ESTER Users manual

September 22, 2012

Contents

Chapter 1

Getting started

1.1 Prerequisites

The ESTER library depends on some external libraries that should be installed in the system, namely:

- BLAS and LAPACK, for matrix algebra
- PGPLOT (CPGPLOT) for graphics output

The library PGPLOT is optative and can be deactivated in the `Makefile` (`make.inc`) setting the variable `USE_PGPLOT=0`.

1.2 Installation

Unpack the file of the distribution (normally `ester.tar.gz`) in the desired directory

```
root$ tar zxvf ester.tar.gz
```

Go to the directory `ester/src`

```
root$ cd ester/src
```

To clean out any previous installation we can do

```
root/ester/src$ make distclean
```

First we have to compile the third-party libraries included in the distribution. This only have to be done the first time (or after a `distclean`). At the moment there is only one library that needs to be build, the package for interpolating opacity tables created by Günter Houdek. After checking the `Makefile` located at `ester/tables/houdek/v9` we do

```
root/ester/src$ make tables
```

Now, we are prepared to compile the main library. The main variables for the compilation should be defined in a file named `make.inc` . Two examples are included, `make.inc.icc` and `make.inc.gcc`, for the Intel compiler and the GNU compiler respectively (tested under Ubuntu 12.04). After setting the proper values for the compilation we execute `make all` or just

```
root/ester/src$ make
```

To remove intermediate files we can also do

```
root/ester/src$ make clean
```

The main library is created in `ester/lib/libester.so`, the include files are in `ester/include` and the executables in `ester/bin`. We can add the latter directory to the system path, for the *bash* shell

```
$ export PATH="your_root_directory/ester/bin:$PATH"
```

or include this line in your `.bashrc` file.

1.3 Checking the installation

To check the functionality of the program we are going to calculate the structure of a star using the default values for the parameters. First we calculate the structure of the corresponding 1D non-rotating star. Change to your working directory and execute

```
$ star1d
```

Then we use the output file (by default `star.out`) as the starting point for the 2D calculation

```
$ star2d -i star.out -Omega_bk 0.7
```

This calculates the structure of a star rotating at 70% of the break-up velocity $\Omega_k = \sqrt{\frac{GM}{R_e^3}}$.

1.4 Using the library

To use the ESTER library in a C program you should write at the beginning of your source file

```
#include "ester.h"
```

To facilitate the process of compiling and linking against the library and all its dependencies, we provide an automatically generated script `ester/utils/ester_build` so, all you have to do is

```
$ ester_build your_cpp_program.cpp -o your_executable
```

Chapter 2

Basic usage

There are three executables provided with the library:

- `star1d` For calculating the structure of a 1D non-rotating star
- `star2d` For calculating the structure of a 2D rotating star
- `gen_output` For generating a custom output file

2.1 Configuration files

The main configuration file for `star1d` and `star2d` is `ester/config/star.cfg`. This file contains the main options for the program, which are

- `maxit` (default 200). Maximum number of iterations. After `maxit` iterations, the program exists normally and the output file is saved, even if it has not completely converged.
- `minit` (default 1). Minimum number of iterations. It may occur that the value of the error for the first iteration is not representative. With this parameter we force the solver to do at least `minit` iterations. This parameter is superseded by `maxit`, for example if `maxit=5` and `minit=10`, the solver will do only 5 iterations.
- `tol` (default 1e-8). The relative tolerance for checking the convergence of the model.
- `newton_dmax` (default 0.5). After one step of the Newton's method, the maximum relative change allowed for a variable is given by `newton_dmax`. If necessary the iteration is relaxed by a parameter h

  $$\vec{x}^{N+1} = \vec{x}^N + h\delta\vec{x}^N$$

  according to this value. This parameter can be used to stabilize the convergence when the initial estimation is far from the solution.
- `output_file` (default `star.out`). Name of the output file.
- `output_mode` (default `b`). Type of the output file `b` for binary and `t` for text output.
- `verbose` (default 1). Level of verbosity, from 0 (quiet) to 4.

- `plot_device` (default `/NULL`. Plotting device for PGPLOT, see the documentation of PGPLOT for details. For output in a X window use `/XSERVE`. To disable the graphic output use `/NULL`.
- `plot_interval` (default 10). Minimum time in seconds to update the graphic output.

All this options can be specified in the file `ester/config/star.cfg` in the form `option_name=option_value` (one per line) and in the command line as `-option_name option_value`. The options specified in the command line have precedence over those specified in the configuration file.

There are some additional options that can be included in the command line:

`-input_file` *infile*. Use the file *infile* as the starting point for the iteration.

`-i` *infile*. Same as `-input_file` *infile*.

`-o` *outfile*. Same as `-output_file` *outfile*.

`-param_file` *file*. Where *file* contains the parameters of the stellar model to be calculated (see below).

`-p` *file*. Same as `-param_file` *file*.

`-ascii`. Same as `-output_mode t`.

`-binary`. Same as `-output_mode b`.

`-noplot`. Same as `-plot_device /NULL`.

`-v`*n*. Same as `-verbose` *n*.

2.2 `star1d` input parameters

The input parameters for `star1d` can be passed in the command line or in a text file specified with the option `-param_file` *file* (or just `-p` *file*). It can also be used simultaneously, in this case the parameters given in the command line take precedence over those specified in the file. In the text file they are written in the form `param_name`=*param_value* and in the command line as `-param_name` *param_value*. Here is the list of valid parameters

- `ndomains`. The number of subdomains to use.
- `npts`. Number of points in each subdomain. It is specified as a comma-separated list. If only one value is specified, it will be used for all the subdomains, for example:

  ```
  star1d -ndomains 4 -npts 20,20,20,20
  ```

  is equivalent to

  ```
  star1d -ndomains 4 -npts 20
  ```

- `xif`. The position of each subdomain as a comma-separated list. The list should contain the first and the last points of the entire domain (that should be 0 and 1), having a total of `ndomains`+1 values. If only one value is specified (γ), the positions are calculated using the formula

$$\texttt{xif(i)} = 1 - \left(1 - \frac{i}{\texttt{ndomains}}\right)^{\gamma} \quad \gamma > 0$$

where $\gamma = 1$ corresponds to equally-spaced subdomains, for $\gamma > 1$ they are more concentrated near the surface, and the opposite for $\gamma < 1$.

- `M`. The mass in units of solar mass.
- `X`. Mass fraction of hydrogen.
- `Z`. Mass fraction of metals.
- `Xc`. Fraction of the hydrogen abundance present in the convective core. The profile of hydrogen abundance will be in the form

$$X(\vec{r}) = \begin{cases} \texttt{X} \times \texttt{Xc} & \text{if } \vec{r} \text{ is in the convective core} \\ \texttt{X} & \text{otherwise} \end{cases}$$

  If there is no convective core, this parameter is ignored.
- `conv`. The number of subdomains within the convective core. If `conv=0` the model will be completely radiative.
- `surff`. This parameter is used for truncating the stellar model at some point below the surface. The surface pressure will be `surff` times the "real value and the boundary conditions will be adjusted in consequence. This parameter is provided only for testing purposes as it does not produce an accurate representation of the internal layers of the star. For regular calculations it should be `surff=1`.
- `Tc`. Initial estimation of the central temperature. To be updated during the calculation.
- `pc`. Initial estimation of the central pressure. To be updated during the calculation.
- `opa`. Type of opacity law. Possible values are:
  - `opal`. OPAL opacities.
  - `houdek`. Houdek's interpolation of OPAL opacities (smoother).
  - `kramer`. Kramer's opacity.
- `eos`. Type of equation of state. Possible values are:
  - `opal`. OPAL equation of state.
  - `ideal`. Ideal gas.
  - `ideal+rad`. Ideal gas with radiation.
- `nuc`. Type of nuclear reactions. At the moment, only `simple` is implemented.
- `atm`. Type of atmosphere. At the moment, only `simple` is implemented.

If some parameters are omitted, the program will take the value from the input file (set with `-input_file` or `-i`) or from the default parameters file in `ester/config/1d_default.par` when no input file is specified.

At the moment, the code does not permit to change the number of domains and/or their position when using an input file.

2.3 `star2d` input parameters

The program `star2d` admits the same parameters than `star1d` plus some extra specific options:

- `nth`. The number of grid points in the latitude.
- `nex`. Nuber of radial points in the external domain.
- `Omega_bk`. Angular velocity at the equator in units of the critical velocity $\Omega_c = \sqrt{\frac{GM}{R_e^3}}$.
- `Ekman`. Ekman number.

The default values are written in the file `ester/config/2d_default.par`. Note that the input of `star2d` can be a non-rotating 1D model calculated with `star1d`.

2.3.1 Some recipes

The typical workflow to calculate a model starts with the calculation of the corresponding 1D model and using it as an input for `star2d`. For example, to calculate the structure of a $5M_\odot$ star with OPAL opacity rotating at with $\Omega = 0.7\Omega_c$ we can do:

```
$ star1d -M 5 -opa opal -o model1d
$ star2d -i model1d -nth 24 -Omega_bk 0.7 -o model2d
```

As the code uses the Newton's method, sometimes it is not possible to converge to a solution if the initial estimation is too far from it. In this case we can use some intermediate steps. For example, if we want to calculate the structure of a $2.5M_\odot$ star rotating with $\Omega = 0.9\Omega_c$, we should probably do

`$ star1d -M 2.5 -o model1d -conv 0`	(Deactivate core convection to improve convergence)
`$ star1d -i model1d -o model1d -conv 1`	(Re-activate core convection)
`$ star2d -i model1d -nth 24 -Omega_bk 0.7 -o model2d`	(Using an intermediate value for rotation)
`$ star2d -i model2d -nth 32 -Omega_bk 0.9 -o model2d`	(Calculating the final model)

Executing `star2d` with `maxit=0` can be used to interpolate a model without recalculating it.

`$ star2d -i model -npts` *npts_new* `-nth` *nth_new* `-o model_interp -maxit 0`

Pressing Ctrl-C at any time during the execution of `star2d` will terminate the program, giving the possibility of finish the current iteration and write the result in the output file.

2.4 Generating custom output files

The output files generated by `star1d` and `star2d` contain just the minimal information necessary to reconstruct the code. However, sometimes a more detailed output is required. This can be done using the program `gen_output` included in the distribution. This program reads a template from the standard input and write the result in the standard output. A typical call would be

`$ gen_output` *model_file* `<` *template_file* `>` *output_file*

The template file is a regular text file with the following rules:

- Plain text are copied from the template to the output file. It cannot contain the reserved characters `$` and \.
- Line breaks are ignored. To insert a line break in the output file we have to insert a blank line in the template.
- Variables from the model are written in the form `${`*var*`,`*fmt*`}`, where *var* is the code for the variable (see table below) and *fmt* is a valid format for the C function *printf* (e.g. `%d` for an integer, `%f` for float, `%e` for exponential notation). If `fmt` is omitted `${`*var*`}` the variable is written in binary format.

Table 2.1: Non-exhaustive list of codes for the model variables in the template file

Code	**Description**	**star1d**	**star2d**
nr	# of radial points	*	*
nth	# of points in latitude		*
ndomains	# of domains	*	*
npts	# of radial points in each domain	*	*
xif	Position of each domain	*	*
nex	# of radial points in the external domain		*
surff	Parameter surff (see above)	*	*
conv	# of convective domains	*	*
Omega	Angular velocity at the equator		*
Omega_bk	Angular velocity at the equator in units of the critical velocity		*
Omegac	Critical velocity $\Omega_c = \sqrt{\frac{GM}{R_e^3}}$		*
X	Hydrogen abundance	*	*
Z	Metal abundance	*	*
Xc	Fraction of X at the convective core	*	*
rhoc	Central density	*	*
Tc	Central temperature	*	*
pc	Central pressure	*	*
M	Mass	*	*
Rp	Polar radius	*	*
Re	Equatorial radius	*	*
L	Luminosity	*	*
M/M_SUN	Mass in solar units	*	*
Rp/R_SUN	Polar radius in solar units	*	*
Re/R_SUN	Polar radius in solar units	*	*
L/L_SUN	Luminosity in solar units	*	*
r	Radius	*	*
th	Colatitude		*
rex	External radius		*
phi	Gravitational potential	*	*
phiex	Gravitational potential of the external domain		*
rho	Density	*	*
p	Pressure	*	*

T	Temperature	*	*
w	Angular velocity		*
G	Stream function for the meridional circulation		*
Xr	Hydrogen abundance $X(r,\theta)$	*	*
N2	Squared Brunt-Väisälä frequency (in $\mathrm{rd}^2/\mathrm{s}^2$)	*	*
opa	Type of opacity	*	*
opa.k	Rosseland mean opacity	*	*
opa.xi	Thermal diffusivity (χ)	*	*
opa.dlnxi_lnT	$\left(\frac{\partial \log \chi}{\partial \log T}\right)_{\rho,\mu}$	*	*
opa.dlnxi_lnrho	$\left(\frac{\partial \log \chi}{\partial \log \rho}\right)_{T,\mu}$	*	*
eos	Type of equation of state	*	*
eos.G1	Γ_1	*	*
eos.cp	c_p	*	*
eos.del_ad	∇_{ad}	*	*
eos.G3_1	$\Gamma_3 - 1$	*	*
eos.cv	c_v	*	*
eos.prad	Radiation pressure	*	*
eos.chi_T	$\chi_T = \left(\frac{\partial \log p}{\partial \log T}\right)_{\rho,\mu}$	*	*
eos.chi_rho	$\chi_\rho = \left(\frac{\partial \log p}{\partial \log \rho}\right)_{T,\mu}$	*	*
eos.d	$d = \frac{\chi_T}{\chi_\rho} = -\left(\frac{\partial \log \rho}{\partial \log T}\right)_{p,\mu}$	*	*
nuc.eps	Energy generation rate per unit mass	*	*
nuc.pp	Energy generation rate per unit mass (pp-chain)	*	*
nuc.cno	Energy generation rate per unit mass (CNO cycle)	*	*
Teff	Effective temperature at the surface $T_{\mathrm{eff}}(\theta)$	*	*
gsup	Effective gravity at the surface $g_{\mathrm{eff}}(\theta)$	*	*
D	Radial differentiation matrix $\frac{\partial}{\partial \zeta}$ for 2D models, $\frac{\mathrm{d}}{\mathrm{d}r}$ for 1D models	*	*
I	Radial integration matrix	*	*
Dex	Radial differentiation matrix for the external domain		*
Dt	Angular differentiation matrix $\frac{\partial}{\partial \theta}$ for symmetric variables		*
Dtodd	Angular differentiation matrix for antisymmetric variables		*
Dt2	Second order angular differentiation matrix for symmetric variables		*
It	Angular integration matrix		*

For 2D variables, their values at the collocation points are written in the output file in matrix form. Each line corresponds to a different value of the colatitude θ (i.e. a different column), starting at the equator.

$$\begin{array}{cccc} p(\zeta_0,\theta_0) & p(\zeta_1,\theta_0) & p(\zeta_2,\theta_0) & \cdots \\ p(\zeta_0,\theta_1) & p(\zeta_1,\theta_1) & p(\zeta_2,\theta_1) & \cdots \\ p(\zeta_0,\theta_2) & p(\zeta_1,\theta_2) & p(\zeta_2,\theta_2) & \cdots \\ \vdots & \vdots & \vdots & \end{array}$$

Being ζ the radial spheroidal coordinate. Similarly, 1D variables can be seen as a column vector and are written in one line in the output file, terminated by a new line character. This behavior can be inverted by writing this line in the template file

```
\conf{transpose=1}
```

After this command, the variables will be written row wise, i.e. one line for each value of the radial coordinate. Note that it does not affect variables written in binary format, which are always column wise. To recover the original behavior we use

```
\conf{transpose=0}
```

The original grid does not contain points in the equator and the pole. If we want the values at this points we should write

```
\conf{equator=1}
\conf{pole=1}
```

By default, the output uses cgs units. If we want the normalized values used internally by the code, we simply put

```
\conf{dim=0}
```

These control commands can be written anywhere in the template file, in separated lines, affecting only the code that appears below them.

Let's see an example.

Template file:

```
Model of ${M/M_SUN,%.2f} solar masses and R=${R,%e} cm

rotating with Omega=${Omega_bk,%f} Omegac

${nr,%d} radial points and
${nth,%d} latitudinal points

\conf{pole=1}
\conf{equator=1}
r:

${r,%e}
Pressure:

${p,%.14e}
```

Output file:

```
Model of 2.50 solar masses and R=1.219822e+11 cm
rotating with Omega=0.900000 Omegac
240 radial points and 32 latitudinal points
r:
0.000000e+00 4.944313e+07 1.971944e+08 4.415355e+08 7.796539e+08 ...
0.000000e+00 4.944313e+07 1.971944e+08 4.415355e+08 7.796539e+08 ...
```

```
0.000000e+00 4.944313e+07 1.971944e+08 4.415354e+08 7.796533e+08 ...
0.000000e+00 4.944313e+07 1.971944e+08 4.415352e+08 7.796523e+08 ...
[...]
Pressure:
1.61049808835808e+17 1.61048890365891e+17 1.61035199104197e+17 ...
1.61049808835808e+17 1.61048890354742e+17 1.61035198927083e+17 ...
1.61049808835808e+17 1.61048890265707e+17 1.61035197512689e+17 ...
1.61049808835808e+17 1.61048890088480e+17 1.61035194697311e+17 ...
[...]
```

2.5 Python module

A basic python module for reading the models is included in the distribution. It is located in `ester/python/star.py`. At the moment it only works for models calculated using `star2d`. The variables in the models are defined as *numpy* arrays. Here is a little example:

```
import sys
sys.path.append('path_to/ester/python') # include the full path to the module

from star import * # Loads the module

A=star2d('model_file')  # Loads a model

print A.p[0,0] # Prints the central pressure

A.draw(A.w) # Makes a plot of the differential rotation
show() # Needed in non-interactive mode of matplotlib
```