Simulating Stars — run_star_extras

Ilaria Caiazzo, Jeremy Heyl. TAs: Amber Lauer, Kang Liu.

1 Introduction

Now it's the time to dig into the customization possibilities that run_star_extras gives you! For an introduction to run_star_extras, if you haven't done it already, please take a look at the slides included in this folder.

Run_star_extras is written in Fortran, but if you don't know Fortran don't be afraid, as long as you have coded in any language before, you'll be fine. And if you have any problems, please ask your fellow students and the TAs for help! A list of basic commands in Fortran is available in the appendix to get you started.

I would like to thank Josiah Schwab for letting me use some of his examples.

2 Stopping the run at 2 Rsun on the RGB

We just saw together how to add a stopping condition for $r > 2$ Rsun using run_star_extras (again, see the slides). However, since we are using the `1M_pre_ms_to_wd` test_suite, and since the pre-main-sequence star has a radius bigger than 2 solar radii, the run will stop right at the beginning.

We want the run to stop on the RGB, so after the end of the main sequence. A way to do it, is to add an additional check: is the hydrogen in the center of the star depleted yet? Ask MESA to stop if both conditions are met.

Task: take a look at the `star_info` structure to find what's the name of the component for the central hydrogen. Add a stopping condition when both the radius is bigger than 2 Rsun and the central hydrogen mass fraction is less than 10^{-4}.

Solution:

```
! returns either keep_going, retry, backup, or terminate.
integer function extras_finish_step(id, id_extra)
   use chem_def
   integer, intent(in) :: id, id_extra
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_finish_step = keep_going
   call store_extra_info(s)

   if(s% r(1) > 2 * Rsun .and. s% center_h1 < 1d-4) then
      extras_finish_step = terminate
      s% termination_code = t_xtra1
      termination_code_str(t_xtra1) = 'radius is bigger than 2 Rsun on the AGB'
      return
   endif
end function extras_finish_step
```

3 Stopping the run when the water on Earth boils

Let's imagine that we want to study the destiny of our own planet, and we want to understand how the temperatures on Earth will change as the Sun grows older. If you assume that the Earth is a perfect black body, its equilibrium

temperature is:

$$T_\oplus = T_\odot \left(\frac{R_\odot}{2\,\mathrm{AU}} \right)^{\frac{1}{2}} \tag{1}$$

While the Sun is burning hydrogen on the main sequence, its temperature slowly increases. We want to keep track of the temperature on Earth as the Sun evolve.

Task: Add a new history column for the temperature of the Earth. The routines that you need are `how_many_extra_history_columns` and `data_for_extra_history_columns`. Take a look at Fig. 1 for instructions.

Mind that you don't have to (and shouldn't) add the name you are defining to the history_column.list file.

```
integer function how_many_extra_history_columns(id, id_extra)
   integer, intent(in) :: id, id_extra
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return

   !you have to write here how many new history columns you want
   how_many_extra_history_columns = 0
end function how_many_extra_history_columns


subroutine data_for_extra_history_columns(id, id_extra, n, names, vals, ierr)
   integer, intent(in) :: id, id_extra, n
   character (len=maxlen_history_column_name) :: names(n)
   real(dp) :: vals(n)

   !you need to define a new real here, for example:
   real(dp) :: t_earth

   integer, intent(out) :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return

   !and you have to calculate the value of t_earth, using the values you need from %s (star_info)
   t_earth = ...

   !finally, you have to output the value in the history.data. You have to specify what name you want, for example:
   names(1) = "T_earth"
   !and the value you want to output:
   vals(1) = t_earth

   ierr=0
end subroutine data_for_extra_history_columns
```

Figure 1: Instructions to add an extra history column.

Solution:

```
integer function how_many_extra_history_columns(id, id_extra)
   integer, intent(in) :: id, id_extra
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   how_many_extra_history_columns = 1
end function how_many_extra_history_columns


subroutine data_for_extra_history_columns(id, id_extra, n, names, vals, ierr)
   integer, intent(in) :: id, id_extra, n
   character (len=maxlen_history_column_name) :: names(n)
   real(dp) :: vals(n)

   real(dp) :: t_earth

   integer, intent(out) :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return

   t_earth = s% Teff * sqrt(s% photosphere_r * Rsun / (2.0 * au))

   names(1) = "T_earth"
   vals(1) = t_earth - 273.15

   ierr = 0
end subroutine data_for_extra_history_columns
```

We are interested in monitoring the variation in temperature only until the Earth is still inhabitable. A stopping condition at around 100°C sounds reasonable.

Task: Add a stopping condition when the temperature of the Earth reaches a value that is specified in the inlist. You'll have to use the routine `extra_finish_step` and the controls `x_ctrl` that we saw in the slides.

As a reminder, the `star_info` structure contains a set of controls that can be defined in the controls section of the inlist:

```
&controls
x_ctrl(1) = 2.52
x_integer_ctrl(1) = 30
x_ctrl(2) = 3.2
x_logical_ctrl(1) = .true.
```

and then you can call them in run_star_extras.f as e.g. `s% x_ctrl(2)`.

Hint: As before with the radius, the run starts as a pre-main-sequence star and the temperature on Earth when the Sun was a pre-main-sequence star was already higher than 100°C. We want the run to stop after the Sun as reached the main-sequence. Indeed, we know that right now the temperature on Earth is less than 100°C.

Solution:

```
&controls

      ! temperature stopping condition in degrees Celsius
      x_ctrl(1) = 100

      use_gold_tolerances = .true.
      use_eosELM = .true.
      use_eosDT2 = .true.

      ! check for retries and backups as part of test_suite
      ! you can/should delete this for use outside of test_suite
         max_number_backups = 20
         max_number_retries = 500
         max_model_number = 15000

      initial_mass = 1.0
      initial_z = 0.02d0

      use_Type2_opacities = .true.
      Zbase = 0.02d0
```

```
! returns either keep_going, retry, backup, or terminate.
integer function extras_finish_step(id, id_extra)
   use chem_def
   integer, intent(in) :: id, id_extra
   integer :: ierr
   real(dp) :: t_earth
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_finish_step = keep_going
   call store_extra_info(s)

   t_earth = s% Teff * sqrt(s% photosphere_r * Rsun / (2.0 * au))

   write(*,*) "T_earth", t_earth

   if(t_earth > (s% x_ctrl(1) + 273.15) .and. s% star_age > 4.5d9) then
      extras_finish_step = terminate
      s% termination_code = t_xtra1
      termination_code_str(t_xtra1) = 'T_earth higher than set temperature'
      return
   endif
end function extras_finish_step
```

Bonus task: Instead of defining `t_earth` in both routines (`extra_finish_step` and `data_for_extra_history_columns`) you can store the value of `t_earth` in a `xtra` variable. MESA provides a number variables that are useful for implementing algorithms which require a state. if you just use these variables, restarts, retries, and backups will work without doing anything special. They are named `xtra1 .. xtra30`, `ixtra1 .. ixtra30`, and `lxtra1 .. lxtra30` for double, integer and boolean precision respectively. They are automatically versioned, that is if you set `s% xtra1`, then `s% xtra1_old` will contains the value of `s% xtra1` from the previous step and `s% xtra1_older` contains the one from two steps ago. You can use them also to store variables between routines.

Solution:

```
integer function how_many_extra_history_columns(id, id_extra)
   integer, intent(in) :: id, id_extra
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   how_many_extra_history_columns = 1
end function how_many_extra_history_columns


subroutine data_for_extra_history_columns(id, id_extra, n, names, vals, ierr)
   integer, intent(in) :: id, id_extra, n
   character (len=maxlen_history_column_name) :: names(n)
   real(dp) :: vals(n)

   real(dp) :: t_earth

   integer, intent(out) :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return


   t_earth = s% xtra1

   names(1) = "T_earth"
   vals(1) = t_earth - 273.15

   ierr = 0

end subroutine data_for_extra_history_columns
```

```
! returns either keep_going, retry, backup, or terminate.
integer function extras_finish_step(id, id_extra)
   use chem_def
   integer, intent(in) :: id, id_extra
   integer :: ierr
   real(dp) :: t_earth
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_finish_step = keep_going
   call store_extra_info(s)

   t_earth = s% Teff * sqrt(s% photosphere_r * Rsun / (2.0 * au))

   s% xtra1 = t_earth

   write(*,*) "T_earth", t_earth

   if(t_earth > (s% x_ctrl(1) + 273.15) .and. s% star_age > 4.5d9) then
      extras_finish_step = terminate
      s% termination_code = t_xtra1
      termination_code_str(t_xtra1) = 'T_earth higher than set temperature'
      return
   endif
end function extras_finish_step
```

Bonus task: Instead of having the run stopping at `t_earth>x_ctrl(1)`, try to put a stopping condition when `t_earth` is equal to `x_ctrl(1)` within a certain precision.

Hint: You will have to use the routine `extras_check_model` and ask for a "redo" with a reduced timestep, as well as `extras_finish_step`.

Solution:

```
! returns either keep_going, retry, backup, or terminate.
integer function extras_check_model(id, id_extra)
   integer, intent(in) :: id, id_extra
   integer :: ierr

   real(dp) :: t_earth, dt_earth, delta
   real(dp), parameter :: epsilon = 1d-6

   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_check_model = keep_going


   t_earth = s% Teff * sqrt(s% photosphere_r * Rsun / (2.0 * au))

   s% xtra1 = t_earth

   dt_earth = t_earth - (s% x_ctrl(1) + 273.15)
   delta = dt_earth / (s% x_ctrl(1) + 273.15)


   if(delta > 0 .and. s% star_age > 4.5d9) then
      if(delta > epsilon) then
          extras_check_model = redo
          s% dt = 0.5d0 * s% dt
      endif
   endif

end function extras_check_model
```

```
! returns either keep_going, retry, backup, or terminate.
integer function extras_finish_step(id, id_extra)
   use chem_def
   integer, intent(in) :: id, id_extra
   integer :: ierr
   real(dp) :: t_earth
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_finish_step = keep_going
   call store_extra_info(s)

   t_earth = s% xtra1

   write(*,*) "T_earth", t_earth

   if(t_earth > (s% x_ctrl(1) + 273.15) .and. s% star_age > 4.5d9) then
      extras_finish_step = terminate
      s% termination_code = t_xtra1
      termination_code_str(t_xtra1) = 'T_earth higher than set temperature'
      return
   endif
end function extras_finish_step
```

4 Changing the wind on the AGB

MESA allows you to overrun its physics routines. When you want to implement and test a new physics routine, the first thing to do is to look into the folder `$MESA_DIR/star/other`, where all the hooks are listed as "other" routines, and look for the one corresponding to the physics you want to change. If you open one of them, you can see that many contains examples and suggestions, and that they all contain a `null_other_*` subroutine.

We don't want to edit the subroutine in the file. In general the steps to use a "other" routine are the following:

- Find the `other_*` routine that corresponds to the physics you want to change in the `$MESA_DIR/star/other` folder.
- Copy the corresponding `null_other_*` subroutine into the `run_star_extras.f` file, at the same level as the other subroutines. It can be anywhere in the file after the word `contains`, as long as it's at the same level of indentation as the other subroutines like `extras_controls` for example.
- Change the word `null` with a word that you like in the definition and end of the subroutine (for example, instead of `null_other_wind`, you can call it `lecture_other_wind`).
- Edit the new subroutine to include the physics you want.
- Let MESA know that you want to use the "other" routine. In order to do that, you need to edit both `run_star_extras.f` and the `inlist`. In `run_star_extras.f`, you can see that the subroutine `extras_controls` instructs MESA on where to look for the different subroutines; you need to add an ulterior instruction for your subroutine: `s% other_* => your_*_routine` (for example `s% other_wind => lecture_wind_routine`). In the `inlist` as well, you have to set the command `use_other_*` to `.true.`

For this task, we want to test the effect of a new mass-loss model for the asymptotic giant branch. In the `1M_pre_ms_to_wd`, the default wind model implemented on the AGB is the Blocker formula (Blocker 1995):

$$\dot{M}_B = 1.93 \times 10^{-21} \eta_{\rm B} \left(\frac{M}{M_\odot}\right)^{-3.1} \left(\frac{L}{L_\odot}\right)^{3.7} \frac{R}{R_\odot} \quad [M_\odot\,{\rm yr}^{-1}]. \tag{2}$$

where $\eta_{\rm B}$ is a constant that determines the efficiency of the mass-loss on the AGB. A possible alternative for the mass-loss on the AGB is a chromospheric mass-loss prescription (Schröder & Cuntz 2005):

$$\dot{M}_{SC} = \eta_{\rm sc} \frac{L_* R_*}{M_*} \left(\frac{T_{\rm eff}}{4000{\rm K}}\right)^{3.5} \left(1 + \frac{g_\odot}{4300 \cdot g_*}\right) \quad [M_\odot\,{\rm yr}^{-1}]. \tag{3}$$

where R_*, M_*, and L_* are the stellar radius, mass, and luminosity given in solar units and g_* and $g_\odot$ are the stellar and solar surface gravity, respectively. Again, $\eta_{\rm sc}$ is an efficiency parameter and is around 8×10^{-14} $M_\odot$ yr^{-1}.

Task: Follow the steps listed above to change the mass-loss prescription on the AGB to a Schröder & Cuntz one.

You'll see that in the `other_wind` routine there are a few suggestions and notes. One note says: `don't assume that vars are set at this point, so if you want values other than those given as args, you should use values from s% xh(:,:) and s% xa(:,:) only, rather than things like s% Teff or s% lnT(:), which have not been set yet.` This is because the `other_wind` routine is called before things like `s% Teff` are set. You can find what are the different columns of the arrays `s% xh(:,:)` and `s% xa(:,:)` in the `star_data.inc` file. However, for our purpose, the values given as args are enough (we just have to calculate Teff as $T_{\rm eff} = (L/(4\pi\sigma R^2))^{1/4}$).

Also, if you are using the `star_info` structure, for example to set the η parameter in the `inlist`, you'll have to call it in the routine.

Hint: You want the new wind to be active only on the AGB, so ideally after the star has run out of helium in its core. Remember that you can access all the commands in the `inlist` through the `star_info` structure, including the `use_other_wind` command (`s% use_other_wind`).

Solution:
I copied the `null_other_wind` routine in my `run_star_extras.f`, I renamed it to `lecture_other_wind` and I coded in the new wind:

```
! these routines are called by the standard run_star check_model
contains

subroutine lecture_other_wind(id, Lsurf, Msurf, Rsurf, Tsurf, w, ierr)
   use star_def
   integer, intent(in) :: id
   real(dp), intent(in) :: Lsurf, Msurf, Rsurf, Tsurf ! surface values (cgs)
   ! NOTE: surface is outermost cell. not necessarily at photosphere.
   ! NOTE: don't assume that vars are set at this point.
   ! so if you want values other than those given as args,
   ! you should use values from s% xh(:,:) and s% xa(:,:) only.
   ! rather than things like s% Teff or s% lnT(:) which have not been set yet.
   real(dp), intent(out) :: w ! wind in units of Msun/year (value is >= 0)

   ! defining new useful variables:
   real(dp) :: surf_grav, surf_grav_sun, eff_temp

   integer, intent(out) :: ierr

   ! you have to call the star_info structure
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return

   ! calculating surface gravity and effective temperature:

   surf_grav = standard_cgrav * Msurf / Rsurf**2.

   surf_grav_sun = standard_cgrav * Msun / Rsun**2.

   eff_temp = (Lsurf /(boltz_sigma * 4 * pi * Rsurf**2.))**(1./4)

   ! calculating the mass loss:
   w = s% x_ctrl(1) * ((Lsurf / Lsun) * (Rsurf / Rsun) / (Msurf / Msun)) * &
      (eff_temp / 4000)**(3.5) * (1.0 + surf_grav_sun / (4300 * surf_grav))

   ierr = 0
end subroutine lecture_other_wind
```

...

...then I added the new routine to the list in `extras_controls`:

```
subroutine extras_controls(id, ierr)
   integer, intent(in) :: id
   integer, intent(out) :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return

   s% extras_startup => extras_startup
   s% extras_check_model => extras_check_model
   s% extras_finish_step => extras_finish_step
   s% extras_after_evolve => extras_after_evolve
   s% how_many_extra_history_columns => how_many_extra_history_columns
   s% data_for_extra_history_columns => data_for_extra_history_columns
   s% how_many_extra_profile_columns => how_many_extra_profile_columns
   s% data_for_extra_profile_columns => data_for_extra_profile_columns

   s% other_wind => lecture_other_wind
end subroutine extras_controls
```

...

...and I set the `other_wind` routine to be active only after the end of the horizontal branch in the `extras_finish_step` routine:

```
! returns either keep_going, retry, backup, or terminate.
integer function extras_finish_step(id, id_extra)
   use chem_def
   integer, intent(in) :: id, id_extra
   integer :: ierr
   type (star_info), pointer :: s
   ierr = 0
   call star_ptr(id, s, ierr)
   if (ierr /= 0) return
   extras_finish_step = keep_going
   call store_extra_info(s)


   if(s% center_he4 < 1d-5) then
      s% use_other_wind = .true.
   endif

end function extras_finish_step
```

A Basic Fortran

Variables

Declaring variables:

```
! declare an integer variable
integer :: i

! declare a double precision variable
real (dp) :: a

! declare a boolean variable
logical :: boo

! declare a 1d array with 10 elements
real(dp), dimension(10) :: vec
```

Assigning variables:

```
! boolean
boo = .true.
boo = .false.

! arrays
vec(1) = 3.2
vec(2:6) = 1.0
vec(7:10) = 0.2
```

Logic

In Fortran there are two equivalent ways of comparison, a textual form and a symbolic form:

text form	symbol form	description
.gt.	>	greater than
.lt.	<	less than
.ge.	>=	greater than or equal to
.le.	<=	less than or equal
.eq.	==	less equal to
.ne.	/=	not equal to

```
! these are the same
(a .gt. 0)
(a > 3.2)
```

There are 3 logical operators `.and.`, `.or.` and `.not.`

```
! true when 0 < i < 10
((i > 0) .and. (i < 10))

! true when i /= 0,1
(.not. ((i .eq. 0) .or. (i .eq. 1)))
```

Example of a if cycle:

```
if (x > 12) then
   a = 3.2d2
else if (x < 3) then
   a = 7.0
else
   a = 0.0
end if
```