STARS

draft version, October 28, 2012

Contents

1 Introduction **5**
- 1.1 About the code . . . 5
- 1.2 About this document . . . 6
  - 1.2.1 Motivation . . . 6
  - 1.2.2 Structure . . . 6
  - 1.2.3 Further work . . . 6

2 Basic user guide **9**
- 2.1 Files . . . 9
- 2.2 Running the code . . . 9
- 2.3 Viewing output . . . 10
- 2.4 Examples of more general operation . . . 10
  - 2.4.1 Evolving a PMS model to a homogeneous ZAMS model . . . 10
  - 2.4.2 Changing the mass of a ZAMS model . . . 11

3 Input and output **13**
- 3.1 `data (1)` . . . 13
- 3.2 `COtables (10)` . . . 18
- 3.3 `physn.dat (11)` . . . 18
- 3.4 `nrate.dat (13)` . . . 18
- 3.5 `modin[2] (30)` . . . 18
- 3.6 `nucmodin[2] (31)` . . . 19
- 3.7 `out[2] (32)` . . . 19
- 3.8 `plot[2] (33)` . . . 20
- 3.9 `modout[2] (34)` . . . 21
- 3.10 `nucmodout[2] (35)` . . . 21
- 3.11 `syntha,synthb,synthc[2] (36,37,38)` . . . 22
- 3.12 `surface[2],centre[2] (39,40)` . . . 22
- 3.13 `sprocess[2] (41)` . . . 22
- 3.14 `montage[2] (42)` . . . 23

4 Code structure **25**
4.1 Architecture . . . 25
4.1.1 Common blocks . . . 25
4.1.2 I/O and linking . . . 25
4.1.3 Code units . . . 26
4.2 Subroutines . . . 26

5 Input physics **29**

6 Advanced user guide **31**

A STARS cheat sheet **33**

B Veteran tips **35**
B.1 Comments in I/O files . . . 35
B.2 Command line tools for manipulating plain text . . . 35
B.3 Separating models in subdirectories . . . 36

C Matrix inversion **39**

1 Introduction

1.1 About the code

The code documented here was originally written by Peter Eggleton in the early 1970s. Features of the original code were outlined in four main papers that described the adaptive non-Lagrangian mesh (Eggleton, 1971), the treatment of convection (Eggleton, 1972), the method for computing the equation of state (Eggleton et al., 1973) and the code's output for the evolution of a $4\,M_\odot$ star (Eggleton, 1973). Pols et al. (1995) detailed a number of updates to the code that were added in the intervening 22 years. That version of the code was used to produce the grid of models presented by Pols et al. (1998) and was previously distributed as the `ds2000` version of the code.

The next major update was made my John Eldridge as part of his PhD. John created a new opacity routine that included data for variable abundances of carbon and oxygen (Eldridge & Tout, 2004). A further version of the code was available. I refer to it as `ds2004`.

Around the same time, Richard Stancliffe, also as part of his PhD, worked on getting the code to accurately model thermally-pulsing asymptotic giant branch (TPAGB) stars. He subsequently continued working on the code and ultimately produced the version that is documented here, which I refer to as `bs2007`. Because the nature of much of the code is unchanged, much of this documentation is relevant to previous versions of the code. However, if you are working with an older version, beware that there are differences and no mention is made of them here.

The contents of the code papers are summarized below.

Citation	**Content**
Eggleton (1971)	Mesh spacing.
Eggleton (1972)	Convection and semiconvection.
Eggleton et al. (1973)	Equation of state.
Eggleton (1973)	Example evolution.
Pols et al. (1995)	Summary of updates, notably EoS and physical data.
Schröder et al. (1997)	Convective overshooting algorithm.
Eldridge & Tout (2004)	New opacity routines.
Stancliffe, Tout & Pols (2004)	TPAGB-specific mesh-spacing function.
Stancliffe et al. (2005)	Minor element nucleosynthesis update.
Stancliffe et al. (2007)	Thermohaline mixing.
Stancliffe & Eldridge (2009)	Example binary evolution.

If you have used the code for published work, citations to these papers, as appropriate, would be greatly appreciated.

1.2 About this document

Warrick Ball began writing this document in some of the time between the completion of his PhD and the start of his first postdoctoral position. The content is drawn from a number of documents that collectively provided some form of guidance for using the code. Chapter 2 is inspired by the short user guide that existed on the old website. Most of Chapter 3 is taken from Richard Stancliffe's own documentation for `bs2007` that he wrote around the time of its release. Most of the technical details in 4 are Warrick Ball's own original creation, with some input from Peter Eggleton's documentation for EV/TWIN. Chapter 5 is a consolidation of many years of IoA PhD theses, including those of Warrick Ball, John Eldridge and Richard Stancliffe. Some elements were also taken from the EV/TWIN writeup. Chapter 6 is a hybrid of Richard Stancliffe's tutorials and the 'recipes' that were on the website.

The 'cheat sheet' (Appendix A) was created by Warrick Ball, although many STARS users may have created their own. Appendix B was created for the first time for this document and draws on the collective experience of past and present members of the Cambridge Stellar Evolution group. Finally, Appendix C is a near-verbatim copy of an obscure plain-text file written by Peter Eggleton for EV/TWIN. Warrick Ball was surprised to find that Richard Stancliffe saw it for the first time in 2012!

1.2.1 Motivation

The persistent lack of a primary user guide for the STARS code presented a steep learning curve for new users. Each newcomer to the code would have to cobble together the available documents, some of which are only available from obscure sources, and try to make sense of how to operate the code. Even then, only after some time would new users be able to streamline their operation of the code and use it efficiently. The objective of this documentation is to resolve these problems. Hopefully, now, new users are be able to start using the code much more readily than before and capitalize on the collective experience of established users.

Alas, this motivation is expected to be largely unrewarded and can only propel the author's spirit so far. If you find that this documentation or the website have fallen into something resembling disrepair, its probably because a previous maintainer became too preoccupied with trying to establish a career!

1.2.2 Structure

This document is intended as both a user guide and a reference document and it need not be approached in a linear fashion. Chapter 2 is intended for new users to get a feel for simply running the code and viewing output. The specific task at hand can be formulated and a decision made about how to use the code to achieve that by controlling the input and output, which is described in Chapter 3. Examples of more advanced use is found in Chapter 6 but the material in that section presumes the user is familiar with Chapter 3. This should not be seen as a major problem because the user can refer to Chapter 3 to find the relevant information when necessary.

Chapters 4 and 5 provide technical details about the code. The design of the code and a breakdown of its components are provided in Chapter 4. Chapter 5 describes the physical processes that are included in the code and, to some extent, how they are implemented. If you intend to publish results that are produced by the STARS code, you should be familiar with Chapter 5.

The appendices provide additional material that is not important but might be useful. Appendix A can be printed and placed somewhere nearby as a reminder of the output options and controls. Appendix B contains some useful tricks for using the code effectively. Again, these are not important but you may find them useful. Finally, Appendix C is a reproduction of Peter Eggleton's detailed description of the matrix inversion algorithm. It is the only documentation of these subroutines that anyone is aware of so it is included here for completeness and posterity.

1.2.3 Further work

This documentation is still incomplete and the following material needs to be created.

- A brief outline of using the code to create ZAMS models and change their masses (Section 2.4).
- A description of the modeling of physical processes and of the implementation of those models. i.e. Chapter 5.
- The advanced user guide. i.e. Chapter 6.
- Completion of the description of the subroutines (Section 4.2).
- A description of how to use the IoA grid to run jobs would be useful (Appendix B).
- A new diagram of the code structure (Section 4.2).
- Appendix C is presently nearly a pure copy-'n-paste of Peter's plain-text document. It should be TEX-ed.

2 Basic user guide

This sections describes the contents of the archive that is available on the website and the basic operation of the code. It is presumed that you are able to compile FORTRAN 77 code into an executable. Command line instructions are given for GNU/Linux operating systems.

2.1 Files

Once you have downloaded the standard archive, extract it in a suitable directory. e.g. `/home/user/stars/`. This creates several folders and a few files in the current directory. The archive does *not* create a zeroth-order folder like `stars/`. The contents of the archive are tabulated below.

`dat/`	Contains tables of physical data and other constants, including the opacities, nuclear reaction rates and spline coefficients.
`obj/`	Initially empty but, after compilation, contains all the object files for the subroutines.
`src/`	Contains the FORTRAN source code.
`Makefile`	Compilation script.
`run`	Execution script.
`data`	Controls for operation of the code (see Section 3.1).
`modin`	The stellar model data (see Section 3.5).

2.2 Running the code

The compilation script (i.e. `Makefile`) is relatively simple. It is based on *A Simple Makefile Tutorial* [1], which Warrick Ball found on the website of Bruce Maxwell. By default, the `Makefile` uses `gfortran` to compile the code. You should check the file in anticipation of compilation errors and for compatibility with your system. This documentation does not address such issues. When ready, compile the STARS executable by running `make`. This creates the object files in the `obj/` directory, links them and creates the STARS executable file, `bs`. You should now be ready to run the code.

To run the code, execute the script `run`. In GNU/Linux, you will probably have to enter `./run` at the terminal. The code should give a few lines of output to indicate that it is initializing model and then sit quietly. The STARS code is now computing the evolution of the default stellar model, which is a $1\,M_\odot$ pre-main-sequence star with a central temperature around 10^6 K. The code should run until the star reaches the helium flash, where it stops. The terminal should show a whole lot of lines starting with

[1] http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

```
OPACITY OUT OF RANGE TF FR FKL FKH
```

followed by four numbers. When the terminal becomes available again, the run is complete. Depending on the speed of your computer, this should take a few minutes.

2.3 Viewing output

The output of the model is written to several output files. First, let's create an HR diagram of the star's evolution. In your favourite plotting program, plot column 4 of `plot` against column 5. For example, to plot the output using GNUPLOT, run

```
plot "plot" using 4:5
```

Column 4 is the effective temperature in kelvin and column 5 the logarithm of the luminosity in $L_\odot$. Remember that Hertzsprung–Russell diagrams have $T_{\rm eff}$ reversed, so the x-axis is backwards. A complete list of the columns of `plot` is included in Section 3.8. So, if you wanted to plot the central density against central temperature, you could instead plot column 73 against column 74.

The second output file is `out`. You may open this in a text editor to view its contents. The file begins with the basic initial conditions of the model and then provides summaries of the star's structure and, every 100th model, detailed profiles of the structure and composition. The output is described in detail in Section 3.7 and the parameters that control the output are specified in the file `data`, which is described in Section 3.1.

The third file is `modout`. This file provides stellar models in the same format as input models. If you want to use a previous stellar model to create a new one, you will want to copy parts of this output into `modin`. The format of `modin` and `modout` is described in Section 3.5.

2.4 Examples of more general operation

There are two purposes to running the code: either to model the evolution of the star or to create an initial model by taking a known stellar model and telling the code to modify it in some way. e.g. by adding or subtracting mass. The first case is as described above. With the appropriate `modin` file, just run the script (`run`) and then view the output.

In the second case (creating new models), you must set an appropriate stop condition for the code and copy the output models from `modout` into `modin`. The code parameters are contained in `data` but several examples are provided below to help your understanding of how the code works. You may read through Section 3.1 before trying these but it's up to you.

2.4.1 Evolving a PMS model to a homogeneous ZAMS model

How do we evolve a PMS model to the ZAMS? The PMS model must be able to radiate thermal energy and contract but it mustn't start the transformation of chemical elements through nuclear fusion. To achieve this kind of control, STARS has entries in the `data` file that enable, disable or scale various physical processes.

In our case, we want to disable all nuclear reactions. To do so, use your favourite plain text editor (e.g. vim, EMACS, gedit) to change the first line of `data` to

```
1   199   40   10   15   15    3    1    1    0    0    0    1
```

These three numbers (`IX`, `IY` and `IZ`) decide whether or not the chemical composition is allowed to change owing to hydrogen burning, helium burning or subsequent reaction chains (carbon burning and beyond). Note that energy is still produced by these reactions: only the transmutation of elements is stopped.

In addition, we don't need the model to run for as long. Judging from the full evolution we did before, 3200 steps should do. To tell the code to stop after 3200 steps, change the first line of `modin` to

```
1.000794E+00  6.308266E+03  0.000000000E+12  9.989717E+48  2.000000E+03  0.000000E+00  199 3200  0    1
```

The modified number tells the code how many timesteps to take before stopping.

Now, run the code again by executing `run`. It will eventually stop, as before, but it won't shouldn't provide any errors. This is because our stop condition (3200 steps) allows the code to exit normally, whereas previously the code stopped because it could not successfully evolve the star past the helium flash.

You can inspect the output again. It is worth plotting the first the timestep number (column 1) against the effective temperature or surface luminosity (columns 4 and 5). You'll see that both stop changing just before the evolution stops. This is the indication that the model has reached the main sequence and stopped evolving. You should also plot the timestep number against the timestep (column 27) to see that the timestep begins to increase rapidly because the model is no longer really changing with time.

2.4.2 Changing the mass of a ZAMS model

As a final exercise in the code's basic usage, we will change the mass of the ZAMS model produced above. To do so, we will keep nuclear transformations off, fix the model in thermal equilibrium and tell the code to add a small amount of mass at each timestep.

Our starting model is now the last model from the previous evolution, so we must first copy the new model to `modin`. To achieve this, you need to put the last 399 lines of `modout` in `modin`. You can do so however you please but you may want to backup `modin` before doing so. At the Linux command line, for example, you could copy last 399 lines by typing

```
rm modin
tail -399 modout > modin
```

You could also use a text editor: just copy to `modin` everything in `modout` from (and including) the line

```
1.000794E+00  1.068356E+07  2.515366186E+08  9.989717E+48  2.000000E+03  0.000000E+00   199  3200  3201     0  6.669E+16  0.000E+00
```

to the end of the file.

To keep the model in thermal equilibrium, we simply tell the code that it cannot generate energy through contraction or expansion. Change the first line of `data` to

```
1   199  40  10  15  15   3   1   0   0   0   0   1
```

Now, all the output models will be in precise thermal equilibrium.

To tell the code to add mass to the model, modify the fourth number on the 18th line of `data` to something small. e.g. 10^{-9}.

```
18  1.00E+08 0.00E+00 0.00E-01 1.00E-9 0.00E+00 3.00E-01 1.00E-03
```

This tells the code to increase the total mass of the model by a fraction 10^{-9} each year.

Now, execute `run` again. As in the previous run, the code terminates without any error message. To see what you have done, plot the timestep number (column 1) against the total mass of the star (column 6). You should find that the model grew up to a total mass of about $5.35\,\mathrm{M}_\odot$. If you wanted higher mass models, you could allow the code to run for longer. Each 1000th model is saved to `modout`, so you also have new input models for $M/\mathrm{M}_\odot = 1.41$, 1.96 and 3.74. You can use them as new input models in the same way as we did above: just copy them to `modin`. (You can change the frequency of the output models using the `data` parameter `NSAVE`. See below.)

3 Input and output

The STARS code reads and writes to files by linking the default FORTRAN 77 output files, which have names matching `fort.*`. For each of the following files, the relevant `fort.*` file is specified. The linking is done by the `run` script. Each section is titled by the file in question and the FORTRAN unit is given in brackets. Where an I/O file exists for both star 1 and 2, `[2]` is appended to the filename. For example, `modin` provides the input model for star 1; `modin2` for star 2. The section is marked `modin[2] (30)`. For the file linked to star 2, add 20 to the number in brackets. (e.g. `modin2` is linked to `fort.50`.)

Because the code is designed to handle binary evolution, many of the files are doubled up but have the same format. In general, data files are read from `fort.1?`, I/O for star 1 (or a single star) is done through `fort.3?` and I/O for star 2 is done through `fort.5?`.

3.1 `data (1)`

This file contains controls for how the code operates. There are controls for the nature and frequency of output, the selection of equations that are solved, how the mesh points are distributed and much more. The file has undergone considerable revision over the course of the code's life. The present version appears as below and a description of each parameter and its default value(s) is given in the table that follows. After the functional control parameters, the file also has a large section that briefly recounts some of the information presented here.

```
NH2 ITER1 ITER2 JIN JOUT NCH JP ITH IX IY IZ IMODE
ICL ION IAM IOP INUC IBC ICN IML1 IML2 ISGTH IMO IDIFF
NWRT1 NWRT2 NWRT3 NWRT4 NWRT5 NSAVE NMONT
EPS DEL DH0 DT3 DDD
NE1 NE2 NE3 NB NEV NF J1 J2 IH JH
ID(30) - 3 lines
NE1 NE2 NE3 NB NEV NF J1 J2 IH JH
ID(90) - 3 lines
ISX(45) - 3 lines
DT1 DT2 CT(10)
ZS ALPHA CH CC CN CO CNE CMG CSI CFE
RCD OS RML RMG ECA XF DR
RMT RHL AC AK1 AK2 ECT TRB
IRAM IRS1 VROT1 IRS2 VROT2 FMAC FAM
IVMC TRC1 IVMS TRC2 MWTS IAGB ISGFAC FACSGMIN SGTHFAC
```

Name	Description	Default
NH2	The desired number of mesh points. If different from that in modin, the code interpolates the given model to give the new one, provided that NCH is greater than or equal to 1.	199
ITER1	The maximum number of iterations allowed on the first timestep.	10
ITER2	The maximum number of iterations allowed on later timesteps.	10
JIN	The number of independent variables of the H and DH arrays to be read in.	15
JOUT	The number of independent variables written to output models.	15
NCH	Determines how the model is re-meshed. If NCH=1, the mesh is fixed. If NCH=2, the mesh is allowed to vary but the composition is not modified. If NCH=3, the mesh is allowed to vary and, on the first step, the composition is reset to the values specified in data.	1,2,3
JP	Determines whether the last set of corrections to the previous model is used as the first correction to the current model. If JP=0, the corrections are reset to zero. If JP=1, the last set of corrections to the previous model is used as the first correction to the current model.	1
ITH	The thermal energy generation rate is ITH$\times T\,\partial s/\partial t$. Setting ITH to zero means $T\,\partial S/\partial t$ is ignored. i.e. there is no thermal evolution of the star. This is sometimes useful when creating new models.	1
IX	Determines whether hydrogen is converted into helium. If IX=0, hydrogen is not converted into helium but the energy produced by the nuclear reactions is still included. If IX=1, hydrogen is converted into helium normally.	1
IY	As for IX but controls alpha-burning.	1
IZ	As for IX but controls carbon-burning.	1
IMODE	Determines whether the code operates in pseudo-binary mode or full binary mode. If IMODE=1, only the equations for one star are solved. If IMODE=2, the equations for both stars are solved.	1
ICL	If ICL=1, the effects of Coulomb interactions are included in the calculation of the pressure ionization.	1
ION	The number of elements that for which the the ionization state is calculated. The default value includes hydrogen and helium. The choices 3, 4 and 5 include carbon, nitrogen and then oxygen. i.e. ION=5 calculates the ionization of all five elements.	2
IAM	If set to zero, the program will use integer atomic weights.	1
IOP	Used to select spline interpolation in opacity. If set to one or less the old style opacity tables are used. If IOP is 1, spline interpolation will be used. If set to zero, a linear interpolation in opacity will be used. Setting this to 5 will use the new opacity tables with variable C/O composition.	5
INUC	If INUC$\geq$10, weak screening is employed in the nuclear reaction rates.	0
IBC	No current function.	
ICN	Used for CNO equilibrium on the main sequence. If ICN=1, a baryon correction is applied to the hydrogen evolution equation so that hydrogen is not converted into helium but other elements can achieve their equilibrium abundances. The timestep is also affected when ICN=1.	0
IML1 IML2	Specify the mass-loss rates for stars 1 and 2, respectively. 0 corresponds to no mass loss, 1 to Reimers mass loss, 2 to Blöcker, 3 to Vassiliadis & Wood and 4 to Wolf-Rayet mass loss.	0
ISGTH	Turns on thermohaline mixing. Used if you want to compute models with accretion or to provide extra mixing on the giant branch in low mass stars. See also SGTHFAC.	1

IMO	Turns on the molecular opacity routines in statef if IMO=1. These are only really important for cool ($T < 10^4$ K) stars that are carbon-rich, such as low-mass AGB stars and carbon-enhanced metal-poor stars.	0
IDIFF	Turns on gravitational settling, as well as atomic and thermal diffusion. This probably only matters for low-mass stars without significant convective envelopes.	0
NWRT1	Prints the internal details of every NWRT1th model to out.	100
NWRT2	Prints the internal details of every NWRT2th meshpoint when the internal model is printed to out.	1
NWRT3	The number of 'pages' printed out for every NWRT1th (i.e. detailed) model.	1
NWRT4	Prints a short summary of every NWRT4th model.	1
NWRT5	Prints a one-line summary of each iteration of each model, excluding the first NWRT5 iterations of each model.	0 (2)
NSAVE	An output model is saved to modout every NSAVEth timestep, in the same format as the input model, that can be used for a subsequent run. The final model is automatically saved.	100
NMONT	A detailed model for use as an input file for the Montage post-processing code is printed out every NMONT models. If zero, no output is produced.	0
EPS	The accuracy to which the equations are expected to be solved. EPS must be greater than DH0. i.e. you can't solve to a greater degree of accuracy than the derivatives.	10^{-6}
DEL	The maximum value in ERR for which the whole correction is applied by the solution subroutine. Above this limit, the correction is reduced by a factor of ERR/DEL.	0.01
DH0	Affects the value of the increments of the variables during the numeric differentiation. It is no longer so important, now that everything is in double precision.	10^{-7}
DT3	No current function.	
DDD	Sets the modulus of the total increment that is desired in one timestep.	0.5–4
NE1	The number of 1st order equations that the code uses.	6
NE2	The number of 2nd order equations that the code uses.	5
NE3	This defines a subset of the 1st order equations that may be defined at 3, rather than two, adjacent meshpoints. At present, this is not implemented in the code.	0
NB	The number of boundary conditions possessed by the 1st order equations at the stellar surface.	3
NEV	The number of 'eigenvalues' (i.e. quantites that don't vary with the mesh) used by the model. This is usually the mesh spacing function.	1
NF	The number of variables being passed into the equns routines.	30
J1, J2, IH, JH	Variables for debugging. A suitable choice gives an output via printc that can be used to see if funcs and equns are setting up the difference equations correctly. The default values suppress debugging output.	0, 0, 0, 99

`ID(90)` There are two blocks of numbers for `ID`: one for the evolution package (`funcs1`, `equns1`) and the other for the nucleosynthesis package (`funcs2`, `equns2`). Together with the preceding line (containing `NE1`, `NE2`, `NB`, `NEV`, `NF`, `J1`, `J2`, `IH`, `JH`), these define in what order the variables are solved for (first line), in what order the equations are solved (second line) and in what order the boundary conditions are solved (third line). For the evolution of single stars the usual values are:

```
 1  2  4  5  3  9 10  8  7  6  0  0  0  0  0
 6  7  8  9  4  2  1  3  5  0  0  0  0  0  0
 4  5  6  7  2  3  1  2  3  1  0  0  0  0  0
```

`ISX(45)` 3 lines of 15 numbers that determine which variables of the internal structure are printed to `out`. The first 15 values define what will be placed on the first 'page', with the next two sets of 15 defining the output to the extra pages. The options are:

1. ψ	10. ^{1}H	19. $E_{\rm th}$	28. S
2. P	11. ^{4}He	20. $E_{\rm nuc}$	29. $L/L_{\rm Edd}$
3. ρ	12. ^{12}C	21. E_{ν}	30. μ
4. T	13. ^{14}N	22. δm	31. $\mu_{\rm ideal}$
5. κ	14. ^{16}O	23. k^2	32. $\sigma_{\rm thermohaline}$
6. ∇	15. ^{20}Ne	24. $n/(n+1)$	33. $E_{\rm bind}$
7. $\nabla_{\rm ad}$	16. ^{24}Mg	25. $U_{\rm hom}$	34. $\beta = P_{\rm rad}/P_{\rm tot}$
8. $\nabla_{\rm rad} - \nabla_{\rm ad}$	17. r	26. $V_{\rm hom}$	
9. m	18. L	27. U	

23 is the square of the radius of gyration; 24–26 are homology invariants.

`DT1`	Places a lower limit of `DT1`$\times\Delta t$ on the the size of the next timestep, where Δt is the size of the current timestep.	0.8
`DT2`	Places an upper limit of `DT2`$\times\Delta t$ on the size of the next timestep. If both `DT1` and `DT2` are set to 1, the timestep is constant, unless the model fails to converge, in which case it is reduced by 20% for the next attempt at a solution.	1.2
`CT(10)`	Coefficients used in the mesh spacing function, Q.	
`ZS`	The stars metallicity. Make sure this matches the value in the opacity tables!	0.02
`ALPHA`	The mixing length. The value chosen *should* be based on calibration to a solar model but the default value is not a calibrated value.	2.00
`CH-CFE`	Values for initializing the abundances of ^{1}H through to ^{56}Fe of a model. The metals are expressed as a fraction of the total metallicity. These are only used for ZAMS models (or better still, pre-MS models) with `NCH`=3.	
`RCD`	The diffusion coefficient for convective mixing is `RCD`$\times 2(\nabla - \nabla_{\rm ad})/t_{\rm nuc}$.	10^6
`OS`	Convective overshoot parameter. Zero implies no overshoot.	0 (0.12)
`RML`	Used to set the amount of Reimers (1975) mass loss. This value is the parameter η such that the mass loss rate is $\dot{M} = \eta \times 4 \times 10^{-13}\,{\rm M}_{\odot}\,{\rm yr}^{-1}$.	0
`ECA`	Used in the evolution of `EC`. This is useful when creating pre-MS models.	0
`XF`	Defines the boundary of a core, for printout purposes only, to be when the abundance equals XF.	0.1 (0.3)
`DR`	Defines the boundary between a convection and semiconvection zone, for printout purposes only, to be at $\nabla - \nabla_{\rm ad} =$`DR`.	0.01
`RHL`	Defunct.	0
`AC`	Defunct.	1.0

AK1	Used in the AGB mixing formula (see Stancliffe et al., 2004, for details). Sets β for the H-rich regions.	1.0
AK2	Used in the AGB mixing formula. Sets for the H-poor regions.	10^{-4}
ECT	A constant logarithmic increase or decrease for EC (see Section 3.5), which can be used to push a ZAMS star back up its Hayashi track to make a pre-MS star.	0 (10^{-4})
TRB	Used in setting the surface conditions of binaries. Effectively places the star in a radiation bath of temperature TRB.	0
IRAM	Resets the orbital angular momentum. If IRAM=1, the code recalculates the orbital angular momentum using the period from modin.	0
IRS1	Resets the spin angular momentum of star 1. If IRS1=1, the code sets the spin of star 1 using the rotational speed provided in VROT1.	0
VROT1	The desired rotational speed of star 1, in $\mathrm{km\,s^{-1}}$. Used only if IRS1=1.	
IRS2	As for IRS1, but for star 2.	
VROT2	As for VROT1, but for star 2.	
FMAC	The fraction of matter accreted during mass transfer. Any matter not accreted is lost from the system and carries away angular momentum.	1.0
FAM	The fraction of angular momentum transferred in the accreted matter that is transferred to the accreting star.	1.0
IVMC	Used to turn on the viscous mesh. Used in conjunction with TRC1. Useful for evolving AGB stars and possibly in other situations.	0
TRC1	The mesh point at which the code switches from viscous to non-viscous meshing. The transition must be smooth so there is probably some degree of viscosity at k=TRC1. If TRC1$\geq$NMESH, there is no viscous meshing. It is also unwise to try and lock the majority of the star with the viscous mesh. You have been warned.	700
IVMS	Used to turn on the viscous mesh at the star's surface. Used in conjunction with TRC2. This is useful for suppressing loops in the HR diagram for post-AGB objects (and possibly other shell-burning stars).	0
TRC2	As TRC1.	300
MWTS	Unlike the core viscous meshing, which is designed to be invoked at low timesteps, the surface viscous mesh takes a constant mesh weighting. If this is high enough, the surface mesh points only move around slowly thereby reducing the problem of numerical diffusion. If MWTS=1, the points do not move at all. This limits numerical diffusion but can cause convergence failures.	0.75
IAGB	Switches on the AGB-specific modification by Stancliffe et al. (2004), including the more realistic mixing, the AGB timestep control and the AGB specific mesh-spacing function. Note that you still have to change the coefficients of the mesh spacing function!	0
ISGFAC	Switches on the reduced mixing convergence aid. Used with FACSGMIN.	0
FACSGMIN	Sets the initial mixing reduction factor for assisted convergence. Setting this to 1 is equivalent to setting ISGFAC=0.	10^2
SGTHFAC	Boosts the efficiency thermohaline mixing by a factor of SGTHFAC. It has been suggested that 10^2 is appropriate for getting ^{3}He-induced extra mixing on the giant branch in low-mass stars.	10^2

3.2 COtables (10)

This file contains the opacity tables used for calculating the opacity with variable carbon and oxygen abundances. Each table begins with a row of five numbers. The first three give the hydrogen, helium and metal abundances of the table and the next two give the scaled carbon and oxygen abundances. For carbon, there is no carbon enhancement if this number is 0. If it is 1, the abundance is $1-X-Z$, where X is the hydrogen abundance and Z the metallicity.[1] There then follow 141 lines of 32 numbers. The first number in each line is the temperature in $\log_{10} T$. The remaining numbers are the opacities running from $R = \rho/T_6^3 = 8$ to $R = 7$.

3.3 physn.dat (11)

This contains the input physical data, including the original style of opacity table, the neutrino loss rates and the charged particle reaction rates. The file starts with a single number defining how many opacity tables of different composition there are. This is followed by a line defining the composition of each table using the formula $m = 2X + Y + 1$, where Y is the helium abundance. The n tables follow in 10×9 blocks. After this comes the neutrino loss rate and then the charged particle reaction rates. The latter have been updated to use the most recent rates available in 2004.

3.4 nrate.dat (13)

This contains the reaction rates for the neutron capture reactions used in the nucleosynthesis subroutines. Each set of reaction rates comes as a block of 200 numbers in 20 rows and 10 columns, just like the charged particle reaction data.

3.5 modin[2] (30)

These are the input model files for the stellar models. The file `modin` corresponds to star 1 or a single star. The file `modin2` corresponds to star 2 and it not included in the basic download. The first lines of both files is of the format

```
1.000000E+00  2.173234E+03  0.000000000E+10  1.323350E+81  2.000000E+10  0.000000E+00   199 99900     0     1  6.680E+16  0.000E+00
```

The numbers are the star's mass, the current timestep size, the model's age, the period of the binary,[2] the total mass of the binary, an artificial energy generation term, the number of mesh points in the model, the desired number of models to be computed, the starting model number, a number determining whether this is star 1 or star 2 of a binary, the pressure in the H-burning shell and the pressure in the helium burning shell.

There then follow the lines describing the model itself. There can be up to 15 variables in this file at present and all are currently used.[3] In order, the variables are

[1] Shouldn't helium be included too?

[2] The code always thinks you are dealing with a binary, even if you only care about evolving one star.

[3] This is not *strictly* true. In `modin2`, the value of the orbital angular momentum is not used. One could replace this with the eccentricity of the orbit but that would stop `modin` and `modin2` from being interchangeable.

1. $\log f$ a parameter related to the electron degeneracy
2. $\log T$ log of the temperature in Kelvin
3. $X_{\rm O}$ the mass fraction of ^{16}O
4. $\log m$ log of the mass in units of 10^{33} g
5. $X_{\rm H}$ the mass fraction of ^{1}H
6. $\partial Q/\partial K$ the gradient of the mesh spacing function
7. $\log r$ log of the radius in units of 10^{11} cm
8. L luminosity in units of 10^{33} erg s^{-1}
9. $X_{\rm He}$ the mass fraction of ^{4}He
10. $X_{\rm C}$ the mass fraction of ^{12}C
11. $X_{\rm Ne}$ the mass fraction of ^{20}Ne
12. $X_{\rm N}$ the mass fraction of ^{14}N
13. $H_{\rm orb}$ the orbital angular momentum in code units
14. $H_{\rm spin}$ the spin angular momentum of the star
15. $X_{^3{\rm He}}$ the mass fraction of ^{3}He

3.6 `nucmodin[2]` (31)

These are the input model files for the nucleosynthesis routines. The first line of each file is the same as the first line of `modin` or `modin2` file. There then follow the lines of the model itself. Each line has 50 numbers. These are the abundance by mass fraction of all the elements used by the nucleosynthesis subroutines. In order, they are

1. g, gallinoes
2. n
3. ^{2}H
4. ^{3}He
5. ^{7}Li
6. ^{7}Be
7. ^{11}B
8. ^{13}C
9. ^{14}C
10. ^{15}N
11. ^{17}O
12. ^{18}O
13. ^{19}F
14. ^{21}Ne
15. ^{22}Ne
16. ^{22}Na
17. ^{23}Na
18. ^{24}Mg
19. ^{25}Mg
20. ^{26}Mg
21. ^{26}Al$^{\rm m}$
22. ^{26}Al$^{\rm g}$
23. ^{27}Al
24. ^{28}Si
25. ^{29}Si
26. ^{30}Si
27. ^{31}P
28. ^{32}S
29. ^{33}S
30. ^{34}S
31. ^{56}Fe
32. ^{57}Fe
33. ^{58}Fe
34. ^{59}Fe
35. ^{60}Fe
36. ^{59}Co
37. ^{58}Ni
38. ^{59}Ni
39. ^{60}Ni
40. ^{61}Ni
41. ^{1}H
42. ^{4}He
43. ^{12}C
44. ^{14}N
45. ^{16}O
46. ^{20}Ne
47. Unused
48. Unused
49. Unused
50. Unused

3.7 `out[2]` (32)

This file is useful as a reference for what is going on as a star evolves. The file starts with a copy of the data block used for the star's evolution. Below this is a print out of the first line of `modin` (or `modin2`). The structure of the remaining output depends on the values that were chosen in `data` (see Section 3.1). Typically, the code produces a short four line summary of the model (the frequency of which depends on the value of `nwrt4`), like

```
1       447  dt/age/MH/MHe  tn/tKH/Mb  P/rlf/dM LH/LHe/LC Lth/Lnu/m  H1      He4     C12     N14     O16     Ne20    He3       psi    dens    temp
2    1.0000 1.977573088E+08 1.067E+10 1.443E+81 9.904E-01-2.764E-04 0.37009 0.60941 0.00001 0.00570 0.00895 0.00198 0.00001  -1.539  2.1457  7.1818  cntr
3    0.0001 4.541134415E+09 1.306E+07-1.253E+02 1.620E-33 2.154E-02 0.70000 0.28000 0.00346 0.00112 0.00964 0.00198 0.00003 -15.823 -6.7799  3.7594  srfc
4    0.0000  0.6645  0.3145 2000.0000 0.000E+00 0.000E+00    0.0001 0.37009 0.60941 0.00001 0.00570 0.00895 0.00198 0.00001  -1.539  2.1457  7.1818  Tmax
```

The first column contains the number of the model (in this case `447`) and the mass of the star being evolved, in solar units (`1.0000`). The next number is the location in mass of the boundary of the hydrogen-burning shell and the last is the location in mass of the boundary of the helium-burning zone. The locations of these boundaries are determined by the parameter `XF` in `data`. Specifically, the boundaries are placed where $X <$`XF`.

In the second column, we have the current timestep (1.98×10^8 yr) and, below it, age (4.54×10^9 yr) of the model, both in years. The two numbers below these are the masses of hydrogen (`0.6645`) and helium (`0.3145`)

in the star, in solar units. The next column gives the nuclear and Kelvin-Helmholtz timescales in years and the mass of the binary system in solar units.

In column four, we find the period of the binary system in days, the difference between the stellar and Roche Lobe radii (in units of the stellar radius) and the rate of mass transfer. Next come two columns giving information about the luminosity of the system. The first of these columns contains the luminosities from the fusion of hydrogen, helium and carbon.[4] The second column gives the thermal and nuclear luminosities. The units are $\mathrm{L}_\odot$ and the m at the base of this column is the mass location of the region of maximum temperature.

The remaining entries give the abundances of nuclear species as well as the electron degeneracy parameter Ψ, the density and the temperature at the centre, surface and point of maximum temperature, as noted in the last column by `cntr`, `surf` and `Tmax`.

After this block come another 4 lines proving information about the model. They take a form like

```
1     0.0000   0.976  -0.976   1.000  -1.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000 -1.1941  0.0753 -.04571 -0.0009 -0.0110   447
2   199 199     109    -109       3      -3       0       0       0       0       0       0       0
3     0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000 1.261E+01 7.306E-31 0.000E+00
4          1       1       1       0       0       0       0       0       0       0       0       0       0       0       0
```

On the first line, the first number is the mass interior to the point at which carbon burns. The next twelve are the masses at which the model changes from convective regions to radiative regions (and vice versa). The next numbers are $\log_{10} mr^2k^2$ evaluated at some point close to the surface[5], a homology constant k^2, $\log_{10}(R_*/\mathrm{R}_\odot)$, $\log_{10}(L_*/\mathrm{L}_\odot)$ and the model number again.

On the second line, the first number is the mesh point marking the outer boundary of the hydrogen-burning region (which can be a shell or a core). The next number is the same quantity for the helium-burning region. The last 12 numbers of the second line say where the boundaries between convective and radiative regions are. Semi-convective boundaries are determined by the parameter `DR` in `data` and are reported where $0 < \nabla - \nabla_{\mathrm{ad}} <$`DR`. Boundaries between convective and semi-convective regions are marked by negative signs. Convective-radiative boundaries have no sign.

On the third line, the first three numbers are the mass locations of the points of maximum hydrogen, helium and carbon energy generation. The next 12 numbers are the mass locations of the boundaries of burning regions. The final three numbers on this lines are the values of the energy generation rates for hydrogen, helium and carbon at their respective maxima.

On the last line, the first three numbers are the the mesh point numbers of the points of maximum hydrogen, helium and carbon energy generation. The next twelve numbers are the mesh point numbers of the location of burning shell boundaries. The last value is $d\log r/d\log m$.

Every `NWRT1th` model, the code produces a detailed profile of the stellar model. The columns to be printed are selected via `ISX` in `data`.

3.8 `plot[2]` (33)

These are output files that are useful for plotting the characteristics of the models as they evolve. It is possible to use this file to create Kippenhahn diagrams of the interior. Each line corresponds to one stellar model and the columns are, from left to right,

1. N — model number
2. t — age in years
3. $\log_{10}(R_*/\mathrm{R}_\odot)$ — logarithm of radius in solar units
4. $\log_{10} T_{\mathrm{eff}}$ — logarithm of effective temperature in kelvin
5. $\log_{10}(L_*/\mathrm{L}_\odot)$ — logarithm of luminosity in solar units
6. $M_*/\mathrm{M}_\odot$ — mass in solar units

[4] Actually, the third number is the luminosity of all nuclear burning that *isn't* hydrogen or helium.

[5] It is not clear what this number actually represents nor why you would want it. It may be changed or the documentation updated.

7.	$M_{\rm H}/{\rm M}_\odot$	mass of H-exhausted core in $\rm M_\odot$
8.	$M_{\rm He}/{\rm M}_\odot$	mass of He-exhausted core in $\rm M_\odot$
9.	$L_{\rm H}$	logarithm of hydrogen luminosity in solar units
10.	$L_{\rm He}$	logarithm of helium luminosity in solar units
11.	$L_{\rm C}$	logarithm of carbon luminosity in solar units
12–23.	$m_{\rm conv}$	mass co-ordinates of convective boundaries
24.	$m_{\varepsilon_{\rm H,max}}$	mass co-ordinate of maximum energy generation from H-burning
25.	$m_{\varepsilon_{\rm He,max}}$	mass co-ordinate of maximum energy generation from He-burning
26.	$\log_{10}\kappa$	logarithm of the opacity
27.	Δt	timestep in years
28.	$X_{{\rm H},s}$	surface abundance of hydrogen
29.	$X_{{\rm He},s}$	surface abundance of helium
30.	$X_{{\rm C},s}$	surface abundance of carbon
31.	$X_{{\rm N},s}$	surface abundance of nitrogen
32.	$X_{{\rm O},s}$	surface abundance of oxygen
33.	$X_{^3{\rm He},s}$	surface abundance of 3He
34.	$R_*/R_{\rm RL}$	radius as fraction of Roche lobe radius
35.	J_1	spin angular momentum of star
36.	$P_{\rm bin}$	period of binary in days
37.	$r_{\rm sep}$	separation of binary
38.	M_1+M_2	binary mass
39.	$J_{\rm orb}$	orbital angular momentum
40.	J_1+J_2	total spin angular momentum
41.	$J_{\rm tot}$	total angular momentum
42.	$\omega_{\rm orb}$	angular freq. of orbit
43.	ω_1	angular freq. of star
44.	I_1	moment of inertia of star
45.	$I_{\rm orb}$	moment of inertia of orbit
46.	$\dot{M}_*$	mass loss rate
47–58.	$m_{\rm shell}$	mass location of burning shell boundaries
59–70.	$m_{\rm th}$	location of boundaries of regions where thermohaline mixing is active
71.	$M_{\rm conv\text{-}env}$	mass in the convective envelope (if there is one)
72.	$R_{\rm conv\text{-}env}$	radius of the base of the convective envelope (if there is one)
73.	$\log_{10}\rho_{\rm c}$	logarithm of central density
74.	$\log_{10}T_{\rm c}$	logarithm of central temperature

3.9 `modout[2] (34)`

These are where the code outputs models to. To restart a run from a given point, you need to copy the appropriate model from `modout` and put it into `modin`. Usually you would simply use the last output model but you could choose any model from `modout`. The format of `modout` is identical to that of `modin`. The frequency with which models are written to the output is controlled by `NSAVE` in `data`. The last model of a run is always printed.

3.10 `nucmodout[2] (35)`

`nucmodout` is to `nucmodin` what `modout` is to `modin`.

3.11 `syntha,synthb,synthc[2] (36,37,38)`

These contain the detailed models for the nucleosynthesis files. There are 3 `synth` files: `syntha`, `synthb` and `synthc`. `syntha` has the lightest species in it starting from the gallinoes and going up to ^{22}Ne. `synthb` has ^{22}Na up to ^{34}S and `synthc` has the iron group elements as well as the nucleosynthesis code's values of the structural isotopes ^{1}H, ^{4}He, ^{12}C, ^{14}N, ^{16}O and ^{20}Ne. All the abundances are printed with with the meshpoint number in the first column and the mass co-ordinate in the last column.

Detailed nucleosynthesis models are printed out with the same frequency and number of mesh points as the detailed structure models. These settings are controlled by `NWRT1` and `NWRT2`. They are not printed if you are not using the nucleosynthesis routines.

3.12 `surface[2],centre[2] (39,40)`

These files contain the abundances of all the isotopes in the structure code at the surface and centre in the order they are listed above in `nucmodin` (see Section 3.6). They are preceded by the model number and the age to aid plotting.

In addition, `surface` also contains data that is useful for computing yields. The abundances are followed firstly by the wind mass-loss rate (in $\mathrm{M}_{\odot}\,\mathrm{yr}^{-1}$) and the mass-loss rate from non-conservative mass transfer (or mass lost during common envelope evolution). The final entry is the timestep in years. From all this data, one can compute the yield for every species using a separate FORTRAN program. This is not presently included with the code distribution but it is the maintainer's intention to include it later.

3.13 `sprocess[2] (41)`

Originally, these file were going to be used to supply input to a post-processing code to compute *s*-process nucleosynthesis in AGB stars. This has largely been abandoned in favour of using the MONTAGE code to do post-processing in general (see the next section). However, this file does contain useful information like the temperature in the various burning shells. The entries are

1. the model number.
2. the model age in years.
3. the mass co-ordinate of the H-burning shell.
4. the mass co-ordinate of the He-burning shell.
5. the intershell mass (i.e. 3-4, remember this was originally designed for AGB stars!)
6. the stellar mass.
7. the abundance of ^{14}N in the ashes of the H-burning shell.

8–19. the temperature at the convective boundaries, in units of 10^8 K.

20–31. the density at the convective boundaries.

32. the temperature at the base of the convective envelope.
33. the temperature in the H-burning shell (at the point that $X_{\mathrm{H}} = 0.345$).
34. the temperature at the base of the H-burning shell (at the point that $X_{\mathrm{H}} = 0.05$).
35. the temperature in the He-burning shell (at the point that $X_{\mathrm{He}} = 0.45$).
36. the temperature at the base of the He-burning shell (at the point that $X_{\mathrm{He}} = 0.05$).
37. the Mira pulsation period in days, based on the pulsation law in Vassiliadis & Wood (1993).
38. the absolute value of the mass-loss rate in $\mathrm{M}_{\odot}\,\mathrm{yr}^{-1}$.

3.14 `montage[2] (42)`

This is currently very experimental output. The hope is to be able to take output from the stellar evolution code and feed it in to a post-processing nucleosynthesis routine (specifically Ross Church's MONTAGE program) to calculate nucleosynthesis for a wider range of isotopes (including the *s*-process isotopes) than the evolution code can handle.

The output first throws out a line including the number of mesh points, the model number, age in seconds followed by the hydrogen and helium luminosities in $L_\odot$. There then follows a detailed model with the temperature in K), density in $g\,cm^{-3}$, mixing length in cm, convective velocity in $cm\,s^{-1}$, radius in cm, mass in $M_\odot$ and the ^{1}H, ^{4}He, ^{12}C, ^{14}N and ^{16}O abundances printed for each mesh point.

4 Code structure

A new section describing some of the broader things used in the code so that people know what they're dealing with when they start modifying it.

4.1 Architecture

The code has a few consistent design features across all the subroutines that should be known when making modifications. Above all, users who wish to tamper with the code must be familiar with the FORTRAN 77 standard. One example of such a 'feature' is the fixed width of input lines (64 characters) unless the appropriate compiler flag is issued at compilation. Another 'feature' is the reservation of the first 7 characters of each line for indexing.

This section does *not* claim to *justify* these design choices. It only describes what they are.

4.1.1 Common blocks

The code uses 'common' blocks. These are basically a way of declaring global scope for variables, which introduces a number of things that must be kept in mind when viewing or modifying the code.

First, note that a single common block is effectively one large array with names for the elements of the array. This means that, given a common block with two elements ...`varA,varB`..., the coder can access `varB` by referring to `varA(2)`. Unless specifically instructed, this doesn't, by default, produce any warnings or errors when compiling. There are (or, at least, *were*) parts of the code where this trick was used. Warrick Ball is sure that the nuclear reaction rates were abused in this manner.

Secondly, though less critically, variable names need not be the same between subroutines. For example, the variables in common block `INF` go by many names across different subroutines. When tracking a variable through the code, keep in mind that it might appear to vanish when really it's just changed its name.

4.1.2 I/O and linking

All of the code's input and output is done by reading from or writing to the default output files, which have filenames that match `fort.*`. For example, when the code says `WRITE(32,99003)`, it writes output to `fort.32` using the fixed format defined by the line prefixed with `99003` in the same source file. The link to real filenames (e.g. `COtables`, `out`) is made by the execution script, `run`. Thus, trying to run `bs` directly probably won't work because the wrong data will be read.

The `fort.*` files should be cleaned up by `run` before and after execution to avoid confusion. If a run is aborted for some reason, the `fort.*` files may persist but they shouldn't cause any problems.

4.1.3 Code units

The code generally defines masses in units of 10^{33} g, lengths in 10^{11} cm and luminosities in 10^{33} erg s^{-1}. In this form, many factors of 10^{11} cancel out so the code is a bit neater. In these 'Eggleton' units, the solar mass, luminosity and radius are 1.989, 3.844 and 0.696. Tinkerers should be aware of these units when adding new routines. Adrian Potter at least once found his diffusion coefficients were out by a factor of 10^{22}. Eggleton units were to blame!

4.2 Subroutines

A list of all of the subroutines, in alphabetical order, is given below. The relevance of the subroutines vary wildly. Some are mostly fixed black boxes (e.g. `difrns`); others must be modified extensively when including new processes (e.g. `funcs1`).

`compos`	Prevents abundances from becoming negative by rounding them to zero when smaller than 10^{-12}.
`consts`	Sets up physical constants.
`diffusion`	
`diffusion2`	
`difrns`	Varies the equations to calculate numerical derivatives that populate the matrix that is inverted.
`divide`	Performs the matrix inversion. Intimately connected to `elimn8`. See Appendix C for details about how the inversion is performed.
`elimn8`	Does some matrix multiplication to streamline the inversion. Intimately connected to `divide`. See Appendix C for details about how the inversion is performed.
`equns1`	Sets up difference equations for the structure and main composition variables.
`equns2`	Sets up difference equations for the minor composition variables.
`fdirac`	Calculates Fermi–Dirac integrals by using the fitting function described by Eggleton et al. (1973).
`funcs1`	Evaluates functions of the variables that are required for the difference equations. In effect, most of the structure equations (e.g. hydrostatic equilibrium) are described here.
`funcs2`	Evaluates functions of the minor composition variables that are to be calculated.
`main`	The primary execution loop. This is where the code starts.
`massloss`	Calculates mass-loss rates. Mass loss rates were previously computed in `funcs1` but it is now done in this separate subroutine. See the entries for `IML1` and `IML2` in `data` (Section 3.1) for the available options.
`netyield`	
`neutron`	
`nucrat`	Returns nuclear reaction rates.
`nucrat2`	
`opacty`	Old opacity routine. Called by using `IOP`=0 or 1.
`opspln`	Combined with `spline`, calculates spline interpolations.
`pressi`	Calculates the contribution of pressure ionization in the equation of state.
`printa`	The main input subroutine. Also provides initial output and advances the evolution by one timestep at a time.
`printb`	The main output subroutine.
`printc`	Produces additional debugging output, which is suppressed by default. See the entry for `J1`, `J2`, `IH` and `JH` in `data` (Section 3.1).
`remesh`	Re-initializes the mesh, if necessary. Only called on the first iteration.

`solver`	Primary controller for the Newton–Rhapson iterations of the relaxation algorithm. This subroutine contains most of the convergence tests.
`spline`	Combined with `opspln`, calculates spline interpolations.
`statef`	The equation of state subroutine.
`statel`	A driver for the equation of state that checks whether it needs to be recalculated. It basically prevents the code from recalculating the state variables when it isn't necessary.
`xopac`	New opacity routine by Eldridge & Tout (2004). Called by using `IOP=5`, which is the default choice.

5 Input physics

6 Advanced user guide

A

STARS cheat sheet

Plot

1. N
2. t
3. $\log_{10}(R_*/\mathrm{R}_\odot)$
4. $\log_{10} T_{\mathrm{eff}}$
5. $\log_{10}(L_*/\mathrm{L}_\odot)$
6. $M_*/\mathrm{M}_\odot$
7. $M_{\mathrm{H}}/\mathrm{M}_\odot$
8. $M_{\mathrm{He}}/\mathrm{M}_\odot$
9. L_{H}
10. L_{He}
11. L_{C}

12–23. m_{conv}

24. $m_{\varepsilon_{\mathrm{H,max}}}$
25. $m_{\varepsilon_{\mathrm{He,max}}}$
26. $\log_{10}\kappa$
27. Δt
28. $X_{\mathrm{H},s}$
29. $X_{\mathrm{He},s}$
30. $X_{\mathrm{C},s}$
31. $X_{\mathrm{N},s}$
32. $X_{\mathrm{O},s}$
33. $X_{^3\mathrm{He},s}$
34. R_*/R_{RL}
35. J_1
36. P_{bin}
37. r_{sep}
38. M_1+M_2
39. J_{orb}
40. J_1+J_2
41. J_{tot}
42. ω_{orb}
43. ω_{l}
44. I_{l}
45. I_{orb}
46. $\dot{M}_*$

47–58. m_{shell}

59–70. m_{th}

71. $M_{\text{conv-env}}$
72. $R_{\text{conv-env}}$
73. $\log_{10}\rho_{\mathrm{c}}$
74. $\log_{10} T_{\mathrm{c}}$

Out and code variables

1. ψ
2. p
3. ρ
4. T
5. κ
6. ∇_a
7. ∇
8. $\nabla_r-\nabla_a$
9. m
10. X_{H}
11. X_{He}
12. X_{C}
13. X_{N}
14. X_{O}
15. X_{Ne}
16. X_{Mg}
17. r
18. L
19. $\varepsilon_{\mathrm{th}}$
20. $\varepsilon_{\mathrm{nuc}}$
21. ε_ν
22. δm
23. k^2
24. $n/n+1$
25. $d\log r/d\log m$
26. $d\log p/d\log m$
27. u
28. S
29. L/L_{Edd}
30. $w\cdot l$

1. $\log f$
2. $\log T$
3. X_{O}
4. $\log m_{33}$
5. X_{H}
6. C
7. $\log r_{11}$
8. L_{33}
9. X_{He}
10. X_{C}
11. X_{Ne}
12. X_{N}

Mesh Function

$$Q = \log\left(\frac{1}{c_6}\left(\frac{m}{M_*}\right)^{2/3}+1\right) - c_3\log\left(\frac{r^2}{c_8}+1\right) + c_7\log\left(\frac{T}{T+c_0}\right)$$
$$+c_4\log p + c_5\log\left(\frac{p+c_9}{p+c_{-1}}\right) + c_2\log\left(\frac{p+c_{10}}{p+c_{-1}}\right)$$

$c_0 = 20\,000\,\mathrm{K}$
$c_{-1} = 10^{10c_1}$

B Veteran tips

This appendix outlines tips for the code that you might find useful. Each subsection describes a trick that at least one previous user has used to make their use of the code more efficient. If you have your own dark methods, share them with us and they may find themselves included here!

B.1 Comments in I/O files

All of the STARS code's input and output is in the form of fixed-format plain-text files. This section briefly discusses a few useful ways of abusing the fixed formatting. In essence, all the tricks make use of the fact that any extra information, outside the bounds of the fixed formatting, is ignored.

For example, the first line of `data` looks like

```
 199   40   10   15   15    3    1    1    1    1    1    1
```

You might forget what these numbers mean. The code only knows to read precisely the correct number of numbers in each line, so we can remind ourselves what the last, say, three numbers are by leaving notes to the right-hand side.

```
 199   40   10   15   15    3    1    1    1    1    1    1    ... NCH JP ITH IX IY IZ IMODE
```

Naturally, you can write endless remarks after the last line, too. The default `data` file has such a block after the controls specified in Section 3.1.

The fixed formatting actually allows one to store a complete sequence of evolutions in a single `modin` file. For example, suppose we created and evolved a $5\,\mathrm{M}_\odot$ ZAMS model by allowing the default $1\,\mathrm{M}_\odot$ model to contract on the ZAMS, adding mass until $M_* = 5\,\mathrm{M}_\odot$ and finally evolving the model. After allowing the default model to contract, you would normally copy the last (or other suitable) output model from `modout` to `modin`. Instead copy the model to the beginning of `modin` *but leave the old model in place*. The code only reads the number of lines it needs and then stops. The extra lines that correspond to the old models will not be read. This is useful if you are experimenting with a series of models to achieve something slightly exotic. If you find you have created an unworkable model, you can easily roll back to a previous, working, model.

B.2 Command line tools for manipulating plain text

Because the input and output is all in fixed-format plain-text files, it is easy to manipulate using basic command line tools in GNU/Linux and other Unix-like operating systems.

`head` Reads the first 10 lines of a file. If the first `N` lines are desired, you can use `head -N <file>`.

`tail` Like `head` but for the last 10 (or `N`) lines of a file. Useful for retrieving the last model from `modout`, which occupies the last $2N+1$ lines of the file.

`awk` A fairly complicated tool but capable of printing individual columns by using `awk 'print $a,$b' <file>`, where `a` and `b` are numbers. So, for example, to extract only the 4th and 5th columns of plot (for a theoretical HR diagram), you would enter `awk 'print $4,$5' plot` and the result would be displayed at the standard output.

`grep` Returns lines that match a given expression. Useful for isolating parts of `out`. Its cousin `egrep` matches regular expressions.

`sed` A tremendously powerful command line tool that can perform a number of tasks including (but far from limited to) printing a range of lines, printing every *n*th line and deleting blank lines.

If you find yourself frequently gathering up fragments of the STARS output, it is recommended that you consider using some of these tools.

As an example, suppose we wanted to retrieve the model profiles in `out` without the line breaks every 10th line. The command

```
grep -A 219 '  K  ' out
```

returns the 219 lines following a match to ' K ', which always precedes the profiles. To delete the blank lines, use `sed` to delete lines that simply start and end.

```
grep -A 219 '  K  ' out | sed '/^\$/d'
```

where I have used the appropriate regular expression. The syntax `sed '/expr/d'` deletes lines on which `expr` appears. You may also wish to delete the 'header' line of the models, in which case

```
grep -A 219 '  K  ' out | sed '/^\$/d' | sed '/  K  /d'
```

will suffice.

If you master the command line tools above, you will find yourself able to effortlessly make elaborate constructions from the code output.

B.3 Separating models in subdirectories

If you intend to run large numbers of models, having the executable and I/O in a single folder causes problems because of the I/O linking. In short, two runs of the code in the same place will try to link the same files and, unsurprisingly, quickly produce garbage. The solution is to separate separate runs into separate folders and modify `run` to link the data files in their new relative locations.

The script `run` starts with variables that specify where files are.

```
#!/bin/csh

set BSDIR=.
set DATDIR=$BSDIR/dat
set EXEDIR=$BSDIR
```

Specifically, they say where the physical data files (e.g. opacities) and the executable are. It is presumed that the model input is in the same folder as the script.

Suppose we want to run 3 models with masses 1, 3 and $5\,M_\odot$ in parallel and that we have the relevant input files available. Suppose also that STARS is being run in the current directory `/home/user/stars/`. Create

subdirectories *m*1/, *m*3/ and *m*5/ and move the model files (`data`, `modin` and, if necessary, `nucmodin`) to the relevant subdirectory. The executable script must be modified to reflect the locations of all the information.[1]

```
#!/bin/csh

set BSDIR=/home/user/stars
set DATDIR=$BSDIR/dat
set EXEDIR=$BSDIR
```

Now, to evolve the $1\,M_{\odot}$ model, change to the relevant subdirectory (`cd m1`) and run `../run`. If you prefer, you can create a symbolic link to the script with `ln -s ../run run`. All of the output is produced in the relevant subfolder.

You may use this technique to create large numbers of models and even multiple layers of subfolder. You could have folders matching `stars/popI/m??.?` and `stars/popII/m??.?` to represent different masses and metallicities but you must be careful to make sure that the appropriate opacity tables are linked for the different models. The executable script is neatly organized and straightforward to edit and you are invited to do so to streamline your own use of the code.

[1] You can do this with relative folders like `..` but absolute folders are used here.

C Matrix inversion

The following section is drawn from an obscure plain-text file called `writeup.fig` that was included with Peter Eggleton's documentation from TWIN/EV. It is possibly the only extant explanation of how the matrix of derivatives is manipulated, by subroutines `divide` and `elimn8`, to improve the inversion that is required for the relaxation method. It also explains the meaning of the parameters `NE1`, `NE2`, `NE3`, `NB`, `NEV`, `NF`, `J1`, `J2`, `IH`, `JH`, as well as the arrays `ID`. These are all specified in `data` and tell the code what the structure of the matrix of derivatives is.

Below is a transcription of Peter Eggleton's original documentation. It is not quite verbatim but it is believed that his meaning and, to some extent, style have been preserved. Each part of the explanation begins with the present state of the matrix of derivatives, which shall be called A. Its components are, crudely,

$$A_{ij} = \frac{\partial(\text{equation})_i}{\partial(\text{variable})_j}, \tag{C.1}$$

where 'equation' is either a difference equation or boundary condition and 'variable' is an intrinsic code variable. For illustrative purposes, the system of equations has been trivialized down to

- 6 variables: f, T, m, X, r and L;
- 3 meshpoints: surface, interior and centre, from left to right;
- 1 eigenvalue: C, at the far right;
- 3 zeroth-order surface BCs in the first three rows;
- 1 first-order surface BCs in the fourth row;
- 5 first-order DEs;
- 1 second-order DE;
- 5 first-order DEs again;
- 1 first-order central BC; and
- 4 zeroth-order central BCs.

This gives 19 equations in 19 variables.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	*				+														
L=r2T4		*			+	+													
mdot=f(T)		+	*																
h(f)X'=0	+			*			+			*									
r'=C/d(f)	+				*		+				*								+
L'=Ce(T)		+				*		+				*							+
P'=Cg(r)	*				+		*				+								+
T'=CL		*				+		*				+							+
m'=C			*						*										*
X"=R(T)		+		*				+		*				+		*			
r'=C/d(f)											*		+				*		+
L'=Ce(T)												*		+				*	+
P'=Cg(r)							*						*				+		+
T'=CL								*						*				+	+
m'=C									*						*				*
h(f)X'=0							+			*						*			
r=0																	*		
L=0																		*	
m=0															*				

The matrix has a block tri-diagonal structure. The variables and the equations have been ordered (i.e. permuted relative to their numbering in the code, which is indicated along the bottom, and down the right-hand side) so that very significant terms are on the leading diagonal—except for the bottom right-hand element. However, this element gets a significant value in the course of the elimination of the material below the leading diagonal (see later).

V . . . . V E (actual)
1 2 4 5 7 8 6 (1245876)
FSBC
SDE FDE . . . FDE
6 2 4 1 3 5 (642135)
FCBC ZCBC . ZCBC ZSBC . ZSBC
4 3 2 1 2 3 1 (4231231)

The above rows give the values for the array `ID`, except that (a) I 'misremembered' the ordering of L, r and consequential on that the ordering of the L', r' FDEs, and the L, r ZCBCs. This permutation doesn't matter at all. (b) The real code has 5 composition Variables instead of 1 (X). The 4 extra composition Variables are numbered 3, 9, 10, 11. The 4 extra SDEs are numbered 7, 8, 9, 10, as are the 4 extra FSBCs. The 4 extra FCBCs are numbered 5, 6, 7, 8.

The 4 below m means that the m-variable is stored as `H(4,K)` at meshpoint k. The 3 beside ZSBC means that in the ZSBC portion of subroutine `equns`, `EQU(3)` $= L - r^2T^4$. The 1 beside FDE means that in the FDE portion of `equns`, `EQU(1)` $= P_k - P_{k-1} - 0.5C(g_k + g_{k-1})$; and so on. In the illustration, P is assumed to be a function of f only, rather than of f, T, and its gradient P' to be C times a function of r only, etc.

* means a significant term; + is a minor term, might be comparable but need not be. Every block that contains either *s or +s also in practice contains more non-zero but unimportant terms; all blocks that have neither are completely empty, except for the 3rd and 4th block in the bottom row. Since the 'central' meshpoint is actually half a meshpoint in from the centre (to avoid singularities like m/r^2), the real ZCBCs involve *both* innermost meshpoints.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	1			+	+	+													
L=r2T4		1		+	+	+													
mdot=f(T)			1	+	+	+													
h(f)X'=0	+			*			+			*									
r'=C/d(f)	+				*		+				*								+
L'=Ce(T)		+				*		+				*							+
P'=Cg(r)	*				+		*				+								+
T'=CL		*				+		*				+							+
m'=C			*						*										*
X"=R(T)		+		*				+		*				+		*			
r'=C/d(f)											*		+				*		+
L'=Ce(T)												*		+				*	+
P'=Cg(r)							*						*				+		+
T'=CL								*						*				+	+
m'=C									*						*				*
h(f)X'=0							+			*						*			
r=0																	*		
L=0																		*	
m=0															*				

Successive calls to `difrns` in `solver` set up successive rows of blocks in the matrix. In between these calls, Gaussian elimination proceeds via calls to `elimn8` and `divide`.

The first call to `divide` multiplies the top 3 rows by the inverse of the leading 3×3 block, reducing this block to unity. The block to its immediate left is stored (and also the column on the far right, which is not illustrated: when operating on the array A in $Ax = y$, the same operations are done on y as on A, but we only illustrate A).

After the first call to `divide`, we call `difrns` again to fill the next 6 rows (SBC2 and DE1). Then a call to `elimn8` eliminates the blocks below the leading unit matrix: see next Fig.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	1			+	+	+													
L=r2T4		1		+	+	+													
mdot=f(T)			1	+	+	+													
h(f)X'=0				*	+	+	+			*									
r'=C/d(f)				+	*	+	+				*								+
L'=Ce(T)				+	+	*		+				*							+
P'=Cg(r)				+	+	+	*				+								+
T'=CL				+	+	+		*				+							+
m'=C				+	+	+			*										*
X"=R(T)		+		*				+		*				+		*			
r'=C/d(f)											*		+				*		+
L'=Ce(T)												*		+				*	+
P'=Cg(r)							*						*				+		+
T'=CL								*						*				+	+
m'=C									*						*				*
h(f)X'=0							+			*						*			
r=0																	*		
L=0																		*	
m=0															*				

Then in `divide` we divide the current 6 rows (SBC2 and DE1) by the leading 6×6 matrix: next page.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	1			+	+	+													
L=r2T4		1		+	+	+													
mdot=f(T)			1	+	+	+													
h(f)X'=0				1						+	+	+							
r'=C/d(f)					1					+	+	+							+
L'=Ce(T)						1				+	+	+							+
P'=Cg(r)							1			+	+	+							+
T'=CL								1		+	+	+							+
m'=C									1	+	+	+							+
X"=R(T)		+		*				+		*				+		*			
r'=C/d(f)											*		+				*		+
L'=Ce(T)												*		+				*	+
P'=Cg(r)							*						*				+		+
T'=CL								*						*				+	+
m'=C									*						*				*
h(f)X'=0							+			*						*			
r=0																	*		
L=0																		*	
m=0															*				

Next, we fill up the next 6 rows (DE2, DE1) in `difrns`. Then we use `elimn8` to eliminate everything below that part of the leading diagonal which is already reduced to unit matrices. See next fig.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	1			+	+	+													
L=r2T4		1		+	+	+													
mdot=f(T)			1	+	+	+													
h(f)X'=0				1						+	+	+							
r'=C/d(f)					1					+	+	+							+
L'=Ce(T)						1				+	+	+							+
P'=Cg(r)							1			+	+	+							+
T'=CL								1		+	+	+							+
m'=C									1	+	+	+							*
X"=R(T)										*	+	+		+		*			
r'=C/d(f)										+	*	+	+				*		+
L'=Ce(T)										+	+	*		+				*	+
P'=Cg(r)										+	+	+	*				*		+
T'=CL										+	+	+		*				+	+
m'=C										+	+	+			*				*
h(f)X'=0							+			*						*			
r=0																	*		
L=0																		*	
m=0															*				

Now use `divide` to convert the leading 6×6 array of rows DE2, DE1 to unity: next fig.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	1			+	+	+													
L=r2T4		1		+	+	+													
mdot=f(T)			1	+	+	+													
h(f)X'=0				1						+	+	+							
r'=C/d(f)					1					+	+	+							+
L'=Ce(T)						1				+	+	+							+
P'=Cg(r)							1			+	+	+							+
T'=CL								1		+	+	+							+
m'=C									1	+	+	+							*
X"=R(T)										1						+	+	+	+
r'=C/d(f)											1					+	+	+	+
L'=Ce(T)												1				+	+	+	+
P'=Cg(r)													1			+	+	+	+
T'=CL														1		+	+	+	+
m'=C															1	+	+	+	*
h(f)X'=0							+			*						*			
r=0																	*		
L=0																		*	
m=0															*				

Now fill up the last 4 rows (CBC2, CBC1) in `difrns`, and `elimn8` what is below the unit part of the leading diagonal: next fig.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	1			+	+	+													
L=r2T4		1		+	+	+													
mdot=f(T)			1	+	+	+													
h(f)X'=0				1						+	+	+							
r'=C/d(f)					1					+	+	+							+
L'=Ce(T)						1				+	+	+							+
P'=Cg(r)							1			+	+	+							+
T'=CL								1		+	+	+							+
m'=C									1	+	+	+							+
X"=R(T)										1						+	+	+	+
r'=C/d(f)											1					+	+	+	+
L'=Ce(T)												1				+	+	+	+
P'=Cg(r)													1			+	+	+	+
T'=CL														1		+	+	+	+
m'=C															1	+	+	+	+
h(f)X'=0																*	+	+	+
r=0																+	*	+	+
L=0																+	+	*	+
m=0																+	+	+	*

This must have the effect of putting some non-trivial value in the bottom right-hand corner, which was previously empty. In fact, the previous `elimn8` step will put there the product of two starred values—one at the far right of the last $m' = C$ row, and one in the last ($m = 0$) row—in the previous fig. This should leave a reasonably invertible 4×4 block in the bottom right: next fig.

eqn var	f	T	m	X	r	L	f	T	m	X	r	L	f	T	m	X	r	L	C
P(f)=g(r)	1			+	+	+													
L=r2T4		1		+	+	+													
mdot=f(T)			1	+	+	+													
h(f)X'=0				1						+	+	+							
r'=C/d(f)					1					+	+	+							+
L'=Ce(T)						1				+	+	+							+
P'=Cg(r)							1			+	+	+							+
T'=CL								1		+	+	+							+
m'=C									1	+	+	+							*
X"=R(T)										1						+	+	+	+
r'=C/d(f)											1					+	+	+	+
L'=Ce(T)												1				+	+	+	+
P'=Cg(r)													1			+	+	+	+
T'=CL														1		+	+	+	+
m'=C															1	+	+	+	+
h(f)X'=0																1			
r=0																	1		
L=0																		1	
m=0																			1

We can now proceed by back-substitution to obtain the required dC, dL, dr, ... , df at central meshpoint, dL, dr, ... , df at intermediate meshpoint and finally dL, dr, ... , df at surface meshpoint.

It should be clear that while some permutations of variables and/or equations have a trivial effect—e.g. interchanging r and L, or f and T—others permutations can have a completely disruptive effect — e.g. interchanging m and X. Determining a viable permutation is as much an art as a science, in my experience.

Suppose we wish to add a few extra equations, as I did recently: the moment of inertia I such that

$$I' = \frac{2}{3} m r^2 C, \tag{C.2}$$

(remembering that $C = m' = dm/dk$), and $P_{\rm rot}$, $H_{\rm orb}$ and e: the last 2 are the orbital angular momentum and the eccentricity. The extra BCs are $I = 0$ at the centre, and at the surface

$$\frac{\rm d}{{\rm d}t}\left(H_{\rm orb} + 2\pi I/P_{\rm rot}\right) = ... \tag{C.3}$$

$$\frac{{\rm d}H_{\rm orb}}{{\rm d}t} = ... \tag{C.4}$$

$$\frac{{\rm d}e}{{\rm d}t} = ... \tag{C.5}$$

The unspecified derivatives involve tidal friction, etc. We have apparently 1 new variable (I) and 3 new eigenvalues ($P_{\rm rot}$, $H_{\rm orb}$, e). There would be KM+3 new equations (KM = no. of meshpoints, = 3 in the illustration): KM-1 DE1s (for I) and 4 BCs, i.e. 1 CBC1 and 3 SBC1s. This looks OK, but apparently it isn't: the 3 new eigenvalues would contribute elements at the *right*-hand end of the top row of blocks, but the the 3 new SBC1s would require some non-trivial elements at the *left*-hand end, in order to give an invertible 6×6 block at the top left.

It is better in this case to reverse the order in which we scan through the meshpoints, putting the surface at the bottom right and the centre at the top left. This will give

One can squeeze in 3 new Variables (I, $H_{\rm orb}$, e) along the top, between m and X at each meshpoint, the new eigenvalue at the end, after C, the 3 new SBC1s after $\dot{m}$, the 3 new DE1s after $m' = C$, and the 1 new CBC1 after $m = 0$. This allows the leading diagonal matrices to be invertible.

The moral seems to be that it would have been better to write the code so that the centre of the star is at the top left and the surface at the bottom right. Apparently one should start at the end which has the *fewest* BCs. This is exactly the opposite of what I supposed in 1969 when I wrote the code. It should be possible to rewrite the solution package to work from centre to surface, but my heart sinks at the thought of it. Besides, one might wonder whether the saving of CPU time is worth it. The current version requires inverting 9×9 matrices, apart from the first and last; revised as I suggest this would be reduced to 7×7. But actually the figures are 13×13 and 11×11, since I have ignored 4 of the 5 composition equations. The CPU time goes as the cube of the size of the matrix.

Bibliography

Eggleton P. P., 1971, MNRAS, 151, 351

Eggleton P. P., 1972, MNRAS, 156, 361

Eggleton P. P., 1973, MNRAS, 163, 279

Eggleton P. P., Faulkner J., Flannery B. P., 1973, A&A, 23, 325

Eldridge J. J., Tout C. A., 2004, MNRAS, 348, 201

Pols O. R., Tout C. A., Eggleton P. P., Han Z., 1995, MNRAS, 274, 964

Pols O. R., Schroder K. P., Hurley J. R., Tout C. A., Eggleton P. P., 1998, MNRAS, 298, 525

Reimers D., 1975, Memoires of the Societe Royale des Sciences de Liege, 8, 369

Schröder K. P., Pols O. R., Eggleton P. P., 1997, MNRAS, 285, 696

Stancliffe R. J., Eldridge J. J., 2009, MNRAS, 396, 1699

Stancliffe R. J., Tout C. A., Pols O. R., 2004, MNRAS, 352, 984

Stancliffe R. J., Lugaro M., Ugalde C., Tout C. A., Görres J., Wiescher M., 2005, MNRAS, 360, 375

Stancliffe R. J., Glebbeek E., Izzard R. G., Pols O. R., 2007, A&A, 464, L57

Vassiliadis E., Wood P. R., 1993, ApJ, 413, 641