MESA Summer School 2022: Session 1 Labs

Sunny Wong and Meridith Joyce

Abstract

This lab accompanies the first lecture of the 2022 MESA Summer School. Here, we introduce the MESA software and teach you how to navigate the code base. During this session, you will learn how to locate and change parameter settings, control output, and modify the source code via `run_star_extras`.

Google Drive link with repositories and solutions:

The following link hosts two folders: `lecture1.zip`, which contains a clean version of the working repository, and `solutions.zip`, which includes solutions to all pieces of this lab. Please consult this only if you get lost. While it is important that you do not fall behind, you will learn more if you issue the commands yourself.

https://drive.google.com/drive/folders/124eHggIEgsSvR_vYl9uCUcXlbQyhpnJL?usp=sharing

1 Setting environment variables (microlab 0)

This section will cover:

- Environment variables
- What is `.bashrc`, and where is it?

To run MESA, you must set *environment variables* that point your system to the correct paths for dependent packages. These are `$MESA_DIR`, `$MESASDK_ROOT`, and `$OMP_NUM_THREADS`.

To avoid having to set your environment variables manually every time you run MESA, you can define these variables within the shell start-up file, which is executed whenever a user opens a new shell. The default for most Linux distributions is the `bash` environment, which is controlled by the `.bashrc` file. You can check which shell your system uses by running

```
echo $0
```

If this returns `bash`, then you will edit `.bashrc`. If this returns `zsh` (as on some Macs), you will edit `.zshrc` (substitute the appropriate file in subsequent instructions accordingly).

The `.bashrc` file is located at `$HOME/.bashrc`. Check that you can locate the `.bashrc` file by running

```
ls -a $HOME
```

Here is an example from a machine that uses bash as its shell

```
# the location of your top-level MESA directory
export MESA_DIR=/home/mjoyce/MESA/mesa-r22.05.1

# the location of the SDK
export MESASDK_ROOT=/home/mjoyce/MESA/mesasdk
source $MESASDK_ROOT/bin/mesasdk_init.sh

# the number of threads (= 2x number of cores) you choose to devote to MESA calculations
export OMP_NUM_THREADS=8
```

Note that `/home/mjoyce/MESA/` is a path on my (mjoyce's) machine. Yours will be different.

2 Opening and modifying defaults files

This section will cover

- how to change control parameters
- the meanings of different inlist sections
- organizational structure of code base (e.g. noting that `opacity` defaults are located in a different place than `controls.defaults` and `star_job.defaults`)
- controlling output (history vs profile)
- how to turn off/on `pgstar`

2.1 Compile and Run

Each time you want to start a MESA project, you should make a new copy of the `star/work` directory.

```
cp -r $MESA_DIR/star/work my_new_project
```

In this case, we have prepared and provided a work directory for you. Download, unpack, and enter this work directory.

```
unzip lecture1.zip
cd lecture1
```

This directory evolves a $1.5\,M_\odot$ star from the main sequence to hydrogen exhaustion. To build your executeable (`star` or `binary`), which is the program that is built by the compiler and runs your simulation, issue the commands

```
./clean
./mk
```

Now you can run the code via

```
./rn
```

Your run should stop at time step 271.

2.2 Minilab 1: Using inlists

Meanwhile, take a look at the `inlists`, which are Fortran namelists that contain value definitions for all of the parameters of your run. `MESA/star` currently has 5 inlist sections:

- `star_job` - options for the program that evolves the star
- `eos` - options for the MESA eos module
- `kap` - options for the MESA opacities module, abbreviated `kap`
- `controls` - options for the MESA star module
- `pgstar` - options for on-screen plotting

The distinction between `star_job` and `controls` can be a little subtle, and sometimes there are overlapping controls. In general, `star_job` contains options that answer questions like:

- how should MESA obtain the initial model?
- are there any changes MESA should make to the initial model?
- which nuclear net and reactions should MESA use?

whereas `controls` contains options that answer questions like:

- what conditions should cause MESA to stop evolving the model?
- which angular momentum transport processes should MESA consider?
- what numerical tolerances should MESA's solvers use?

The `eos` and `kap` contain options for the equation of state and opacity modules, respectively, and answer questions like:

- which equation of state (EOS) components should be used and where (in terms of state variables) should they be blended?
- what composition should be assumed when calculating the opacity?

MESA's many inlist options have their default values set in `*.defaults` files. These files also provide documentation on the options, including possible values and references to the source material.

For a specific module, this file (or files) is located in <module>/defaults/. You can directly read these defaults files as plain text. Alternatively, they are rendered as part of the online documentation.

The individual module defaults appear under Module Documentation

- `$MESA_DIR/eos/defaults/eos.defaults`
- `$MESA_DIR/kap/defaults/kap.defaults`

while the `star` (and also `binary` and `astero`) defaults appear under Reference

- `$MESA_DIR/star/defaults/star_job.defaults`
- `$MESA_DIR/star/defaults/controls.defaults`
- `$MESA_DIR/star/defaults/pgstar.defaults`

They are roughly sorted into groups of related options. When you're searching for an option, see if it seems to match any of the section headings and then look there first. If that fails, try searching for some key words.

Note that inlists can point to other inlists. This can be useful for keeping things organized. In the `lecture1` directory, `inlist` points MESA to `inlist_project` and `inlist_pgstar`.

2.3 Controlling Output

By default, MESA stores output files in the `LOGS/` directory. The two output file types are

- `history.data`, which contains *evolutionary information*. This stores the value of scalar quantities (e.g., mass, luminosity) at different timesteps, and
- `profileX.data`, which contains *structural information*. This gives you spatially varying quantities within the model at fixed time t (e.g. density, pressure, etc as a function of radius or $m/M_\star$).

You can adjust the frequency of these outputs in the inlists.

The contents of MESA's output files are **not** directly controlled via inlists. The default output is set by the files

```
$MESA_DIR/star/defaults/history_columns.list
$MESA_DIR/star/defaults/profile_columns.list
```

In order to customize the output, you copy these files to your work directory.

```
cp $MESA_DIR/star/defaults/history_columns.list .
cp $MESA_DIR/star/defaults/profile_columns.list .
```

Then, open up `history_columns.list` or `profile_columns.list` in a text editor and comment/uncomment any lines to add/remove the columns of interest ('!' is the comment character.)

3 Minilab 2: compare two MESA output tracks

This section will cover

- renaming output files
- locating inlist parameter keys
- graphically comparing output between runs with different parameter assignments

Having run through a $1.5M_\odot$ model with the default value of the mixing length parameter $\alpha_{\rm MLT}$ (take a look at `controls.default` to see what the default value is), now run another model with $\alpha_{\rm MLT} = 1.5$. Make sure you do not overwrite the first output directory. Using your favorite plotting tool*, compare the two models on the HR diagram.

HINT 1: You can specify the directory in which MESA stores output files.
HINT 2: For each of the output directory and $\alpha_{\rm MLT}$, is the option a `star_job` or `controls` option?

*While we understand that each scientist has their preferred tools for data analysis, we are best equipped to help you with the MESA in-house plotting program `pgplot` or the Python tool `py_mesa_reader`, available on Github.

If you are having trouble with plotting, we provided a Fortran file `hr.f` that makes use of pgplot to generate a HR diagram. To use, first edit `hr.f` to provide it with the appropriate pathnames to your history files, then use the terminal command

```
make
```

to generate an executable `hr`, and finally run this executable

```
./hr
```

Click here for solutions

4 Maxilab 1: Modify stopping condition in run_star_extras

This section will cover

- how to find, move, and compile the `include` file
- MESA flow-of-control
- data structures
- code organization
- how to write a custom stopping condition

Take a look at your `run_star_extras.f90` file, located within the `src` directory. You can do this by opening the files in your text editor of choice or by using the terminal command:

```
cd ./src
less -S ./run_star_extras.f90
```

The stock version of `run_star_extras.f90` is quite boring. It "includes" another file which holds the default set of routines.

```
include 'standard_run_star_extras.inc'
```

which is located in the `$MESA_DIR/star/job/` directory.

The routines defined in the `standard_run_star_extras.inc` file are the ones we will want to customize. Because we want these modifications to apply only to this working copy of MESA, and not to MESA as a whole, we want to replace this `include` statement with the contents of the included file.

4.1 Microtask: Replacing include statement

As we will do whenever we want to customize our `run_star_extras.f90` file when starting from a fresh working directory, replace the `include` line, with the the contents of `$MESA_DIR/star/job/standard_run_star_extras.inc`.

Before we make any changes, we should check that the code compiles.

```
cd ../
./mk
```

HINT 1: The command to insert the contents of a file in `emacs` is `C-x i <filename>`, in `vim` is `:r <filename>`, or you can just copy and paste.
HINT 2: If it doesn't compile, double check that you cleanly inserted the file and removed the include line.

4.2 The `star_info` structure and `star_data` module

The `star_info` data structure contains all the information about the star that is being evolved. By convention, the variable name `s` is used throughout `run_star_extras.f90` to refer to this structure. In Fortran, the percent (%) operator is used to access the components of the structure. (So you can read `s% x = 3` in the same way that you would read `s.x = 3` in C or Python.)

The `star_info` structure contains the stellar model itself (i.e., zoning information, thermodynamic profile, composition profile). The `star_data` module contains the definition of the `star_info` structure and more specifically, these components are listed in the file `$MESA_DIR/star_data/public/star_data.inc`, which points to 3 other files:

```
include "star_data_step_input.inc"
    !input for taking a step
include "star_data_step_work.inc"
    !working storage for data derived from inputs while taking a step
include "star_data_procedures.inc"
    !procedure pointers for hooks and such
```

These "included" files are also located in `$MESA_DIR/star_data/public/` . Most useful quantities about the stellar model are most likely in the `step_input` or `step_work` files; When in doubt, just search in that directory for your keyword, e.g.

```
cd $MESA_DIR/star_data/public
grep -i "Teff" *
```

In addition, `star_info` contains the values for the parameters that you set in your controls inlist (i.e., `initial_mass`, `xa_central_lower_limit`), although they are not contained in `star_data.inc` and associated files.

4.3 User-specified inlist controls

There is one set of controls that will prove useful time and time again when using `run_star_extras.f90`, and these are `x_ctrl`, `x_integer_ctrl`, and `x_logical_ctrl`. These are arrays (of length 100 by default) of double precision, integer, and boolean values, respectively. You can set the elements in your inlists

```
&controls
  x_ctrl(1) = 3.14
  x_ctrl(2) = 2.78
  x_integer_ctrl(1) = 42
  x_logical_ctrl(1) = .true.
/ ! end of controls inlist
```

and access them later on as part of the star structure (i.e., `s% x_ctrl(1)`, etc.). With these controls, you can specify parameters in your inlists instead of hard-coding them in `run_star_extras.f90`.

4.4 Physical constants (and const_def.f90)

As we already saw, MESA defines its constants in `MESA_DIR/const/public/const_def.f90`. MESA uses cgs units unless otherwise specified. The most common non-cgs units are solar units. Since the `run_star_extras` module includes the line

```
use const_def
```

you can access these definitions within `run_star_extras`. Using these built-in constants ensures that you are using exactly the same definitions as used throughout MESA.

4.5 Math functions

In `run_star_extras.f90`, you can readily access mathematical functions, such as `safe_log10` and `sinh`, found within `$MESA_DIR/math/public/math_lib_crmath.f90`, thanks to the line

```
use math_lib
```

4.6 Flow of control

MESA's flow of control is described by Figure 1. Here you will note that subroutines in your updated `run_star_extras.f90` file correspond to blocks in the flowchart: `extras_controls`, `extras_startup`, `extras_start_step`, `extras_check_model`, `extras_finish_step` and `extras_after_evolve`. When you wish to insert additional tasks into MESA's execution, you will need to modify subroutines in the appropriate location. For example, an assignment made inside the `extras_start_step` subroutine will be executed *once per evolve loop*, whereas an assignment made inside `extras_check_model` will be made *once per solver iteration*. There can be many solver iterations per evolve loop, and one evolve loop corresponds to one time step.

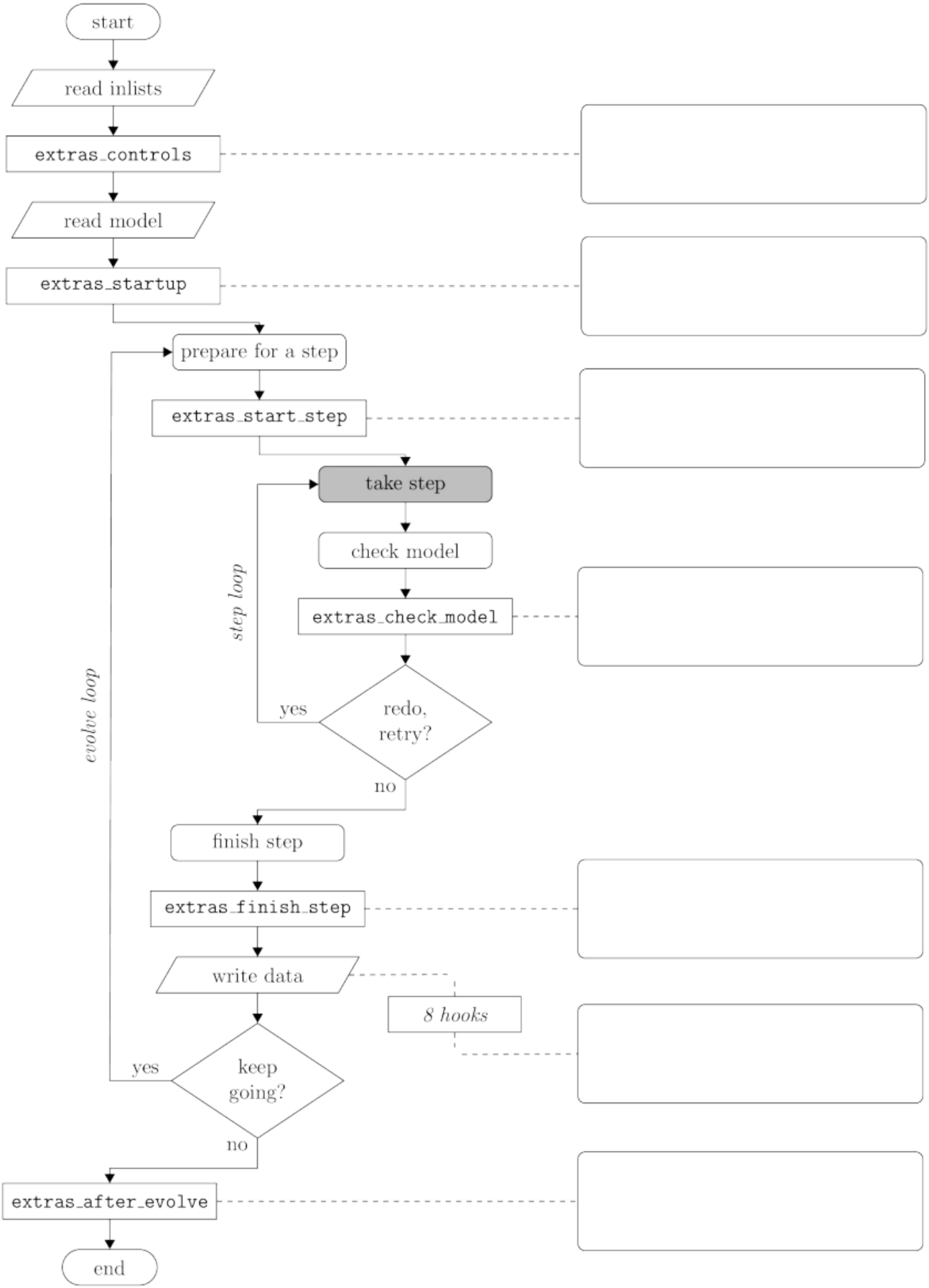

Figure 1: Flow of control for an evolutionary step in MESA.

4.7 Task (example): Adding a stopping condition

Suppose we are trying to fit observations of a star, and we want to stop our calculation when the stellar model reaches a certain region of the HR diagram. Although stopping conditions such as `Teff_upper_limit` exist within `controls.defaults`, they do not bracket a region in the HR diagram. How can we do this?

First, look at how the routines in `run_star_extras.f90` fit into a MESA run (Figure 1). To decide whether to stop, I want to check the effective temperature and luminosity of the stellar model after each evolutionary (e.g. time) step. Thus, I want the subroutine that is called after each time step, which is `extras_finish_step`.

Now, I need to figure out how to access information about the conditions at the stellar photosphere. I go to `star_data/public/` and start looking around. If I search for the word "photosphere," I can find what I'm looking for – `photosphere_L` and `Teff`.

Looking through `star_data_step_work.inc` shows that `photosphere_L` is in units of `Lsun`, which is defined in `$MESA_DIR/const/public/const_def.f90`. Note the many other constants that are defined in `const_def.f90`.

I want my stopping condition to be user-editable, so instead of hard-coding a value, I will specify `x_ctrl` in my inlist, within the `controls` section,

```
! measured logTeff and uncertainty in logTeff
x_ctrl(1) = 3.7d0
x_ctrl(2) = 0.01d0

! measured logL and uncertainty in logL
x_ctrl(3) = 0.85d0
x_ctrl(4) = 0.01d0
```

and then access this value in my code. While we're editing the inlist, comment out the H-depletion stopping condition, as this may conflict with our desired stopping condition. Also make sure to set $\alpha_{\rm MLT}$ back to its default value – otherwise our new stopping condition won't work.

Now, edit the `extras_finish_step` function within `run_star_extras.f90` so that the code understands the new `x_ctrl` parameters we've introduced. We assign these in terms of `star_info`, or `s`, attributes:

```
      ! returns either keep_going or terminate.
      ! note: cannot request retry; extras_check_model can do that.
      integer function extras_finish_step(id)
         integer, intent(in) :: id
         integer :: ierr
         type (star_info), pointer :: s
         !
         real(dp) :: logTeff_lo, logTeff_hi
         real(dp) :: logL_lo, logL_hi
         real(dp) :: logTeff, logL
         !
         ierr = 0
         call star_ptr(id, s, ierr)
         if (ierr /= 0) return
         extras_finish_step = keep_going

         ! upper and lower limits of logTeff
         logTeff_lo = s% x_ctrl(1) - s% x_ctrl(2)
         logTeff_hi = s% x_ctrl(1) + s% x_ctrl(2)

         ! upper and lower limits of logL
         logL_lo = s% x_ctrl(3) - s% x_ctrl(4)
         logL_hi = s% x_ctrl(3) + s% x_ctrl(4)

         ! actual logTeff and logL of model
         logTeff = safe_log10(s% Teff)
         logL = safe_log10(s% photosphere_L)

         write(*,*) 'logTeff and logL are', logTeff, logL
```

```
         if ((logTeff <= logTeff_hi) .and. (logTeff >= logTeff_lo) &
              .and. (logL <= logL_hi) .and. (logL >= logL_lo) ) then
            write(*,*) 'The MESA model has reached the bracketed region'
            extras_finish_step = terminate
         end if

         ! to save a profile,
            ! s% need_to_save_profiles_now = .true.
         ! to update the star log,
            ! s% need_to_update_history_now = .true.

         ! see extras_check_model for information about custom termination codes
         ! by default, indicate where (in the code) MESA terminated
         if (extras_finish_step == terminate) s% termination_code = t_extras_finish_step
      end function extras_finish_step
```

Now, recompile your working directory

```
./mk
```

You will need to do this step each and every time you edit `run_star_extras.f90`! You will not need to do this when you edit only inlist files.

We can now start the model again from the beginning

```
./rn
```

This run should halt around step 401.

Note that, though not necessary for this exercise, we could limit the time step via controls such as `delta_lgTeff_limit` and `delta_lgL_limit`, such that `MESA` doesn't take steps that are too big for the bracketed region.

4.7.1 Compilation errors

Sometimes the compilation will fail with for example the following message:

```
Symbol 'logteff_lo' at (1) has no IMPLICIT type
```

In this case it is because I forget to declare the data type of the variable `logTeff_lo`, i.e. the following line near the top of the function is missing:

```
real(dp) :: logTeff_lo
```

Other common compilation errors are those indicating that MESA cannot locate `makefile_header` or some other component of the code. 99% of the time, these occur because an environment variable has not been properly specified, or there is an error with file paths. To check, you can compare the output of the `pwd` command to the output of `echo $MESA_DIR`.

5 Maxilab 2: Changing input physics

This section will cover:

- overriding MESA's built-in routines
- replacing a static variable with an adaptive variable

MESA provides hooks to override or modify many of its built-in routines. These routines mostly affect things that occur within "take step" box of the flowchart (Figure 1), referred to as "other" routines. There are two main steps needed to take advantage of this functionality: (1) writing your own "other" routine; and (2) instructing MESA to use your routine instead of its own.

First, navigate to `$MESA_DIR/star/other`, where you will see a set of files named with the pattern `other_*.f90`. In general, find the one corresponding to the physics (or numerics) that you want to alter. Open one up and read through it. Many of the files contain comments and examples.

Note that we do not want to directly edit these files. Instead we want to copy the template routine into our copy of `run_star_extras.f90` and then modify it there. The template routines are usually named either `null_other_*` or `default_other_*`.

In this example, we will focus on `other_alpha_mlt.f90`. Open up this file. Copy the subroutine `default_other_alpha_mlt` and paste it into your `run_star_extras.f90`. It should be at the same "level" as the other subroutines in that file (that is, contained within the `run_star_extras` module.).

```
subroutine default_other_alpha_mlt(id, ierr)
    integer, intent(in) :: id
    integer, intent(out) :: ierr
    type (star_info), pointer :: s
    integer :: k
    ierr = 0
    call star_ptr(id, s, ierr)
    if (ierr /= 0) return
    s% alpha_mlt(:) = s% mixing_length_alpha
end subroutine default_other_alpha_mlt
```

Note that this routine already does something; `default_other_alpha_mlt` sets the mixing length parameter within the star uniformly to the `mixing_length_alpha` set in the `controls` section of the inlist. Rename the subroutine to `lecture1_other_alpha_mlt`, and replace the line

```
s% alpha_mlt(:) = s% mixing_length_alpha
```

with

```
do k = 1, s% nz
    s% alpha_mlt(k) = s% mixing_length_alpha
end do
```

In Fortran, you can write expressions that operate on the whole array at once, e.g.
`s% alpha_mlt(:) = s% mixing_length_alpha`.
However, it is often simplest to explicitly set the value of `alpha_mlt` (or some other array) one value at a time, by using a loop.

While we're looking code with a loop, it is a good time to mention that in MESA, the outermost zone is at `k=1` and the innermost zone is at `k=s% nz`. Arrays may also have a size in excess of `nz`, but only elements `1` through `nz` are meaningful.

5.1 Task: changing the mixing length parameter

Use the `other_alpha_mlt` routine to change the mixing length parameter from a fixed, static value adopted at every layer of the model to the depth-dependent function:

$$\alpha_{\rm MLT} = \left(\frac{\alpha_2 - \alpha_1}{2}\right) \tanh\left(\frac{M_r - M_0}{\Delta M}\right) + \left(\frac{\alpha_2 + \alpha_1}{2}\right) \tag{1}$$

where M_r is the enclosed mass, $M_0 = 1\,M_\odot$, $\Delta M = 0.1\,M_\odot$, $\alpha_1 = 1.5$, and $\alpha_2 = 2$.

The lower left panel in the `pgstar` plots displays the value of `s% alpha_mlt`, so you should be able to easily check if it looks OK. $\alpha_{\rm MLT}$ should asymptotically approach 1.5 in the core and reach 2 at the surface.

You can receive valuable MESA bonus points if your routine allows for user-specified values of ΔM and M_0, α_1 and α_2.

HINT 1: After you locate within `star_data` the variable name for the enclosed mass, check its units. You may need to do a unit conversion (or not).
HINT 2: Read the top of `other_alpha_mlt.f90`. It contains instructions on how to activate `other_alpha_mlt`.

Click here for solutions

6 BONUS maxilab exercise: adding an extra profile column

Using what we have learned about modifying `run_star_extras`, add an additional `profile_columns` output named `alpha_mlt`.

HINT 1: identify the subroutine that controls profile output
HINT 2: note that you will have to change the number of extra profile columns—which is zero by default—as well as collecting the `alpha_mlt` quantity. This will involve touching two subroutines: `how_many_extra_profile_columns` and `data_for_extra_profile_columns`.

For time step X of your choice, plot `alpha_mlt` as a function of mass from the associated `profileX.data`.

Click here for solutions

7 Solutions

7.1 Minilab 2 Solutions

You can either simply rename the first directory to avoid over-writing it:

```
mv LOGS LOGS_default
```

and/or add the following line in `controls` when running with the updated $\alpha_{\rm MLT}$:

```
log_directory = 'LOGS_1.5'
```

The tracks should look similar to Figure 2.

Click here to go back to Minilab 2

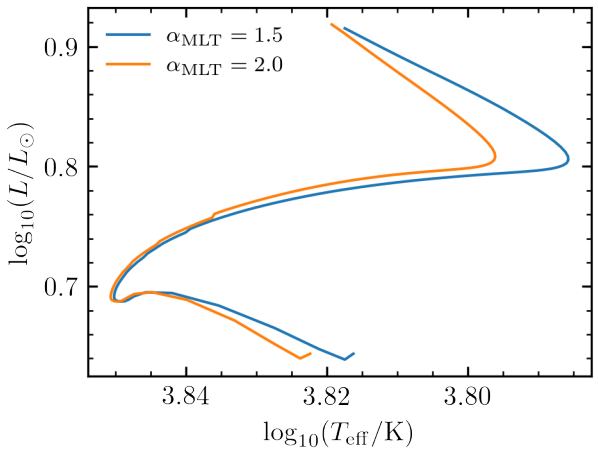

Figure 2: The difference between two $1.5 M_\odot$ stellar tracks with different values for $\alpha_{\rm MLT}$.

7.2 Maxilab 2 Solutions

The following is our new `other_alpha_mlt` subroutine

```
subroutine lecture1_other_alpha_mlt(id, ierr)
    integer, intent(in) :: id
    integer, intent(out) :: ierr
    type (star_info), pointer :: s
    integer :: k
    real(dp) :: delta_M, M0
    real(dp) :: c1, c2, a1, a2
    ierr = 0
    call star_ptr(id, s, ierr)
    if (ierr /= 0) return

    a1 = 1.5d0
    a2 = 2d0

    c1 = 0.5d0*(a2-a1)
    c2 = 0.5d0*(a2+a1)

    delta_M = 0.1d0
    M0 = 1.0d0

    do k = 1, s% nz
        s% alpha_mlt(k) = c1*tanh((s% m(k)/Msun - M0)/delta_M) + c2
    end do

end subroutine lecture1_other_alpha_mlt
```

You will also need to add the line

```
s% other_alpha_mlt => lecture1_other_alpha_mlt
```

in the `extras_controls` subroutine of your `run_star_extras.f90`, and set

```
use_other_alpha_mlt = .true.
```

in the `controls` section of your inlist.

Click here to go back to Maxilab 2

7.3 Bonus Solutions

Add the following lines to the `data_for_extra_profile_columns` subroutine:

```
            names(1) = 'alpha_mlt'
            do k = 1, nz
              vals(k,1)  = s% alpha_mlt(k)
            end do
```

Change the number of extra profile columns by setting

```
how_many_extra_profile_columns = 1
```

in `how_many_extra_profile_columns`.

Click here to go back to bonus maxilab