1

Programme EV, January 2005

A computer programme for single and binary stellar evolution

Peter P. Eggleton
Lawrence Livermore National Laboratory, L-413, 7000 East Ave, CA 94550, USA
and Institute of Astronomy, Madingley Rd, Cambridge CB3 0HA, UK
ppe@igpp.ucllnl.org

1.1 INTRODUCTION

There are a number of papers which describe in outline various aspects of the programme:

Eggleton, P. P. (1971) MNRAS, 151, 351
Eggleton, P. P. (1972) MNRAS, 156, 361
Eggleton, P. P., Faulkner, J. & Flannery, B. P. (1973) A&A, 23, 325
Eggleton, P. P. (1973) MNRAS, 163, 279
Han, Zh., Podsiadlowski, Ph. & Eggleton, P. P. (1994) MNRAS, 270, 121
Pols, O. R., Tout, C. A., Eggleton, P. P & Han, Zh. (1995) MNRAS, 274, 964
Nelson, C. A. & Eggleton, P. P. (2001) ApJ, 552, 664 (2001)
Eggleton, P. P. (2001) ASP Conf. 229, 157
Eggleton, P. P. & Kiseleva-Eggleton, L. G. (2002) ApJ, 575, 461

The first paper describes the treatment of thin shell sources, and by extension the more general treatment of the automatic positioning of the mesh points in the non-Lagrangian mesh which is used, the second describes the treatment of convective and semiconvective mixing, and the third the equation of state. The fourth paper comments briefly on the applicability of the programme to the evolution of a star of $4\,M_\odot$ between the main sequence and the onset of carbon ignition. The fifth is a recent rather brief update on the procedures used, and the sixth is an update on the equation of state, opacity and nuclear reaction network. The seventh concentrates on conservative binary-star evolution; in particular it concentrates on the massive parallelisation of more-or-less the code described here. The eighth outlines a non-conservative model which is (optionally) implemented in this programme, and the ninth implements it in the context of RS CVn and Algol binaries. A reference to one or more of these papers, as appropriate, would be appreciated, if you find the programme at all useful. The programme has been quite substantially modified, usually for the better, since 1971, but this writeup (not intended for publication) is the only record of the current version. You can reference it as 'private communication 20xx'.

The programme is set up to evolve stellar models with just 199 meshpoints in them (although up to 499 are allowed by dimension statements), distributed between the photosphere (optical depth $\sim$ 2/3) and the centre. During an iteration it solves implicitly not just the usual 4 stellar structure equations, but a larger set which includes in addition 5 composition-change equations and also 2 equations governing the positioning of meshpoints within the star – hence 11 simultaneous equations altogether. Recent versions have included 13, 19 or 34 simultaneous equations. I believe the extra equations contribute substantially to numerical stability, although not in all conceivable circumstances. They do mean that stars which develop thin burning shells as they approach or climb the giant branch can be evolved remarkably easily, in contrast to more normal procedures where the mesh is fixed (*i.e.* Lagrangian) during one time step, and the composition is updated after the time step. They also mean that semiconvective mixing can be taken care of without any extra algorithm: the implicit solution of the diffusion equation governing composition changes in convective regions automatically includes semiconvective mixing where it is required. The simultaneous solution of the structure, mesh and composition, allied to the fact that the difference equations are differentiated numerically rather than analytically, allows the code to be relatively short ($\lesssim$ 3000 lines); this further means that it is relatively easy to operate, and particularly to modify.

When I say '19 simultaneous equations' I do not mean '19 first-order differential equations'. This particular version has a mixture of 5 second-order differential equations for composition, i.e.

$$\frac{1}{\rho r^2}\frac{\partial}{\partial r}\rho\sigma r^2\frac{\partial X}{\partial r} = \left(\frac{\partial X}{\partial t}\right)_m + R_{\rm nuc} \qquad ,$$

8 first-order differential equations like

$$\frac{1}{\rho}\frac{\partial p}{\partial r} = -\frac{Gm}{r^2} + \frac{2}{3}\Omega^2 r \qquad ,$$

and 6 zero-order equations like

$$\frac{d\,I\Omega}{dt} = \text{surface terms involving magnetic braking and tidal friction.}$$

I am assuming here that the rotation rate Ω is uniform through the star (because of magnetic fields); but if you believe you know an equation, say a second-order diffusion-like differential equation, governing the redistribution of angular velocity throughout the star, you can write it in yourself.

The present version of the code normally includes 5 composition variables within the same solution step, these variables being ^{1}H, ^{4}He, ^{12}C, ^{16}O and ^{20}Ne. ^{14}N is included indirectly, by appealing to conservation of CNO nuclei. There is also an optional second solution step in which 6 composition variables are solved simultaneously (^{1}H, ^{3}He, ^{4}He, ^{12}C, ^{14}N, ^{16}O), but the structure and mesh are kept fixed in this step in the interest of economy of computer time. This optional second step has not been updated for some time, and so is probably no longer compatible with the other subroutines, but it shouldn't be difficult to update and enlarge it if required. It assumes, of course, that certain further nuclear species, such as ^{13}C, ^{15}N, ^{17}O and ^{18}O (not to mention ^{2}H, ^{7}Be and ^{7}Li) are in transient equilibrium. I am indebted to Drs B. P. Flannery and S. Ramadurai for writing portions of this.

To start some evolution, you have to specify two masses and a binary period; then when the first star's radius reaches its Roche lobe radius, mass is lost at a rate proportional to the cube of the fractional excess of the star's radius over its lobe's radius. For evolution of a single star, just choose an orbital period so large that there is no prospect of Roche lobe overflow. Note that because of the fact that the mesh-spacing is computed along with the structure, the inclusion of RLOF is almost trivial: it is only necessary to change one boundary condition, from 'M = given' to 'dM/dt = - const. $*$ max (0, $[\ln\{R_{\rm star}/R_{\rm lobe}\}]^3$)'. While following the evolution of a lobe-filling, mass-losing star, we store its mass-loss history and use this as input for a subsequent calculation of the companion's evolution as a gainer. If however you wish to follow the evolution through a possible contact phase, then you would do best to solve both components *simultaneously*: see TWIN below.

I have recently introduced an alternative way of doing RLOF, in which the mass transfer rate is computed by integrating some simplified equations for Bernoulli flow over the layers of a star between its surface and its inner critical surface (i.e. its Roche lobe). This does not make much difference in practice, if only one star fills its Roche lobe; but it can apply more seriously in contact binaries, where *both* stars fill their Roche lobes, presumably by very slightly different amounts.

An important recent change is to the equation of state, by Dr O. R. Pols. This now includes an approximation to coulomb interaction between the electrons and ions. The effect is to bring the EoS into much better agreement with sophisticated computations by Rogers & Iglesias (ApJS, 79, 507, 1992) and Mihalas, Däppen & Hummer (ApJ, 331, 815, 1988), but at very little extra cost in cpu time. The treatment of pressure dissociation has also been improved. Drs Pols and C. A. Tout have also updated the physical input tables, as described below.

A recent version (since ~ June 2000) has been developed to work on massively parallel machines, by C. A. Nelson. However, the code included in this package is essentially the non-parallel core of a potentially parallelisable system. It is set up in principle to do a 3D grid of binary runs, each element of the (equally log-spaced) grid being defined by a primary mass, a mass ratio, and an initial period. Of course one can choose to do only one grid point; and one can further choose to do only one component, *i.e.* a single star, by putting the binary companion in a suitably long-period orbit and not bothering to evolve it.

Recently (May 2003) I have added an algorithm which, at the core He flash of a low-mass star, replaces the pre-He-flash model by a post-He-flash (ZAHB) model with the same core mass and total mass. This then allows the evolution to continue. So far, I have only done this for solar metallicity, but I hope to get around to doing this for the other metallicities tabulated. Interested readers can probably do this themselves, following some guidelines given later.

The current version optionally includes models of mass loss (ML), magnetic braking (MB) and tidal friction (TF), and allows for the possibility that the orbit starts out eccentric, with unsynchronised stellar spin, and then pseudo-synchronises, circularises, and ultimately synchronises. It requires that one solve a larger set of equations: 13 or 19 instead of the usual 11. This part of the code is fairly novel, and by no means guaranteed (but nothing else is guaranteed either). It can be avoided by putting certain input parameters to zero, and solving only the smaller set of

11 equations. The effect of GR on the orbit (period and eccentricity) is always included, even in runs labeled 'conservative'.

Fairly recently (~ April 2002) I have developed what I call the 'TWIN' code, which solves the two stars *simultaneously.* It can therefore allow for ML, MB and TF in *both* components. Dr R. F. Webbink is working on a formulation of luminosity transfer within a contact envelope, so with any luck we shall be able to do contact binaries as well. But even apart from contact binaries, it allows one to evolve binaries where, for example, there is MB and TF in *both* components.

Several different options are available, which can be selected by appropriate choice of numbers in the input 'configuration file'. Regarding the components separately, the options might be described as (a) 'Fully Conservative', FC, (b) 'Partially Non-Conservative', PNC, or (c) 'Wholly Non-Conservative', WNC. More specifically:

FC: the component's mass is constant, except (at suitably short binary periods) for RLOF

PNC: in addition to possible RLOF, a star might lose mass according to some prescription which is a function of surface values M, R, L

WNC: in addition to both the above, there may be mass loss (such as a dynamo-driven stellar wind) which is a function of $P_{\rm rot}$ as well as M, R, L. Evidently to determine how $P_{\rm rot}$ evolves with time, we need some extra differential equations, e.g. for the moment of inertia.

Considering that a binary has two components, we can imagine 9 possibilities, but only 4 make any real sense: (i) FC + FC, (ii) PNC + FC, (iii) WNC + FC, and (iv) WNC + WNC. In the first three of these, the fact that $*2$ is FC means that it is possible to follow the entire evolution of $*1$ before computing separately the entire evolution of $*2$ – except that if $*2$ fills its own Roche lobe at some stage (which it normally will) the evolution is invalid beyond that point. This makes the evolution relatively cheap in cpu time, since the maximum number of equations being solved simultaneously at any one time is 11 for (i), 13 for (ii) and 19 for (iii). Option (iv) requires the TWIN version with (currently) 34 equations.

1.2 THE PACKAGE

The following will be sent, if required:

(1) The Fortran programme, consisting of a main routine and 35 subroutines. It is a little less than 3000 lines long. It is in double precision, and is liable to underflow interrupts on a machine which cannot take products under about 10^{-60}. About 12 - figure accuracy is what is actually required; this is because the difference equations which are set up are differentiated numerically rather than analytically. The advantage of numerical differentiation is that it is possible to change the equations very easily, if one wishes to modify the programme. The package also includes physical data for solar composition, and a range of ZAMS models to start from: $\log M = -1.0\ (0.025)\ 2.3$.

(2) Tables of physical data, for various zero-age metallicities. This is described more fully in the sixth paper listed. Crudely, each table contains: opacities for up to 10 compositions at various stages of hydrogen, helium and carbon depletion; weak-interaction neutrino loss rates from Itoh *et al.* (ApJ, 339, 354, 1989, and ApJ, 395, 622, 1992, with errata); nuclear reaction rates for 20 reactions from Caughlan & Fowler (Atomic Data & Nuclear Data Tables, 40, 284, 1988) and Caughlan *et al.* (1985, Atomic Data & Nuclear Data Tables, 35, 198, 1985). The opacities are from Rogers & Iglesias (ApJS, 79, 507, 1992), from Alexander & Ferguson (ApJ,

437, 879, 1994) for temperatures below $10^{3.8}$ where molecules matter, and from Hubbard & Lampe (ApJ, 156, 795, 1969), modified by Itoh *et al.* (ApJ, 273, 774, 1973) for relativistic electrons, where degeneracy is important. Some of these data tables were retabulated fairly finely at equal increments of 0.05 in log T (3.0 to 9.3) and 0.25 in log ρ (-12.0 to 10.25). The published data fill only portions of this rectangle (by and large the areas of astrophysical interest), but we have padded out the rest of the rectangle with junk that fits on smoothly. The retabulation, where necessary, was by a quadratic interpolation in the actual tables. Naturally the finer retabulation that we use has no more information in it than the original tables. The code incorporates an efficient spline-interpolation procedure from C. A. Tout and L. Dray. There is also a file of ZAMS models to go with each metallicity.

(3) A test suite of 11 problems, with the input files that set them up, and also some output files from running them. These include construction of a ZAMS, evolution of single stars, evolution (conservative and non-conservative) of binaries, and construction of ZAHB and ZAHeMS stars, He and C/O white dwarfs, and pre-MS stars. These tests are discussed a little more fully in the last Section of this writeup.

With most of these files gzipped, they amount to about 50 Mbytes. I can also send a relatively short package, omitting all the output files (10 Mbytes), and a really short package of 2MB, where you will have to create some of the input files by running a job which uses a minimal input file. Of course, nobody worries about 50MB these days, but experience tells me that the shorter the file the more likely it is to arrive intact.

1.3 THE SUBROUTINES.

The purposes of the main routine and 35 subroutines can be summarised as follows. They are divided into five groups for convenience.

1: aamain.f ....

MAIN – This sets up a triple DO loop to step through a grid of binary-star parameters: primary mass, mass ratio and period. Only certain masses and mass ratios are readily allowed: those where both masses are of the form $10^{0.025n}$, with $-40 \leq$ n $\leq$ 92. This is because it reads the ZAMS models from a file storing those particular masses. With a little bit of trouble one can set up an alternative file of ZAMS models. The period can be chosen more arbitrarily, as any multiple $>$ 1 (though safer to have $\gtrsim$ 1.05) of the period at which the primary would fill its Roche lobe on the ZAMS. The last part of MAIN first evolves the primary, storing its mass-exchange history, and then (optionally) the secondary using the stored history to determine its accretion rate. A single star is just a special case of a binary star that you can take to be rather wide (and you don't compute the companion's evolution); and if you don't want a grid of models, but just one case, you only have to make sure that the loop parameters allow just one case, for each loop.

SETSUP – Initialises some mathematical and physical constants; reads in tables of opacities, nuclear reaction rates, neutrino rates, atomic statistical weights, ionisation energies, bolometric correction and $B-V$ colour etc.

OPSPLN – creates spline coefficients for opacity interpolation; used only once, at start of run.

SPLINE – calculates cubic spline coeffs for OPSPLN

PRUNER – This reads the output file of the primary and prunes it down to a shorter file from which the secondary can read its accretion history. The final product of a binary run is a file (fort.9) with the pruned output of the secondary following immediately on from the pruned output of the primary. I use this file as input to a separate plot routine.

LT2UBV – Computes $M_{\rm V}$, $B-V$ and $U-B$ from given L, T, using tables.

2: bbegin.f ....

STAR12 – The main evolutionary loop. At each of a specified number of time steps it calls the solution algorithm (dsolve.f; see below), a Newton-Raphson package which solves a large number of implicit simultaneous difference equations, to obtain the distribution of density, temperature, radius, luminosity, mass, five major composition variables, and (optionally) some quantities related to binarity: eccentricity, orbital angular momentum etc. Then, also optionally, it can do a second Newton-Raphson solution of a further collection of difference equations for a set of minor composition variables. Evolution is terminated when one or other of various conditions, set out in MAIN, is satisfied.

BEGINN – Mainly handles input of stellar models; it is called twice for a binary, once for $*1$ and once for $*2$. In the latter case it also reads back in data from the evolution of $*1$, such as the companion mass and the orbital angular momentum, that provide boundary conditions for the evolution of $*2$.

FGB2HB – When a low-mass ($\lesssim 2\,M_\odot$) model reaches the onset of the core He flash, this subroutine constructs a ZAHB star with approximately the same core mass and total mass, so that the calculation can continue. For a binary, it only works for $*1$ at present. It only applies to initial solar composition, at present.

NEXTDT – In between timesteps, it chooses – not always very effectively – the length DT of the next time step. The algorithm assumes that one wants a roughly constant mean-modulus (MM) change in the independent variables, so that if the MM change increases DT decreases, and vice versa.

UPDATE – In between timesteps, it updates the independent variables, storing the second-last as well as the last converged model.

BACKUP – In circumstances where DT has to decrease *rapidly*, e.g. the end of the MS, or the onset of RLOF, NEXTDT usually doesn't work very well: DT is not cut back enough, and the model fails to converge. In this case the code restarts from the *second*-last converged model, with DT decreased. This may happen as much as 5 times, in the transition from nuclear-timescale to thermal-timescale evolution.

OUTPUT – Every so often (according to a given datum) the interior details of a model are stored. They can be used to restart the run, if required.

3: cfuncs.f ....

FUNCS1 – Evaluates, at each mesh point in turn, those functions of the independent variables which are required in setting up the difference equations.

EQUNS1 – Sets up the difference equations, which may be a mixture of first and second order. Its input is the set of functions which are output from FUNCS1.

NAMES1 – For my TWIN model (both components simultaneously), this routine places the data for FUNCS1, either for $*1$ or for $*2$ as required, in locations with recognisable names. It also puts the output of FUNCS1 into locations for $*1$ or $*2$, as appropriate. Some locations are reserved for variables which depend jointly on both components.

NAMES2 – As NAMES1, but for EQUNS1 rather than FUNCS1.

PRINTB – Prints out stellar models, either in summary or in detail as required. It is also the place to evaluate integrals for, say, the gravitational/thermal binding energy, and the convective envelope turnover time of a model that has just been solved.

REMESH – Before doing any evolution, this puts the mesh points at equal intervals of the mesh-spacing function, if they are not there already (*e.g.* because one has decided to use a different mesh-spacing formula, say putting mesh points closer together near the centre, though consequentially further apart elsewhere). It can also interpolate to give say a 500 meshpoint model from a 200 meshpoint model. It further computes an initial guess for certain independent variables, e.g. potential and moment of inertia, that are not necessarily included in the given initial grid of models (which are all non-rotating).

POTENT – Works out the difference in potential between the surfaces through L1 and L2.

DJNVDH – Works out the mass-loss rate of luminous stars accoding to the formulation of de Jager, Nieuwenhuizen & van der Hucht (1988; A&AS, 72, 259)

STATEL – A storage buffer between FUNCS1 and STATEF (see below), to save recomputing the equation of state when only a variable that does not effect the state of the gas (e.g. radius) differs from the last call to STATEF.

CHECKS – Prevents abundances (also orbital eccentricity) from reaching very small numbers (like 10^{-100}), and also from going negative by small amounts. It sets them to zero if they are smaller than 10^{-12}.

FUNCS2 – As FUNCS1, but for minor composition variables only. However unlike FUNCS1 the various functions are differentiated analytically, thus saving computer time but at the expense of making alterations more difficult.

EQUNS2 – As EQUNS1, but relating to FUNCS2. Like FUNCS2, EQUNS2 involves analytic rather than numerical integration. Neither FUNCS2 nor EQUNS2 has been updated recently to keep pace with changes in the other subroutines, so they probably need some revision.

4: dsolve.f ....

SOLVER – Along with the next three subroutines, this is a very general solution package for systems of simultaneous difference equations, some first order and some second order, with appropriate boundary conditions at each end. It does not depend on the particular character of the stellar structure equations. Its purpose is to achieve, by Newton-Raphson iteration, a solution to the equations evaluated in EQUNS1 (or EQUNS2).

DIFRNS – This interacts with either FUNCS1 and EQUNS1, or FUNCS2 and EQUNS2, to set up the particular difference equations required. It evaluates the derivatives of these equations numerically, by varying each independent variable in turn, and passes the derivatives to SOLVER so that it can implement its Newton-Raphson scheme for solving the difference equations. But there is the option of evaluating the derivatives analytically: this saves computer time (hardly at a premium these days) but requires more human time in order to get these derivatives exactly right.

DIVIDE – Does the necessary matrix inversion. This is a custom-built matrix inverter, which I believe is a good deal smarter the anything I was able to get off the shelf.

ELIMN8 – Does some matrix muliplication required for successive elimination.

PRINTS – If required, this prints out considerable detail of the matrix inversion process during the Newton-Raphson iteration. It can be very useful for debugging, but you had better be sure that you know how the programme is supposed to work.

PRINTC – As PRINTS, still more optional detail.

5: estate.f ....

STATEF – Equation-of-state subroutine; computes all the required quantities which are functions only of the state of the gas, determined by electron degeneracy, temperature and composition.

FDIRAC – Evaluates an approximation to the Fermi-Dirac integrals.

PRESSI – Used by STATEF to compute a rather *ad hoc* approximation to pressure dissociation. It also contains an approximation to the Coulomb interaction.

NUCRAT – Evaluates the nuclear reaction rates, both for the structure, mesh and major composition package (FUNCS1, EQUNS1) and the minor composition package (FUNCS2, EQUNS2).

OPACTY – uses spline interpolation for opacity from tables.

There are also a couple of plotting programs which I use *after* rather than *during* a run.

1.4 THE INDEPENDENT VARIABLES.

For a single star, or each component of a rudimentary binary, there are 11 independent variables which have to be guessed, and then solved for, at each meshpoint. They are:

(1) ln f - a dimensionless quantity closely related to electron degeneracy: for the case where electrons are non-degenerate and non-relativistic, $f \sim 10^8 \rho / T^{1.5}$

(2) ln T - logarithmic temperature (Kelvins)

(3) $X16$ - fractional abundance by mass of ^{16}O

(4) m - mass (10^{33} gm)

(5) $X1$ - the abundance of ^{1}H

(6) C - the gradient of mesh-spacing function $Q(f, T, m, r)$ with respect to meshpoint number K. C does not vary with K, the meshpoint number, although it varies with time. It is in effect an eigenvalue

(7) ln r - logarithmic radius (10^{11} cm)

(8) L - luminosity (10^{33} erg/s). Not logged, because it may be negative

(9) $X4$ - the abundance of ^{4}He

(10) $X12$ - the abundance of ^{12}C

(11) $X20$ - the abundance of ^{20}Ne

For a more sophisticated binary, including mass loss, magnetic braking, rotation (uniform but time-varying) and tidal friction, a further 8 variables are stored:

(12) I - the moment of inertia of the interior material (10^{55} gm.cm^2)

(13) $P_{\rm rot}$ - the rotation period (days) of the star (here taken to be independent of depth, so that it is an 'eigenvalue', like C above)

(14) ϕ - the centrifugal-gravitational potential (ergs).

(15) $\phi_{\rm s}$ - the potential at the surface, minus the potential on the L1 surface (ergs); also an 'eigenvalue'.

(17) $H_{\rm orb}$ - the orbital angular momentum (10^{50} gm.cm^2/sec); also an 'eigenvalue'.

(18) e - the eccentricity: also an 'eigenvalue'.

(19) ξ - the flux of mass towards $*2$ and away from $*1$ (10^{33} gm/sec); a function of depth, but zero below the L1 surface.

(20) $M_{\rm B}$ - the total mass of the binary; depleted by wind in either or both stars, but not by mass transfer between the stars. Also an 'eigenvalue'.

I have left (16) blank, available for some other function. I have allocated space for 40 variables altogether, thinking of the first 16 as intrinsic to $*1$, the last 16 as intrinsic to $*2$, and the intermediate 8 as relating to *both* components, e.g. the orbital angular momentum and the eccentricity. The intermediate 8 can include a mass transfer flux between the components – which might be in either direction in a contact binary. In the TWIN mode, variables 25 – 40 are the same as 1 – 16, but for $*2$. It is already clear that 40 variables is nothing like enough, and I propose to increase this to 100 though probably not soon.

These variables are stored in a matrix H(J,K), with J = 1, KVB and K = 1, KH. A similar matrix DH(J,K) stores guesses for the increments of these variables in the current timestep. The solution package iterates to find the exact values of the DH's to satisfy the difference equations and the boundary conditions outlined below.

If the option of using the 'minor' composition variables, via FUNCS2 and EQUNS2, is used, there are a further seven variables which are $X(^1\mathrm{H})$, $X(^4\mathrm{He})$, $X(^{12}\mathrm{C})$, $X(^{14}\mathrm{N})$, $X(^{16}\mathrm{O})$, $X(^{24}\mathrm{Mg})$ and $X(^3\mathrm{He})$. These variables can be used, somewhat strangely, as a separate set from the major composition variables $X1, X4, X12, X16$ and $X20$ above, although in one mode of operation they can be related. The dimensions of H, DH are currently 40 variables by 500 meshpoints. At present, the FUNCS2/EQUNS2 package is incompatible with the normal use – it needs a bit of work.

1.5 THE DIFFERENCE EQUATIONS.

Define the following:

$A \;\equiv\; -d\ln P/dm \;=\; Gm/4\pi r^4 P$ (but optionally modifieed by rotation),

$\nabla \;\equiv\; d\ln T/d\ln P \;=\; \nabla(L, m, r, \chi, \nabla_a, CF)$, a function given by either radiative transport or mixing length theory (MLT), as appropriate. $\chi(\equiv P\kappa/T^4)$, $\nabla_a(\equiv$ adiabatic gradient) and $CF(\equiv C_P\rho\kappa T\sqrt{(C_P\nabla_a T)})$ are functions of state and composition only.

$\dot S \;\equiv\; dS/dt$ at constant meshpoint, *i.e.* non-Lagrangian time derivative

$\dot m \;\equiv\; dm/dt$ at ....

$\dot X \;\equiv\; dX/dt$ at ....

$\epsilon \;\equiv\; \epsilon_{\rm nuc} + \epsilon_\nu - T\,DS/Dt$ (Lagrangian time derivative)

$= (\epsilon_{\rm nuc} + \epsilon_\nu - T\dot S) + (C_P T d\log P/dm)(\nabla - \nabla_a)\dot m$

$\equiv\; (E_1) + (E_2)\dot m$, say (defining E_1, E_2)

$Q \;\equiv\; -c_1 m^{2/3} + c_2\ln P$ (actually a somewhat more complicated expression is used)

$Q' \;=\; dQ/dm \;=\; -2c_1/3m^{1/3} - c_2 A$

$m' \;\equiv\; dm/dk \;=\; C/Q'$, C being the eigenvalue quantity stored as independent variable (6) in the previous section

$R_{\rm nuc} \;\equiv\;$ Lagrangian rate of consumption of fuel = $\epsilon_{\rm nuc}$ divided by energy per unit mass from the relevant nuclear fuel

σ_0 $\equiv wl$, the convective mixing rate of composition, w being the turbulent velocity and l the mixing length from MLT:

$$\frac{DX}{DT} \equiv \left(\frac{\partial X}{\partial t}\right)_Q - \left(\frac{\partial X}{\partial m}\right)_t \left(\frac{\partial m}{\partial t}\right)_Q = \frac{1}{\rho} \nabla .(\rho wl \nabla X) + R_{\rm nuc} \quad .$$

However the wl used here is not quite the same as the wl obtained from MLT, partly because we add in a model of 'convective overshooting'. If we write the expected MLT value approximately as wl = coeff. $\times$ $[\nabla_{\rm r} - \nabla_{\rm a}]^{1/3}$ – see below for the definition of the square brackets – then we use an overshooting estimate which is wl = coeff. $\times$ $[\nabla_{\rm r} - \nabla_{\rm a} + \nabla_{\rm OS}]^2$. Using the square rather than the cube root smooths the transition from mixing to no mixing. Obviously we will get mixing in the region $\nabla_{\rm r} \geq \nabla_{\rm a} - \nabla_{\rm OS}$, rather than in the classical Schwarzschild core where $\nabla_{\rm r} \geq \nabla_{\rm a}$. In addition I typically multiply this wl by 10^{-2} or 10^{-4} (i.e. a data number called CRD), to make convergence better with (I hope) marginal loss of accuracy. $\nabla_{\rm OS}$ is a smallish quantity, given on p21, and calculated from two data numbers (COS, CU) and the ratio of radiation pressure to gas pressure.

$\sigma \equiv \sigma_0 (4\pi r^2 \rho)^2 / m'$ is the quantity appearing in the difference equations below, to allow for the fact that the equations are written with differentiation w.r.t m, not r.

NB HERE AND THROUGHOUT, [X] MEANS X IF X IS POSITIVE, AND ZERO IF X IS NEGATIVE.

Then the difference equations which the program solves are (qualitatively):

(1) $\sigma_{\rm k+1/2}(X_{\rm k+1} - X_{\rm k}) - \sigma_{\rm k-1/2}(X_{\rm k} - X_{\rm k-1}) \;=\; (\dot{X}_{\rm k} + R_{\rm nuc,k}) m'_{\rm k}$
$\quad - \;(X_{\rm k+1} - X_{\rm k})[\dot{m}_{\rm k}] + (X_{\rm k} - X_{\rm k-1})[-\dot{m}_{\rm k+1}]$

(2) – (5) similar to (1), for different composition variables: $X4, X12, X16, X20$.

(6) $\log P_{\rm k+1} - \log P_{\rm k} \;=\; -(Am')_{\rm k+1/2}$

(7) $r^2_{\rm k+1} - r^2_{\rm k} \;=\; (m'/2\pi\rho r)_{\rm k+1/2}$

(8) $\log T_{\rm k+1} - \log T_{\rm k} \;=\; -(\nabla A m')_{\rm k+1/2}$

(9) $L_{\rm k+1} - L_{\rm k} = (m' E_1)_{\rm k+1/2}$
$\quad + (m'E_2)_{\rm k}[\dot{m}_{\rm k}]$ - $(m'E_2)_{\rm k+1}\ [-\dot{m}_{\rm k+1}]$

(10) $m^{2/3}_{\rm k+1} - m^{2/3}_{\rm k} \;=\; (2m'/3m^{1/3})_{\rm k+1/2}$

The additional equations associated with ML/MB/TF are

(11) $I_{\rm k+1} - I_{\rm k} \;=\; (2m'r^2/3)_{\rm k+1/2}$

(12) $\phi_{\rm k+1} - \phi_{\rm k} \;=\; (Gmm'/4\pi r^4 \rho)_{\rm k+1/2}$

(19) $\xi_{\rm k+1} - \xi_{\rm k} \;=\; (\sqrt{[2\phi_s]}/r\ m')_{\rm k+1/2}$, if $\phi > 0$; $= 0$ otherwise.

There are only 13 equations, because out of the 19 variables 6 (identified as 'eigenvalues') are constant throughout the star. In the TWIN version, the corresponding numbers are 25, 34 and 9.

The square brackets used above and below are the same as defined above in capitals after σ. The suffix $k+1/2$ means an arithmetic average, *i.e.* a central-difference scheme is used. Equations (6) – (9) are a fairly standard centrally-differenced approximation to the usual stellar structure equations, except that the Lagrangian heat time derivative is replaced by a non-Lagrangian derivative along with an 'upstream' approximation for the advection term. Equation (10) determines the mesh spacing, such that meshpoints are put at equal intervals of the mesh-spacing function Q. The actual functional form of Q is a little more elaborate than the one given above. It is a sum of six terms, two of which depend on P only, two on T only, one on r only and one on m only. The first to fifth equations are composition-change equations, with convective mixing modeled by a diffusion approximation, and with another 'upstream' advection term in the time derivative.

The boundary conditions associated with the above equations are:
At the next-to-surface and next-to-centre meshpoints (K = 2 and KH − 1), the second order equations (1) to (5) each have a first order boundary condition

(1a, 2a, 3a, 4a, 5a, 1b, 2b, 3b, 4b, 5b) $\sigma_{\mathrm{k}\pm 1/2}(X_{\mathrm{k}} - X_{\mathrm{k}\pm 1}) = (\dot{X}_{\mathrm{k}} + R_{\mathrm{nuc,k}}).(m_{\mathrm{k}} - m_{\mathrm{k}\pm 1})$

making 10 such equations in all.

At the surface (K = 1)

(6a) $dM/dt \; = \; -\,\mathrm{CML}\,.\,|\dot{M}_{\mathrm{DDW}}(R, M, L, P_{\mathrm{rot}})| - \mathrm{CMJ}\,.\,|\dot{M}_{\mathrm{JNH}}| - \mathrm{CMR}\,.\,1.3\times 10^{-5} LM/|E_{\mathrm{B}}|$
$\qquad - \mathrm{CMS}\,.\,[\ln(R/R_{\mathrm{L}})]^3 - \mathrm{CMT}.\xi + \mathrm{CMI}\,.\,M,$
giving a combination of dynamo-driven wind, the luminous-star wind of de Jager et al (1988), a Reimers-like wind/superwind on the AGB, two versions of Roche-lobe overflow, and an artificial term that can be used for running up or down the ZAMS; the terms are explained in a little more detail below. Mostly the CM coefficients are either 0, to exclude the term, or 1, to include it; but CMS, CMT and CMI may need non-zero values other than 1.
(7c) $L \; = \; \pi acr^2T^4$
(8c) $P\kappa/g \; = \; 2/3$ – more or less
(9c) ϕ = surface value of potential
(10c) $d(I\Omega)/dt = \ldots$, the rate of change of angular momentum of the star, carried away by stellar wind $|\dot{M}_{\mathrm{DDW}}|$ or lost to the orbit by tidal friction ($\Omega \equiv 2\pi/P_{\mathrm{rot}}$)
(11c) $\phi_s = \phi$
(17c) $dH_{\mathrm{orb}}/dt = \ldots$, rate of change of *orbital* angular momentum, including tidal friction which exchanges ang. mom. between spin and orbit
(18c) $de/dt = \ldots$, rate of circularisation due to tidal friction
(20c) dM_{B}/dt = sum of winds from $*1$ and $*2$; M_{B} is the *binary* mass.

At the centre (K = KH)

(6d) $m = 0$
(7d) $L = 0$
(8d) $r = 0$
(9d) $I = 0$
(19d) $\xi = 0$.

Although there are apparently 19 variables in all, as listed in Section 1.3, actually there are only 13 at each meshpoint, plus the 'eigenvalues' C, $P_{\mathrm{rot}}, e, H_{\mathrm{orb}}, \phi_s$, M_{B}. The eigenvalues are stored for convenience at each meshpoint but have the same value at all. We therefore have 6 + 13.KH unknowns. The 8 first-order and 5 second-order difference equations (1) – (12), (19) are defined obviously at KH − 1 and KH − 2 meshpoints respectively, so that along with the

10 first-order boundary conditions (1a) – (5b) and the 14 zero-order boundary conditions (1a) – (19d) we have 6 + 13.KH equations, fortunately enough.

The mass-loss term $|\dot{M}_{\rm DDW}|$ comes from my simplistic model of the combination of dynamo action and magnetic field dissipation. The luminous-star wind of de Jager et al (1988) is taken from the Tchebyshev polynomial fit in their paper. The superwind term is Reimers-like for red giants and subgiants, where $E_{\rm B}$ – the binding energy of the envelope – is comparable to GM^2/R, but grows very large in red supergiants because $E_{\rm B}$, including the energy of ionisation etc. can go through zero at large enough radius and luminosity.

The RLOF mass-transfer term can be treated in two ways. On the one hand, in the cheaper version – where equation (19) is ignored – $\dot{M}$ is evaluated only at the surface, rather than integrated through the layers above the L1 surface. The more sophisticated (and more expensive) option, equation (19), is selected in the $\dot{M}$ boundary condition by choosing CMT in (6a) to be non-zero rather than zero (10^{-2} is OK). The latter option also applies to the TWIN version, with equation (19) being modified to give the possibility of flow in either direction, as the calculation demands. Some or all of the mass-change terms can be switched off by setting the appropriate CMx coefficient to zero.

The quantities $A, \nabla, ..., \sigma$ listed at the beginning of this section are all functions of the variables at a single meshpoint. They are evaluated in FUNCS1. The 13 difference equations, (1) – (12), (19), and also the 24 boundary conditions, (1a) – (19d), are functions of these quantities at one, two or three adjacent meshpoints, and are evaluated in EQUNS1. The subroutine DIFRNS calls FUNCS1 at each meshpoint as it passes through the mesh, and stores the last three sets of values so that when DIFRNS calls EQUNS1 it gives EQUNS1 as input the output of FUNCS1 at the last three meshpoints. In fact DIFRNS calls FUNCS1 20 times at each meshpoint, varying each of the 19 variables in turn by a small amount, to obtain a matrix of derivatives.

In technical detail, the form of the difference equations unfortunately depends on whether the surface, or alternatively the centre, is treated as the 'first' meshpoint. The code starts at whichever end is defined as 'first', and then sweeps through the mesh evaluating (a) functions and their derivatives at successive meshpoints, (b) difference equations and their derivatives at two or three successive meshpoints. The issue of what is upstream and what is downstream partly depends on this choice, as does the sign of dm/dk. This is handled a bit clumsily at present: the quantity dm/dk, and all other quantities in FUNCS1, have the same sign independent of direction, and the direction is allowed for only in EQUNS1, where KL = 0, KQ = 1 means that the surface is first, while KL = 1, KQ = -1 means that the centre is first.

There is a file in this directory called crib.txt which identifies the variables, the functions and the equations by name and also by number in the corresponding arrays.

Note that the heat equation implemented in this code is NOT CORRECT. Did you spot the two deliberate mistakes in the equations $\epsilon = ... = ...$ at the beginning of this section? Firstly, what we *want* is $DU/Dt + P\, D(1/\rho)/Dt$, which is not $T\, DS/Dt$ but $T(DS/Dt + \Sigma \mu_i\, DX_i/Dt)$, where the μ_i are the chemical potentials. Secondly, DS/Dt is not $\partial S/\partial T\,.\,DT/Dt + \partial S/\partial P.DP/Dt$, but needs an extra $\Sigma\ \partial S/\partial X_i.DX_i/Dt$. These two omitted terms don't cancel out; they amount to the omission of a term (I think) $\Sigma\ (\partial H/\partial X_i)_{P,T}\,.\,DX_i/Dt$, H being the enthalpy. I have omitted this through laziness, because it would be quite a lot of work to put it in, and I don't think there can be many circumstances where it matters. It can only contribute significantly if the composition is changing on a thermal timescale; but usually the composition is changing on a nuclear timescale.

In fact the difference equations in the code all differ in small ways from the difference equations as set out above. It might be helpful to distinguish three aspects of a particular variable: what it is physically, what is useful computationally, and how it is stored. Consider the physical variables m, r and L. Numerically, there is good reason to prefer the variables $m^{2/3}, r^2$ and $L^{2/3}$. This is because all three vary linearly with $\log P$, $\log T$ and $\log \rho$ near the centre, and this means that the difference approximations to their differential equations will be more accurate there. But actually $L^{2/3}$ is not very good, because L can be negative near the centre, and pass through zero on the way out. My numerical variable is therefore a bit of a monster:

$$\lambda \;\equiv\; \frac{L\sqrt{m_c^{2/3} + m^{2/3}}}{m^{1/3}} \ ,$$

where m_c is a characteristic 'central mass' given by $m_c = (3.5/\rho_c^2)\,(P_c/G)^{3/2}$. Central values P_c, ρ_c taken from the *previous* timestep rather than the current timestep are used. The stored variables are different again. I would prefer $\log m, \log r$ and $\log L$, mainly because I use changes in the stored variables to dictate the next timestep (see Section 1.9), and a mean-modulus change of say .01 would be meaningful if logs were used. However the logs are all undefined at the centre. I have ended up with an awkward compromise: m, L are stored as such, and not used in determining the next timestep, while r is stored as $\log(r_c^2 + r^2)$ with r_c being a 'central radius' which is just a small constant. It would probably be better if r_c, like m_c, were computed explicitly, from P_c, ρ_c of the previous model, but this is low down my list of priorities. The stored r-variable is used in determining the next timestep.

It will be clear to the reader that I am a fanatic for *implicit* calculations. It might seem easier to do quite a lot of the calculation explicitly, i.e. use values from the previous timestep rather than the current timestep. But it is my rather considerable experience that solving simultaneously for *all* current unknowns is a lot more robust than solving for some current unknowns while using previous values for some others. In this code I make three exceptions: $m_c, t_{\rm nuc}$ and $E_{\rm B}$. These are computed in printb.f from the previous model. The last two, nuclear timescale and envelope binding energy, are used in the the calculation of the surface mass-loss-by-wind recipe, (6a) above. That t_{nuc} is not shown there is because I have somewhat abbreviated the statement. If I replaced the constant r_c by the variable value I suggest above, it would also be explicit and calculated in printb.f.

1.6 THE SOLUTION PACKAGE

The subroutine SOLVER, along with DIFRNS, ELIMN8, DIVIDE and PRINTC, is a fairly self-contained package intended to solve a general collection of zero, first and second order difference equations with associated boundary conditions, some at each end. A 'zero-order' DE is a statement like boundary condition (20c), which determines an 'eigenvalue' like $M_{\rm B}$ rather than a function of position like P, ρ. The independent variables to be solved for can be a mixture of KEQ 'actual variables' (*i.e.* quantities which actually do vary across the mesh), and KEV 'eigenvalues', by which I mean quantities which don't vary through the mesh. There are KVB = KEV + KEQ 'apparent variables'; and there are KEV + KEQ.KH unknown quantities to solve for. Suppose there are KE1 first and KE2 second order equations (KEQ = KE1 + KE2); the former can be defined at KH − 1, and the latter at KH − 2 meshpoints. I take each second order equation to have one boundary condition at each end. The first order equations have say KBC boundary conditions at the start (K = KH), and so there must be KE1 − KBC + KEV boundary conditions at the end (K = 1) to close the system. The stellar evolution equations as outlined in Section 1.4 correspond to KE1 = 8, KE2 = 5, KEQ = 13, KBC = 5, KEV = 6 and KEQ = 19. It is permissible that a subset KE3 of the KE1 first order difference equations may involve quantities defined at three, rather than just two, adjacent meshpoints, although the present program does not use this (*i.e.* KE3 = 0).

A recent innovation is an integer KL whose value determines whether the 'start' is at the surface (if KL = 0) or at the centre (if KL = 1). Previously the start was always at the surface. However numerical efficiency requires one to start at the end with the *smaller* number of boundary conditions. Since this is now the centre, thanks to the surface boundary conditions (9c) – (20c), I have switched to KL = 1.

Although the difference equations are ultimately functions of the independent variables, it is convenient to treat them as functions of an intermediate set of functions which are evaluated at each meshpoint. For example, though $f, T, X1, X4, X12, X16$ and $X20$ are truly independent variables, as set out in Section 1.3, the difference equations are written naturally in terms of quantities like $P, \rho,$ *etc*, which are themselves functions of $f, T,$ *etc*. The subroutine FUNCS1 evaluates these intermediate functions, all of which must be definable at a single meshpoint. These intermediate functions (KFN of them) are passed to EQUNS1, which has access to these functions at not just the current, but also the previous and the ante-previous meshpoints. It is assumed for the present that no difference equation depends on more than three adjacent meshpoints.

In my new TWIN version, the intermediate functions are divided, like the independent variables, into three groups: those relating to ∗1, those relating to both stars, and those relating to ∗2. I allow for 100 functions in all, the first 42 for ∗1, the next 16 for both, and the last 42 for ∗2.

It should be clear that SOLVER has to handle the independent variables in not just any old order but in a fairly definite order. This is because the first KBC equations, *i.e.* the central boundary conditions (6d) – (19d), do not necessarily depend on all the independent variables, but only (at least in principle) on a subset of them. If the first submatrix to be inverted, the KBC by KBC matrix of derivatives of the boundary conditions with respect to some of the variables, is singular we would be in trouble. In practice, the equations (6d) – (19d) of Section 1.4 'mainly' determine the independent variables m, L, r, I and ξ. But these are not the first three variables as listed in Section 1.3, because I didn't realise this when I first set up the H(J,K) array. It is necessary therefore to give SOLVER a permutation of the order of the independent variables. Even if one had noted this point to start with, such a permutation is still convenient, because for example one might add some further or alternative variables and/or equations later on, and it would be inconvenient to have the variables H in one order in some models, and in another order in others.

It is similarly necessary to be able to notionally permute the equations as well as the variables. For example, you might want to try the effect of different difference approximations for the same differential equation: one version might be the regular one and the other might be defined as some hitherto unused number. You can then summon whichever you want by giving different permutations as input. Finally, it is similarly convenient to be able to permute the boundary conditions, independently of the difference equations. For the present, SOLVER cannot handle more than 40 difference equations in 40 independent variables, because of dimension statements. But computers are so much bigger than they were 30 years ago that it shouldn't now be difficult or expensive in cpu time to do more.

In a separate file (writeup.fig) of this subdirectory there is an illustration of the way in which the matrix equations of the Newton-Raphson iteration are manipulated. I hope this will assist the serious reader in deciding how to introduce new equations into the system (or eliminate some that might be thought unnecessary). The file is a *text* file, not TeX or LaTeX.

SOLVER requires a bunch of input numbers, which are listed below. They are either in its calling sequence (ITER, KD(130), KT5 and JO), the blank COMMON block (H, DH, EPS, DEL, DH0, KH), or the COMMON block /QUERY/ (KOC). KD is a vector of integers, H and DH are arrays of reals. DH is both input (an initial guess, sometimes just zeros) and output (hopefully the converged solution).

Input for SOLVER:

H - the array of independent variables for the previous model in an evolutionary sequence; or a guess at an initial model if starting a new sequence

DH - as input, a guess at the increment of H in the current timestep; as output, should be the converged solution for the increment. The usual input is just the output of the previous timestep, but in some circumstances it is all zeros.

KOC - either 1 or 2. With 1, SOLVER goes to FUNCS1 and EQUNS1, and differentiates their output numerically. With 2, SOLVER goes to FUNCS2 and EQUNS2, and expects the differentiation to have been done analytically within these subroutines.

ITER - the maximum number of iterations allowed in one timestep (can be different, *e.g.* larger, on the first timestep). If SOLVER hasn't reduced the error of an iteration sufficiently, the program exits shortly. A procedure outside the solution package can then restart from the *second*-last converged model, with a shorter timestep.

EPS - the accuracy sought in a converged model. If ERR, which is the mean modulus change in DH in the latest iteration, is less than the datum EPS, the model is taken to have converged. EPS $= 10^{-6}$ is usually OK.

DEL - if ERR is moderately small it is reasonable to apply directly to DH the corrections to it which SOLVER has computed in the course of an iteration. But I have often found that if ERR is on the large side it is safer to apply only a fraction of the computed corrections. Convergence will be slower, of course, but is often more reliable. DEL is the maximum value of ERR for which the whole correction is applied. For ERR $>$ DEL, the corrections are reduced by the factor DEL/ERR. Usually DEL $= 0.01$.

DH0 - during the numerical differentiation of the intermediate functions which FUNCS1 computes, the independent variables each have to be varied in turn by a small amount: this small increment is dictated by DH0. Now that the whole thing is in double precision DH0 doesn't matter so much. Usually DH0 $= 10^{-7}$.

KH - the number of meshpoints in the model. Usually KH = 199.

KT5 - SOLVER writes one line of output for each iteration, indicating the maximum modulus correction to each variable in turn, and where in the mesh this occurred. But to save output this line is omitted for the first KT5 iterations (= 2 usually), since the code takes more than two iterations only if it is not working very well, i.e. (usually) if the evolution is accelerating but the timestep is not decreasing fast enough.

KD(130) - One lot of 10, and 3 lots of 40, integers. The first 10 are KE1, KE2, KE3, KBC, KEV, KFN and IL, as defined in the first two paragraphs of this section, and JH1 – JH3. These last 3 are only used for debugging; a suitable choice of them gives lots of output from PRINTC, which may allow one to check that FUNCS and EQUNS are setting up correctly the difference equations intended. Default values 0, 0, 0 suppress this output. The first set of 40 integers is the permutation of the independent variables described above, the second is the permutation of the equations, and the third is the permutation of the boundary conditions. The normal values (for the particular set of 19 equations I tend to use currently) are

(8, 5, 0, 5, 6, 50, 1, 3*0),
(7, 8, 4, 12, 19, 5, 3, 9, 10, 11, 1, 2, 14, 6, 13, 15, 18, 17, 20, 21*0),
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 19, 27*0) and
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 17, 18, 20, 6, 7, 8, 9, 19, 21*0).

The above set of numbers includes the extra equations that I have recently added for I (moment of inertia), $\phi, P_{\rm rot}, \phi_s, H_{\rm orb}, e$ and the binary mass $M_{\rm B}$. These extra equations model the evolution (for $*1$ only, not $*2$) of the rotation and orbit as influenced by a model of magnetic braking combined with tidal friction, as well as a model for the mass-loss rate in RLOF based on an integrated mass flux through the levels above the critical surface.

All of these numbers have to be read in BEGINN. Usually H as input will be the output of a previous run – one had to start somewhere, but nowadays I can usually find some previous model that can be used as, or pseudo-evolved into, the starting point for some new run.

Note that BEGINN reads in a vector KD(260), which is two lots of KD(130): one for FUNCS1/EQUNS1 and one for FUNCS2/EQUNS2. With the latter all zero, the 'minor composition' option is not currently used.

In the file called writeup.fig, referred to above, I illustrate the process of Gaussian elimination and back-substitution that is involved. This file also comments on how one chooses the permutations in order to make sure that the matrices that are to be inverted are in fact invertible. This is important if one wants to add some new equations. This is as much an art as a science. One needs to have a feeling for which of the many variables that might appear in a new boundary condition (for instance) is the *principal* one that the BC determines.

1.7 THE STELLAR STRUCTURE PACKAGE

FUNCS1 is mainly where the physics of stellar evolution goes, except that FUNCS1 calls STATEF which gives the equation of state (EoS). Actually, FUNCS1 calls STATEL, which sometimes calls STATEF. This is so that if you are in the process of differentiating numerically wrt r, say, you don't have to reevaluate the EoS, whereas for $\log T$, say, you do.

FUNCS1 computes the mixing-length approximation to the temperature gradient. It solves the cubic equation analytically, rather than approximately. FUNCS1 also sets up the minimal 'reaction network', which only involves hydrogen burning via the pp chain and CNO bi-cycle (with ^{3}He, ^{13}C and ^{15}N assumed in equilibrium, though not ^{14}N and ^{16}O), helium burning mainly by 3α and ^{12}C$(\alpha,\gamma)^{16}$O, and carbon, oxygen and neon burning just by ^{12}C+^{12}C, ^{16}O+^{16}O, ^{20}Ne$(\alpha,\gamma)^{24}$Mg and a few related (α, γ) and (γ, α) reactions. This limited network results ultimately in a mixture of ^{24}Mg and ^{28}Si (except it doesn't often get there in practice ....).

Another of FUNCS1's contributions is to evaluate the terms in the mesh-spacing equation, *i.e.* in Q and Q' (Section 1.4). Q is actually a rather more sophisticated function than Section 1.4 implies, but not vastly more. At the end of FUNCS1 the surface boundary conditions are evaluated.

FUNCS1 knows whether it is dealing with $*1$ or $*2$ of a binary by looking at JB, which is either 1 or 2. For JB = 1 it computes the evolution of rotation, eccentricity and orbital period according to naive prescriptions for the ML rate of stellar wind, the AML rate because of magnetic braking in this wind, and the exchange of angular momentum between the spin and the orbit because of tidal friction. The same processes in $*2$ are ignored. When JB = 2, FUNCS1 uses the already known mass-loss and mass-transfer history of $*1$ to provide a boundary condition for $*2$.

It would not be possible to determine the entire evolution of $*1$ before starting on that of $*2$ unless we make the assumption that the only aspect of $*2$ that influences $*1$ is $*2$'s mass. It is not necessary that $*1$ evolve conservatively, since we can keep track separately of mass lost to infinity by wind and mass lost to $*2$ by transfer. But we cannot allow $*2$ to have its own wind-driven mass loss, since that will depend, at the very least, on the luminosity, radius and rotation period of $*2$ as well as its mass.

The above has to be qualified if we are in the TWIN mode, solving both stars simultaneously. Then FUNCS1, at each meshpoint, does (a) things relating to $*1$ only, (b) things relating to $*2$ only, and (c) things that relate to both stars (e.g. heat transfer that depends on the difference in enthalpies). In the TWIN mode it is an index JS which says which star we are in. The fact that in the non-TWIN mode JS = 1 and JB indicates which star we are in, while in the TWIN mode JB = 1 and JS says which star we are in, is a tedious complexity that I will have to eliminate some time. The procedure is rather complex at the moment: I haven't had time to write something simple.

1.8 THE EQUATION-OF-STATE PACKAGE

The EoS has an unusual form in that it gives such quantities as $P, \rho, C_{\rm P}, \ldots$ as functions of the two independent variables f, a parameter related to electron degeneracy, and T, the temperature. In addition, of course, the EoS depends on the abundances of the various elements, which are given constants for present purposes. The choice of f rather than say ρ or P as independent variable is based on the fact that several physical processes, notably electron degeneracy in cores and ionisation in envelopes, are *explicit* functions of f, but not of ρ or P. So also would be such extra processes as pair production and inverse β-decay, although I have not actually programmed those. By using explicit formulae the computation is rendered very efficient; there is no need to invert a complicated highly non-linear relation $\rho = \rho(f, T)$ in order to determine the electron degeneracy parameter f which is needed in the Fermi-Dirac integrals.

Electron degeneracy is normally represented as a quantity ψ, which appears in such Fermi-Dirac integrals as

$$N_{\rm e}\rho \;=\; {\rm const.} \int_0^\infty \frac{p^2 dp}{e^{Emc^2/kT-\psi}+1} \quad .$$

The quantities p, E are the dimensionless momentum and energy of an electron, related by $E = \sqrt{1+p^2} - 1$; N_e is the number of free electrons per a.m.u., which itself depends on ψ via ionisation (see below). The quantity f gives ψ explicitly, by definition, as

$$\psi \ = \ \ln \frac{\sqrt{1+f}-1}{\sqrt{1+f}+1} + 2\sqrt{1+f} \quad , \quad \text{so that} \quad \frac{d\psi}{df} \ = \ \frac{\sqrt{1+f}}{f} \quad .$$

In terms of a further function $g(f,T)$ defined by $g = (kT/mc^2)\sqrt{1+f}$, the Fermi-Dirac integral above can be approximated, to about 3 s.f. for *all* physical f, T, by

$$N_\mathrm{e}\rho \ = \ \text{const.} \frac{f}{1+f} \{g(1+g)\}^{3/2} \frac{\sum_0^3 \sum_0^3 a_{ij} f^i g^j}{(1+f)^3(1+g)^3} \quad ,$$

where the a_{ij} are a set of 16 constant coefficients found by least-squares. It is easy to see that in four limiting circumstances we have:

$$f \ll 1 \ , \ g \ll 1 : \qquad \psi \ \sim \ \ln\frac{f}{4} + 2 \ , \ g \ \sim \ T \ , \ N_\mathrm{e}\rho \ = \ \text{const.}\, 4a_{00} T^{3/2} e^{\psi - 2}$$

$$f \ll 1 \ , \ g \gg 1 : \qquad \psi \ \sim \ \ln\frac{f}{4} + 2 \ , \ g \sim \ T \ , \ N_\mathrm{e}\rho \ = \ \text{const.}\, 4a_{03} T^{3} e^{\psi - 2}$$

$$f \gg 1 \ , \ g \ll 1 : \qquad \psi \ \sim \ 2\sqrt{f} \ , \ g \sim \ \frac{\psi T}{2} \ , \ N_\mathrm{e}\rho \ = \ \text{const.}\, a_{30} (\psi T)^{3/2}$$

$$f \gg 1 \ , \ g \gg 1 : \qquad \psi \ \sim \ 2\sqrt{f} \ , \ g \sim \ \frac{\psi T}{2} \ , \ N_\mathrm{e}\rho \ = \ \text{const.}\, a_{33} (\psi T)^{3} \quad .$$

Similar approximations exist for the pressure and the internal energy U, or equivalently the entropy S, of the free electrons. The coefficients are to be found in the subroutine fdirac.f.

A second virtue of the above approximation is that, because it is closely based on the analytic expansions of the integral in its various limiting regimes, its partial derivatives, even up to third derivatives, are reasonably accurate, and can also be written down analytically. Since much of the physics we need involves derivatives of P, S w.r.t. T, ρ, it is important that the derivatives are also accurate.

Ionisation is expressed by a number of equations of the form

$$\frac{N_{\mathrm{H}^+}}{N_\mathrm{H}} \ = \ \frac{\omega_{\mathrm{H}^+}}{\omega_\mathrm{H}} e^{-\psi - \chi_\mathrm{H}/kT} \quad ,$$

where the ω's are statistical weights and χ_H is the ionisation potential; this has to be solved along with $N_{\mathrm{H}^+} + N_\mathrm{H} \ = \ X$, where X is the given abundance of hydrogen by mass. Obviously these two equations give both N_H and N_{H^+} simply and explicitly in terms of ψ, T or equivalently of f, T. Similar equations (3 rather than 2) give the helium ionisation equilibrium. A slight complication is that for hydrogen we also have to consider the *molecular* equilibrium, but it turns out that this only means solving a quadratic rather than a linear equation for N_H.

Two substantial problems remain, one of which is an artefact of the choice f of independent variable, and one of which is a problem however we choose the independent variables. In order, (a) f or ψ becomes indeterminate if the gas is fully non-ionised, so that there are no free electrons, and (b) the ionisation equation above breaks down at high density ('pressure' ionisation), because the atomic structure of ions is strongly modified when the ions are so closely packed together that their Bohr radii are less than their separation. I don't have an answer to (a), but mercifully most *stellar* material is hot enough, dense enough, or dilute enough, that at least some atoms are ionised. I help this along by approximating both Si and Fe as *wholly* ionised, even although they are not.

We deal with (b) approximately, by assuming that both the ω's and the χ's are dependent on $N_e\rho$ and T, and *only* on these. Thus we add to the Helmholtz free enery a term of the form $\Delta F \equiv N_e F_0(N_e\rho, T)$. Then in the ionisation equation we have to add to ψ the additional chemical potential term $(kT)^{-1}(\partial\Delta F/\partial N_e)_{\rho,T}$. This works because $(\partial\Delta F/\partial N_e)_{\rho,T}$, unlike ΔF, is a function of N_e and ρ only through the combination $N_e\rho$, and this in turn is a function only of the input variables f, T. We must also for thermodynamic consistency add terms $\rho^2(\partial\Delta F/\partial\rho)_{N_e,T}$ to the pressure and $-(\partial\Delta F/\partial T)_{N_e,\rho}$ to the entropy.

1.9 THE TIME-STEP PACKAGE

The selection of the next timestep (in NEXTDT) is more of an art than a science. This is at least partly because *all* stellar models in assumed hydrostatic equilibrium break down ultimately, approaching either degenerate He ignition, degenerate C ignition, or a core-collapse supernova. But other less obvious pitfalls are commonly encountered before then. They may require the timestep to be cut down drastically, but temporarily, and it is often difficult to spot this in advance. Therefore the code will sometimes fail to converge; but it often recovers from this by retreating two timesteps and trying again with (say) half the timestep.

Apart from a predictable breakdown at a late stage in evolution, one sometimes gets temporary breakdowns at earlier stages. Mostly these are associated with one or other (or in tiresome cases more than one) of the following:

(i) Evolution accelerates rapidly at the end of core-burning phases; sometimes DT isn't cut back rapidly enough to cope.

(ii) When a convective core (or envelope) has a boundary moving into a formerly non-convective region, possibly with a composition gradient, a composition discontinuity is liable to build up. This is physically OK, but may give the code a problem. For the way a composition discontinuity propagates with an implicit diffusive treatment of convective mixing is by having X (the abundance) change rapidly at one meshpoint while hardly changing at all at both adjacent meshpoints. Then when the intermediate X has almost reached uniformity with the convective region, the same thing starts happening one point further out (or in). It is therefore difficult to extrapolate reliably to the next model from the present and previous models, specifically when the rapid change in X at one meshpoint is beginning to die out while that at the next meshpoint is beginning to pick up. One might expect that a short enough timestep would do it, and that is generally right; but the step has to shorten very rapidly at a rather unpredictable point, and preferably also increase again rather rapidly once the difficulty is (temporarily) over.

(iii) On a line roughly coinciding with the blue edge of the Cepheid strip, the depth of the convective envelope is very sensitive to effective temperature. The convective part of the envelope may seek to expand or contract by 10 or more meshpoints in the course of a single timestep. The difference equations are especially non-linear in this circumstance, and it is somewhat amazing that the Newton-Raphson technique (which assumes local near-linearity) works at all. In practice it usually does, but sometimes it needs a temporarily shortened timestep.

(iv) The onset of RLOF is similarly a very non-linear process, on top of which the rate of evolution commonly speeds up by a factor of ~ 100.

The code chooses the next DT on the basis of

(a) certain input numbers, viz. CDC(1) – (5), CT1, CT2 and CT3

(b) the DT you give it at the start of a run - normally the DT from the end of a previous run, but you can vary it

(c) internally generated numbers SUM, CDD and IHOLD: SUM $\equiv \Sigma\Sigma|DH(I,K)|$/KH, the mean-modulus change of the model in the timestep. The sum is over all meshpoints K, but only over a selection of the variables I, the selection being on the basis of experience. CDD is the target value of SUM – one wants to choose timesteps such that SUM $\sim$ CDD. CDD is made to vary somewhat as a function of evolutionary state. During core He burning, it is desirable to have CDD smaller than during core H burning, and during double-shell-burning it is desirable to have it larger. For the present, as a result of trial-and-error, I set

CDD = CDC(1) = 0.01 for evolution from the ZAMS to He ignition

CDD = CDC(1)*CDC(2) = 0.0025 for evolution during core He burning

CDD = CDC(1)*CDC(3) = 0.01 for further evolution until the He shell nearly catches up with the H shell

CDD = CDC(1)*CDC(4) = 0.04 for double-shell-burning.

For IHOLD, see below.

(d) whether the model has converged or not on its last timestep.

Assuming convergence, for the moment, the code tries to choose DT so that FAC≡SUM/CDD is the same (unity) from one timestep to the next. So if the timestep DT just finished gives a certain FAC, the next timestep should be DT′ = DT/FAC. But to prevent DT from fluctuating too rapidly (which often seems to cause numerical problems), DT′/DT is restricted to lie in the range (CT1, CT2), typically (0.9, 1.05), when things are running reasonably smoothly.

When things are not running smoothly, the first symptom is that the number of iterations needed to reach the required accuracy rises, from a typical 2 to a permitted maximum of say 5 (*i.e.* KR2). Usually the timestep starts cutting back, but often not fast enough. If even 5 iterations fail to give convergence, the code restarts from the anteprevious model, with DT decreased by a given factor CT3. Sometimes this is enough, and after 5 timesteps (determined automatically by IHOLD), DT starts selecting itself again. Sometimes the code staggers on for a few timesteps, then fails again to converge and restarts again, as above, with a further decrease of DT. On occasion 5 such restarts occur before the code picks up its usual rhythm. The worst case is that the code continues decreasing DT by further factors of CT3 without ever reaching convergence, but DT is only allowed to decrease to 10^6 secs before the code terminates.

The following stratagem, to be done by hand rather than automatically, can sometimes push the code through some convergence failure (except of course for the terminal difficulties in the first paragraph of this Section). Retreat to the last stored model of the last run that went well, and evolve only so many timesteps that the code terminates normally a few steps before the problem. Then halve (or quarter) DT manually, and prevent it from increasing by putting CT1 = CT2 = 1.0 (or maybe 0.95, 1.05). If it appears to be settling down after the difficulty, remember to relax CT1, CT2 a bit, or the calculation may take forever.

For ∗2 of a binary, the problem is compounded. The same DT as ∗1 will seldom do, since at an early stage ∗2 may be evolving more slowly than ∗1 and at a later stage faster. Also, one or other star may need to accelerate strongly, while the other does not. NEXTDT knows ∗1's timestep history while trying to decide on ∗2's next timestep. Mainly it operates in the same way as for ∗1 (above), while looking for the timesteps in ∗1 that include the beginning and the end of ∗2's timestep; but if this would encompass a large number of timesteps of ∗1 the timestep for ∗2 is cut back a bit.

The process for ∗2 works least well early in RLOF, where the mass-transfer rate grows rapidly from zero to the thermal timescale. The code at present never quite reduces the timestep of ∗2 to what I would like: typically there are half-a-dozen timesteps that are about twice the size that would be desirable. Somewhat surprisingly the code seems to converge without difficulty, but ∗2 changes more in one timestep than one would wish on the grounds of accuracy. This seems to be because (a) the thermal timescale of ∗2 at this point is usually quite a lot shorter than ∗1's, and (b) the rate of mass gain of ∗2 is obtained from linear interpolation in ∗1's history, and so is in effect *constant* for a while and then increases as a step-function. Perhaps I need non-linear interpolation, but since the variation of $\dot{M}$ in ∗1 is itself a very rapid acceleration, non-linear interpolation would seem even more problematic than linear.

A further complication, though apparently not a major one, is that in the interest of economy the code flip-flops between ∗1 and ∗2, doing say 200 steps of the first and then enough steps of the second to catch up. Without this, ∗1 might be evolved to a pre-supernova, only to discover that ∗2 came into contact with ∗1 while ∗1 was still in the main-sequence band.

1.10 THE INPUT AND OUTPUT DATA

The following data numbers all have to be assigned. Some are never, or very seldom, changed; some are changed in order to specify what kind of evolutionary run you want; and some have to be changed, or are changed automatically, in response to way the run is progressing. It is of course something of an art form to come up with changes that allow you to cope with some unpredicted breakdown. There are a few numbers whose choice seems fairly delicate for getting the thing to run smoothly. They mostly have to do with choosing the best timestep, and the way that timestep should increase or decrease in response to the evolution. Some rules of thumb were given in the previous Section. The values which I usually assign are given below in brackets (except for long vectors).

I find it simplest to use the default names fort.n (on some computers ftn0n) for the file that is read or written to unit n. Files 1 – 3, 8, 9, 13 – 15 are used as output, and files 11, 12 and 16 – 21 are used as (relatively) fixed input; some files (13 – 15) can be used as output in one run and input in the next run. File 22, a 'configuration file', is changed occasionally, and only file 23 (another configuration file) is normally changed from job to job, defining what particular masses and period are required. Starting with files 12 – 21, in reverse order, and finishing with files 22 and 23:

....

Fort.21 contains data for converting from $L, T_{\rm e}$ to $M_{\rm V}$, $B-V$, $U-B$, and some nuclear data (Q-values for nuclear reactions and neutrinos, numbers for computing screening corrections) and atomic data (ionisation potentials, statistical weights, atomic masses and numbers).

....

Fort.20 contains the OPAL opacity data, combined with molecular opacities, and it also contains nuclear reaction rate data and neutrino loss rates:

KCSX : The number of different sub-compositions tabulated, ≤ 10, currently 8 or 9

CZS : the metallicity

CH : the initial hydrogen mass fraction, which I usually take to correlate with CZS, CH = 0.76 – 3*CZS

CSX(10) : Numbers which describe indirectly the 8, 9 or 10 sub-compositions

CS(90,127,10) : The opacity tables

CHAT(8920) : Neutrino loss rate and nuclear reaction rate tables.

....

Fort.19 contains just 4 numbers, MLO, DM, MHI, KDM, defining the masses of the models stored in fort.16 and fort.17 below. The present numbers say that the ZAMS models provided in fort.18 have log masses ranging from MLO = – 1.00 to MHI = 2.30 inclusive, in steps of DM$\equiv \Delta \log_{10} M$ = 0.025; and that the corresponding summary in fort.17 is KDM = 5 times more finely spaced.

....

Fort.18 contains a short summary of every fourth model in a ZAMS run (executed previously) where the mass increases from 0.1 to 200 $M_\odot$ in equal increments of $\log_{10} M$. For every twentieth model there are also interior details like $\log \rho, \log T, \log P, \ldots$. The file was output (on fort.1) of a previous job.

....

Fort.17 contains a 'pruned' version of fort.18, with just 5 lines for every fourth model. The pruning is done automatically at the start of each job; so fort.17 is not actually needed as data.

....

Fort.16 contains the interior details of every twentieth ZAMS model on fort.18, in the form needed to start a new run (which is different from the form to be seen in fort.18). These are output from the same previous run, pseudo-evolving a star that gained mass yet didn't evolve (much), i.e. a ZAMS run. More on this later. But the output file of one run is suitable to be the input file of another run. The 133 ZAMS models on fort.16 have to correspond of course to the metallicity of the opacity table of fort.20. There is no marker between one model and the next, but the first line of each model says how many following lines belong to it. In fact they all have 199 grid-points, and so 200 lines when we include the first line. The first line, which one sometimes edits by hand – but which one can also superseded by putting some replacent numbers in fort.23 – contains the following:

SM : The stellar mass, in solar units. All of SM – ENC below can be re-initialised by data in fort.23.

DTY : The next time-step, in years.

AGE : The age, in years.

PER : The orbital period, in days. For a single star, take say 1.0d10

BMS : The total mass of the binary. The star you are evolving is always in a binary, but you can make the binary so wide (using PER above) that it is effectively single. BMS may change with time if stellar-wind mass loss from $*1$ is included.

ECC : The orbital eccentricity; will go to zero if tidal friction is included, and if the binary is close enough.

P1 : The *rotational* period of the star, which approaches pseudosynchronism with the orbital period if tidal friction is included.

ENC : A constant (in space, but not in time) energy generation rate that can be added to the usual nuclear term, for constructing Hayashi-track pre-MS stars from ZAMS stars. See CEA, CET below (fort.22). ENC is the starting value for the run, and it is made to increase, on a timescale 1/CET (yrs) towards an asymptotic value CEA (ergs/gm/sec). (0.0)

KH : The number of mesh-points in the input model to be read in shortly.

KP : The number of time-steps you want to take.

JMOD : The number of time-steps it has already taken. If starting from the ZAMS, you can reset JMOD = 0, and in the first step nuclear and thermal evolution are ignored, which can be useful.

JB : For JB = 1, the star is treated as the primary of a close binary, and will lose mass when it fills its Roche. For JB = 2, it is the secondary. The code runs the primary first, and then puts its output through a subsidiary (very simple) subroutine called pruner.f, which produces a file containing mass as a funtion of time. With JB = 2, the secondary uses this file to gain mass at the appropriate rate as a function of time. What happens when the secondary also fills its Roche lobe? If you find out, let me know. In the latest code, the switch from $*1$ to $*2$ is handled automatically, and can happen every (say) 200 timesteps of $*1$. Thus if $*2$ fills its Roche lobe fairly early on, we haven't wasted time evolving $*1$ to a supernova.

JIN : The number of independent variables of the H, DH arrays to be read in. This may be increased early in the run by adding in some estimates for extra variables satisfying extra equations. I do this in REMESH, which is called once only, almost at the beginning of a run. (11, 16, 24)

JF : This is a tedious number introduced so that (a, JF = 0), if one is reading a model, say from the ZAMS library, that does *not* include the two variables $I(m)$ and $\phi(m)$, then REMESH will put in an initial guess for them, but (b, JF = 2), if one is reading a model already computed by one of the code versions that solve for $I(m), \phi(m)$, then the stored solution will not have this $I(m), \phi(m)$ overwritten by REMESH. There must be a better way of doing this in a general way, but I have not come up with it.

H(40,500) : The details of an initial model, normally an output model of a previous run. You can use as many as 499 meshpoints (not 500, for some reason), but mostly I have stuck with 199.

Although the models in fort.16 have specific values of SM, ... JMOD, these can be superseded by values you select in fort.23 below.

....

Fort.13 – fort.15 optionally contain similar models which can be read in as data (according to what one puts in fort.23), and which were output from previous runs. As output, fort.15 contains every NSV'th model (or pair of models, for a binary) computed in a run. Fort.13 and fort.14

contain the last and the second last model that were stored. The code alternates between 13 and 14, overwriting anything previous, so it is not necessarily clear which is the last and which the second last without looking at their first lines.

....

Fort.12 contains a 'canonical' ZAHB model which is used if a low-mass star reaches the He flash: the canonical model is pseudo-evolved to reach both the right surface mass and the right core mass (approximately). Fort.11 contains one integer, a zero. If while the code is running, you want to terminate it cleanly, rather than just kill it, put a non-zero integer in here. Have you heard of 'interactive computing'? This is the nearest I get to it.

....

Fort.22 and fort.23 are 'configuration files', determining the kind of job to be done. Fort.22 is occasionally changed, and fort.23 usually changed, from one run to another. The following are read from fort.22, in subroutine BEGINN. There are 29 rows, but doubled (for binaries), and with a blank line in between, since the first lot applies to ∗1 of a binary and the second, which may be different, to ∗2.

First row:

KH2 : The number of mesh-points you want; if this differs from KH (above) the code should interpolate in the given model to produce a new one; but you must also set JCH (below) to ≥ 2 to implement this change (199)

KR1 : The maximum number of iterations allowed on the first few timesteps (12)

KR2 : The maximum number of iterations allowed on later timesteps (5)

JCH : If JCH = 2, 3 or 4, the code (in REMESH) initialises the model in various ways, *e.g.* by using a new mesh-point distribution function, so that the new model is to be got by interpolation in an old model. For a ZAMS model, you can initialise to a given uniform $X(^1\mathrm{H})$, ... $X(^{16}\mathrm{H})$ by taking JCH = 4. You can also change the mass, preferably by just a small amount, with JCH > 2. (1, 2, 3 or 4)

KTH(1) – (4), *alias* KTH – KZ:

KTH : ϵ_{th} = KTH ∗ (T DS/Dt); so you can ignore T DS/Dt if you want (1 or 0)

KX : D$X(^1\mathrm{H})$/Dt = KX ∗ (burning rate of ^{1}H); so you can ignore the composition change while keeping the energy production (1 or 0)

KY : The same, for ^{4}He (1 or 0)

KZ : The same, for ^{12}C and ^{16}O (1 or 0)

Second row:

KCL(1) – (7), *alias* KCL, KION, KAM, KOP, KCC, KNUC, KCN:

KCL : Unity includes the Coulomb correction to pressure etc; zero suppresses it. (1)

KION : EoS does the ionisation of the first KION elements in the list H, He, C, N, O, Ne, Mg, Si, Fe. No other elements are included. KION = 5 is about optimal. Don't try 9. (5)

KAM : Not currently used

KOP : If unity, code should use spline interpolation in tables of opacity; if zero, simple bilinear interpolation (1)

KCC : Not currently used

KNUC : Not currently used

KCN : If 0, gives standard nuclear network. If 1, gives a CNO equilibrium fudge for ZAMS models: see FUNCS1 (0)

Third row:

KT(1) – (4), *alias* KT1, KT2, KT3, KT4:

KT1 : Print internal details of every KT1'th model (20 or n200)

KT2 : Print internal details at every KT2'th meshpoint of the KT1'th model (1 or 2)

KT3 : Print KT3 'pages' of details for every KT1'th model (1, 2 or 3)

KT4 : Print a five-line summary of every KT4'th model (1, 2 or 4)

KT5 : Print a one-line summary of each iteration of each model, except for the first KT5 iterations of each model (0 or 2)

KSV: an output model is stored in fort.15 after every KSV'th timestep in a run, in the form needed as input for a further run. The last model of a run is automatically also stored, in fort.13 or fort.14. (5000)

Fourth row:

EP(1) – (3), *alias* EPS, DEL, DH0:

EPS : The accuracy to which SOLVER is expected to solve the equations (10^{-6})

DEL : See section 1.5 (10^{-2})

DH0 : See section 1.5 (10^{-7})

CDC(1) – (5): Coefficients determining the mean increment, modulus-wise, that you would like in one timestep (.01, .25, 1.0, 4.0, 1.0)

Next 14 rows (5 – 18):

KD(260) : See section 1.5

Next 3 rows (19 – 21):

KSX(45) : The first 15 integers identify the quantities, such as $\log\rho, L, X(^4\mathrm{He})$ ... , which are to be printed in columns on the first 'page' of structure details for every KT1'th model (see above). The next two lots of 15 relate to the optional further 'pages'. The variables are: 1 - ψ, 2 - P, 3 - ρ, 4 - T, 5 - κ, 6 - ∇_a, 7 - ∇, 8 - $\nabla_r - \nabla_a$, 9 - m, 10 - H1, 11 - He4, 12 - C12, 13 - N14, 14 - O16, 15 - Ne20, 16 - Mg24, 17 - r, 18 - L, 19 - ϵ_{th}, 20 - ϵ_{nuc}, 21 - ϵ_ν, 22 - δm, 23 - not used, 24 - $n/n+1 = d\log\rho/d\log p$, 25 - $d\log r/d\log P$, 26 - $d\log m/d\log P$, 27 - U (int. en.) , 28 - S, 29 L/L_{Edd}, 30 - $w_{\mathrm{conv}}l$, 31 - μ, 32 - wt, 33 - ν_{e}, 34 - ν_{e0}, 35 - w_{conv}, 36 - I, 37 - ϕ, 38 - Fm, 39, 40, 41 - not used, 42 - ΔL, 43 - Δ(enth.), 44 - v^2, 45 - fac.

Next 3 rows (22 – 24):

KN : The number of variables that will be used for determing the next t-step

KJN(1) – KJN(40) : the first KN of these identify the variables to be used

Row 25:

CT1 : The next timestep cannot (normally) be less than CT1 times present timestep (0.8, 0.9 or 1.0)

CT2 : The next timestep cannot be greater than CT2 times present timestep. If both CT1 and CT2 are 1.0, then the timestep is constant, of course (which is useful for constructing a ZAMS by artificial 'mass-gain') – except that if a model fails to converge the timestep will be multiplied by CT3 (next) (1.1, 1.05 or 1.0)

CT3 : when the solution package fails to converge, the code retreats to the second-last converged model, and continues with the timestep decreased by the factor CT3. (0.3 or 0.5)

CT(1) – (10): coefficients used in the mesh-spacing function Q

Row 26:

CC, CN, CO, CNE, CMG, CSI, CFE : values for initialising $X(^{12}\mathrm{C})$, ... $X(^{56}\mathrm{Fe})$, as fractions of the total metallicity Z (= CZS, above); only used for ZAMS models, with JCH (above) = 4 (0.176, .052, 0.502, 0.092, 0.034, 0.072, 0.072)

Row 27:

CALP : The mixing-length ratio (2.0)

CU : Along with COS and CPS (next), a 'convective overshoot' parameter; see CRD below (0.1)

COS : A convective overshoot parameter for H-burning cores; see CRD below. Zero implies no overshoot. (0.12)

CPS : as COS, but for He-burning cores. (0.12)

CRD : The diffusion coefficient σ for convective mixing is taken to be CRD times the 'legitimate' rate from mixing-length theory; except that an approximate multiple of $[\nabla_r - \nabla_a]^{1/3}$ is replaced by the same multiple of $[\nabla_r - \nabla_a + \nabla_{\mathrm{OS}}]^2$, where

$$\nabla_{\mathrm{OS}} \equiv \frac{\mathrm{COS}}{(2.5 + 20\beta_* + 16\beta_*^2)\,(\mathrm{CU}\,.\,\partial \log m/\partial \log P + 1)} \quad , \quad \beta_* \equiv \frac{P_{\mathrm{rad}}}{P_{\mathrm{gas}}} \quad .$$

The usual CRD is 10^{-4}.

CXB : Defines the boundary of a core to be at $X(^1\mathrm{H})$ or $X(^4\mathrm{He})$ = CXB. Mainly for printout; but the H-boundary is also taken to be the base of the envelope when integrating the envelope binding energy. If you know a better definition, use it. (0.15)

CGR : Defines the boundary between a convection zone and a semiconvection zone, for printout purposes only, to be at $\nabla_r - \nabla_a + \nabla_{\mathrm{OS}} = \mathrm{CGR}$ (0.001)

CEA : A constant energy rate ENC (see fort.18 above) can be added to $\epsilon_{\mathrm{nuc}} + \epsilon_{\mathrm{th}} - \epsilon_\nu$. An increasing ENC can push a star back from the ZAMS to the Hayashi track. CEA and CET (next) determine how ENC changes with time (1.0E2)

CET : See CEA above. The equation for the growth of ENC with time is $d\mathrm{ENC}/dt$ = ENC.CET.(1 − ENC/CEA), so that ENC increases exponentially on the assigned timescale 1/CET (yr), until saturating at ENC ~ CEA. (1.0E-6)

Row 28:

CMS : At the surface,

$$\dot{M} = -\,\mathrm{CMS}{*}\,[\log(r/r_{\mathrm{lobe}})]^3 - \mathrm{CMT}{*}\xi + \mathrm{CMI}{*}m - \mathrm{CMR}{*}1.3\times10^{-5} * L * m/|E_{\mathrm{BE}}|$$
$$- \,\mathrm{CMJ}{*}\dot{M}_{\mathrm{JNH}} - \mathrm{CML}{*}\zeta(L, r, m, P_{\mathrm{rot}});$$

so you can include if you like one or other of two versions of mass transfer by RLOF (CMS, CMT), a constant mass-gain/loss rate, mainly for running up or down the ZAMS (CMI), a Reimers-like mass-loss rate (CMR), a mass-loss rate for luminous stars (de Jager et al 1988; CMJ), and a mass-loss rate as obtained from a simplistic dynamo theory (CML). Units are: CMT ($M_\odot$/yr), CMI (yr^{-1}), CMR, CMJ, CML (dimensionless). Remember what [. . .] means (p8). (0.0, or 1.0D4)

CMT : See CMS above. CMS, CMT are *alternatives*; set one to zero. (0.0, or 1.0D-2)

CMI : See CMS above (0.0, ± 5.0D-9 or ±1.0D-6)

CMR : See CMS above (0.0 or 0.2 or 1.0)

CMJ : See CMS above (0.0 or 1.0)

CML : See CMS above (0.0 or 1.0)

CHL : A factor multiplying the rate of ang. mom. loss associated with the rate of mass loss ζ, according to the same dynamo model. (0.0 or 1.0)
CTF : A factor multiplying an expression for the rate of tidal friction. (0.0 or 0.01)
CLT : A coefficient used in the estimation of heat flux between components in contact. It doesn't really work yet.
Row 29 (last row):
CPA : 'partial accretion': the fraction of one star's wind that is accreted by the other. What if both stars have winds? You tell me. (0.0)
CBR : 'bipolar re-emission': the fraction of material accreted by a star that is ejected in bipolar jets. Needed for CVs, LMXBs. (0.0)
CSU : 'spin-up', specifically of the gainer due to accretion. CSU is the specific angular momentum, relative to orbital, taken out of the orbit by material leaving the L1 point, acquiring ang. mom. due to Coriolis force, and landing on the other star – so orbital ang. mom. is converted to gainer's internal ang. mom.
CSD : 'spin-down'; the same process also spins down the loser, I suppose, though not by as much.
CDF : this is used to convert a step-function into a 'smoothed' step function: see below.
CGF : this multiplies the rate of angular-momentum loss due to GR, so you can ignore it if you wish.
There are 3 more numbers available following these, which I don't use at present.

You may recall that I defined $[x]$, alias the triangular function $T(x)$, as x when x is positive, and zero when x is negative. The derivative of this is the Heaviside step function $H(x)$. Since neither function is differentiable at $x = 0$, they pose a problem when one is doing the numerical differentiation in the solution package dsolve.f. Surprisingly, the code usually works anyway, but convergence seems better if I 'smooth' them. The smoothed step function is

$$\mathrm{SMF}(x, d) = \max\left[0, \frac{(x+d)^2}{(x+d)^2 + d^2}\right] ,$$

where d is a positive quantity of suitable dimensionality multiplied by CDF, typically 0.01, to make it small. My approximation to $T(x)$ is $x + d$ times this.

Single-star runs and TWIN binary runs need one lot of the above, and binary (non-TWIN) runs need two, the second following the first directly in the same fort.22.

....

Fort.23 is eight rows with various control numbers. You have to decide on each of four options, each giving two alternatives. The options are
(a) single stars or binary stars
(b) new, i.e. starting from scratch (ZAMS), or old, i.e. starting from the end of a previous run.
(c) independent evolution, or *simultaneous* evolution, of the components.
(d) a 'one-shot' or 'grid' run. A grid means several runs, one after the other (but simultaneous using the massively parallel version, not described here), with the three parameters of primary mass, mass ratio and orbital period being cycled through. One-shot means what it says.
Not all 16 possibilities make sense: e.g. if you are doing simultaneous evolution, you won't want single stars. Many but not all of the remaining possibilities should be viable.

The first row is integers ISB, KTW, IP1, IM1, IP2, IM2, KPT, KP:
ISB: single or binary. ISB = 1 implies only single stars to be computed; ISB = 2 gives binaries.
For *single* stars, you may still use the outer (first) cycle for masses. The inner 2 cycles are

automatically set to do only one case each. The mass ratio and the period are of course virtually ignored, but have to be supplied. The period should be so large that there is no danger of RLOF (*e.g.* XL = 7.0, meaning a period of $\sim 10^7$ d).

KTW: 1 for normal operation; 2 for TWIN mode, where both stars are solved *simultaneously.*

IP1: the number (13 − 16) of the file (fort.13 − fort.16) where the initial model for ∗1 is to be taken from. ZAMS models are on fort.16.

IM1: the sequential number of the model required on fort.IP1. This is computed automatically, from later data, if the ZAMS file fort.16 is used, so that if IP1 is 16, it doesn't matter what value you give for IM1, but you have to give a value.

IP2: as IP1, but for ∗2.

IM2: as IM1, but for ∗2.

KPT, KP: KPT is the maximum number of timesteps for each component (2000 to 4000 for fairly complete evolution). Do approximately KP of ∗1, then enough of ∗2 to catch up with ∗1, then another ∼ KP of ∗1, etc, so that if ∗2 breaks down before ∗1 you don't waste a lot of calculation on ∗1. You will seldom get *exactly* the number of timesteps that you ask for. For single stars, KP is set to KPT automatically.

The next 3 rows are parameters for 3 nested loops (mass, mass ratio, and initial period) to be run through. Each loop has: starting value; increment; number of cases (1 more than the number of increments).

The first loop is: $\log_{10}$ (mass, solar units), starting at ML1, increasing by steps of DML to ML1 + (KML − 1) . DML

The second loop is: $\log_{10}$ (mass ratio in sense $\{*1/*2\}$), starting at QL1, increasing by steps of DQL to QL1 + (KQL − 1) . DQL

The third loop is: X ≡ $\log_{10}$(orbital period /period necessary for ∗1 to fill its Roche lobe when still on the ZAMS), starting at XL1, increasing by steps of DXL to XL1 + (KXL − 1) . DXL.

Obviously you can arrange for only one model (single or binary) to be done, by taking KML = KQL = KXL = 1.

The next (fifth) row is ROT, KR and EX:

rotational period for each star = rotational breakup period ∗ 10**ROT, if KR = 1

rotational period for each star = max(1.05 ∗ rotational breakup period, orbital period ∗ 10**ROT), if KR = 2 or 3.

KR = 3 is an option for a binary where you wish to start at such an orbital period and eccentricity that, as a result of tidal friction, it will later have a target eccentricity and period. Forgetting about intrinsic spin, this means

$$P_0 = P\left[\frac{1-e^2}{1-e_0^2}\right]^{3/2} \quad .$$

EX is the initial eccentricity, or the target eccentricity if KR = 3.

The sixth row is eight numbers AX(8), and an integer JMX. The AX's are optional replacements for the values of SM, DTY, ... , ENC that the code would normally pick up in fort.IP1 from some previous run, or from the ZAMS library (fort.16 above). JMX similarly is an optional replacement for JMOD. They are only applied if they are non-negative. Thus you can replace only one, or several.

The object of the optional replacement is this. Suppose you want to model a particular binary, and believe starting values ($1.8 + 1.6\,M_\odot, 3\,\mathrm{d}$) will do. The nearest (log) masses in the library are 0.25 and 0.20, so set the M and Q loops accordingly (ML1 = 0.25, QL1 = 0.05, with KML = KQL = 1). But then in the sixth row put SM = 1.8, BM = 3.4 (*binary* mass). You won't know what value of X corresponds to 3 d, so also put PER = 3.0 in the sixth row. You can set the rotational period also (P1): the same for both components, I'm afraid. There are complexities here that I probably won't resolve till I retire; e.g. in some operational modes one doesn't follow the rotation of the *star*, only the orbit; or you might follow the rotation of ∗1 but not ∗2. Don't give a short ($\lesssim 10^6$ d) rotational period unless you are computing the rotational evolution, since the star will keep rotating at the same rate for all time and perhaps become rotationally unstable as a red supergiant.

The last three lines are a set of criteria to determine when the run is to be ended, e.g. when the age is greater than 2×10^{10} yr. You'll have to read the end of printb.f to figure them out. Not all of the available 21 numbers are used.

....

Fort.13 and fort.14 are used by the code for storage of intermediate or final models (in the same format as the input models of fort.16 above). This can be used for continuing a run with a further run. The machine alternates between 13 and 14, overwriting previous models. You therefore have access only to the last two stored models. It may not be clear till you look inside them which actually has the last model, although the one-line output on fort.8 contains the relevant IP, as well as the termination code (see below) and a few other numbers.

In addition to getting the last model of a run on fort.13/14, the parameter KSV in fort.22 says that the code stores every KSV'th timestep in fort.15. I usually don't bother, and take KSV = 5000, which is more than most runs take. There is little point in using KSV for non-TWIN binaries, since model 500 of ∗1 will not necessarily be contemporaneous with model 500 of ∗2; however fort.13/14 will give contemporaneous pairs.

The major useful output of a run comes out on fort.1, for ∗1 of a binary or for a single star, and on fort.2 as well if binary. This output has the same format as the input file fort.18. After ∗1 has finished, fort.1 is 'pruned' down and put in fort.3, where it serves as input for the next part of the evolution of ∗2: ∗2 has to know the mass-loss and angular-momentum-loss history of ∗1 in order to accrete mass at the right rate.

After a binary run is completed, the fort.2 output of ∗2 is also pruned down. Then fort.9 is created, which is first fort.3 (the pruned version of ∗1's run) followed by the pruned-down version of fort.2. Fort.9 is output which I often send to a plotting routine (a separate code, evplot.f) to look at.

The evolution code creates a tiny fort.8, a single line (per run) with integers identifying how the ∗1 and ∗2 runs ended (impending SNEX, lifetime > 20 Gyr, *etc*). These termination codes are listed in the next Section (and also at the beginning of fort.8). If several runs are done in a loop, they accumulate in fort.8 and fort.9 (and fort.15, if used), but the other output files are overwritten.

1.11 MODES OF OPERATION, AND BREAKDOWN

The usual mode is 'legitimate' evolution, say starting from the ZAMS and continuing till the code finally breaks down at degenerate C ignition, or a bit beyond non-degenerate C ignition. For binaries, there are some other ways to break down. A parameter JO indicates roughly why

a run stopped: for a binary there would be one value for each star. Some 'breakdowns' are not fatal, as listed below. The JO values are:
-1 – STAR12 – no timesteps required
0 – STAR12 – finished required timesteps OK
1 – SOLVER – failed; backup, shorten timestep; not fatal
2 – BACKUP – timestep reduced below set limit
3 – NEXTDT – $*2$ evolving beyond last $*1$ model; not necessarily fatal, go back to $*1$ and continue
4 – PRINTB – $*1$ radius exceeds lobe radius by set limit (e.g 5%)
5 – PRINTB – age greater than set limit
6 – PRINTB – C-burning luminosity exceeds set limit (*i.e.* close to SNEX)
7 – PRINTB – $*2$ radius exceeds lobe radius by set limit (0%, i.e. contact, or *reverse* RLOF)
8 – PRINTB – close to He flash; not fatal, automatically restarts on ZAHB
9 – PRINTB – massive ($> 1.2\,M_\odot$) degenerate C/O core
10 – PRINTB – $|\dot{M}_1|$ exceeds set limit
11 – NEXTDT – impermissible FDT for $*2$
13 – PRINTB – reached a ZAHB model equivalent to a post-He-flash model; not fatal
14 – PRINTB – sometimes the composition profile goes funny, for no good reason. This may get noticed in PRINTB, and the code should then back-track and restart; not fatal
15 – STAR12 – job terminated by hand.
12, 22, 32 – subsets of 2, but with non-zero central H, zero central H and non-zero central He, and zero central H *and* central He, respectively.

The set limits are in the vector UC, read in as part of fort.23.
JO = 32 is much the same as JO = 6, indicating that a supernova explosion is imminent.

If you want to evolve a star up to some particular point, say where central H is 0.05, and then stop, the simplest way is to put an extra JO code in printb.f, along with the others there:

IF (DABS(DMT).GT.30.0D0*PX(9)/TKH) JO = 10 ! already there
if (sx(10,2).lt.0.05d0) jo = 16 ! new line.

It is not very transparent that sx(10,2) is the central H abundance. SX stores up to 45 variables at each meshpoint: a DATA statement in printb.f identifies them (not all are used). For some reason which I forget, sx(i,2) and not sx(i,1) is the central value.

But as well as doing legitimate evolution, one can also do pseudo-evolution: moving the star up or down the main sequence by adding or subtracting mass while not allowing the composition to change; generating a low-mass HB star from a high-mass one (thus avoiding the He flash) by losing surface mass, but allowing the hydrogen shell to burn outwards while preventing central helium from burning; generating a pre-main sequence Hayashi-track star from a ZAMS model by putting a uniform artificial energy source in the star; generating a white dwarf sequence; *etc.*

The input data file fort.22 above has entries that determine what kind of run is to be done. For example, you can move up or down the ZAMS by setting CMI ($\equiv d\ln m_*/dt$) in fort.22 to (say) $\pm 5.10^{-9}$ (yr^{-1}) instead of zero. With DTY = $5.73992.10^5$, this means that the mass is multiplied by 10 in the course of 800 timesteps. Prevent DTY from varying during evolution by putting CT1 = CT2 = 1 in fort.22. Choose the right number of steps (KPT) to get you to where you want. The final output model will be put in fort.13 (or 14), and intermediate models at every KSV'th timestep are stored in fort.15. If you are 'evolving' through the region $\sim 1 - 2\,M_\odot$, you may need smaller timesteps, since the convective boundaries tend to move rather rapidly in this region. DTY, and also KPT, are fed in by fort.23 rather than fort.22.

Note that we have two definitions of 'ZAMS', both idiosyncratic. In one the star has an absolutely uniform composition in ^{1}H, ^{4}He, ^{12}C, ^{14}N, ^{16}O, ^{20}Ne, ^{24}Mg, ^{28}Si and ^{56}Fe. Since ^{12}C burns very rapidly to ^{14}N in moderate to high mass stars, evolution from this 'ZAMS' model starts with a jerk (unless you use a *very* small timestep). A reasonable ploy is to begin by taking a single timestep of 'normal' length, *i.e.* about 0.002 - 0.005 times the expected lifetime in the MS band. This will, if it works at all, relax the C/N into equilibrium, for those upper MS models where *pp* is not the dominant energy source.

In the second definition, ^{12}C/^{14}N equilibrium of a sort is obtained, with a uniform distribution of all the other species, by solving for the C and N abundances only. The best ZAMS is likely to be obtained by (a) going *down* the MS with a completely uniform composition and with the composition change (and the thermal energy term) suppressed, to get a star of say 0.08 or 0.1 $M_\odot$; and then (b) go *up*, allowing the C/N to equilibrate (KCN = 1) but otherwise suppressing composition changes. Throw away the (a) ZAMS, and keep the (b) one. The point about the particular values suggested for CMI and DTY is that one evolves through the region $1 - 2\,M_\odot$ on something like a thermal timescale, which is also about the timescale on which actual stars get there from the Hayashi track.

From a ZAMS model with CN equilibrium one can generally evolve very easily to the He flash (in low-mass stars), degenerate C ignition (intermediate-mass stars), or the end of C burning (massive stars). Recently I have incorporated an algorithm that, when the He flash is reached, restarts the star on the ZAHB, so that one can in fact continue to C ignition in low-mass stars as well. One can let the star run entirely automatically, taking $\sim 3000 - 5000$ timesteps, or if one is prepared to intervene a bit one can do it in only 1000 timesteps. 'Intervention' means varying CDD from one portion of run to the next. A value like 0.01 is OK on the MS, but maybe should be less for central He burning, and more for steady shell burning, whether single or double shell. The value of 0.01 seems to work all the way, with the code having to backtrack and halve the timestep, maybe more than once, in some difficult regions, and with somewhat unnecessarily slow progress in shell burning.

The problem of the He flash is difficult, because an implicit centrally differenced scheme appears to be unstable, generating saw-tooth oscillations, once a part of the star starts to evolve on a much more rapid than thermal timescale. The problem is possibly compounded by my adaptive mesh, although I have no evidence that it is either worse or better than with a fixed mesh. A different form of differencing is probably required if one wants to follow the flash in detail − and I suspect one ought to do it in 3-D, not 1-D. I have not persevered with the He flash in 1-D because one can easily construct 'post-He-flash' models by a roundabout route:

(1) Evolve a ZAMS $2.25\,M_\odot$ model till its central He has just ignited (non-degenerately, by a whisker). Use KY = 0, so that although the He burns (ie. provides energy) it is not consumed.

(2) Subject it to fairly rapid mass loss using a negative CMI, but prevent this from contributing thermal energy (KTH = 0) until the surface mass is down to what you want.

(3) Finally switch off the mass loss, and let the shell burn outwards till it reaches the core mass you want, while continuing to suppress the He evolution in the core (KY = 0).

A model produced in this way is stored in fort.12, and is varied automatically to produce the ZAHB star required after the He core flash for low-mass stars. However, I've only done this (so far) for Z = 0.02.

With a bit of practice one can produce by similar stratagems
(a) 'ZAHeMS' stars
(b) He, or C/O, white dwarfs
(c) pre-MS fully convective (*i.e.* Hayashi track) stars.
In some cases, one might edit the input model by hand, say moving the H-shell in or out by a meshpoint, or even removing it altogether if it is fairly near the surface and you don't want it.

The parameter ENC can be used to construct pre-MS Hayashi track stars, from ZAMS ones. ENC is a constant energy source (constant in space, not time), normally zero. Make it non-zero and increase it gradually (at rate CET), keeping KTH = KX = 0, so that the star swells up and its centre cools down. There is not a lot of point in doing pre-MS stars with this code, because I have not yet added in the species ^{2}H, ^{3}He, ...; maybe I'll get around to it sometime. ENC can also be used to make luminous 'zero-age' white dwarfs.

It is possible to remove the composition equations altogether from the solution step, so that the composition remains fixed at each meshpoint even although the mass will vary at each meshpoint because of the adaptive grid. To do this you need different values for the integers KD(1) - KD(130) in fort.22, viz. (grouping in 10 and 3 lots of 40): (5, 0, 0, 3, 1, 18, 0, 0, 0, 0), (1, 2, 4, 8, 7, 6, 34*0), (6, 7, 8, 9, 10, 35*0), (6, 7, 8, 6, 7, 8, 34*0). This would allow you to reduce the core mass of an HB star while simultaneously reducing its total mass, not necessarily at the same rate. I hope this also illustrates the intended flexibility of the solution package: you can add in extra differential equations that you want solved along with the structure, but you can also easily ignore them in circumstances where you temporarily don't need them.

The programme can handle semidetached binaries rather easily, on the assumption that ∗1, the more massive star *initially*, fills its Roche lobe first, and that ∗2 never fills its Roche lobe. Actually, there is plenty of opportunity for ∗2 to fill its own lobe. It is pretty well bound to, at a late stage when ∗1 has finished its evolution and becomes an NS or WD. But it may well fill it before then, particularly while ∗1 still is semidetached so that the stars come into contact. I have developed a code which will solve both stars simultaneously at this stage (i.e. 34 equations instead of 11 or 13 or 19; but possibly one can simplify by forgetting about the nuclear evolution). This code has an algorithm for energy transfer in the outer layers of the two components, but it is very tentative so far. It doesn't give the answers I want.

The two figures, on pages 33 and 35, show some results for two binaries, with initial parameters $8+7\,M_\odot; 3.5\,\mathrm{d}$ and $2+1.8\,M_\odot; 3.5\,\mathrm{d}$. The first binary was evolved conservatively, which implies for present purposes that both components were non-rotating, or at least rotating very slowly. The second was evolved non-conservatively, and started with a mild eccentrity and with non-corotation, but it was sufficiently close that the orbit circularised and ∗1 synchronised fairly quickly. For each binary, say (A) and (B), the 6 panels show the following:

(a) The theoretical HRD for both components. In (A), ∗1 became a He star after its first episode of RLOF, and then expanded back to the GB (Case BB), but reached C ignition, and presumably an SN explosion, at almost the same time as it re-reached its Roche lobe. In (B), ∗1 evolved to RLOF, but only after losing a considerable amount of mass ($\sim 0.5\,M_\odot$) by stellar wind as a red subgiant. RLOF left it a He WD of $0.3\,M_\odot$. ∗2 evolved to the GB, and was *just* able to ignite He before it would have filled its own Roche lobe. It shrank back, and did not in fact fill its lobe until it had started on the AGB. Presumably there would be a common-envelope episode, perhaps leaving a CO WD paired with a He WD.

(b) The quantities $\log r_{\rm star}$ and $\log r_{\rm lobe}$ as functions of mass, for each of ∗1 and ∗2. (B) terminated

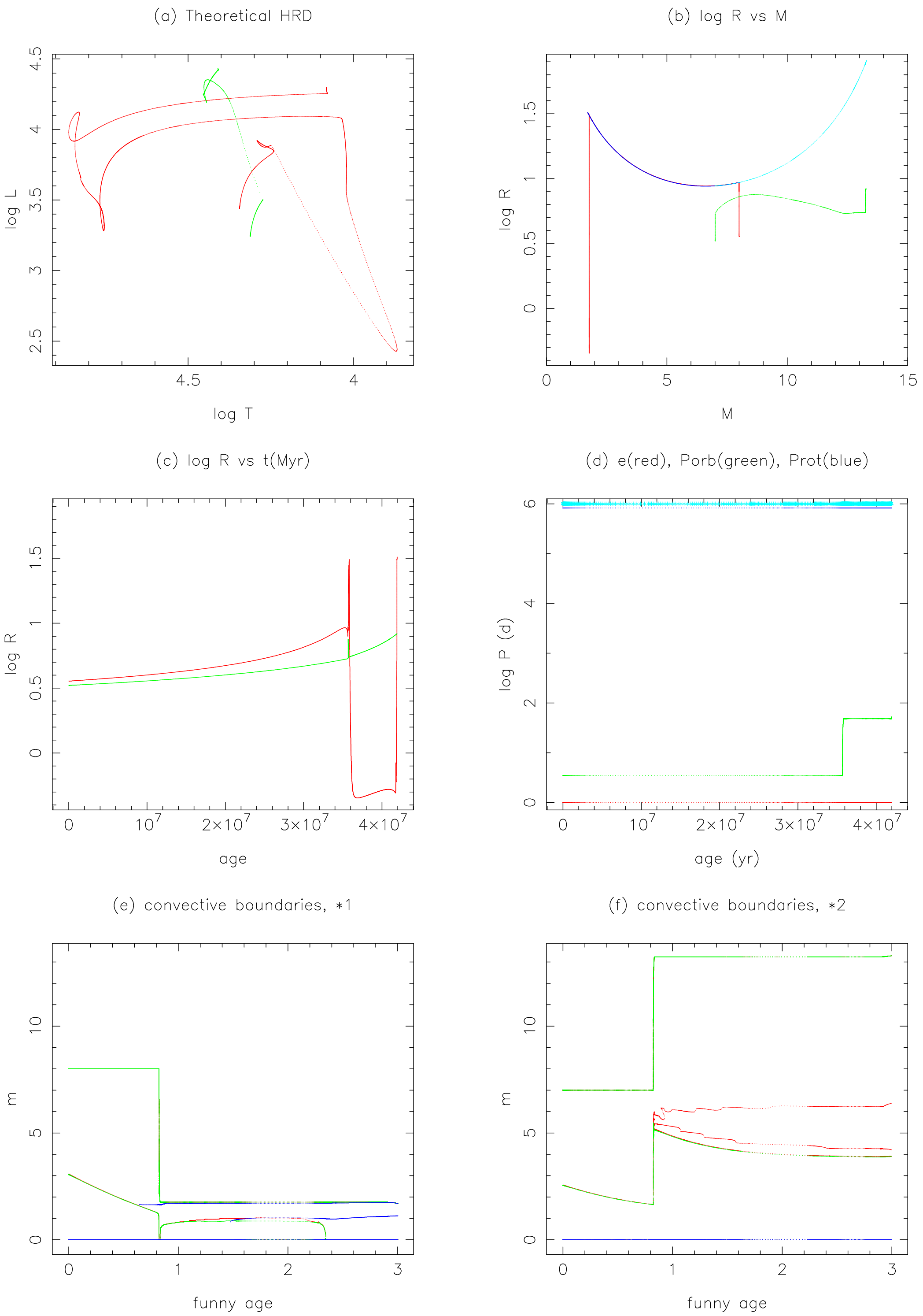

Fig A – Evolution (conservative) of a binary with initial parameters ($8 + 7\,M_\odot; 3.5\,\mathrm{d}$).

when $*2$ reached its Roche lobe (reverse RLOF), following at a substantial interval after forward RLOF from $*1$.

(c) $\log r_{\rm star}$ for each star, as functions of time.

(d) The eccentricity, orbital period and rotational period of $*1$, as functions of time. In (B), the orbit circularised and the periods synchronised in $\sim 300\,$Myr.

(e) The internal structure of $*1$ as a function of time. Curves show the location of the surface, the convective and semiconvective boundaries, and the shell where hydrogen or helium abundance is 0.15. The time coordinate is logarithmic, $-\log(1-t/t_0)$, with t_0 being just beyond the end of the evolution.

(f) The same for $*2$.

1.12 TWELVE TEST JOBS

I have already mentioned that there are several different kinds of job that one might wish to do with basically the same code: construct a ZAMS, evolve a single star, evolve a binary, evolve a grid of single stars or of binaries, etc. One philosophy is to have say 12 different versions of the code, to cope with 12 different tasks; another is to put 12 different options in the same code. I have tried both philosophies at different times, and I can only say I found the latter to be much the simpler in the long term, though not in the short term.

I am developing a suite of test jobs which is supposed to cover most of the tasks that I have carried out at some time in the past. At present this comprises 12 tasks or groups of tasks, which in a few words are:

run01: Construct a table of ZAMS models, at equal intervals of $\log M$, possibly with different metallicity than the 'standard' solar Z: I selected $\Delta \log M = 0.025$, $-1 \leq \log M \leq 2.3$, somewhat arbitrarily. Higher masses are quite feasible, but I haven't often computed them.

run02: Evolve a one-parameter grid of (effectively) single stars, with equally-spaced $\log M$ selected exactly from the ZAMS compilation of run01. If you want to do some rather boring isochrones for single stars, you may want a finer grid in log M, but probably over a more limited mass range, say 1 - 30 $M_\odot$. If it is for non-solar metallicity, you will probably have to do a few manipulations with run01 first.

run03: Evolve non-conservatively a star with a mass somewhat different from any in the Table in run01. It is a 'pre-solar' star, starting on the ZAMS with slightly greater mass than the Sun, and with rapid rotation (3 d). Dynamo activity, mass loss and magnetic braking bring it in $\sim 4.6\,$Gyr to solar mass and rotation rate.

run04: Evolve non-conservatively $*1$ only, but allowing for partial accretion on $*2$ of wind from $*1$, and also for bipolar re-emission of some (but in the examples here all) of the accreted mass. This may be relevant to CVs, and to LMXBs.

run05: Evolve conservatively a (3-parameter) grid of 9 binaries where both masses come from the Table created by run01 – so that (log) mass-ratios are necessarily 0.025, 0.05, ... Only one value of $\log q$, 0.1, was used, but 3 values of $\log M_1$ and 3 of $\log P$. Periods are also equally spaced logarithmically. The subdirectory also contains the initial and output data for the run producing Fig A.

run06: As run05, but only 2×2 models, and non-conservative. The subdirectory also contains the initial and output data for the run producing Fig B.

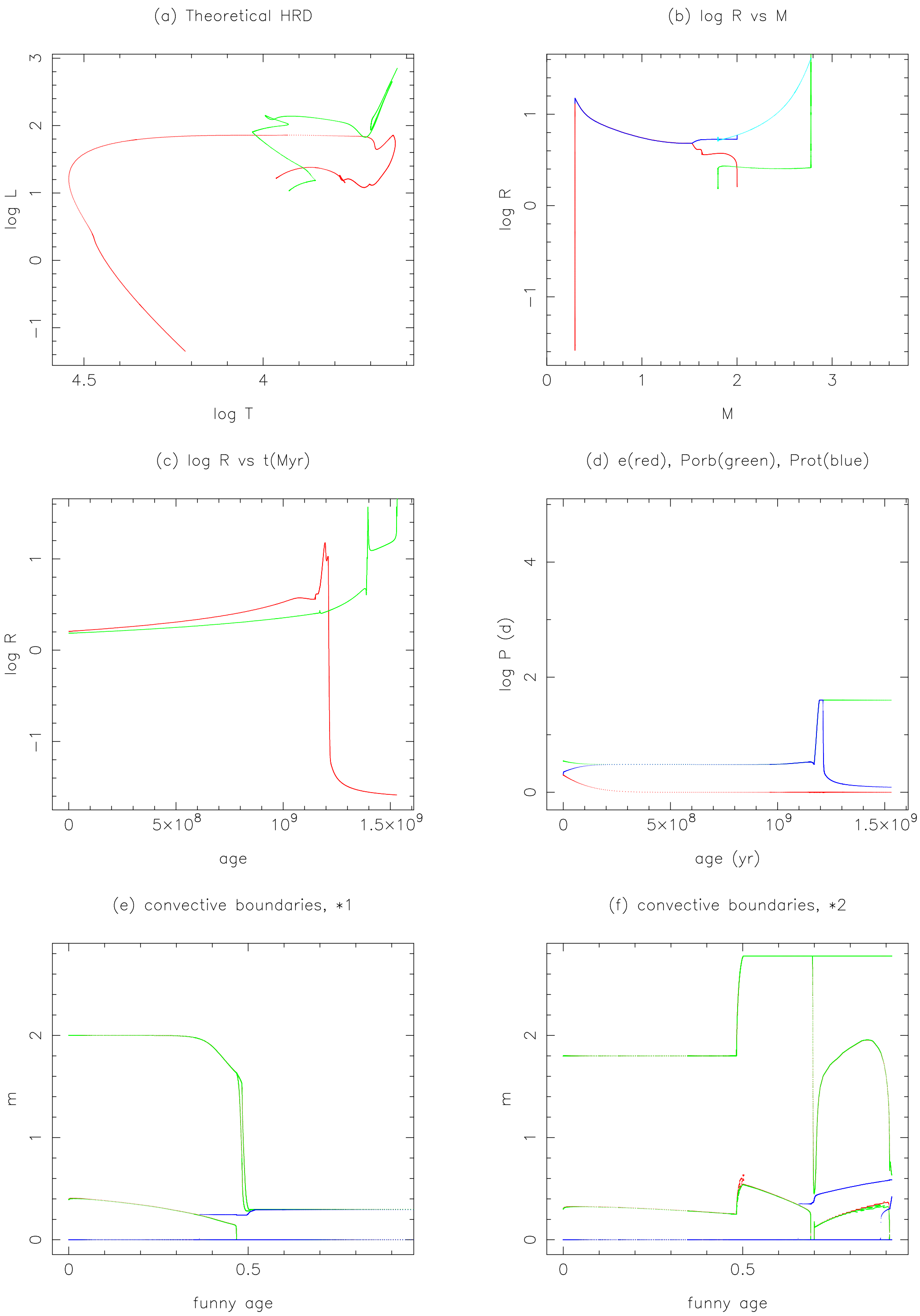

Fig B – Evolution (non-conservative) of a binary with initial parameters $(2 + 1.8\,M_{\odot}; 3.5\,\mathrm{d})$.

run07: This is 3 sub-jobs. (a) Construct a 'Zero-Age Horizontal Branch' model from a ZAMS model: evolve a $2.25\,M_\odot$ star in which the composition change, but not the energy generation, due to He burning is suppressed. (b) Construct a ZAHB of stars with a given core mass, starting from the 1100th timestep (12th stored model) of the previous run and imposing a fixed mass-loss rate. This run ended when the star was stripped down to a He core of $\sim 0.29\,M_\odot$, and He was no longer able to burn. Run (c) started from the 800th timestep (17th stored model) of run (b), a He star of $\sim 0.32\,M_\odot$, and added mass to it to create a ZAHeMS which went up to $\sim 21\,M_\odot$. The code couldn't get beyond that: I'm not clear why.

run08: 4 sub-jobs, the last two of which end with star 1 as a He WD and a CO WD respectively, so you can have starting models of such stars for other purposes. (a) is a binary run from run06, where $*1$ becomes a He WD of $0.295\,M_\odot$. In (b), a small H-rich envelope is stripped off by artificial mass loss. (c) Then *add* mass artificially. Note that added mass *always* has the same composition as the photospheric layer – I'd like to change that – so that if you added mass directly in (b) you would get a series of nova outbursts, or more likely it would just break down. Run (c) broke down at $1.34\,M_\odot$, when He was in the course of pycnonuclear ignition. (d) This starts from a ZAHeMS model – from run07 – of $0.531\,M_\odot$, and ends with a cooling sequence of CO WD models; they have a substantial He envelope.

run09: (a) Construct pre-main-sequence, Hayashi-track, models with $0.5\,M_\odot$ and increasing luminosity. This is done by adding in an artificial energy-generation rate that is constant throughout the star and increasing with time. (b) Also follow the contraction of a pre-MS protostar. Starting from the last model of the previous subjob, let it contract thermally. Since deuterium, ^{3}He, . . . are not included, an important stage is missed out; but one could no doubt add them to the 5 elements normally included. (c) Also starting from the last model of (a), accrete mass on a thermal timescale, suppressing fuel consumption. This could generate PMS starting models of higher mass than $0.5\,M_\odot$. However as the mass and luminosity go up the accretion rate does not, so the model would converge to the ZAMS, like run01. It might be good to have a given accretion rate that goes up proportionally to the luminosity.

run10: Use the TWIN model to follow three binaries (BY Dra, RZ Eri, Z Her) where non-conservative processes in *both* stars are likely to be important.

run11: Use the TWIN model to follow a very close $1.25 + 1\,M_\odot$ binary into RLOF, and then into contact.

run12: Use the TWIN model to do RLOF including a simple expression for the rotational spin-up of the gainer.

The number of equations solved is the basic 11 in 01, 02, 05, 07 and 08, 13 in 03, 19 in 04 and 06, 6 or 11 in 09 and 34 in $10 - 12$.

All of these tasks can be done with the same computational 'core' code, but with different options triggered in the 'shell' around it. In the core we have to have some free parameters that multiply certain bits of physics: parameter zero means the bit of physics (say, thermal energy generation, or RLOF, or stellar wind) is suppressed, and parameter unity means that it is included. We also have input parameters that determine which kind of computation the core is to do. I'm afraid this has turned out to be quite complicated for the TWIN mode. KTW = 1 means do the components semi-independently, KTW = 2 means solve them simultaneously, often by doing (in effect) a loop DO II = 1, KTW. Although each shell separately might be supposed to be fairly simple, I have opted for a single shell that tries to subsume *all* of the above tasks. If one could be confident that the core code would never change, it would make sense to have have

12 different simple shells; but in practice I have found that the core is always developing, even if only in minor ways, and that it is a lot of trouble to keep checking that a new version of the core is compatible with all 12 shells: I find it slightly less trouble with a single more complex shell.

The set of files that I will send on request, including the code itself, can include 12 subdirectories called run01, run02, ..., run12. Each contains the appropriate fort.22 and fort.23 (but renamed f22 and f23) to do an example from the list above. Some subdirectories contain two or more sets. Some subdirectories also contain the output files that are obtained, typically fort.9 (abbreviated to f9). I wish I could say that I did all twelve runs with *exactly* the same code (both core and shell). In practice I made a few modifications on the way, hoping that I did this sufficiently carefully that the new code would still do the previous tasks in the same manner.

1.13 SUGGESTIONS ON HOW TO USE THE CODE

I have not attempted to make my code very user-friendly, but if you have a stellar evolution problem that you think you might use it for, here is what I suggest. Firstly, set up a directory (I'll call it the main directory) which contains the compiled code, 'ev', and the (fairly) standard files fort.11, fort.12, and fort.16 to fort.21; also the subdirectories run01 – run12, copied from the library. Then suppose, for example, that you have just read Griffin (2002) – AJ, 123, 1001 – and want to model o Leo, a ($2.12 + 1.87\,M_\odot$; 14.5 d) binary whose $*1$ is right in the middle of the Hertzsprung gap ($5.7\,R_\odot$). You might be interested in its future evolution or its past, or just its present structure. Read the list in the previous section of the runs in run01 – run12 to see which is most like what you want to do. If you are only interested in $*1$ you might think of run03 – single star only – but if you are interested in both then run05 or run06 might be necessary. Run05 evolves both stars, with conservative RLOF if you want to consider future evolution. o Leo looks like a fairly straightforward pre-RLOF system, not very far from RLOF. But if you want to know whether tidal friction should have circularised the orbit by now (it is observed to be circular) you might want run06, which puts in tidal friction, spin, and eccentric orbits. Run06 is 'singly non-conservative', i.e. tidal friction, magnetic braking and mass loss are included but only for $*1$. Conceivably you might want the whole hog – these effects computed in *both* components – which would mean run10. But let's say you think run06 will do.

Copy the f22 and f23 files of run06 to fort.22 and fort.23 in the main directory. Fort.23 will need some editing, but not fort.22. Edit the first 6 lines of fort.23 from

```
2   1  16  1  16  1  3000  200
0.00D0   0.30D0   2
0.10D0   0.05D0   1
0.60D0   0.30D0   2
-0.3D0   2   0.30D0
-1.0D0   -1.0D0   0.0D0   -1.0D0   -1.0D0   0.3D0   -1.0D0   0.0D0   1
```

to

```
2  1  16  1  16  1  1000  200
0.325D0   0.30D0   1
0.05D0   0.05D0   1
0.60D0   0.30D0   1
-0.3D0   3   0.30D0
```

2.12D0 -1.0D0 0.0D0 1.45D1 3.99D0 0.0D0 3.0D0 0.0D0 0
[last line is
SM DTY AGE PER BMS ECC P1 ENC JMX]
Considering only the changes:
3000 → 1000 1000 timesteps will probably do
0.00D0 → 0.325D0 M1 is 2.12, so log M1 ~ 0.325; multiples of 0.025 are convenient
2 → 1 only do 1 case, not 2, with log M1 incremented by 0.3. Obviously the value of increment is irrelevant if you want to do only one case
0.10D0 → 0.05D0 Q=M1/M2, so log Q ~ 0.05; again use nearest multiple of 0.025
2 → 1 again to do only one case
KR: 2 → 3 ; leave EX at 0.3; see below
-1.0D0 → 2.12D0 replace ∗1's approximate mass SM ($10^{0.325}\,M_\odot$) by the precise mass required
-1.0D0 to 1.45D1 supersede the default initial orbital period by 14.5d
-1.0D0 → 3.99D0 replace approximate binary mass BMS ($10^{0.325}+10^{0.275}\,M_\odot$) by the precise value required
0.3D0 → 0.0D0. Because KR = 3 and EX = 0.3, the actual initial orbital period will be set such that the system has the right angular momentum to reach P = 14.5d when $e = 0.0$, starting from $e = 0.3$.
-1.0D0 to 3.0D0 start with spin periods of 3 days each, say
1 → 0 the first 'timestep' will simply be a convergence to the correct masses, with no thermal evolution.
The last row of numbers, identified as SM, ... JMX, are potential replacements for default values of starting quantities. They only replace the default if, individually, they are non-negative. ECC is the eccentricity.

An alternative line SM, ..., JMX might be
-1.0D0 -1.0D0 0.0D0 1.45D1 -1.0D0 0.0D0 3.0D0 0.0D0 1
which would make do with the approximate SM and BM.

When I run this, I find that the eccentricity only drops to 0.082 by the time the radius has increased to its present value, but that is probably not bad: an increase by a factor of 2 or 3 in the strength of tidal friction will give $e \sim 0$ (though so would a factor of 10^6). However the stellar-wind algorithm produced a mass loss of $0.03\,M_\odot$, so one should presumably do the calculation again starting with a slightly greater mass. This would probably also require a slightly shorter initial period.

Summary

A typical Algol comprises
Two stars of quite different sizes.
    The one that is small
    Does nothing at all,
But the other evolves to a crisis.

It expands, and continues to grow
Till it reaches Roche-lobe overflow.
    Then it transfers some mass
    In a thin stream of gas
That falls on the other below.

The period starts to increase,
But the other star's not left in peace.
    It starts to grow fast,
    Overtaking at last,
And that's when the run has to cease.