popular F90 compilers and sets the appropriate flags. If none of them is found in the system, the user will have to specify manually the compiler and its options. Currently supported compilers include those from GNU, Intel, IBM or the Portland Group. See section 4.1 for information on compiling this version. A battery of tests has also been included to verify that the code works properly (section 4.1).

The other important change has to do with the handling of input/output files. The whole process has been restructured and simplified. The actual Fortran code now works only with a very specific fixed binary file format, both for inputs and outputs. A Python program (run_nicole.py) now does all the parsing of human readable files and creates the fixed-format file that the Fortran code will use. The input files with model atmospheres or observations can now be given in many different and convenient formats, which will then be transformed by the Python wrapper into NICOLE's own native binary format. This means that the user can now supply data in formats that include ASCII (same as in previous versions), IDL savefile or NICOLE's native format. The plan is to include in the near future also support for FITS files (e.g., to invert directly data from Hinode) or some numerical simulation codes. The input files containing the basic run parameters, spectral grid to use or spectral line data, abundances, etc have now been merged into one single file using a popular parsing standard. The format of this input file is now much more flexible and the user can easily track all of the run parameters. The price to pay for this increased flexibility and convenience is that NICOLE must now be run through the Python wrapper. The user is NOT supposed to run the Fortran executable directly. See section 3 for more information.

The following input files have become obsolete and have been removed (most of their functionality has been incorporated in NICOLE.input): the wavelength grid, ABUND and NLTE_lines. There is now no difference between inversion and multiple inversion modes (and similarly between syntehsis and multiple synthesis). A single inversion or synthesis is simply one

where npix=1.

A minor change is that the magnetic field is now given in terms of its components instead of its modulus and angles as in previous versions. This makes the inversion better conditioned.

Chapter 3

Quick start

This chapter is for the impatient type. The first step is obviously to uncompress and unpack the distribution. Under UNIX, do:

```
tar zxf nicole*.tar.gz
```

then copy the relevant Numerical Recipes routines to the numerical_recipes/ directory. The files to be copied are: convlv.f90, four1.f90, fourrow.f90, nr.f90, nrtype.f90, nrutil.f90, pythag.f90, realft.f90, ludcmp.f90, lubksb.f90, svbksb.f90, svdcmp.f90, tqli.f90 and twofft.f90. Once you have done this, go to the main/ directory to create the makefile and compile with

```
./create_makefile.py
make clean
make nicole
```

If you get any errors after running create_makefile, refer to chapter 4.1 for instructions on how to manually configure it. If you wish to include compiler options, e.g. to specify optimization parameters, you may do so with the –otherflags switch. For instance:

```
./create_makefile.py --otherflags='-fast -O3'
```

If the code compiled successfully you may run the included tests to make sure it works properly:

```
cd ../test
./run_tests.py
```

To run the tests in non-interactive mode, e.g. on platforms where jobs need to be submitted to a queue system (typically supercomputers), refer to the procedure described in section 4.1 below.

Note: Often times you may want to create a new makefile using the same options as the last time. This is particularly useful e.g. when you update the source code distribution with a new release of the code. If the source code structure hasn't changed then you don't need to run creat_makefile.py at all. However, there are times when a new version of NICOLE includes new source files or perhaps some files no longer exist or have been moved. Thus, it is always recommended to run create_makefile.py again every time you install a new version of NICOLE. To save you some hassle, you can use the –keepflags option to instruct create_makefile.py to use exactly the same command-line arguments that you used the last time. These arguments are saved as comments in the makefile so they can be read and reused by create_makefile.py at a later time.

If everything went smoothly, go into the run/ directory.

Note: NICOLE works internally with binary files (the format is described below). Binary files can be written in two different ways, usually called big-endian and little-endian style. Some hardware platforms, such as PCs (generally speaking, machines based on the Intel processor architecture), use the little-endian format whereas others (e.g., Motorola or PowerPC architectures) use big-endian. In order to ensure compatibility among files created in different machines, NICOLE will always read and write files consistently using the little-endian form even when it runs on big-endian machines. This process should be completely transparent to the user except that some file transfer clients (typically some ftp programs) are too smart and will modify binary files (swap bytes) when transferring between machines with different endianness in an attempt to make the file compatible with the target machine. Such modification is not necessary for NICOLE files and in fact would result in file corruption. If you experience trouble running the code after transferring model or profile files via ftp, this is probably the reason.

3.1 The synthesis mode

In this mode you will be computing synthetic spectra from one or more prescribed model atmospheres. You will need the following files:

3.1.1 The ATOM file

This file is necessary only when we are computing NLTE lines. If you are working in LTE only, you don't need to read this. The ATOM file resides in the directory where NICOLE is executed. It has the same format as the model atoms used in MULTI, **except** that comments are signaled by an

exclamation mark (!) instead of an asterisk (*). If the ionization stage is in roman numbers, you might have to change that too (i.e., replace "Ca II" with "Ca 2"). So if you got your ATOM file from MULTI, make sure to edit it and replace the asterisks with exclamation marks. Some model atoms have a GENCOL section at the end, which supplies the parameters for the collisional routine GENCOL. That section contains a grid of temperatures under the key TEMP, which specifies the number of temperatures in the grid (ntemp), and then the temperature grid itself, T(1:ntemp). Unlike MULTI, NICOLE requires that ntemp and the temperature grid T(1:ntemp) be all in one line, without any line breaks. Keep in mind that each line in the model atom is limited to a maximum of 500 characters (this limit can be changed in the declaration of array ColStr in forward/NLTE.f90). Similarly, the rest of the temperature dependent data in the GENCOL section must be in one line.

There are some MULTI features that are not supported in the current NICOLE version. For example, you cannot have bound-bound transitions with IW=1. ITRAD=4 is not supported either. Everything else should work as in MULTI, although we have only tested the code with a 6-level Ca and a 14-level O atom. If you find it to run (or not) with other atoms, please let us know.

Only one ATOM file may be used in a given run. This means that you are limited to the calculation of NLTE lines (as many as desired) existing in the model atom employed plus an arbitrary number of lines from any other element(s) treated in LTE.

Blends are treated consistently by NICOLE and computing blended lines does not require any additional configuration by the user. However, the departure coefficients for the NLTE lines are computed neglecting blends (usually a good approximation unless such blends produce significant distortions of a NLTE line core).

Note: The calculation of the magnetic field inclination and azimuth in the line-of-sight coordinates has a singularity when both components are zero. To avoid numerical accuracy issues, a magnetic field component is considered zero by the code when it is below a threshold of 10^{-6} G. This threshold is defined as a parameter (ZeroFieldThreshold) in forward/profiles.f90.

3.1.2 The LINES file

NICOLE needs to know the atomic line data of the transitions you want to synthesize or invert. This is a configuration ASCII file in the ConfigObj standard format. Comments are marked by the # character. Everything following a # will be ignored. The file is divided in different sections. The beginning of a new section is marked by a string enclosed in square

brackets ([ ]). In the case of the LINES file each section corresponds to a spectral line. Inside a section we can have different parameters that define the spectral line. Each parameter goes in a separate line and the symbol = is used to separate the field from its value. For example, a section defining a spectral line would look like this.

```
[FeI 6301.5]
   Element=Fe
   Ionization stage=1
   Wavelength=6301.5080
   Excitation potential= 29440.17 cm-1 # 3.65 eV
   Log(gf)=-0.59
   Term (lower)=5P2.0
   Term (upper)=5D2.0
   Collisions=Unsold
   Width=2
```

The parsing is insensitive to case and indentation. The various sections and the lines within each section can appear in any order. Some fields are mandatory (wavelength, ionization stage, etc) and others are optional (collisions, width, etc) and have default values. Allowed fields in specifying a spectral line are the following (mandatory unless noted otherwise):

- Element: Atomic element symbol (two characters). Case insensitive
- Ionization stage: 1 for neutral, 2 for singly ionized or 3 for doubly ionized. Higher ionization stages are not supported
- Wavelength: Central wavelength in Å
- Excitation potential: Can be given in units of eV (default) or cm^{-1}. To specify units, follow the numeric value with either eV or cm-1 (see example above)
- Log(gf): Self-explanatory
- Term (lower): A string of the form 2D1.5 (see also the example above). The first character must be a number with the lower level multiplicity (2s+1). The second character represents the orbital angular momentum. From the third character to the end of the string we specify the total angular momentum J (may be non-integer)
- Term (upper): Same as above for the upper level

- Collisions: (Optional, default=1). This is a flag indicating the treatment for collisional line broadening. Use 1 or the string Unsold to use the Unsold formalism (Unsold 1955). Set it to 2 or the string Barklem to use the approach of Barklem, Anstee and O'Mara (1998). This is the preferred option since it is more realistic, but it requires setting also the fields Damping sigma or Damping alpha which are line dependent. Note, however, that in this version of NICOLE collisions with neutral Helium are neglected compared to collisions with neutral Hydrogen. Normally this is a good approximation. Option 3 allows you to introduce manually the values of the radiative, Stark and van der Waals damping constants (γ_r, γ_{Stark}, γ_{vdW}). In this mode you can specify the additional fields **Gamma Radiative**, **Gamma Stark** and **Gamma van der Waals** (see below).

- Damping sigma: (Optional, but mandatory if Collisions is set to 2 or Barklem). Sigma parameter (σ) when using the Barklem et al formalism, in units of Böhr radius squared.

- Damping alpha: (Optional, but mandatory if Collisions is set to 2 or Barklem). Alpha parameter (α) when using the Barklem et al formalism. This parameter is dimensionless

- Gamma Radiative: If **Collisions** is set to 3, this sets the value of the damping constant γ_r in units of 10^8 rad/s. The default would be the value obtained with **Collisions = 1** (i.e., using the Unsold formula).

- Gamma Stark: If **Collisions** is set to 3, this sets the value of the damping constant γ_{Stark} in units of 10^8 rad/s per 10^{12} perturbers per cm^{-3} at T=10,000 K. The code assumes a temperature dependence of $T^{0.17}$. The default would be the value obtained with **Collisions = 1** (i.e., using the Unsold formula).

- Gamma van der Waals: If **Collisions** is set to 3, this sets the value of the damping constant γ_{Stark} in units of 10^8 rad/s per 10^{16} perturbers per cm^{-3} at T=10,000 K. The code assumes a temperature dependence of $T^{0.38}$. The default would be the value obtained with **Collisions = 1** (i.e., using the Unsold formula).

- Damping enhancement: (Optional, default=1). Additional multiplicative factor to apply to the collisional damping

- Width: (Optional, default=2). Distance in Å from line center at which the line has a significant opacity. This is used to speed up the calculation when a wide wavelength range is used. To be on the

safe side this parameter should be set to a large value. For most photospheric lines, a value of 1 Å is sufficient but for some strong chromospheric lines larger values are needed.

- Mode: (Optional, default=LTE). Can be either LTE or NLTE. If the line is NLTE then some additional parameters are needed (see below).

- Transition index in model atom: (Optinal, but needed if Mode is NLTE). Index of the transition in the model atom that corresponds with this line. For example if this line is the first transition in the model atom, set this value to 1.

- Lower level population ratio: (Optional, default=1, used only when Mode is NLTE). Sometimes, the transition being referenced in the model atom is actually a multiplet and we are defining one of the lines of that multiplet. This parameter specifies the fraction $\frac{n_l}{n}$ for the lower level, where n_l is the population of the sublevel for the line we are defining and n is the population of the level defined in the model atom.

- Upper level population ratio: (Optional, default=1, used only when Mode is NLTE). Same as above for the upper level.

The values in the LINES file can be overriden by the NICOLE.input file, as explained below. This is done so that one can have a centralized database of spectral information and be able to make temporary adjustments to the atomic parameters for the current run without having to modify the central database. There is a sample LINES file in the run/ directory of your distribution.

In case of conflict between parameters specified in the ATOM and LINES (see below) files (e.g., the $\log(gf)$ in LINES is not compatible with the g and F parameters in ATOM), then the NLTE iteration is done using the values in ATOM and the atomic populations obtained are used for a final Stokes formal solution with the parameters in LINES.

3.1.3 The NICOLE.input file

There is a sample NICOLE.input file in your distribution. This is where we tell NICOLE exactly what we want to do. It is a configuration ASCII file in the ConfigObj standard format. Comments are marked by the # character. Everything following a # will be ignored. The file may contain sections. The beginning of a new section is marked by a string enclosed in square brackets ([ ]).

In each line the field label is separated from the value by the symbol = (e.g., Mode=Synthesis). The parsing is insensitive to case and indentation. If you don't remember exactly the field label, just write your best guess. The Python wrapper run_nicole.py will produce an error if it encounters an incorrect line and will show one or more helpful suggestions.

The various sections and the lines within each section can appear in any order. The main body of the file (before the beginning of any sections) has the main parameters that control the behavior of the code. Some of them are optional and have default values. The following is a list of these parameters (mandatory unless noted otherwise):

- Command: (Optional, default=../main/nicole). Starting in v2.0, the Fortran executable is not launched directly by the user. Instead, it is spanned by the Python script run_nicole.py. Here you can specify what that command is. This is important because typically you need a more complicated form in order to launch the parallel version. For instance, using mpich2 to run on 8 processors, the command could look like this:

  ```
  mpirun -n 8 ./nicole
  ```

  In that case, we would specify:

  ```
  Command=mpirun -n 8 ./nicole
  ```

  When supplying a string argument, as shown in this example, it is generally a good idea to enclose it within quotes to make sure that the parser interprets it as a single value. Otherwise, if we had for instance commas in the expression, it could be broken into a list of arguments.

  ```
  Command='mpirun -n 8 ./nicole'
  ```

- Cycles: Number of cycles in the run (Optional, default=1). This needs to be specified either in NICOLE.input or NICOLE.input_1. Each cycle is governed by a separate NICOLE.input file with the suffix _n (where n is the cycle number). If the file corresponding to a given cycle doesn't exist, NICOLE will try to use the values in NICOLE.input. Typically, if we wish to have a run of 3 cycles one sets Cycles=3 in NICOLE.input (or NICOLE.input_1) and then create the files NICOLE.input_2 and NICOLE.input_3 with the parameters of the second and third cycles, respectively. **Important:** Because of

how the code is structured, all the input files required in any given cycle need to exist at the beginning of the run. So even if you use the output of one cycle to feed the input of another, or even if you use a file to serve simultaneously as input and output (both of which are legitimate strategies), you need to have at least a dummy file that exists and contains the right dimensions before you start the run. Also, all input files for cycles two and above must be in NICOLE's native format. This is because the format conversion is performed by the Python wrapper run_nicole.py and it only works on the first cycle.

- Start cycle (Optional, default=1): You may want to skip one or more cycles. For instance, assume that your run was terminated halfway during the second cycle and you wish to resume it. You would need to use the Restart option (see below) and skip the first cycle when you rerun the code. Use this field to specify in which cycle you wish to begin execution. This field is only read in NICOLE.input or NICOLE.input_1.

- Mode: Can be the word Synthesis, Inversion or Convert (actually, only the first character is checked). Synthesis and inversion are self-explanatory. The third mode, convert is used to convert the geometrical height scale in a model to optical depth, or vice-versa. In this mode the code will take one of them (depending on the value of Height Scale described below), compute the other, write the output model file to disk and exit.

- Input model: Name of the file containing the input model atmosphere. In synthesis mode, this is the atmosphere for which the spectral profiles are computed. In inversion mode this is the starting guess for the model. The file may contain one or many models. In synthesis mode, one set of Stokes I, Q, U and V is produced for each model. In inversion mode, if the number of models is smaller than the number of profiles then the last model is repeated to pad the calculation and ensure that all of the profiles are inverted. If the number of models is larger than the number of profiles, then the last models are ignored. For more information on the possible formats for this file, see section 3.1.4

- Input model 2: Same as above but for the second component in case of a 2-component run. The filling factor, macroturbulence and stray light parameters in this model will be ignored. They are taken from the first model.

- Output profiles: Name of the file that will be written by NICOLE with the output profiles. In synthesis mode this is the final result of the calculation. In inversion mode it contains the fits produced. For more information on the format of this file, see section 3.1.8.

- Heliocentric angle: (Optional, default=1). Cosine of the heliocentric angle (usually denoted as μ in the literature).

- Observed profiles (Optional, but mandatory if Mode is Inversion): Name of the file with the input observed profiles to invert. For more information on the format of this file, see section 3.1.8.

- Restart: (Optional, deafult=0) Set to 1 to resume a previous run that was interrupted. See section 3.2.9. A value of -1 means to remove output files and then do normal run (not restarting previous calculation). Use this to make sure you don't accidentally mix older preexisting files with new results.

- Output model: (Optional, but mandatory if Mode is Inversion). Name of the file with the output model atmosphere resulting from the inversion. In synthesis mode, the full model including gas pressure, density, electron pressure and optical depth scale (which might have been calculated internally by NICOLE) is written. For more information on the possible formats for this file, see section 3.1.4

- Output model 2: Same as above but for the second component in case of a 2-component run.

- Formal solution method: (Optional, default=0). There are several formal solution methods for the Stokes radiative transfer equation currently implemented in NICOLE, including the Hermitian method of Bellot Rubio, Ruiz Cobo and Collados Vera (1998), the Weakly Polarizing Media (WPM) approximation described in Trujillo Bueno & Sánchez Almeida (1999), DELO (Rees et al 1989), Bezier splines (De la Cruz Rodríguez and Piskunov 2013) and short characteristics (Kunasz and Auer 1987). See the comments in the header of subroutine *formal_solution* in forward/forward.f90 for details. The WPM approximation is faster, but not always applicable, while the Hermitian, DELO and Bezier methods are of general validity (provided, of course, that the spatial grid in the model atmosphere is fine enough). There is an automatic formal solver selector implemented in NICOLE, which will check at each wavelength whether WPM is suitable or not. If it is, it will be used. Otherwise, the cubic Bezier method is chosen.

We strongly recommend you to leave this value set to 0 (auto). Otherwise: 1-Cubic DeloBezier, 2-Cuadratic DeloBezier, 3-Bezier scalar, 4-Hermitian, 5-WPM, 6-DeloLinear, 7-DeloParabolic, 8-SC.

- Stray light file: (Optional). In real observations, especially when observing relatively dark structures such as sunspots, one normally has some amount of scattered light from the surrounding regions contaminating the observed signal. This stray light might come from scattering in the Earth atmosphere, from internal reflections in the telescope/instrument system, etc. If you would like to contaminate your synthetic spectrum with stray light, you must enter the stray light fraction in the model atmosphere file (see section 3.1.4 below) and give here the name of a file with the stray light profile. Leaving this field blank is equivalent to setting the amount of stray light to 0 in the model atmosphere and will result in no contamination of the synthetic profile. This profile can also be used as a prescribed external atmosphere that coexists within the spatial resolution element with the atmosphere undergoing synthesis or inversion. The stray light profile must be in units of the quiet Sun continuum at disk center.

- Printout detail: (Optional, default=1). This switch controls how much information is printed out to screen during normal operation. Higher values correspond to more detailed information (and more cluttering). A value of 1 is normally a good choice.

- Noise: (Optional, default=1e-3). Estimation of the noise in the observations (relative to the average disk center quiet Sun continuum intensity). This is used to compute the inversion weights so that a value of χ^2=1 corresponds to a fit at the noise level. If the weights are supplied manually using a Weights.pro file, then this value has no effect.

- Acceptable Chi-square: (Optional, default=0). This parameter is used to avoid (or at least minimize) the effects of local minima. If the inversion results in a χ^2 value worse than this parameter, the code will discard this result and run again with a randomized initialization.

- Maximum number of inversions: (Optional, default=5). How many inversions will be tried to reach the acceptable χ^2 before giving up and picking the best result of the multiple inversion attempts.

- Maximum inversion iterations: (Optional, default=25). Upper limit to how many iterations will be performed in an inversion cycle.

- Always compute derivatives: (Optional, default=Yes). If set to No, then the derivatives for the response functions are not recomputed after successful iteration steps. Only when an inversion step fails to improve the χ^2 then the derivatives are recalculated. This will save some time and often works almost as good as when recalculating the derivatives.

- Centered derivatives: (Optional, default=0). If set to 1, the code will compute response functions using *centered* derivatives with respect to each parameter. It is more accurate, but the inversion becomes almost a factor $\times 2$ slower.

- Gravity: (Optional, default= 2.7414e+4). Surface gravity (in cm s^{-2}). If not specified, the solar value is adopted.

- Regularization: (Optional, default=1.0). During inversion operations, a regularization term is added to the χ^2 function when the model has fluctuations or other generally undesirable behavior. In this manner, we establish a preference for well-behaved (e.g., smooth) models. This factor weights the regularization term. Setting it to 0 means no regularization at all. Higher values make the code care more about model smoothness than quality of the fit.

- Update opacities every: (Optional, default=10). Background opacities don't change significantly over the narrow wavelength range spanned by a spectral line. This parameter specifies the wavelength intervals (in Å) at which the background opacities will be recomputed. Setting this to 0 makes the code recompute opacities for each wavelength point. Larger values avoid too frequent recalculations and therefore save some time.

- Negligible opacity: (Optional, default=0.0). Opacity threshold to avoid wasting time computing lines that are too weak to contribute to the spectrum. If the ratio of line opacity to continuum opacity at 500 nm is smaller than this threshold, the line is neglected. This condition is tested at each depth-point and wavelength.

- Continuum reference: (Optional, default=1). Switch to control the normalization of the spectral profiles. Select 0 for no normalization (output will be in 10^{14} c.g.s. units, e.g. erg cm^{-2} s^{-1} cm^{-1} for flux and the same per strad for intensity). If you use this mode for inversions, make sure to normalize your input observed profiles to 10^{14} first; 1 for HSRA continuum intensity **at disk center** at a wavelength in the middle of each spectral range (default); 2 for normalization to

HSRA continuum intensity **at disk center** at 5000 Å; 3 for normalization to HSRA continuum intensity at a wavelength in the middle of each spectral range and at the heliocentric angle of the observations (spcified above); 4 for **local** normalization to the first point of each region (useful to normalize to the local continuum) but note that this option eliminates all information on absolute photometry. Therefore the temperature scale retrieved is not physical.

- Continuum value: (Optional, default=1). This is simply a normalization factor that will be applied to the input profiles. This is useful for instance if you are using data directly from an instrument in units of counts. You may want to divide by the number of counts of the average quiet Sun intensity to have a proper normalization (but it needs to be consistent with the Continuum reference parameter above).

- Impose hydrostatic equilibrium: (Optional, default=Yes). If set to Yes, the electron pressure, gas pressure and density of the input model are re-evaluated to put the model in hydrostatic equilibrium. If No, then the values in the input model are used. In inversion mode, the model is always put in hydrostatic equilibrium. The hydrostatic equilibrium uses the electron pressure at the top of the atmosphere as a boundary condition, so at least that one value needs to be properly set. However, the stratification obtained is nearly insensitive to the boundary condition except at the higher layers.

- Input density: (Optinal, default=Pel). The electron pressure, electron number density, gas pressure and gas density are related variables. Given one of them, the others can be determined univocally. This switch defines which one of the four is provided as input. The other two will be comptued by the code. Possible values for this field are Pel, Nel, Pgas or Dens.

- Height scale: (Optinal, default=Tau). The depth scale can be specified either as a geometrical scale (z in km) or as the continuum optical depth at 500 nm. Set this variable to either z or tau to use either scale. The other will be computed automatically by NICOLE from the temperature plus density or pressure (electron or gas).

- Keep parameter (Optional, default=0): Certain model parameters are computed internally by NICOLE by solving the equation of state, the ionization equilibrium and/or the molecular chemical equilibrium. It is possible to supply such parameters in the input model and instruct NICOLE to use those values instead of overriding them with its calculations. To do this set this field to 1 (this line can appear

multiple times with different parameters). Possible accepted values for parameter are: Gas_p, Rho, nH, nHminus, nHplus, nH2, nH2plus.

- Eq of state (Optional, default=nicole): This is a switch to specify how to compute the electron pressure Pe from T and Pg, and conversely the gas pressure Pg from T and Pe. NICOLE has three different methods implemented (see Socas-Navarro et al 2014). Possible values are: a)NICOLE to use our own method described in the paper above. This is probably the best compromise between speed, accuracy and stability; b)ANN to use artificial neural networks trained with precomputed values of (T,Pg,Pe). This is the fastest method but not as accurate as the others. c)WITTMANN to use the method described in Wittmann (1974), which in turn is an improvement over the procedure described by Mihalas (1967).

- Eq of state for H (Optional, default=nicole): For the calculation of background opacities and some other quantities such as collisional rates, the code needs the relative distribution of some H states both in atomic and molecular form (the H ions H, H^+ and H^- and the molecules H_2 and H_2^+). There are several options for this switch: a)NICOLE to use the native method described in Socas-Navarro et al (2014). This is the default option and is probably the best compromise between speed, accuracy and stability; b)ASENSIO2 to use the chemical equilibrium method of Asensio Ramos (Asensio Ramos 2003) but restricted to only two molecules in the chemical equilibrium; c)ASENSIO273 to use the same method with the full list of 273 possible molecules; d)WITTMANN to solve for the H populations as in Wittmann (1974), which in turn is an improvement over the procedure described by Mihalas (1967).

- Pe consistency (Optional, default=1e-3): Depending on how the equation of state is solved, it could be that the Pe, Pg values are not consistent due to the approximations employed. This means that if we take a pair (T,Pe) to compute Pg and later recompute Pe from this (T,Pg), the new Pg obtained will in general differ from the original one. If *Pe consistency* is set to a value lower than 10, the Pe value obtained will be iterated until Pe and Pg are consistent within that tolerance. In principle this could slow down the procedure but in practice the difference in computing time should be negligible in almost all situations.

- Opacity Package (Optional, default=Asensio): Package to use for the computation of background continuum opacities from most common

contributors in the visible and infrared. Possible values: a)ASENSIO, b)WITTMANN, c)SOPA . This last option is hidden in the normal distribution because we haven't been able to adapt the original routines to conform with the standards. As such, it might cause problems with compilation and other issues. If there is any powerful reason why you feel that you'd need to use this package, please contact the authors for a SOPA-enabled version.

- Opacity Package UV (Optional, default=TOP): Package to use for the computation of background continuum opacities from the most common contributors in the UV. Possible values: a)TOP, b)DM. The first option (TOP) will make use of photoionization cross-sections tabulated by The Opacity Project/The Iron Project for most neutral and singly ionized elements between Z=1 and Z=26. For Fe we use the data provided by Bautista (1997, A&AS 122, 166) and by Nahar and Pradhan (1994, J. Phys. B 27, 429), using the smoothing technique of Allende Prieto (2008, Phys Scr 133). The second option (DM) uses the approximation of Dragon and Mutschlecner (1980, Apj 239, 104). These contributions are considered only at wavelenghts below 4000 Å.

- Start X position: (Optional, default=1). The model atmosphere and the observed profile files may contain many models or profiles. This parameter specifies at which position to start the calculation. It can be useful when resuming a previous run that was aborted for some reason. When working with a 3D cube you can specify a smaller field by specifying also **End X position**, **Start Y position** and **End Y position**. Pixel positions go from 1 to n_x and/or n_y. Note that setting these values specifies a range of pixels to compute. If you don't want a smaller subfield but rather to restart the computation from a given point onwards, then use the **Start irec** field explained below.

- Start irec: (Optional, default=1). If a computation is interrupted for some reason and you wish to restart it at a given position, set this field to the restart pixel. If you are working with a datacube, keep in mind that irec=(x-1)$\times n_y$+y.

- Debug mode: (Optional, default=0). Set this to a non-zero integer to switch on debug mode. In this mode, NICOLE will produce a core file with debugging information, useful to trace back any possible crash. One file is created by each process (in case of parallel run) at the start of each synthesis or inversion cycle. Upon completion, the file is removed. Therefore, if a crash should occur you will end up with a

file named core_0.txt in your directory. In the case of the MPI version you'll get core_1.txt, core_2.txt ..., one for each process. To find out which process crashed, search for the string ABORTING in the core files (for example, in Unix use grep ÄBORTING...c̈ore*txt. Debug mode will slow down code execution slightly but not dramatically. You can use the procedure read_debug in the idl/ directory to read the debug information. Some debug levels will make the code crash upon errors or warnings, others will make NICOLE reject the current iteration and try to recover. The precise meaning of each level can be found in the header of the main/debug.f90 source file.

- Height scale: (Optional, default=tau). Set to z or tau to specify the height scale for the input model.
- Optimize grid: Only for synthesis mode! If this switch is set, the model is reinterpolated to have a better vertical sampling (but keeping the same number of grid points) before solving the radiative transfer.

NICOLE.input can have an arbitrary number of sections defining the various wavelength ranges that we want to operate on (whether in synthesis or inversion mode). Each section starts with the label [Region 1], [Region 2]... etc. Inside each region we define the following parameters:

- First wavelength: Starting wavelength of the range in Å.
- Wavelength step: This is the sampling, can be given in Å(default) or in mÅ. For the latter, simply follow the number with the string mA.
- Number of wavelengths: Self-explanatory
- Macroturbulent enhancement: (Optional, default=1). Macroturbulence is specified as a velocity and therefore it scales linearly with wavelength. However, the spectral resolution delivered by a spectrograph instrument may vary from one region to another depending on the order employed and other specifics of the configuration. This parameter multiplies the width of the macroturbulence Gaussian in the current region.

At least one region must be defined. NICOLE.input also contains definitions of the lines to synthesize/invert. These are specified in separate sections. Each one of these sections start with the label [Line 1], [Line 2], ... etc. Inside each one of these sections we specify the parameter Line with the identification of the line in the LINES database file. For instance:

```
Line=FeI 6301.5
```

The line must be defined in LINES. In this section we can also override the values given in LINES simply by specifying again the parameter with a different value. At least one line must be defined (for continua calculation, set the Log(gf) to a very small value)

When doing inversions, the number of nodes to be used must be specified in a section of the configuration file. It starts with the line:

[Nodes]

In this section we specify the number of nodes. Valid tokens are: Temperature, T, Velocity, Microturbulence, Macroturbulence, Bz, Bx, By, Stray light, Filling factor, Abundances, Sx, Sy. Here is an example of the nodes section:

```
[Nodes]
# Nodes for first atmospheric component
#   Commented in parenthesis are default values for different cycles
Temperature=4       # (4, 8, 10)
Velocity=1          # (1, 4, 6)
Bz=1                # (1, 4, 4)
Bx=1                # (1, 2, 2)
By=1                # (1, 2, 2)
Microturbulence=0   # (0, 2, 2)
Macroturbulence=0   # (0, 1, 1)
Abundances=0        # (0, 0, 0)
Stray Light=0       # (0, 1, 1)
Filling Factor=0    # (0, 0, 0)
```

By default NICOLE will set the nodes equispaced through the atmosphere. You may override this and explicitly set the location of the nodes by supplying a comma-separated list of heights. The whole list must be enclosed within quotes (”). For instance:

```
[Nodes]
Temperature=" -5.5 , -4 , -3.5 , -2 , 0 , 2"
```

For NLTE calculations we can include a section to override the default parameters that control the NLTE iteration, convergence, etc:

- Elim: (Optional, default=1E-3 for synthesis, 1E-4 for inversions). The NLTE iteration will stop when the maximum relative change in the atomic level populations, considering all levels and all depthpoints, is below this value.

- isum: (Optional, default=1). Which statistical equilibrium equation will be replaced by the particle conservation equation to close the system.
- istart: (Optional, default=1). Switch to control the initial guess for the atomic level populations. 0: Initialize the solution of the statistical equilibrium for zero radiation field; 1: Start with LTE populations.
- CPER: (Optional, default=1.0). Artificially enhance collisional rates by this factor.
- Use collisional switching: (Optional, default='n'). If set to 'yes' use the collisional switching scheme, in which the NLTE iteration is started with a high value for CPER which is then gradually decreased as the iteration progresses until it finally becomes unity.
- NMU: (Optional, default=3). Number of points in the angular quadrature.
- QNORM: (Optional, default=10). Wavelength normalization value in km/s.
- Formal Solution: (Optional, default=1). Switch to control the formal solution routine to be employed. Set to 1 to use Delo bezier splines or 2 for short characteristics.
- Linear formal solution: (Optional, default=0). If set to 1 forces the solution of the radiative transfer in the NLTE module to use the linear approximation. This is usually more stable but somewhat less accurate than the default.
- Optically thin: (Optional, default=1e-3). Monochromatic optical depth above which we consider the atmosphere to be transparent.
- Optically thick: (Optional, default=1e3). Monochromatic optical depth below which we consider the atmosphere to be thick and use the diffusion approximation (but neglecting the source function gradient)
- Vel Free: (Optional, default='y'). Whether or not to use the velocity free approximation.
- NGACC: (Optional, default='y'). Whether or not to use NG acceleration.
- Max Iters: (Optional, default=500). Maximum number of allowed interations in the NLTE computation.

- Lambda Iterations: (Optional, default=3). Number of lambda iterations to perform at the beginning of the NLTE computation.

- Ltepop: (Optional, default='nicole'). Set to either 'multi' or 'nicole' to switch between two different approaches in the calculation of the LTE populations. In the MULTI approach, the sum of statistical weights that appears in the Saha equation is done over the finite number of levels considered in the model atom. In the NICOLE approach, tabulated partition functions are used to determine an approximate sum over an infinite number of levels.

- Elements to ignore in backgroudn opacities: (Optional, default="). If the NLTE calculation includes the bound-free transitions treated in detail, we might want to exclude that element in the calculation of background opacities. A typical example is the H atom. All background opacity packages in NICOLE consider the opacity produced by bound-free transitions of neutral hydrogen. If we are computing a H atom that includes those transitions then it's a good idea to put H in this field. The background opacity package would then skip the computation of opacities due to neutral H bound-free transitions.

Finally, we can optionally have a section about abundances in NICOLE.input. It is defined in its own section, which starts with the line:

[Abundances]

This section can have the following optional parameters:

- Abundance set: (Optional, default=grevesse_sauval_1998). NICOLE has several abundance sets preset in the code (actually, they are in the Python wrapper). Current options are: Grevesse_Sauval_1998, Asplund_et_al_2009, Thevenin_1989, or Grevesse_1984

- Abundance file: (Optional). If present this is the name of an ASCII file containing the abundances for the 92 first elements. Each line has an element, defined by its two-letter symbol followed by the = sign and its abundance on the usual log scale that has H as 12. For instance

H=12

Inside this Abundances section we can have a subsection named [[Override]] in which we can specify a discrete set of elements for which we want

to override the default values (whether from the hardwired databases or from the external file). This is done simply by including the element to override in this subsection. A full example would be:

```
[Abundances]
Abundance set=Thevenin_1989
[[Override]]
O=600 ppm
He=0.095
```

In the override subsection, the abundances can be specified also in parts per million simply by following the number with the ppm string. Note that, if you have defined abundances in the model atmosphere, those will have precedence and will not be overridden by the values in this section. The abundances specified in NICOLE.input will be used only if the model atmosphere file has zeros in the abundance array.

3.1.4 The model atmosphere file

This file contains one or more model atmospheres for which the synthetic spectrum will be computed. Units for the various physical variables are c.g.s. (K for temperature, g cm^{-3} for density, dyn cm^{-2} for pressure, cm s^{-1} for velocity, G for field strength and degrees for field inclination and azimuth) except for z which is height in km (positive values correspond to higher layers). The line-of-sight velocity sign follows the Astrophysical convention where positive values represent downflows (redshift). The magnetic field in this version is defined primarily by its components with respect to the line of sight. B_long represents the longitudinal (along the line of sight) component, whereas B_x and B_y are the two components on the plane of the sky. The coordinates x and y are arbitrary but they define the reference frame for the linear polarization Stokes Q and U. The "local" variables refer to the solar reference frame. The inclination is defined with respect to the vertical, so 0 is field pointing up, 90 is horizontal and 180 is pointing down. The azimuth reference is arbitrary but the absorption profile for linear polarization is such that Q is positive and U is zero when the azimuth is 0 or 180.

The file can be in one of several formats:

ASCII

This is the same format used in versions prior to 1.6. Have a look at the HSRA.model file included in your distribution. Any line starting with a !

symbol is a comment and will be ignored. The first non-comment line in this file must be the string:

Format version: 1.0

This is an internal identifier which is used for backwards compatibility. Don't change that line or NICOLE will complain about it and refuse to work. The second non-comment line contains two or three real numbers. The first one is the macroturbulent velocity, in cm/s. After computing the synthetic profile, it will be convolved with a Gaussian whose half-width is this value. However, note that if a file named Instrumental_profile.dat exists in the running directory, then the macroturbulence will be ignored (see 3.1.5 below). The second number in this line is the fraction of stray light that will be added to the synthetic profile. It must be in the range [0,1]. Note that the stray light parameter may also be used to account for a magnetic filling factor.

The following lines describe the depth-dependence of the model. NICOLE will read eight columns, which stand for $\log(\tau_{5000})$, temperature (in K), electron pressure (in dyn/cm^2), microturbulence (in cm/s), longitudinal magnetic field (in Gauss), line-of-sight velocity (in cm/s), transverse field (i.e., on the plane of the sky) on the x direction (in Gauss) and transverse field on the y direction (in Gauss), respectively. The $\log(\tau_{5000})$ scale does not need to be equispaced and it may be either increasing or decreasing, but it must be monotonic. Instead of electron pressure, the third column may be electron number density, gas pressure or density (all in c.g.s units), depending on what has been specified in the parameter Input Density in NICOLE.input

IDL savefile

We can use an IDL savefile as inputs. The file may contain the following variables: z, tau, t, el_p, gas_p, rho, v_los, v_mic, b_los_x, b_los_y, b_los_z, b_x, b_y, b_z, nH, nH$^-$, nH$^+$, nH^2, nH^{2+} as arrays with dimensions (nx, ny, nz). Additionally, the following (nx,ny) arrays might exist: keep_el_p, keep_gas_p, keep_rho, keep_nH, keep_nHminus, keep_nHplus, keep_nH2, keep_nH2plus, v_mac, stray_frac, ffactor. Finally, the array abundance (nx,ny,92) can be used to specify the chemical composition of the atmosphere, in Astrophysical logarithmic units. All variables are in c.g.s. units (tau actually refers to the base-10 logarithm of the optical depth at 500nm). The keep_xxx variables are actually flags (even though they are defined as real*8 numbers to maintain the same datatype in the entire model). Each flags specifies whether NICOLE should compute that

variable internally (keep_xxx=0, the default behavior) or use the value supplied in the model (keep_xxx=1).

Sometimes, when creating or manipulating arrays where one of the dimensions has only 1 element, IDL will supress such dimension. Before you write the IDL savefile with the data, please make sure that the model variables have three dimensions (or two in the case of v_mac, stray_frac, ffactor, and the keep flags). If you are having trouble with the array dimensions you can also include nx, ny and nz in the IDL savefile and NICOLE will use them to interpret the arrays correctly. As a last resort, if the arrays are not three-dimensional and nx, ny or nz are not properly specified, NICOLE will try to interpret the missing dimensions as being 1 (starting with z, then y and finally x).

NICOLE native format

NICOLE works internally with binary direct-access little-endian files. This is true even if you are running on a big-endian machine. NICOLE recognizes the endianness of the machine and, if running on big endian, will do byte swapping before reading and writing to disk. Output models produced by the code are in this format.

The record size for model files is 22*nz+11+92 real numbers of kind 8 (meaning, 8 bytes per number). Each record corresponds to one model atmosphere and has all variables stored sequentially in the following order: z(1:nz), log tau_500(1:nz), t(1:nz), gas_p(1:nz), rho(1:nz), el_p(1:nz), v_los(1:nz), v_mic(1:nz), b_long(1:nz), b_x(1:nz), b_y(1:nz), b_local_x(1:nz), b_local_y(1:nz), b_local_z(1:nz), v_local_x(1:nz), v_local_y(1:nz), v_local_z(1:nz), nH(1:nz), nHminus(1:nz), nHplus(1:nz), nH2(1:nz), nH2plus(1:nz), v_mac, stray_frac, ffactor, keep_el_p, keep_gas_p, keep_rho, keep_nH, keep_nHminus, keep_nHplus, keep_nH2, keep_nH2plus, abund(1:92). The "local" b and v represent the magnetic field and velocity vectors in the local solar frame of reference. These parameters are not used by NICOLE directly. They are used by the incline.py program to pre-process a 3D model cube, transforming it from vertical to an inclined line-of-sight. During this transformation, the local b and v are used to compute b_long, b_x, b_y and v_los, which are the variables needed by NICOLE. In summary, if you use incline.py, then b_local and v_local are used but v_los, b_long, b_x and b_y are ignored. If you don't use incline.py then it's the other way around.

The keep_xxx variables are actually flags (even though they are defined as real*8 numbers to maintain the same datatype in the entire model). Each flags specifies whether NICOLE should compute that variable internally (keep_xxx=0, the default behavior) or use the value supplied in the model (keep_xxx=1).

The model file contains npix+1 records (npix being the number of models in the file, equal to nx×ny). The first record is actually a signature to allow the code to recognize the file and also to provide the nx, ny and nz parameters needed to dimension the variables. This signature has the following format. The first 11 bytes contain the string nicole2.3bm. The following 5 bytes are 0s, and then come two 32-bit integers representing nx and ny and one 64-bit integer with nz. From there on the record is padded with zeros.

3.1.5 The instrumental profile file (optional)

The instrumental profile is the response of the instrument to a monochromatic beam. Often this is modeled with the macroturbulence assuming that the instrumental profile is a Gaussian but sometimes this approach does not suffice. If we know the instrumental profile of our instrument (e.g., from theoretical considerations or by measuring the spectrum produced when the system is illuminated by a laser beam) we can specify it by including a file in the running directory named Instrumental_profile.dat. One can also use the suffix _1, _2, etc to have a different file for each cycle of the run. If the file exists, NICOLE will read and use it. Make sure you have spelled the name correctly, though. If the Printout Detail parameter in NICOLE.input is 1 or higher, NICOLE will print the following message:

```
Reading Instrumental_profile.dat
```

This is a binary file with as many records as pixels in the field of view (i.e., $nx \times ny$) plus one. This has changed from versions prior to 2.6 to allow for an instrumental profile that varies over the field of view, as is the case with some instruments. Note that each profile might contain different spectral regions. We then need to include the appropriate profile for each region in each profile. The instrumental profile needs to have its peak at the first point, then have the right-hand side of the profile function run through half of the region profile and, finally, from the midpoint to the last we have the left side of the profile. Each record has $n\lambda$ 8-byte numbers. The first record contains some metadata. The first number specifies the number of points in the instrumental profile for the first spectral region. If there are more regions, then the second number has the number of points in the instrumental profile for the second region, and so on. The number of regions must match what has been specified in NICOLE.input. The user is responsible to ensure that this file is consistent with the rest of inputs. No sanity checks are performed by the code. After this first record with metadata, we have all other records with the instrumental profile(s) for each pixel. Each record contains the instrumental profile, starting with

the peak, with the same wavelength sampling as the spectral region and normalized so that the total area is unity (otherwise the synthetic profile normaliazation is affected by the profile area). Obviously, the number of points in all of the instrumental profiles together must not exceed the total number of wavelengths being computed.

If the instrumental profile is specified, macroturbulence is ignored. It is recommended to test this option with a simple synthesis of a narrow line (without micro- or macroturbulence), observing the broadening produced when the instrumental profile is used.

3.1.6 The departure coefficients file (optional)

Unless you really know what you are doing, you can (and probably should) skip this section. We can specify departure coefficients that will be applied by multiplying the opacity (lower level departure coefficient) and emissivity (upper level departure coefficient) of the lines computed by NICOLE. This could be potentially useful to apply *ad-hoc* NLTE corrections to the lines. To do so you need to have a file named depcoef.dat. The file is in binary format with a record length of $2 \times nl \times nz$ reals of kind=8, where nl is the number of lines and nz the number of vertical grid points in the model atmosphere. For each x pixel the file has a record that contains for each line the departure coefficients for the lower level at all depth-points and then similarly for the upper level.

Note that the departure coefficients need to be in the same grid as the model atmosphere. Also, it is assumed that the coefficients are ordered starting at the top of the atmosphere. If this file is read and the departure coefficients are used, NICOLE will print the following message (assuming that Printout Detail in NICOLE.input is 1 or higher):

```
Reading depcoef.dat
```

There is an alternative use of the depcoef.dat file in which you can supply actual populations directly (actually, one introduces the ratio n/g which is what the code will use internally). To do this, introduce the following line in NICOLE.input:

```
Depcoef behavior= 2
```

and for each level write the populations divided by the statistical weight (g) as explained above (remember: starting from the top of the atmosphere).

3.1.7 Running NICOLE in synthesis mode

Ok, so now that we know all the input files, what they do and how they are written, we only have to run the Python wrapper:

```
./run_nicole.py
```

at your command prompt. This program, more than a wrapper, is almost an entire code by itself. It creates the input files for the Fortran code and runs it. The input files created by run_nicole.py start with a double underscore (e.g., __inputmodel.bin). To avoid accidentally overwriting your own files, it is strongly recommended that you do not use any files starting with a double underscore and leave this prefix for NICOLE and its support programs.

You have to be aware that, if the output files already exist, NICOLE will not replace those files. Instead, it will overwrite the records that it has computed in the present run. Other preexisting records in the file will be preserved. This can be good in some situations but bad in others. For example, NICOLE has a "Restart" mode in which it is possible to resume a run that was interrupted before completion. Or sometimes one uses the input model file with the guess model atmosphere to contain also the atmospheres produced by the inversion. In these situations it is good to have the ability to use existing output files. However, there may also be situations in which one inadvertently overwrites parts of a preexisting files and ends up with unexpected results. If a given output file exists before the run, NICOLE will issue a warning but proceed anyway. If you want to make sure that your run is fresh new, set Restart to -1 (see Section **??** above) and the output files will be removed at the beginning of the run.

When you run the code you may get some warning messages about the line not being optically thin at the surface. This is because the model does not extend high enough. In any case, the values for τ_ν that NICOLE is reporting are $\sim 10^{-2}$, which is good enough for our purposes here, so don't worry too much about it. Ok, so everything worked fine and now you have a new file in your directory with the synthetic data. Congratulations! You have run your first calculation with NICOLE!

If you specified an output model file in NICOLE.input, NICOLE will output the actual model that it used to do the synthesis. This might not coincide exactly with the model you supplied due to a number of reasons. First of all, if you have the optimize grid option set, NICOLE has reinterpolated your model to a more suitable grid. Moreover, some of the variables defining the plasma state are computed by NICOLE's equation of state routines, possibly overriding any values you might have supplied. For instance, if you set input density to electron pressure, all variables will be recomputed from T and Pe: density, gas pressure and number densities of neutral hydrogen, ionized hydrogen, negative hydrogen ion, hydrogen molecule and singly ionized hydrogen molecule. You can instruct NICOLE to retain the original values you supplied in the input model for any of these

parameters by setting the various keep flags, as explained in section 3.1.3. So, for instance, if you wish to use the gas density, gas pressure and neutral hydrogen density provided in your input file, you would need to set *Keep Rho*, *Keep Gas_p* and *Keep nH* to 1 in NICOLE.input (see section 3.1.3).

If the NLTE inversion does not converge (to see this you may need to have the printout parameter set to 3 or greater in NICOLE.input) try playing around with the parameters in the NLTE section of NICOLE.input. The first thing to try would be to use linear interpolation by setting *linear formal solution* to 1. You can then try changing the formal solution from 1 to 2 or viceversa.

The format of the file that you have just obtained, HSRA.pro, is explained in section 3.1.8 below. You can read it in IDL with the read_profile.pro procedure. On exit, NICOLE produces a file with profiling information, named profiling_n.txt, where n is the process number. In case of the serial build there is only one file but the parallel build produces one file per process. This file shows the code execution time and also breaks down how this time is spent inside the most time-consuming routines. Note that some of the routines profiled are actually called by others. For example, solvestat is called by forward, which in turn is called by compute_dchisq_dx. Because of this the percentages don't add up to 100%.

3.1.8 The profile file

This file contains one or more spectral profiles (observed or synthetic). The file can be in one of several formats:

ASCII

This file is arranged in five columns and has as many rows as wavelengths in your spectrum. The first column contains the wavelength in Å. Columns two through five contain the Stokes I, Q, U and V parameters, respectively. The Stokes parameters are normalized according to the corresponding setting in NICOLE.input

IDL savefile

We can use an IDL savefile as inputs. The file must contain the following variables defined: stki, stkq, stku and stkv. They must be arrays of 3 dimensions: nx, ny and nλ (where nλ is the number of wavelengths in the wavelength grid). If several regions are defined then the profiles for different regions are listed sequentially for each x, y point.

Sometimes, when creating or manipulating arrays where one of the dimensions has only 1 element, IDL will supress such dimension. Before you write the IDL savefile with the data, please make sure that all stki, stkq, stku and stkv have three dimensions. If you are having trouble with the array dimensions you can also include nx, ny and nλ in the IDL savefile and NICOLE will use them to interpret the arrays correctly. As a last resort, if the arrays are not three-dimensional and nx, ny or nz are not properly specified, NICOLE will try to interpret the missing dimensions as being 1 (starting with nλ, then y and finally x).

NICOLE native format

NICOLE works internally with binary direct-access little-endian files. This is true even if you are running on a big-endian machine. NICOLE recognizes the endianness of the machine and, if running on big endian, will do byte swapping before reading and writing to disk. Output models produced by the code are in this format.

The record size for profile files is 4*nλ real numbers of kind 8 (meaning, 8 bytes per number). Each record corresponds to one set of the 4 Stokes profiles and has all variables stored sequentially in the following order: I(λ_1), Q(λ_1), U(λ_1), V(λ_1), I(λ_2), Q(λ_2), U(λ_2), V(λ_2), ... ,I($\lambda_{n\lambda}$), Q($\lambda_{n\lambda}$), U($\lambda_{n\lambda}$), V($\lambda_{n\lambda}$)

The file contains npix+1 records (npix being the number of profiles in the file, equal to nx×ny). The first record is actually a signature to allow the code to recognize the file and also to provide the nx, ny, nλ parameters needed to dimension the variables. This signature has the following format. The first 11 bytes contain the string nicole2.3bp, followed by 5 bytes with 0s. Then come two 32-bit integers representing nx and ny, and one 64-bit integer with nλ. From there on the record is padded with zeros.

3.2 The inversion mode

3.2.1 Setting the input parameters

At this point you already know almost everything you need to run NICOLE. In this section you will learn how to use the code in inversion mode. First, we have to change the operation mode in the NICOLE.input file. Edit this file and set Mode to “Inversion”. Now the meaning of the fields Input Model and Synthetic Profiles is different. The input model is a starting guess model, and the synthetic profiles are the profiles emergent from the retrieved model that we will obtain as a result of the inversion. So set input model to “guess.mod” and synthetic profiles to “modelout.pro”.

In addition, there are two new fields that we must complete in inversion mode. The Observed Profiles field must be set to the name of a file with the observed profiles that we wish to invert. Let us test the code by using the HSRA.pro profiles that we synthesized above as observed profiles. This way we can check whether or not the retrieved model is similar to the HSRA.model. The other field that we must complete is Output Model which should be set to the name of the file that will contain the model obtained as a result of the inversion. In this example, set it to "modelout.mod".

3.2.2 Running NICOLE in inversion mode

That's all we need to do. Now run NICOLE exactly as in section 3.1 and you will notice some differences. She will take a little longer to run now and will display messages like this:

```
iter=1 Lambda=10. Chisq=3941.23901
```

Each time you see one of these messages, NICOLE has performed a successful inversion iteration step. The numbers in this line have the following meaning. First we have an integer number, iter, which shows how many iterations have been performed so far. Then we have the diagonal element λ, which is a parameter used in the Levenberg-Marquardt algorithm (you don't need to worry about that for the moment). Finally, Chisq is the merit function χ^2, which measures the quality of the fit. A nice fit is obtained when χ^2 is around or below one. Obviously, the fit is still poor in the first iteration. But no sweat. It will improve quickly. However, the actual values of χ^2 depend on the weights employed.

Instead of the message above, you may read something like this from time to time:

```
REJECTED:— iter=9 Lambda=0.100000001 Chisq=0.607707798
```

This means that the model proposed by NICOLE in this iteration does not yield an improvement to the current value of χ^2, and therefore it has been rejected. After three successive failed attempts, NICOLE will quit. If, at this point, the fit is not yet satisfactory, NICOLE will add more degrees of freedom to the model (e.g., will allow for gradients in the magnetic field strength and orientation, more complicated velocity and temperature stratifications, etc.) and restart a new set of iterations from the current guess model.

Another warning message that you might occasionally get is the following:

Clipping temperature

This means that the model proposed by NICOLE has exceeded the allowed temperature range at one or more depth-points. When that happens, she will bring the model back within range by performing a linear transformation. Normally this will not represent any problem, and it is not something you should worry about. The other atmospheric parameters are also monitored and clipped when they get out of range, so you might get similar warnings regarding the magnetic field, the fraction of stray light, etc.

Once the inversion is finished you can check how good it was by reading the profiles and models with the read_profile.pro and read_model.pro IDL procedures.

Congratulations! You have successfully inverted your first spectrum with NICOLE.

3.2.3 Inversion weights

It is often useful to give different weights to the different Stokes profiles, because the amplitudes of I, Q, U and V typically differ in one or two orders of magnitude. NICOLE uses an automatic weighting scheme defined in the compute_weights routine (which can be found in the misc/ directory of your source code distribution). This scheme takes into account the different amplitudes of the profiles and the noise of the observations. However, one may want to fine tune or customize this scheme. Another reason to weight the profiles is to discard telluric lines or any other contamination that may be present in the observations.

The default weights can be overridden by creating a file named Weights.pro in the working directory. This file must have the same format as a regular ASCII profile file (see section 3.1.8), but contains the σ^2 values for I, Q, U and V at each wavelength point. The actual weight is inversely proportional to σ, so for instance if one wants to ignore a given wavelength range, we just set σ^2 to a very large value in this file. Another way to override weights is to set some points in the input value to negative values beyond -10 (meaning larger absolute values).

Whether one uses default or custom weights, NICOLE will produce an output file Weights_used.pro with the weights it has used in ASCII format.

3.2.4 Changing the default number of nodes

Advanced users may want to change the way NICOLE selects nodes for the inversion. This is done in the NICOLE.input_n files (see section 3.1.3).

3.2.5 Monitoring the inversion

After each successful iteration, NICOLE will output the current guess model and the emergent profiles in the files tmp.mod and tmp.pro. If you would like to monitor the progress of the inversion in real time you can plot these files at any time. A tmp.err file with the error bars is also output at each iteration (see section 3.2.6 below).

3.2.6 The error bars

NICOLE will output the error bars for the current guess model as tmp.err, and for the final model in a file named as the output file, but with extension ".err". Basically, the errors file has the same format as a model atmosphere file. All the physical variables are set to -1 at all the depth-points, except at those points where we have "nodes" of the inversion, where the corresponding errors are given.

The error bars must be interpreted with care. First of all, they are absolutely meaningless until the minimum of χ^2 is reached, which means that one shouldn't take the error bars seriously until the inversion is done, and only when a minimum has been reached. Moreover, you must understand that these bars provide the uncertainties on the retrieved free parameters, so they should be interpreted in the following manner. NICOLE will assume that the model sought can be constructed from the starting guess model plus a correction to be determined during the inversion. This correction may be a constant value, a straight line, a parabola, or a higher-degree polynomial, depending on how much freedom is given to a particular physical parameter. For example, NICOLE will start considering a parabolic correction to the temperature (which means three inversion nodes), a linear correction to the l.o.s. velocity (two inversion nodes), and a constant (depth-independent) correction to the magnetic field vector and to the microturbulence (only one node). Then, for example, the three error bars that we will obtain for the temperature define the range of parabolas that are compatible with the observations.

3.2.7 The file maskinvert.dat

This file is optional. If it exists, it specifies which points should be inverted and which ones should not. There is one real number per record. A value

of zero signals the code to ignore this (ix, iy) profile. A value of one signals the code to perform the inversion.

3.2.8 Tips for successful inversions

- Initialize with physically sensible models. Since NICOLE will start with constant corrections to the magnetic field vector and microturbulence, and with linear corrections to the l.o.s. velocity, it is a good idea to initialize with models having a constant magnetic field and a constant or linear l.o.s. velocity.
- If it doesn't work the first time, try with a different initialization. If you see clipping warnings before and in between the rejected iterations, that might be signaling that NICOLE needs to perform corrections that are pushing the model out of the allowed range. Take the tmp.mod model, simplify the depth-dependence of the clipped quantity and use it as a new initialization.
- Sometimes it is a good idea to take the result of an inversion, perturb some of the physical variables, and use it as initialization for a new inversion. You may obtain better fits.
- If you are having problems, check the troubleshooting chapter. You may find useful information there. If everything else fails or you think you have found a bug in the code, please send email to hsocas@iac.es.

3.2.9 Restarting an inversion

If you're running an inversion with many points and your computation is interrupted, you can set the Restart option to 'Y' in NICOLE.input and then simply restart the code. It will look at the output files (Chisq.dat, as well as profile and model output) and skip those points with a χ^2 value that is better than the Acceptable Chi-square field in NICOLE.input. If your computer uses a disk cache (and nowadays pretty much they all do), you will lose whatever results were not physically written to file. One way to avoid this is to include Flush statements to make sure the files get written after each inversion. Unfortunately, Flush is not part of the Fortran 90 standard. When creating the makefile, the create_makefile.py script (see next section) will check if Flush is available in your platform and include it in the source code if/where appropriate.

In order to restart an inversion using the previous data, you simply need to include the following option in your NICOLE.input file.

```
Restart= Yes
```

You may also want to set one of Start irec, Start X position or Start Y position.

3.2.10 Debugging and profiling

NICOLE includes a set of tools to help identify and fix problems. We can activate debug mode in NICOLE.input simply by including this line:

```
Debug mode=1
```

Two things happen in this mode. First, a number of files are created with information that allows one to trace back the causes of a problem, especially a crash. One file is created for each parallel process. They are named core_n.txt (where n identifies the process that created it). These files are simply ASCII text files with debug information. For each pixel, after successfully completing a synthesis or an inversion cycle, the file is destroyed. When a crash happens, the core files are left behind with the current code status. Make a note of what process caused the crash (this is usually printed out by the OS) so you know which core file you need to analyze. The second thing that happens in debug mode is that a number of runtime checks are performed to verify the sanity of the current run. When debug mode=1, an abnormal condition results in a wealth of information printed out to the console but the code will continue executing, hoping to recover from this condition. Setting debug mode to a value larger than 1 signals the code to stop upon encountering such abnormal situations so that you can then inspect the core files produced. The idl directory contains a procedure (read_debug.pro) that can be useful to inspect core files. If you have multiple processes, and therefore multiple core files, you'll need to specify which one to read upon invocation of read_debug.pro. The procedure returns a structure with the most interesting variables at the time of the crash and the current model atmosphere being used by the code.

In addition to the debug mode, we can set a number of other flags in NICOLE.input to output some further information. Do NOT use these options in MPI mode:

- **Output populations:** At the end of each synthesis, create a binary file named Populations.dat. This file is overwritten after each synthesis, including those that are part of the inversion process. It doesn't work in parallel mode, as all processes would be writing concurrently in the same file. The file has binary format and has record length nz. If you are doing a NLTE calculation the file contains the populations of all the atomic levels (one record corresponds to one level). After

the converged level populations it contains the LTE populations used as starting guess. In LTE mode, however, the structure of the file is different. The record length is still nz but now there are only two records. The first one contains n/g for the line lower level and then the same for the upper level. If multiple lines are being computed, only the level populations corresponding to the last line is written.

- **Output continuum opacity:** At the end of each synthesis, create a binary file named Cont_opacity.dat. This file is overwritten after each synthesis, including those part of the inversion process. It doesn't work in parallel mode, as all processes would be writing concurrently in the same file. The file has binary format and has two records of length nz. The first record contains the continuum opacity at the wavelength of the last transition. The second record contains the reference continuum opacity at 500 nm.

- **Output NLTE source function:** At the end of each synthesis, create a binary file named NLTE_sf.dat. This file is overwritten after each synthesis, including those part of the inversion process. It doesn't work in parallel mode, as all processes would be writing concurrently in the same file. The file has binary format and has record length nz. The records are arranged nested first in transitions (including bound-bound and bound-free transitions treated in detail), then an internal loop in frequencies from 1 to the number of wavelengths NQ specified in the model atom for that transition. Each record contains the source function used in the NLTE calculation for that transition and frequency.

Profiling is enabled in NICOLE by default. If for some reason you wish to change this, edit the profiling.f90 file in time_code and change the following line near the beginning:

```
Logical, Save :: Do_profile=.True.
```

to

```
Logical, Save :: Do_profile=.False.
```

It is not recommended to disable profiling because it doesn't have any noticeable impact on performance and it produces valuable information that can be used to optimize NICOLE and diagnose possible bottlenecks in code execution. At the end of the run NICOLE produces an ASCII file named profiling__0.txt (in the case of a parallel run there will be one file for each process, each one with the process number n in the file name,

e.g. profiling_4.txt). The contents of this file are self-explanatory, with the total execution time as well as detailed information for each one of the major routines showing the number of times it has been called, the time spent inside it and what percentage of the total execution time it makes up for.

Chapter 4

Compiling NICOLE

4.1 Creating the makefile

NICOLE uses a few routines from the Numerical Recipes book (Press *et al.* 1986). Unfortunately, due to licensing and copyright issues I am not allowed to include the source code of these routines in the distributions, so you will have to obtain them by yourself. You can type the routines directly from the book, obtain the distribution on a CD-ROM or download them from their website http://www.nr.com.

You must place the following routines in the numerical_recipes/ directory: convlv.f90, four1.f90, fourrow.f90, nr.f90, nrtype.f90, nrutil.f90, pythag.f90, realft.f90, svbksb.f90, svdcmp.f90 and twofft.f90.

New in v2.0 NICOLE has a system analyzer written in Python that will examine your system and automatically create a makefile, similarly to the popular autoconf tool for Linux. Go to the main/ directory and try the following:

```
./create_makefile.py
```

This program will analyze your system and make some slight modifications to the source files to adapt it to your architecture. If everything goes well it will prompt you for authorization to create a new makefile overwriting the old one. If you your compiler was not automatically detected or if you wish to specify a different compiler, use the -h command line flag:

```
./create_makefile.py -h
```

If you wish, you can include optimization and debugging flags by using –otherflags. For instance:

```
./create_makefile.py --otherflags='-g -O3'
```

Once you have the makefile, simply type:

```
make clean
make nicole
```

If everything goes well, you will end up with a fresh new executable nicole file in this directory. If you make any modifications to the files with extension .presource, you will need to run create_makefile.py again before you can recompile with make nicole.

4.2 Compiler notes

4.2.1 Mac and GNU Fortran

The Mac versions of GNU fortran prior to 4.4.x have a bug that make NICOLE crash during the tests 2 to 6 (the first test runs satisfactorily). Please, make sure that you are using a recent version of gfortran for Mac (at least 4.4.x) to avoid this bug. Use gfortran –version to find out your compiler version.

4.2.2 Linux and GNU Fortran

GNU fortran versions 4.4.x in Linux produce a NICOLE executable with two problems: 1)the output is buffered so once you launch the code you may not see anything at all until the excution finishes or the buffer is full. Then all the output is flushed to the terminal at the same time. 2) In inversion mode, the output model file will contain random empty records (records filled with 0s). These problems do not appear with versions 4.6.x.

4.2.3 Intel Fortran

The Intel fortran compiler ifort currently has a bug that makes it place all array temporaries on the stack, regardless of their size. An example may be found here:

```
http://stackoverflow.com/questions/12167549/
program-crash-for-array-copy-with-ifort
```

This can make NICOLE crash under some circumstances. A simple workaround is to use the -heap-arrays flag to specify the maximum array size that may be placed on the stack, i.e.:

```
./create_makefile.py --otherflags='-fast -O3 -heap-arrays 1600'
```

4.3 MPI version

In order to compile the MPI version for parallel computers, use the –mpi command line switch for create_makefile:

```
./create_makefile.py --mpi
```

You can also specify optimization flags with –otherflags as before. Then, compile as usual:

```
make clean
make nicole
```

4.4 Testing the code

Once you have successfully built the code, you can test it with a battery of standard problems:

```
cd ../test
./run_tests.py
```

This will launch the code on several different problems, make sure it runs without crashes and analyze the results to ensure that they are correct.

For the parallel MPI build, you will have to use the –nicolecommand switch to specify how the code is executed. The following is an example:

```
cd ../test
./run_tests.py --nicolecommand='mpirun -n 8 ../../main/nicole'
```

4.4.1 Testing in non-interactive mode

This simple script will not work in situations where the code is not run interactively. This is typically the case when one runs in parallel and/or using a queue system. In that case the procedure is slightly more manual (but still manageable).

Start by cleaning out any previously existing results in the test subdirectories by running ./run_tests.py –clean. Then go into syn1/ and execute the Python launcher. Use the –nicolecommand option to specify how to launch a program in that system. For example, to run on two processors using mpich2 on my dual-core laptop I would do:

```
./run_nicole.py --clean
cd syn1/
./run_nicole.py --nicolecommand='mpirun -n 2 ../../main/nicole'
```

If for some reason this procedure doesn't work, just do a dry run with the Python launcher to prepare the input files and then run nicole manually, like this:

```
./run_nicole.py --clean
cd syn1/
./run_nicole.py --nicolecommand='date'
mpirun -n 2 ../../main/nicole
```

Repeat this in each one of the subdirectories under test/ (syn1, syn2, ..., inv3). Then back to test/ and execute run_test with the –check-only switch. This switch is used to preserve the files in the directory (created with your manual runs before) and check the results in those files.

```
cd ..
./run_tests.py --check-only
```

Just to show another example, this is how we would run the tests on the LaPalma supercomputer of the Instituto de Astrofísica de Canarias:

```
cd test/
./run_tests.py --clear
cd syn1/
./run_nicole.py --nicolecommand='mnsubmit ../jobscript.bash'
cd ../syn2
  (...repeat for each directory)
cd ../inv3
./run_nicole.py --nicolecommand='mnsubmit ../jobscript.bash'

cd ..
./run_tests.py --check-only
```

4.5 Compiling in double precision

It is strongly recommended that you run NICOLE in double precision, especially for inversions or in complex NLTE syntheses.

There are plans to support a switch to select the desired precision at compilation time but unfortunately this is not yet possible in the current NICOLE version. However, there is a quick-and-dirty workaround using flags that most compilers incorporate to select the kind of implicit types. For example, the Intel compiler uses the flag -r8 to signal the compiler that all implicit reals should be of kind=8 (i.e., double precision). We can use this flag alongside a small modification to one of the Numerical

Recipes routines to compile NICOLE in double precision. In the directory numerical_recipes, edit the nrtype.f90 file and change the following section:

```
INTEGER, PARAMETER :: SP = KIND(1.0)
INTEGER, PARAMETER :: DP = KIND(1.0D0)
INTEGER, PARAMETER :: SPC = KIND((1.0,1.0))
INTEGER, PARAMETER :: DPC = KIND((1.0D0,1.0D0))
```

to:

```
INTEGER, PARAMETER :: SP = 8
INTEGER, PARAMETER :: DP = 4
INTEGER, PARAMETER :: SPC = 8
INTEGER, PARAMETER :: DPC = 4
```

To compile with the -r8 (or whatever your compiler uses) flag, use the –otherflags switch when creating the makefile (see Section 4.1). In our example:

```
./create_makefile.py --compiler=ifort --otherflags='-fast -r8'
```

For gfortran, the equivalent to -r8 would be -fdefault-real-8 -fdefault-double-8

4.6 Supported platforms

NICOLE has been tested on a number of different platforms, including the following:

- PC Linux (i686), 32 bit with gfortran, ifort and pgf90 compilers
- PC Linux (i686), 64 bit with gfortran, ifort and pgf90 compilers
- Mac OS, Intel 32 bit with gfortran and ifort compilers
- Mac OS, Intel 64 bit with gfortran and ifort compilers
- PowerPC Linux 64 bit with xlf90 compiler

If you have had success compiling and running the code on other platforms, please let me know the details (and whether you had to work around any problems) by sending email to hsocas@iac.es

Chapter 5

The source code

In the full distribution you will find the following subdirectories:

- main: This is where you will find the main program nicole.f90, some relevant subroutines and the makefile. The executable file will be placed here when you compile NICOLE.
- lorien: This contains the LORIEN kernel, consisting of the lorien module (in the lorien.f90 source file) and all the required subroutines. Check the LORIEN documentation included in its distribution (that you can obtain at the C.I.C, homepage) for detailed information on these subroutines.
- forward: You can find here the module needed for solving the forward problem, i.e. given a model atmosphere, synthesize the emergent profiles. The main routine is forward, in the forward.f90 source file.
- compex: Here reside the compress and expand routines (in the compress.f90 and expand.f90 source files), and their associated subroutines. These routines are used to compress a model atmosphere onto a vector of dimensionless free parameters, and vice versa, i.e. to expand this free parameters vector to a model atmosphere.
- time_code: Routines to profile the code (determine where most of the time is spent)
- misc: These are miscellaneous routines used to print out information, read and write models, profiles, etc.
- run: This is the directory where you can run the examples described in this manual. You will find here the input files referenced in chapter 2

- test: After compiling the code, try running the tests in this directory. Simply run the Python script run_tests.py
- numerical_recipes: Place here the source files from the Numerical Recipes book
- idl: Some useful IDL procedures to read/write files in various formats

5.1 The dependency tree

The following diagram illustrates the NICOLE dependency tree fully expanded, showing all the dependencies required by a given file. This can be obtained by running the following command in the main/ directory:

```
./create_makefile.py --showtree
```

```
File:  ../compex/model_struct.f90
....../main/param_struct.f90
........../main/phys_constants.f90
File:  ../compex/select_number_of_nodes.f90
....../compex/nodes_info.f90
........../compex/model_struct.f90
............../main/param_struct.f90
..................../main/phys_constants.f90
....../main/param_struct.f90
........../main/phys_constants.f90
....../compex/model_struct.f90
........../main/param_struct.f90
............../main/phys_constants.f90
File:  ../compex/compex.f90
....../main/param_struct.f90
........../main/phys_constants.f90
....../compex/model_struct.f90
........../main/param_struct.f90
............../main/phys_constants.f90
....../compex/nodes_info.f90
........../compex/model_struct.f90
............../main/param_struct.f90
..................../main/phys_constants.f90
....../forward/bezier.f90
File:  ../compex/compress_variable.f90
File:  ../compex/randomize_model.f90
```

```
....../main/param_struct.f90
............/main/phys_constants.f90
....../compex/nodes_info.f90
............/compex/model_struct.f90
................/main/param_struct.f90
..................../main/phys_constants.f90
....../compex/model_struct.f90
............/main/param_struct.f90
................/main/phys_constants.f90
File:  ../compex/record_to_model.f90
....../compex/model_struct.f90
............/main/param_struct.f90
................/main/phys_constants.f90
File:  ../compex/nodes_info.f90
....../compex/model_struct.f90
............/main/param_struct.f90
................/main/phys_constants.f90
File:  ../forward/atomic_data.f90
File:  ../forward/eq_state.f90
....../time_code/profiling.f90
....../numerical_recipes/nrtype.f90
....../main/param_struct.f90
............/main/phys_constants.f90
....../forward/atomic_data.f90
....../main/debug.f90
....../forward/lte.f90
............/main/param_struct.f90
................/main/phys_constants.f90
............/compex/model_struct.f90
................/main/param_struct.f90
..................../main/phys_constants.f90
............/forward/line_data_struct.f90
............/forward/atomic_data.f90
File:  ../forward/lte.f90
....../main/param_struct.f90
............/main/phys_constants.f90
....../compex/model_struct.f90
............/main/param_struct.f90
................/main/phys_constants.f90
....../forward/line_data_struct.f90
....../forward/atomic_data.f90
File:  ../forward/line_data_struct.f90
```

```
File:  ../forward/zeeman_splitting.f90
File:  ../forward/forward.f90
....../forward/forward_supp.f90
.........../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../forward/line_data_struct.f90
.........../forward/eq_state.f90
.............../time_code/profiling.f90
.............../numerical_recipes/nrtype.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../forward/atomic_data.f90
.............../main/debug.f90
.............../forward/lte.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.................../compex/model_struct.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
.................../forward/line_data_struct.f90
.................../forward/atomic_data.f90
.........../main/debug.f90
......./main/param_struct.f90
.........../main/phys_constants.f90
......./compex/model_struct.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
......./forward/eq_state.f90
.........../time_code/profiling.f90
.........../numerical_recipes/nrtype.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
.........../forward/lte.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.............../forward/line_data_struct.f90
.............../forward/atomic_data.f90
```

```
....../forward/zeeman_splitting.f90
....../forward/lte.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../compex/model_struct.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.........../forward/line_data_struct.f90
.........../forward/atomic_data.f90
....../forward/NLTE/NLTE.f90
.........../misc/file_ops.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../compex/model_struct.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.........../forward/line_data_struct.f90
.........../forward/forward_supp.f90
.............../main/phys_constants.f90
.............../forward/atomic_data.f90
.............../forward/line_data_struct.f90
.............../forward/eq_state.f90
.................../time_code/profiling.f90
.................../numerical_recipes/nrtype.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.................../forward/atomic_data.f90
.................../main/debug.f90
.................../forward/lte.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
......................../compex/model_struct.f90
............................/main/param_struct.f90
................................./main/phys_constants.f90
......................../forward/line_data_struct.f90
......................../forward/atomic_data.f90
.............../main/debug.f90
.........../forward/gauss_quad.f90
.........../forward/lte.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
```

```
.................../main/param_struct.f90
......................../main/phys_constants.f90
............../forward/line_data_struct.f90
............../forward/atomic_data.f90
.........../forward/eq_state.f90
............../time_code/profiling.f90
............../numerical_recipes/nrtype.f90
............../main/param_struct.f90
.................../main/phys_constants.f90
............../forward/atomic_data.f90
............../main/debug.f90
............../forward/lte.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.................../compex/model_struct.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
.................../forward/line_data_struct.f90
.................../forward/atomic_data.f90
.........../forward/background.f90
............../forward/atomic_data.f90
............../main/debug.f90
.........../forward/zeeman_splitting.f90
.........../forward/bezier.f90
.........../main/debug.f90
.........../time_code/profiling.f90
.........../main/phys_constants.f90
......./forward/atomic_data.f90
......./forward/gauss_quad.f90
......./forward/line_data_struct.f90
......./forward/bezier.f90
......./forward/background.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
......./numerical_recipes/nr.f90
.........../numerical_recipes/nrtype.f90
......./misc/file_ops.f90
......./time_code/profiling.f90
......./main/debug.f90
File:  ../forward/gauss_quad.f90
File:  ../forward/hydrostatic.f90
......./main/param_struct.f90
```

```
.........../main/phys_constants.f90
......./compex/model_struct.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
....../forward/eq_state.f90
.........../time_code/profiling.f90
.........../numerical_recipes/nrtype.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
.........../forward/lte.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.............../forward/line_data_struct.f90
.............../forward/atomic_data.f90
....../forward/atomic_data.f90
....../forward/background.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
....../main/debug.f90
....../time_code/profiling.f90
File:  ../forward/background.f90
....../forward/atomic_data.f90
....../main/debug.f90
File:  ../forward/forward_supp.f90
....../main/phys_constants.f90
....../forward/atomic_data.f90
....../forward/line_data_struct.f90
....../forward/eq_state.f90
.........../time_code/profiling.f90
.........../numerical_recipes/nrtype.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
.........../forward/lte.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
```

```
............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
............../forward/line_data_struct.f90
............../forward/atomic_data.f90
....../main/debug.f90
File:  ../forward/bezier.f90
File:  ../forward/ann/ann_pefrompg.f90
....../forward/ann/ann_pefrompg_data.f90
....../main/debug.f90
File:  ../forward/ann/ann_nh2frompe_data.f90
File:  ../forward/ann/ann_background_opacity.f90
....../forward/ann/ann_background_opacity_data.f90
....../time_code/profiling.f90
....../main/debug.f90
....../forward/background.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
File:  ../forward/ann/ann_nh2frompe.f90
....../forward/ann/ann_nh2frompe_data.f90
....../main/debug.f90
File:  ../forward/ann/ann_nhfrompe.f90
....../forward/ann/ann_nhfrompe_data.f90
....../main/debug.f90
File:  ../forward/ann/ann_pgfrompe.f90
....../forward/ann/ann_pgfrompe_data.f90
....../main/debug.f90
File:  ../forward/ann/ann_pgfrompe_data.f90
File:  ../forward/ann/ann_nhfrompe_data.f90
File:  ../forward/ann/ann_background_opacity_data.f90
File:  ../forward/ann/ann_pefrompg_data.f90
File:  ../forward/ann/ANN_forward.f90
File:  ../forward/hyperfine/zeeman_hyperfine.f90
....../forward/line_data_struct.f90
....../forward/hyperfine/hyperfine.f90
.........../numerical_recipes/nr.f90
............../numerical_recipes/nrtype.f90
.........../main/param_struct.f90
............../main/phys_constants.f90
....../main/param_struct.f90
.........../main/phys_constants.f90
....../time_code/profiling.f90
```

```
File:  ../forward/hyperfine/hyperfine.f90
....../numerical_recipes/nr.f90
.........../numerical_recipes/nrtype.f90
....../main/param_struct.f90
.........../main/phys_constants.f90
File:  ../forward/sopa/sopa.f90
File:  ../forward/NLTE/NLTE.f90
....../misc/file_ops.f90
....../main/param_struct.f90
.........../main/phys_constants.f90
....../compex/model_struct.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
....../forward/line_data_struct.f90
....../forward/forward_supp.f90
.........../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../forward/line_data_struct.f90
.........../forward/eq_state.f90
.............../time_code/profiling.f90
.............../numerical_recipes/nrtype.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../forward/atomic_data.f90
.............../main/debug.f90
.............../forward/lte.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.................../compex/model_struct.f90
......................../main/param_struct.f90
............................../main/phys_constants.f90
.................../forward/line_data_struct.f90
.................../forward/atomic_data.f90
.........../main/debug.f90
....../forward/gauss_quad.f90
....../forward/lte.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../compex/model_struct.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.........../forward/line_data_struct.f90
```

```
.........../forward/atomic_data.f90
......./forward/eq_state.f90
.........../time_code/profiling.f90
.........../numerical_recipes/nrtype.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
.........../forward/lte.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................./main/phys_constants.f90
.............../forward/line_data_struct.f90
.............../forward/atomic_data.f90
......./forward/background.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
......./forward/zeeman_splitting.f90
......./forward/bezier.f90
......./main/debug.f90
......./time_code/profiling.f90
......./main/phys_constants.f90
File:  ../main/phys_constants.f90
File:  ../main/param_struct.f90
......./main/phys_constants.f90
File:  ../main/nicole.f90
......./main/param_struct.f90
.........../main/phys_constants.f90
......./forward/line_data_struct.f90
......./misc/nicole_input.f90
......./compex/compex.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../compex/model_struct.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.........../compex/nodes_info.f90
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................./main/phys_constants.f90
```

```
.........../forward/bezier.f90
....../compex/model_struct.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
....../lorien/lorien.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../compex/compex.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.............../compex/nodes_info.f90
.................../compex/model_struct.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
.............../forward/bezier.f90
.........../forward/forward.f90
.............../forward/forward_supp.f90
.................../main/phys_constants.f90
.................../forward/atomic_data.f90
.................../forward/line_data_struct.f90
.................../forward/eq_state.f90
......................../time_code/profiling.f90
......................../numerical_recipes/nrtype.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
......................../forward/atomic_data.f90
......................../main/debug.f90
......................../forward/lte.f90
............................/main/param_struct.f90
................................/main/phys_constants.f90
............................/compex/model_struct.f90
................................/main/param_struct.f90
.................................../main/phys_constants.f90
............................/forward/line_data_struct.f90
............................/forward/atomic_data.f90
.................../main/debug.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
```

```
................/main/param_struct.f90
....................../main/phys_constants.f90
............/forward/eq_state.f90
................/time_code/profiling.f90
................/numerical_recipes/nrtype.f90
................/main/param_struct.f90
..................../main/phys_constants.f90
................/forward/atomic_data.f90
................/main/debug.f90
................/forward/lte.f90
..................../main/param_struct.f90
......................../main/phys_constants.f90
..................../compex/model_struct.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
..................../forward/line_data_struct.f90
..................../forward/atomic_data.f90
............/forward/zeeman_splitting.f90
............/forward/lte.f90
................/main/param_struct.f90
..................../main/phys_constants.f90
................/compex/model_struct.f90
..................../main/param_struct.f90
......................../main/phys_constants.f90
................/forward/line_data_struct.f90
................/forward/atomic_data.f90
............/forward/NLTE/NLTE.f90
................/misc/file_ops.f90
................/main/param_struct.f90
..................../main/phys_constants.f90
................/compex/model_struct.f90
..................../main/param_struct.f90
......................../main/phys_constants.f90
................/forward/line_data_struct.f90
................/forward/forward_supp.f90
..................../main/phys_constants.f90
..................../forward/atomic_data.f90
..................../forward/line_data_struct.f90
..................../forward/eq_state.f90
......................../time_code/profiling.f90
......................../numerical_recipes/nrtype.f90
......................../main/param_struct.f90
```

```
.............................../main/phys_constants.f90
........................../forward/atomic_data.f90
........................../main/debug.f90
........................../forward/lte.f90
.............................../main/param_struct.f90
................................../main/phys_constants.f90
.............................../compex/model_struct.f90
................................../main/param_struct.f90
..................................../main/phys_constants.f90
.............................../forward/line_data_struct.f90
.............................../forward/atomic_data.f90
......................../main/debug.f90
..................../forward/gauss_quad.f90
..................../forward/lte.f90
......................../main/param_struct.f90
........................../main/phys_constants.f90
......................../compex/model_struct.f90
........................../main/param_struct.f90
.............................../main/phys_constants.f90
......................../forward/line_data_struct.f90
......................../forward/atomic_data.f90
..................../forward/eq_state.f90
......................../time_code/profiling.f90
......................../numerical_recipes/nrtype.f90
......................../main/param_struct.f90
........................../main/phys_constants.f90
......................../forward/atomic_data.f90
......................../main/debug.f90
......................../forward/lte.f90
........................../main/param_struct.f90
.............................../main/phys_constants.f90
........................../compex/model_struct.f90
.............................../main/param_struct.f90
................................../main/phys_constants.f90
........................../forward/line_data_struct.f90
........................../forward/atomic_data.f90
..................../forward/background.f90
......................../forward/atomic_data.f90
......................../main/debug.f90
..................../forward/zeeman_splitting.f90
..................../forward/bezier.f90
..................../main/debug.f90
```

```
.................../time_code/profiling.f90
.................../main/phys_constants.f90
............../forward/atomic_data.f90
............../forward/gauss_quad.f90
............../forward/line_data_struct.f90
............../forward/bezier.f90
............../forward/background.f90
.................../forward/atomic_data.f90
.................../main/debug.f90
............../numerical_recipes/nr.f90
.................../numerical_recipes/nrtype.f90
............../misc/file_ops.f90
............../time_code/profiling.f90
............../main/debug.f90
.........../compex/model_struct.f90
............../main/param_struct.f90
.................../main/phys_constants.f90
.........../forward/line_data_struct.f90
.........../compex/nodes_info.f90
............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.........../time_code/profiling.f90
.........../main/debug.f90
....../forward/forward.f90
.........../forward/forward_supp.f90
............../main/phys_constants.f90
............../forward/atomic_data.f90
............../forward/line_data_struct.f90
............../forward/eq_state.f90
.................../time_code/profiling.f90
.................../numerical_recipes/nrtype.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.................../forward/atomic_data.f90
.................../main/debug.f90
.................../forward/lte.f90
......................../main/param_struct.f90
........................../main/phys_constants.f90
......................../compex/model_struct.f90
........................../main/param_struct.f90
.............................../main/phys_constants.f90
```

```
....................../forward/line_data_struct.f90
....................../forward/atomic_data.f90
............../main/debug.f90
.........../main/param_struct.f90
............../main/phys_constants.f90
.........../compex/model_struct.f90
............../main/param_struct.f90
.................../main/phys_constants.f90
.........../forward/eq_state.f90
............../time_code/profiling.f90
............../numerical_recipes/nrtype.f90
............../main/param_struct.f90
.................../main/phys_constants.f90
............../forward/atomic_data.f90
............../main/debug.f90
............../forward/lte.f90
.................../main/param_struct.f90
......................./main/phys_constants.f90
.................../compex/model_struct.f90
......................./main/param_struct.f90
........................./main/phys_constants.f90
.................../forward/line_data_struct.f90
.................../forward/atomic_data.f90
.........../forward/zeeman_splitting.f90
.........../forward/lte.f90
............../main/param_struct.f90
.................../main/phys_constants.f90
............../compex/model_struct.f90
.................../main/param_struct.f90
......................./main/phys_constants.f90
............../forward/line_data_struct.f90
............../forward/atomic_data.f90
.........../forward/NLTE/NLTE.f90
............../misc/file_ops.f90
............../main/param_struct.f90
.................../main/phys_constants.f90
............../compex/model_struct.f90
.................../main/param_struct.f90
......................./main/phys_constants.f90
............../forward/line_data_struct.f90
............../forward/forward_supp.f90
.................../main/phys_constants.f90
```

```
................./forward/atomic_data.f90
................./forward/line_data_struct.f90
................./forward/eq_state.f90
....................../time_code/profiling.f90
....................../numerical_recipes/nrtype.f90
....................../main/param_struct.f90
........................../main/phys_constants.f90
....................../forward/atomic_data.f90
....................../main/debug.f90
....................../forward/lte.f90
........................../main/param_struct.f90
.............................../main/phys_constants.f90
........................../compex/model_struct.f90
.............................../main/param_struct.f90
.................................../main/phys_constants.f90
........................../forward/line_data_struct.f90
........................../forward/atomic_data.f90
................../main/debug.f90
.............../forward/gauss_quad.f90
.............../forward/lte.f90
................../main/param_struct.f90
....................../main/phys_constants.f90
................../compex/model_struct.f90
....................../main/param_struct.f90
........................../main/phys_constants.f90
................../forward/line_data_struct.f90
................../forward/atomic_data.f90
.............../forward/eq_state.f90
................../time_code/profiling.f90
................../numerical_recipes/nrtype.f90
................../main/param_struct.f90
....................../main/phys_constants.f90
................../forward/atomic_data.f90
................../main/debug.f90
................../forward/lte.f90
....................../main/param_struct.f90
........................../main/phys_constants.f90
....................../compex/model_struct.f90
........................../main/param_struct.f90
.............................../main/phys_constants.f90
....................../forward/line_data_struct.f90
....................../forward/atomic_data.f90
```

```
.............../forward/background.f90
.................../forward/atomic_data.f90
.................../main/debug.f90
.............../forward/zeeman_splitting.f90
.............../forward/bezier.f90
.............../main/debug.f90
.............../time_code/profiling.f90
.............../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../forward/gauss_quad.f90
.........../forward/line_data_struct.f90
.........../forward/bezier.f90
.........../forward/background.f90
.............../forward/atomic_data.f90
.............../main/debug.f90
.........../numerical_recipes/nr.f90
.............../numerical_recipes/nrtype.f90
.........../misc/file_ops.f90
.........../time_code/profiling.f90
.........../main/debug.f90
....../compex/nodes_info.f90
.........../compex/model_struct.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
....../forward/atomic_data.f90
....../misc/file_ops.f90
....../main/debug.f90
....../time_code/profiling.f90
....../forward/background.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
File:  ../main/debug.f90
File:  ../time_code/profiling.f90
File:  ../misc/checknan.f90
File:  ../misc/tolower.f90
File:  ../misc/parab_interp.f90
File:  ../misc/get_lun.f90
File:  ../misc/myflush.f90
File:  ../misc/roman_to_int.f90
File:  ../misc/set_params.f90
....../misc/nicole_input.f90
....../main/param_struct.f90
```

```
.........../main/phys_constants.f90
File:  ../misc/tau_to_z.f90
......./main/param_struct.f90
.........../main/phys_constants.f90
......./compex/model_struct.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
......./forward/background.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
File:  ../misc/write_profile.f90
......./main/param_struct.f90
.........../main/phys_constants.f90
......./forward/line_data_struct.f90
File:  ../misc/read_weights.f90
......./main/param_struct.f90
.........../main/phys_constants.f90
File:  ../misc/write_direct.f90
File:  ../misc/z_to_tau.f90
......./main/param_struct.f90
.........../main/phys_constants.f90
......./forward/eq_state.f90
.........../time_code/profiling.f90
.........../numerical_recipes/nrtype.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../main/debug.f90
.........../forward/lte.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.............../forward/line_data_struct.f90
.............../forward/atomic_data.f90
......./compex/model_struct.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
......./forward/atomic_data.f90
......./forward/background.f90
.........../forward/atomic_data.f90
```

```
.../main/debug.f90
...../main/debug.f90
File:  ../misc/file_ops.f90
File:  ../misc/printout.f90
...../main/param_struct.f90
.../main/phys_constants.f90
...../compex/model_struct.f90
.../main/param_struct.f90
.../main/phys_constants.f90
...../forward/line_data_struct.f90
File:  ../misc/cubic_interp.f90
File:  ../misc/los_to_local.f90
File:  ../misc/smoothed_lines.f90
File:  ../misc/local_to_los.f90
File:  ../misc/read_next_nocomment.f90
File:  ../misc/nicole_input.f90
File:  ../misc/read_profile.f90
...../main/param_struct.f90
.../main/phys_constants.f90
File:  ../misc/compute_weights.f90
...../main/param_struct.f90
.../main/phys_constants.f90
File:  ../misc/los_to_local_errors.f90
File:  ../misc/write_model.f90
...../main/param_struct.f90
.../main/phys_constants.f90
...../compex/model_struct.f90
.../main/param_struct.f90
.../main/phys_constants.f90
...../misc/file_ops.f90
...../forward/line_data_struct.f90
File:  ../misc/toupper.f90
File:  ../numerical_recipes/twofft.f90
...../numerical_recipes/nrtype.f90
...../numerical_recipes/nrutil.f90
.../numerical_recipes/nrtype.f90
...../numerical_recipes/nr.f90
.../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/added.f90
...../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/lubksb.f90
...../numerical_recipes/nrtype.f90
```

```
....../numerical_recipes/nrutil.f90
.........../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/lubksb_dp.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
.........../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/svbksb.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
.........../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/realft.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
.........../numerical_recipes/nrtype.f90
....../numerical_recipes/nr.f90
.........../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/svdcmp.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
.........../numerical_recipes/nrtype.f90
....../main/debug.f90
....../numerical_recipes/nr.f90
.........../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/nr.f90
....../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/pythag.f90
....../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/ludcmp_dp.f90
....../numerical_recipes/nrtype.f90
....../main/debug.f90
....../numerical_recipes/added.f90
.........../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
.........../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/four1.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
.........../numerical_recipes/nrtype.f90
....../numerical_recipes/nr.f90
.........../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/convlv.f90
....../numerical_recipes/nrtype.f90
```

```
....../numerical_recipes/nrutil.f90
............/numerical_recipes/nrtype.f90
....../numerical_recipes/nr.f90
............/numerical_recipes/nrtype.f90
File:  ../numerical_recipes/fourrow.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
............/numerical_recipes/nrtype.f90
File:  ../numerical_recipes/tqli.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
............/numerical_recipes/nrtype.f90
....../numerical_recipes/nr.f90
............/numerical_recipes/nrtype.f90
File:  ../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/nrutil.f90
....../numerical_recipes/nrtype.f90
File:  ../numerical_recipes/ludcmp.f90
....../numerical_recipes/nrtype.f90
....../numerical_recipes/nrutil.f90
............/numerical_recipes/nrtype.f90
File:  ../lorien/SVD_solve.f90
....../numerical_recipes/nr.f90
............/numerical_recipes/nrtype.f90
....../main/debug.f90
File:  ../lorien/lorien.f90
....../main/param_struct.f90
............/main/phys_constants.f90
....../compex/compex.f90
............/main/param_struct.f90
................/main/phys_constants.f90
............/compex/model_struct.f90
................/main/param_struct.f90
....................../main/phys_constants.f90
............/compex/nodes_info.f90
................/compex/model_struct.f90
....................../main/param_struct.f90
............................/main/phys_constants.f90
............/forward/bezier.f90
....../forward/forward.f90
............/forward/forward_supp.f90
................/main/phys_constants.f90
```

```
.............../forward/atomic_data.f90
.............../forward/line_data_struct.f90
.............../forward/eq_state.f90
.................../time_code/profiling.f90
.................../numerical_recipes/nrtype.f90
.................../main/param_struct.f90
...................../main/phys_constants.f90
.................../forward/atomic_data.f90
.................../main/debug.f90
.................../forward/lte.f90
...................../main/param_struct.f90
........................../main/phys_constants.f90
...................../compex/model_struct.f90
........................../main/param_struct.f90
............................../main/phys_constants.f90
...................../forward/line_data_struct.f90
...................../forward/atomic_data.f90
.............../main/debug.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
.........../compex/model_struct.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.........../forward/eq_state.f90
.............../time_code/profiling.f90
.............../numerical_recipes/nrtype.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../forward/atomic_data.f90
.............../main/debug.f90
.............../forward/lte.f90
.................../main/param_struct.f90
...................../main/phys_constants.f90
.................../compex/model_struct.f90
...................../main/param_struct.f90
......................../main/phys_constants.f90
.................../forward/line_data_struct.f90
.................../forward/atomic_data.f90
.........../forward/zeeman_splitting.f90
.........../forward/lte.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
```

```
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.............../forward/line_data_struct.f90
.............../forward/atomic_data.f90
.............../forward/NLTE/NLTE.f90
.............../misc/file_ops.f90
.............../main/param_struct.f90
.................../main/phys_constants.f90
.............../compex/model_struct.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.............../forward/line_data_struct.f90
.............../forward/forward_supp.f90
.................../main/phys_constants.f90
.................../forward/atomic_data.f90
.................../forward/line_data_struct.f90
.................../forward/eq_state.f90
......................../time_code/profiling.f90
......................../numerical_recipes/nrtype.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
......................../forward/atomic_data.f90
......................../main/debug.f90
......................../forward/lte.f90
............................/main/param_struct.f90
.............................../main/phys_constants.f90
............................/compex/model_struct.f90
.............................../main/param_struct.f90
.................................../main/phys_constants.f90
............................/forward/line_data_struct.f90
............................/forward/atomic_data.f90
.................../main/debug.f90
.............../forward/gauss_quad.f90
.............../forward/lte.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.................../compex/model_struct.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
.................../forward/line_data_struct.f90
.................../forward/atomic_data.f90
```

```
.............../forward/eq_state.f90
.................../time_code/profiling.f90
.................../numerical_recipes/nrtype.f90
.................../main/param_struct.f90
......................../main/phys_constants.f90
.................../forward/atomic_data.f90
.................../main/debug.f90
.................../forward/lte.f90
......................../main/param_struct.f90
............................/main/phys_constants.f90
......................../compex/model_struct.f90
............................/main/param_struct.f90
................................./main/phys_constants.f90
......................../forward/line_data_struct.f90
......................../forward/atomic_data.f90
.............../forward/background.f90
.................../forward/atomic_data.f90
.................../main/debug.f90
.............../forward/zeeman_splitting.f90
.............../forward/bezier.f90
.............../main/debug.f90
.............../time_code/profiling.f90
.............../main/phys_constants.f90
.........../forward/atomic_data.f90
.........../forward/gauss_quad.f90
.........../forward/line_data_struct.f90
.........../forward/bezier.f90
.........../forward/background.f90
.............../forward/atomic_data.f90
.............../main/debug.f90
.........../numerical_recipes/nr.f90
.............../numerical_recipes/nrtype.f90
.........../misc/file_ops.f90
.........../time_code/profiling.f90
.........../main/debug.f90
......../compex/model_struct.f90
.........../main/param_struct.f90
.............../main/phys_constants.f90
......../forward/line_data_struct.f90
......../compex/nodes_info.f90
.........../compex/model_struct.f90
.............../main/param_struct.f90
```

```
.................../main/phys_constants.f90
....../time_code/profiling.f90
....../main/debug.f90
File:  ../lorien/adjust_wave_grid.f90
....../main/param_struct.f90
.........../main/phys_constants.f90
```

Chapter 6

Geometry of the magnetic field

The inclination and azimuth given in the atmospheric models as B_local_inc and B_local_azi refer to the local solar frame. This means that the inclination is given with respect to the solar vertical and the azimuth is given with respect to the local E-W solar direction (measured counterclock-wise from the West). When you specify a heliocentric observation angle, NICOLE will use appropriate coordinates conversions to calculate the apparent inclination and azimuth (the inclination and azimuth with respect to the line of sight) so that the emergent profiles obtained are those that one would actually observe.

Unfortunately, a fully detailed calculation would be very complex, and would require the user to introduce information such as the solar pitch angle, which is not always available. Therefore, for a sake of simplicity and convenience, I have decided to assume that the observed spot lies on the E-W line that passes through the center of the solar disk. In this manner, only the heliocentric angle needs to be specified and the coordinates conversion equations simplify to:

$$\cos\gamma = \sin\gamma' \cos\chi' sin\theta + \cos\gamma' cos\theta, \tag{6.1}$$

$$\chi = \chi' \tag{6.2}$$

where γ and χ are the local inclination and azimuth, γ' and χ' are the line-of-sight inclination and azimuth, and θ is the heliocentric angle of the observations. Simple expressions are also obtained for the conversion of the errors in these quantities.

$$\delta(\cos\gamma) = -\sin\gamma\delta\gamma = \cos\gamma'\cos\chi'\sin\theta\delta\gamma' - \sin\gamma'\sin\chi'\sin\theta\delta\chi' - \sin\gamma'\cos\theta\delta\gamma' \,. \tag{6.3}$$

The above-mentioned approximation has no effect on the inferred magnetic inclination, but the magnetic azimuth resulting from an inversion will be correct only if we are observing a location on the solar disk equator. If this is not the case, there will be a constant offset in the azimuth that can be corrected *a posteriori*. As a first-order correction, you may want to add the heliocentric azimuth (i.e., the azimuth of the observed spot measured counterclock-wise from the West direction of the disk equator) to the inferred azimuth.

Chapter 7

Troubleshooting

Below are some problems that you might run into from time to time:

- **I have problem compiling file with MPI.** The current version of the code does not work with OpenMPI. If OpenMPI is the only MPI version available on your machine, you have to install mpich or Intel-MPI locally in your home directory. to be sure that create_makefile.py is using the correct version of the code, do the following: Execute

  ```
  which mpif90
  ```

  If the result is not the correct MPI version, to fix that set the path to it in your .bashrc file and execute

  ```
  source .bashrc.
  ```

  Create make file with:

  ```
  ./create_makefile --compiler='/absolutepathtoyourmpi/bin/mpif90
  ```

  Be sure that the –mpi flag is NOT specified in this case.

- **I get an error about an incorrect magic number in a python script.** Remove all the .pyc files in your directory. These are compiled binaries that Python creates to speed things up. Perhaps the .pyc file it tried to read was created by a different machine.

- **The code crashes on startup.** This is probably caused by an incorrect format of one of the input files. Try replacing your input files, one by one, with the examples included in your distribution. This way you can track down which file is causing the problem. Then, edit the file and check with the corresponding section in this manual to make sure that the format is exactly the one expected by NICOLE.
- **The NLTE iteration doesn't converge.** Try playing around with the parameters in the NLTE section of NICOLE.input. The first thing to try would be to use linear formal solution. If that doesn't work, try changing the formal solution.
- **My synthesis doesn't produce the results I expect.** Perhaps NICOLE is modifying some of the model parameters that you are supplying in the model (pressure, density, Hydrogen level populations, etc). Check the discussion at the end of section 3.1.7.
- **I can't get a good fit to my profiles.** Check the tips on section 3.2.8. If that doesn't help, make sure that your profiles are properly normalized.
- **Something's weird with my inversions** Do you have a leftover Restart=Yes field in your NICOLE.input?
- **I get "line not optically thin at the surface" errors during my inversions.** This error means that your model does not extend high enough. You should add more layers on top of it. For Fe I 6301 and 6302 you should go up to about $\log(\tau_{5000})=-4.5$.
- **I get "line not optically thick at the bottom" errors during my inversions.** This error means that your model does not extend low enough. You should add more layers underneath it. Normally you should be fine if you go below $\log(\tau_{5000})=2$.
- **I get "hydrostatic inaccuracy" errors during my inversions.** This usually happens when temperatures go very low (or there is a steep temperature drop) in the upper layers. Your lines are normally quite insensitive to these upper layers and that is why they do weird things from time to time. But if you cut them out, you may start getting the "not optically thin" errors mentioned above. My advice is that you start iterating with this model, even if the lines are not getting optically thin at the surface. This means that your final result will not be accurate, but it will probably be close enough. Now add the extra layers on top of your model and invert again. That should get rid of the hydrostatic inaccuracy problem.

- **I made a synthesis using a quiet Sun simulation cube from my favorite MHD code and the average continuum is not at 1**. NICOLE uses a set of opacities that are, in general, different from those in the MHD code used to produce the simulation. This means that the $\tau = 1$ level is not the same (assuming that you're using a geometrical height scale) and therefore one is seeing a slightly different layer. This is not really an error, it's just the result of combining results from two different codes and we need a different interpretation. Instead of relying on the NICOLE normalization it would be better to compute intensities from a quiet Sun snapshot and use its average value as the normalization reference.

- **I modified the source code and now something is going wrong.** Try using the make clean command before compiling.

- **I have read through this manual and the troubleshooting section, and I still can't find a solution for my problem.** Ok, if you can't figure it out send an e-mail at hsocas@iac.es.

Chapter 8

Version history

- v0.9 Original version, based on LILIA v4.1
- v1.1 Departure coefficients, instrumental profile, multiple synthesis mode

... and at that point I stopped keeping track. Using GIT since v2.70 for version control

After v2.72, the version numbering scheme has changed to the format YY.MM which represents the release year and month. The first such release was 13.12 in December 2013.

Chapter 9

Bibliography

- Asensio Ramos, A., Trujillo Bueno, J., Carlsson, M., Cernicharo, J. ApJ 2003, 588, L61
- Bellot Rubio, L.R., Ruiz Cabo, B., and Collados Vera, M. A&A 1998, 506, 805.
- Barkelm, P. S., Piskunov, N., and O'Mara, B. J. AAS 2000, 142, 467.
- Kostik, R. I., Shchukina, N. G., and Rutten, R. J. 1996, A&A, 305, 325.
- Mihalas, D. in Methods in Computational Physics, Vol. 7, Alder, B. (ed.), New York; London; Academic Press 1967
- Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. 1986, Numerical Recipes (Cambridge: Cambridge U niv. Press)
- Ruiz Cabo, B., and del Taro Iniesta, J. C. 1992, ApJ, 398, 375.
- Rybicki, G. B., & Hummer, D. G. 1991, A&A, 245, 171.
- Sánchez Almeida, J., and Trujillo Bueno, J. 1999, ApJ, 526, 1013.
- Scharmer, G. B., and Carlsson, M. 1985, J. Comput. Phys., 50, 56.
- Socas-Navarro, H., de la Cruz Rodríguez, J., Asensio Ramos, A., Trujillo Bueno, J. and Ruiz Cobo, B. 2014, A&A, *in preparation.*
- Socas-Navarro, H. and Trujillo Bueno, J. 1998, ApJ, 490, 383.
- Unsold 1955, Physik der Sternatmospharen, 2nd ed