A Brief guide to MESA and analyzing MESA output for evolving Helium stars

Table of Contents

SECTION 1

Some Background Knowledge

In order to quickly catch up to speed with some of the necessary background knowledge for this project, some useful resources are outlined below:

1.1 Background on Stellar Evolution

In order to get familiar with some of the commonly used terms you will encounter, the first five chapters of a course on stellar evolution, courtesy of Utrecht University: `https://www.astro.ru.nl/~onnop/education/stev_utrecht_notes/`, are particularly useful. However, you can also augment your background knowledge with a number of courses that are free of charge on Coursera, with some crash courses on introductory astrophysics on YouTube, and with some online articles. I found WikiPedia to be particularly useful and actively Google'd most things I did not understand. Of course, you can also always consult a good book.

1.2 Background on Fortran and why it could be useful

MESA is written in Fortran, therefore in order to better understand and manipulate files that MESA works with such as the project configuration files "inlist" and "inlist_project", it could be useful to cover an introductory course on Fortran. The Fortran files themselves can be manipulated using your favorite scripting language such as bash or python when you are working with MESA, but of course, you would probably need to understand what you are doing with said files. It is unlikely that you will require to write any Fortran code yourself (unless you choose to), therefore you can cover the very basics of the language. A tutorial I found particularly useful on YouTube you can use:

https://www.youtube.com/watch?v=__2UgFNYgf8

1.3 Background on Python and some Python libraries for data analysis

Manipulating the output data that MESA produces is most easily achieved, in my opinion, with the help of Python. It would also be a good choice because of the rich set of libraries that it can be augmented by for analyzing data, plotting graphs, etc. To get familiar with Python 3+ you can refer to this video tutorial: https://www.youtube.com/watch?v=__2UgFNYgf8. The most useful Python libraries you could use, Matplotlib and NumPy, are also available as tutorials from the same tutor. Matplotlib: https://www.youtube.com/watch?v=wB9C0Mz9gSo, NumPy: https://www.youtube.com/watch?v=8Y0qQEh7dJg , 2

SECTION

Getting Things Running

2.1 Installing MESA and the MESA SDK

The instructions from the official guide that should be your go-to for installing MESA are available here: `https://docs.mesastar.org/en/latest/installation.html`. The instructions for installing the MESA SDK: `https://docs.mesastar.org/en/latest/installation.html`. Assuming you are on a unix-like machine and it is taking you a bit of time to install these two components, I hope that you find the following tips useful.

Your first step should be to consult the MESA SDK website and to ensure that you have installed every package required for the MESA SDK to run that is listed in the "Prerequisites" section for your linux distribution. After you have ensured that you have all the required packages ready to go, download the archive file in the "Download" section. Extract the file and execute the commands line by line from the "Installation" section.

Your MESA SDK should now be ready, which means your next step should be to download MESA itself. Download the MESA archive from the link in the "Download MESA" section. Check the number of threads your computer supports by running *lscpu*, this will be useful for the next step. You should now edit your *.bashrc* file. Modify the example code provided in the "Set your environment variables" section based on your machine and the output of *lscpu* and append it to the *.bashrc* file.

To check your installation has successfully completed, firstly check your environment variables by running *echo $MESASDK_ROOT* and *echo $MESA_DIR* in a new terminal to ensure they are set. Next, run *cd $MESA_DIR*, followed by *./clean* and *./install*. You should be greeted with a message notifying you the installation has successfully completed.

2.2 Installing Python3+, Matplotlib, NumPy, mesa_reader

You can install Python3+ using your linux package manager to obtain python3, for example, with the command *sudo apt-get install python3* on Ubuntu 16.04+ systems. In order to get Matplotlib, NumPy, and mesa_reader, you can run *sudo apt-get install pip3* followed by *pip3 install matplotlib*, *pip3 install numpy*, and *pip3 install mesa_reader*.

2.3 Your first MESA runs

Carefully following the guide sections "Running MESA" available on `https://docs.mesastar.org/en/latest/using_mesa` is your starting point to understand how MESA works and in order to evolve your first 15MSun star! You should also carefully read the section on MESA outputs to understand where run data is stores, since you will use it to examine the results of your experiments. If you are looking for more guidance getting MESA to run and/or if you are looking to get more familiar with MESA, you can also consult the YouTube channel `https://www.youtube.com/channel/UCFY7K6TqhL-opQ1h-l-RYhA`, which I have personally used.

SECTION 3

Skeleton Scripts

In this section I provide you with what I hope you will find useful skeleton scripts that you can utilize to get your experiments up and running and to collect your data faster. Assuming you are running everything smoothly (MESA, Python3, NumPy, Matplotlib, mesa_reader) on your target machine, you are good to go.

3.1 Scripting multiple runs

Lets consider the case in which you would, for example, like to script several runs to collect MESA model and data files for stars with various starting masses and metallicities as they evolve up to the main sequence in order to speed up your runs in the future. A new premain sequence MESA run is required for different starting masses and/or metallicities, however by storing the model files for various starting masses and metallicities, we can simply load them and resume runs from that point onwards into the main sequence in future experiments with the desired configurations for other parameters. You can use the python3 snippet below as your starting guide. It is easily generalized.

```
#in order to make system calls, use os
import os
#to ensure mutual exclusion and to ensure that there are no concurrency issues
#use subprocess
import subprocess

if __name__=="__main__":
```

```
#to open the inlist project file for configurations that we will
#consistently update with our script
inlist_project_file = open("inlist_project", "r")
#read all the lines in the file
#we will write the lines with the updated configurations back
inlist_project_lines = inlist_project_file.readlines()
inlist_project_file.close()

#iterating through a set of configurations for initial mass of a star
for m in [1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3.75, 4, 4.25,
          4.5, 4.75, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10]:
    #iterating through a set of metallicities
    for z in [0.1, 0.2, 0.5, 1, 2]:

        #removing floating point uncertainty, e.g. 3.00000003 instead of 3
        m = round(m, 3)
        z = round(z*0.014, 4)

        #iterating through every line of the project configuration files
        for i in range(len(inlist_project_lines)):

            #scanning for the line that specifies the filename of the
            #saved model and updating it accordingly
            if inlist_project_lines[i][4:23]=="save_model_filename":
                inlist_project_lines[i] = "    save_model_filename = "
                                          + str(m) + "M"
                                          + str(z) + "ZpreMS.mod" + "\n"

            #performing similar operations for initial mass, y, z.
            #you can see this can be generalized easily
            if inlist_project_lines[i][4:16]=="initial_mass":
                inlist_project_lines[i] = "    initial_mass = "
                                          + str(m) + "\n"

            if inlist_project_lines[i][4:13]=="initial_z":
                inlist_project_lines[i] = "    initial_z = "
                                          + str(z) + "\n"

            if inlist_project_lines[i][4:13]=="initial_y":
                inlist_project_lines[i] = "    initial_y = "
                                          + str(round(1.0-z, 4)) + "\n"

        #write the updated configuration project file
```

```
            inlist_project_file = open("inlist_project", "w")
            inlist_project_file.writelines(inlist_project_lines)
            inlist_project_file.close()

            #make a system call to run MESA with this updated project file
            command = "./rn"
            #wait on MESA to finish executing
            process = subprocess.Popen(command)
            process.wait()
            #print(process.returncode) - you can use this for debugging

            #writing the data after the run completes to a file
            # of your choice filename
            data_file_name = str(m) + "M" + str(z) + "ZpreMS.data"
            data_file = open(data_file_name, "w")

            #another system call to copy the history.data file
            #to your file, since it is where the data is stored
            command = "cp LOGS/history.data " + data_file_name
            os.system(command)
            data_file.close()
```

3.2 Script for data analysis

Now lets consider a skeleton script you can use to analyze data with python. Firstly, a brief deviation into mesa_reader: the set of all useful attributes collected during a run: *('model_number', 'num_zones', 'star_age', 'log_dt', 'star_mass', 'log_xmstar', 'log_abs_mdot', 'mass_conv_core', 'conv_mx1_top', 'conv_mx1_bot', 'conv_mx2_top', 'conv_mx2_bot', 'mx1_top', 'mx1_bot', 'mx2_top', 'mx2_bot', 'log_LH', 'log_LHe', 'log_LZ', 'log_Lnuc', 'pp', 'cno', 'tri_alfa', 'epsnuc_M_1', 'epsnuc_M_2', 'epsnuc_M_3', 'epsnuc_M_4', 'epsnuc_M_5', 'epsnuc_M_6', 'epsnuc_M_7', 'epsnuc_M_8', 'he_core_mass', 'c_core_mass', 'o_core_mass', 'si_core_mass', 'fe_core_mass', 'neutron_rich_core_mass', 'log_Teff', 'log_L', 'log_R', 'log_g', 'v_div_csound_surf', 'log_cntr_P', 'log_cntr_Rho', 'log_cntr_T', 'center_mu', 'center_ye', 'center_abar', 'center_h1', 'center_he4', 'center_c12', 'center_o16', 'surface_c12', 'surface_o16', 'total_mass_h1', 'total_mass_he4', 'num_retries', 'num_iter').* Where the set of attributes can be applied will be evident after examining the following snippet, which is also very easily generalized.

The example below plots final radii as a function of mass for Helium stars with various starting masses and metallicities. You can use it as a starting point and modify the files you read and what is plotted easily.

```
#importing the necessary libraries for data analysis
from numpy import *
from matplotlib import pylab
from pylab import *
import matplotlib.pyplot as plt
import mesa_reader as mr

'''
for LaTeX pgf plots in the future if needed

matplotlib.use("pgf")
matplotlib.rcParams.update({
    "pgf.texsystem": "pdflatex",
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})
'''

'''
#sans-serif fonts
plt.rcParams.update({
    "text.usetex": True,
    "font.family": "sans-serif",
    "font.sans-serif": ["Helvetica"]})

'''


#serif fonts:
plt.rcParams.update({
    "text.usetex": True,
    "font.family": "serif",
    #"font.serif": ["Palatino"],
})



#you can use a set of colors in a file for plotting 100s of datapoints/lines
#my file in this instance contained the hexadecimal representations of the
#colors I obtained from free tools online
#color_scales = open("color_scale.txt", "r").readlines()

i = 0
#the colors we will use to label points or lines
colors = ["red", "orange", "yellow", "green", "blue"]

#iterating through masses
```

```
for m in [1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3.75, 4, 4.25, 4.5,
        4.75, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10]:
    #iterating through metallicities
    for z in [0.1, 0.2, 0.5, 1, 2]:

    #eliminating floating point arithmetic uncertainty
    m = round(m, 6)
    a = round(2, 6)
    z = round(0.014*z, 6)

    #the data file we are to read as a string
    data_file_name = str(m) + "M" + str(z) + "Z" + str(a) + "AMS.data"
    #reading the data file with mesa_reader
    h = mr.MesaData(data_file_name)

    #you can examine any set of attributes as per the set specified above
    #you can also examine what other information is available by running
    #print(vars(h))
    R = 10**h.log_R
    log_R = h.log_R
    log_T = h.log_Teff
    log_L = h.log_L
    He = h.center_he4
    C = h.center_c12
    age = h.star_age

    #plotting and labelling the final radius as a function of mass
    if i<5:
        plot(m, log_R[-1], color=colors[i%5],
             label="metallicity = " + str(z), marker="+")
    else:
        plot(m, log_R[-1], color=colors[i%5], marker="+")
    i+=1

'''
    #this example specifies how you would plot Log of Teff
    #vs. Log of R with lines instead of points
        if i%5 == 0:
            plot(log_R, log_T, label=line_label,
                 color = color_scales[i].strip())
        else:
            plot(log_R, log_T, color = color_scales[i].strip())
'''
```

```
#to plot an individual point,
#perhaps one you are examining you can use the line below
#plot(math.log10(320), math.log10(6800),
#      marker="+", color="red", label="SN 2019yvr")

# set title and axis labels
plt.title("Final R vs. Mass for Stars With Starting \
           Mass in [1.75-10]MSun and Z in [0.1,0.2,0.5,1,2]ZSun")
plt.ylabel('Log of Radius (RSun)')
plt.xlabel('Mass (MSun)')

#you can configure your axis "ticks", or scales, using the lines below
plt.xticks([x for x in arange(1.0, 11.0, 0.25)])
#plt.yticks([x for x in arange(0, 400, 10)])

#to display a grid
plt.grid(color = 'grey', linestyle = '-', linewidth = 0.2)

# invert the x-axis (for HRDs, if needed)
#plt.gca().invert_xaxis()

#set the legend to be in the most optimal position
plt.legend(loc = 0)
#display the legend
plt.show()
```

SECTION 4

Using the Swinburne Supercomputing OzSTAR Facility

If you end up requiring to use the OzSTAR supercomputer, this section is intended to give you some tips on some of the tools I found most useful. One thing to note is that you will not be able to display pgstar or use X11. The official extensive OzSTAR documentation is available here: `https://supercomputing.swin.edu.au/docs/index.html`.

4.1 Installing/Loading MESA

To check if the version of MESA you require is already available, consult with your project supervisor or try to run *module spider mesa*. You may require to consult your project supervisor/admin to use it regardless.

If you need to install a newer version of MESA, the procedure would only require you to install MESA itself, as the SDK is already present. Send the tar archive over to your working directory (ensure you are not using your home directory, since it is limited in size) using the scp command from your machine using the command *scp -r /path_to_file/file.tar OzSTAR_username@ozstar.swin.edu.au:/path_to_directory*, and follow the installation instructions as you have priorly on your local machine.

4.2 Loading modules

One thing to note is that components such as Fortran and Python3 must be loaded prior to their use on the supercomputer. The syntax is *module load *modulename**. For example, to load python3 you can run *module load python* to display a list of available options followed by say *module load python/3.8.5*. To load fortran, you can run *module load ifort*, followed by say *module load ifort/2018.1.163-gcc-6.4.0*.

4.3 Slurm, its operations, and a skeleton script

If you would like to run a modestly sized/big job on the supercomputer, you should run it through the slurm utility. In order to do this, you should provide a bash script akin to the following:

```
#!/bin/bash
#
#SBATCH --job-name=res_tests_extended
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --time=100:00:00
#SBATCH --mem=16G
module load python/3.8.5
export OMP_NUM_THREADS=32
python3 res_tests_extended.py
```

If you are unfamiliar with bash, you can note in the above that you can load any number of modules you require to use in sequence, you can execute your script in your favorite language, and you can vary the parameters of the job easily. In order to schedule this job with slurm, suppose the bash file is saved under the filename *res_tests_extended.sh*. We execute it by running *sbatch res_tests_extended.sh*. In order to monitor your jobs and to see if you are currently in the queue, you can run *squeue | grep "*OzSTAR_username*"*. To cancel a job, you can run *scancel *job-number**. Job numbers can be obtained using the squeue command. Output you are used to seeing in your terminal will be available in the directory from which you executed the script under **job-number*.out*. This summarizes most of the utilities you would need to use OzSTAR, however you can consult the official documentatino for more information.