tulips

Eva Laplace

Oct 19, 2021

CONTENTS

1 Documentation 3

2 Tulips diagrams 53

3 Additional options 55

4 Indices and tables 57

5 Contributing 59

6 Acknowledgments 61

Python Module Index 63

Index 65

TULIPS

Tool for Understanding the Lives, Interiors, and Physics of Stars

This is an alpha version and is not meant for use by external parties.

CHAPTER

ONE

DOCUMENTATION

Welcome to the `tulips` documentation! The `tulips` python package creates visualizations of stars based on output from the MESA stellar evolution code. TULIPS represents stars as circles with varying size and color. Click the links below to learn more about TULIPS through examples, tutorials, and detailed documentation.

1.1 Installation

The easiest way to install `tulips` and all its dependencies is through pip:

```
pip install astro-tulips
```

Note: Python 2 is not supported

1.1.1 Installing from source

To install from source, git clone the directory, then install:

```
git clone https://bitbucket.org/elaplace/tulips.git
cd tulips
python3 setup.py install
```

The latest development version is available on the `development` branch. To create a local installation call:

```
pip install -e .
```

1.1.2 Additional dependencies

You need to have mesaPlot installed for reading MESA output files (see mesaPlot). `tulips` relies on the `ffmpeg` software. It is automatically installed through the `imageio_ffmpeg` package. You also need to have Latex installed to properly display the diagrams. The easiest way to do this is by installing TexLive.

Mencoder

The `mencoder` software may be needed on some platforms for creating the movies. To install `mencoder` on Windows, you can download MPlayer. On Ubuntu, you can install `mencoder` via:

```
sudo apt install mencoder
```

Warning: These additional dependencies are crucial. Without them, you will not be able to create TULIPS animations.

1.1.3 Running on Windows

When getting `tulips` installed on Windows, it is important to add the dependencies to your system environment variable 'Path'. You can do this by going to Start > Settings > Info > Advanced > Environment Variables. Here you can add a new variable to 'Path' in the System Variables. This variable should contain the path to the installed program.

Note: You can check if you installed the program (e.g. `mencoder`) with `<program_name> -version`. This should return the version of the program.

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.2 Getting started

Here, we will explain the basics that are needed to get started with `tulips`. `tulips` creates visualizations of stellar evolution based on output from the MESA stellar evolution code. To read MESA output files, `tulips` uses the mesaPlot package. Here we will show you how to load a MESA model and how you can explore its properties. We will use jupyter notebooks to show the tutorials. With this application you can edit and run python code via a web browser. Have a look here for information on how to download and use it.

1.2.1 Loading a MESA model

To show the functionalities of the `mesaPlot` output, we use a pre-computed MESA stellar model of an $11M_\odot$ star. In order to explore the properties of this model, we can create a `mesaPlot` object called `m11`, in which all the information about the stellar model will be stored. MESA produces two types of output files:

- `history.data` files that contain the evolution of one-dimensional properties of the stellar model, such as its radius.
- `profile.data` files that contain a snapshot of the interior stellar structure at a particular moment in time. For example, this file can contain the temperature as a function of the radius/mass coordinate

In the example below, we show how to load the output of a MESA history file with `mesaPlot`, after which we can access its properties.

[1]:
```
# Interactive matplotlib plotting for jupyter lab
%matplotlib inline
```

(continues on next page)

(continued from previous page)

```
# If you use jupyter notebook
# %matplotlib notebook

import matplotlib.pyplot as plt
import mesaPlot as mp

EXAMPLE_DIR = "../../tulips/MESA_DIR_EXAMPLE/"

m11 = mp.MESA() # Create MESA object
m11.loadHistory(filename_in=EXAMPLE_DIR + 'history.data')
```

Tip

To copy all lines of code in a cell, you can just click on the copy button in the upper right corner!

1.2.2 Accessing properties

After the history output file has been loaded, properties such as the mass, age or radius of the star can be accessed through the `hist` property.

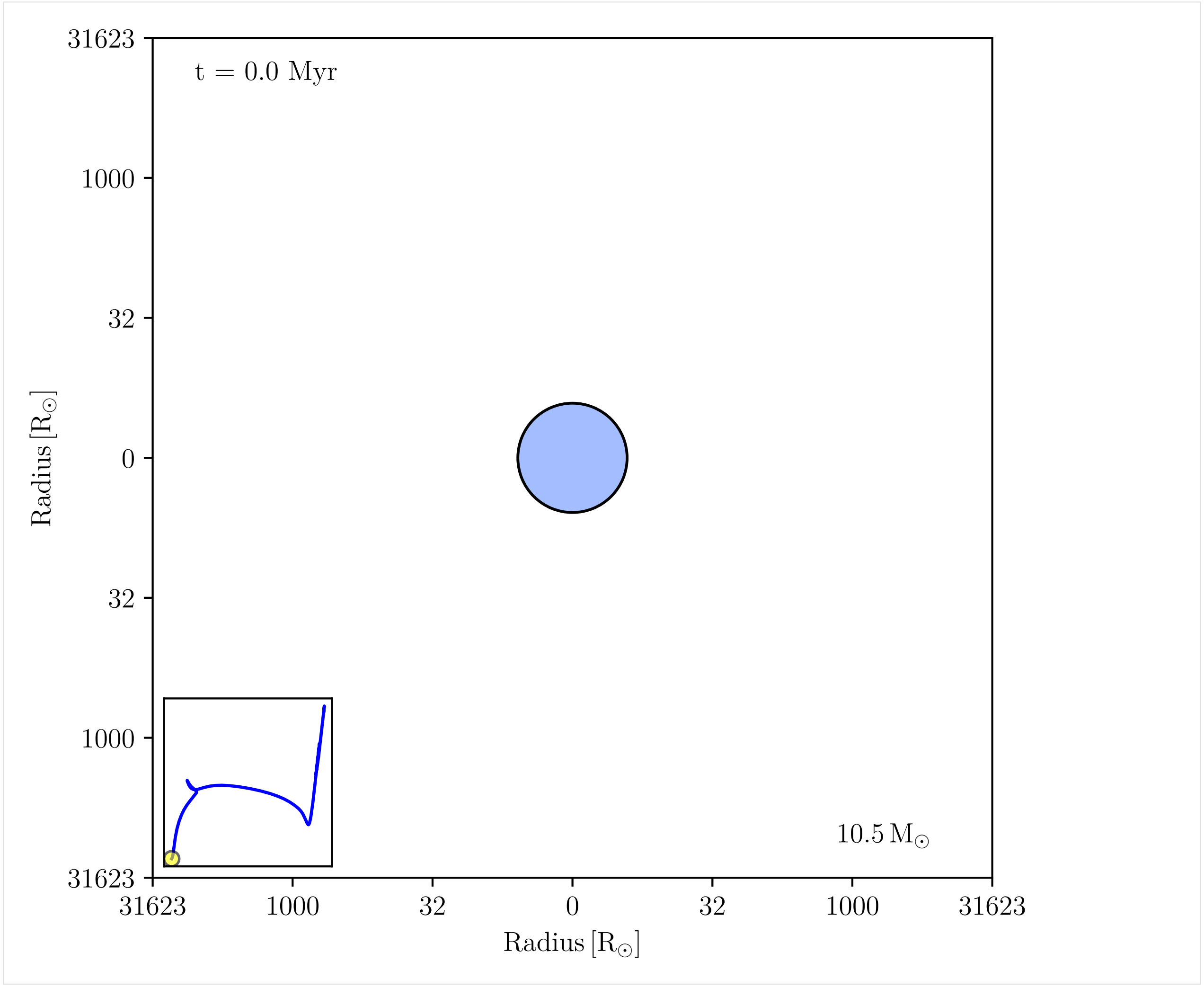
[3]:
```
star_age = m11.hist.star_age # Access star age
log_R = m11.hist.log_R # Access star log radius
```

In order to have a look at the different properties that are stored, type `m11.hist.` and press `Tab`. This will show you a list of the information that can be accessed. You can use these properties to make simple plots. For extensive documentation about `mesaPlot`, have a look at the mesaPlot GitHub page.

1.2.3 Creating your first tulips diagram

Now that we have explored the basic properties of our star, we are ready to visualize it using `tulips`! `tulips` is capable of generating different kinds of diagrams, which are shown at the *Homepage*. For demonstration, we will plot one of these diagrams: the perceived color diagram. This can be done using just one command, passing our MESA object `m11` to the `perceived_color` function from `tulips`. It will show us the perceived color and size of the star at a single point in time, specified by the time index `time_ind`. For now, we only want to plot at `time_ind = 0`.

[4]:
```
import tulips

tulips.perceived_color(m11, time_ind=0, fig_size=(8,6))
plt.show()
```



Here is your first `tulips` plot! We see a blue $10.5 M_\odot$ star. The x and y axis show the radius of the star. The color represents the temperature of the star in the same way we would observe it: blue is hot, red is cooler. On the bottom left we see an inset Herzsprung Russel diagram, showing that this star is just at the beginning of the main sequence. We can use this first diagram to verify if everything looks fine, before moving on to making an animation.

Note

`time_ind` refers to an index that follows every model stored in a MESA `history.data` file. This means the first model is at index 0 and the last is at index -1. It depends on your model which age this corresponds to. The powerful thing about `time_ind` is that it lets you find out some specific times very easily through numpy functions. For example, if you want to create a model of a star at the point when it finishes hydrogen burning, you can look for the model where the central hydrogen mass fraction is very small, like `end_hburn_index = np.where(m11.hist.center_h1 < 1e-4)[0][0]`.

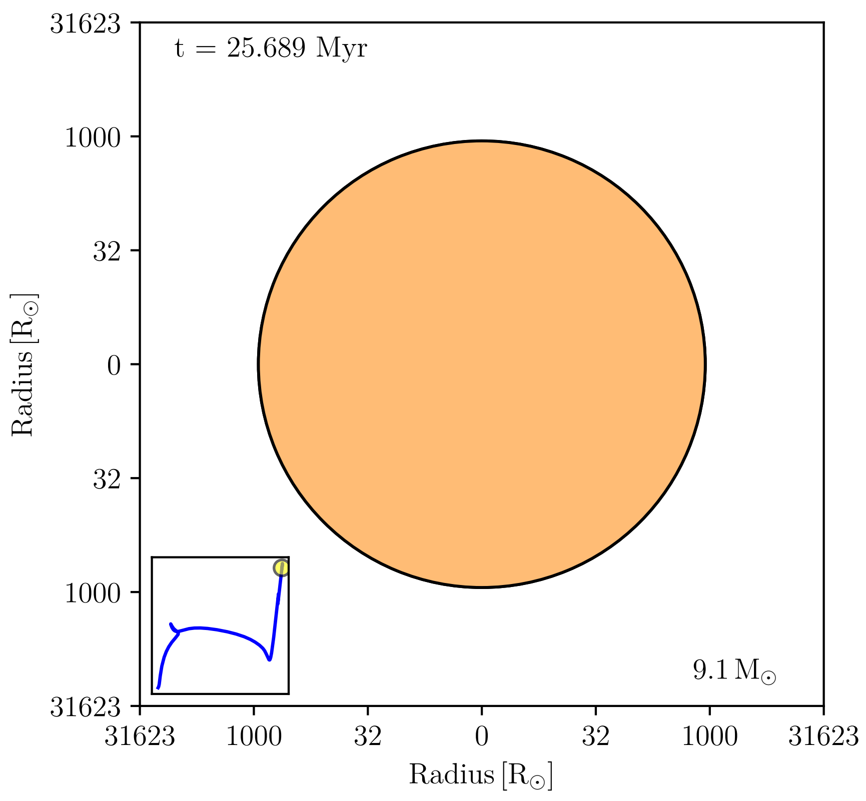
1.2.4 Creating your first tulips animation

Once you are happy with it, it would be interesting to see how the star is evolving and how that reflects in its appearance. Therefore, we can create an animation of the evolution of the star. To view the animation, we first have to save it. Afterwards we can open it with the `Video` function from the `IPython` library. It is not possible to see the animation without saving it. Let's create an animation of our stellar model:

```
[5]: tulips.perceived_color(m11, time_ind=(0, -1, 10), fps=30, output_fname="perceived-color-
     ↪test")
     plt.show()
```

```
  0%|          | 0/492 [00:00<?, ?it/s]
```

```
Creating animation
```



```
[6]: from IPython.display import Video

     Video("perceived-color-test.mp4", embed=True, width=700, height=600)
```

```
[6]: <IPython.core.display.Video object>
```

Now, `time_ind` contains the start index, the end index, and the time step. The `fps` keyword (which stands for "frames per second") helps control the speed of the resulting video. More information and all the options can be found in the function docstring and in the `tulips` API. The function docstring can be found by typing `tulips.perceived_color()` and press `Shift + Tab` inside the parenthesis. The animation is saved as .mp4 file in a directory, which you can specify with `output_fname`. You can then call the `Video` function with the specified name. We can now see the star changing over time in radius, color and mass while it follows its track on the Herzsprung Russel diagram.

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.3 Perceived color diagram

In this notebook we will demonstrate how to create one of the possible `tulips` diagrams. The perceived color diagram is useful when you want to display a property that applies for the entire star, like radius and color. `tulips` visualizes the star as a circle, where its radius represents the actual radius, or any other physical quantity that you're interested in. This circle is filled with a color, which indicates the temperature of the star, according to how it would appear on the sky. Therefore, the diagram is an approximation of how the star would be perceived by the human eye.

1.3.1 Load example model

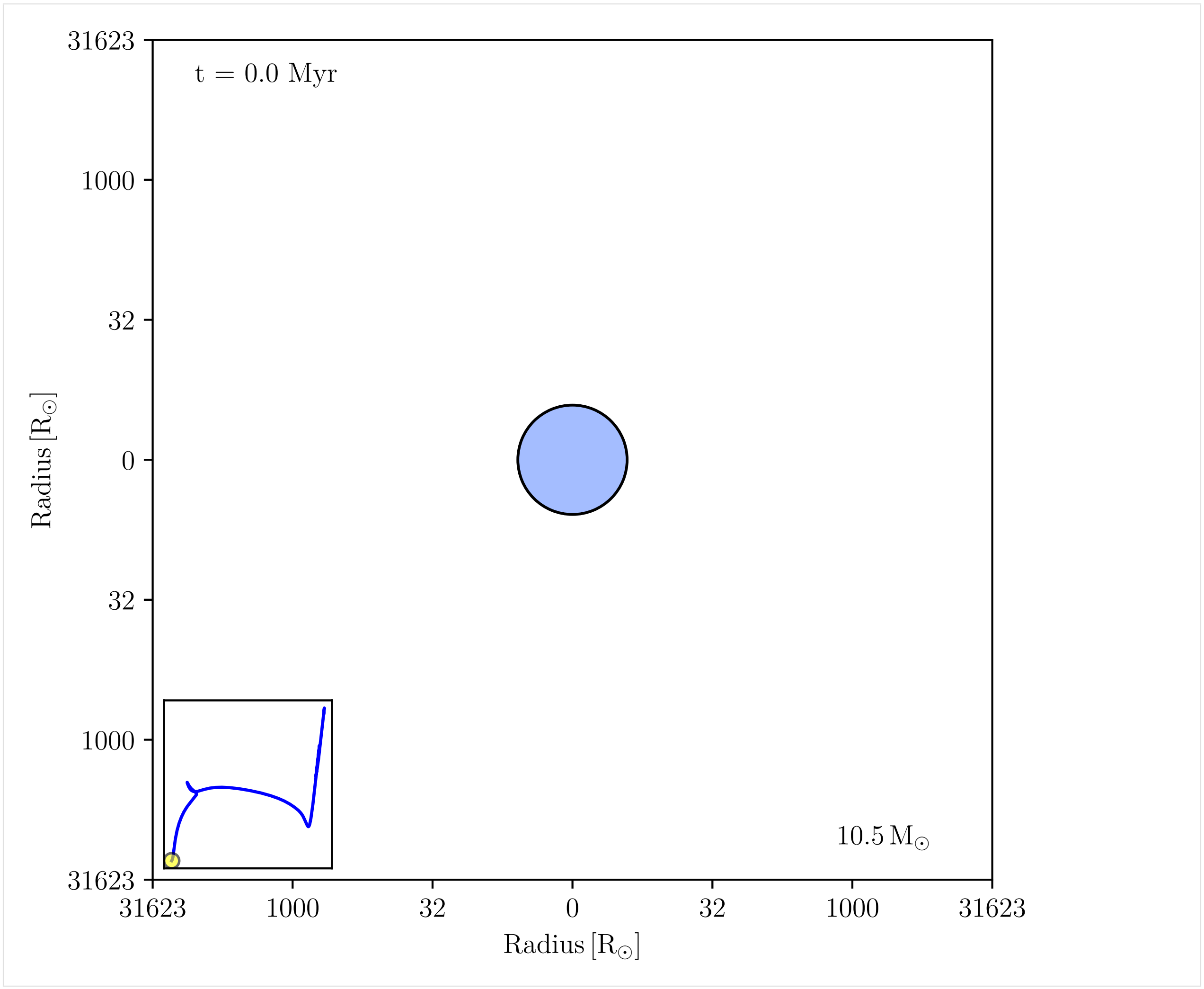
Let's plot a diagram of a $11 M_\odot$ star at the beginning of its evolution. As explained in the *Getting Started* page, we first need to load in a MESA model with `mesaPlot` and store it in an object called `m11`.

[1]:
```
# Interactive matplotlib plotting for jupyter lab
%matplotlib inline

# If you use jupyter notebook
# %matplotlib notebook

import matplotlib.pyplot as plt
import mesaPlot as mp
import tulips

# Specify  directory of MESA model
SINGLE_M11_DIR = "../../tulips/MESA_DIR_EXAMPLE/"

m11 = mp.MESA()
m11.loadHistory(filename_in=SINGLE_M11_DIR + "history.data")
```

1.3.2 Create perceived color plot

Now we are ready to create the perceived color diagram of this star. We can do this by passing the MESA object to the `perceived_color` function. Let's plot an instance of the star at `time_ind=0`. (Have a look at the *Getting Started* page on how to specify this time parameter)

[3]:
```
tulips.perceived_color(m11, time_ind=0, fig_size=(8, 6))
plt.show()
```



The radius of the star is shown on the x and y axis. Its blue color shows how it would be perceived by the human eye, and indicates a relatively hot star.

Note

The color is determined using the `colorpy` package, which approximates the spectrum of a star using its effective temperature. According to the 1931 color matching functions of the Commission Internationale de l'Eclairage, this spectrum is converted to the approximate RGB color the human eye will perceive.

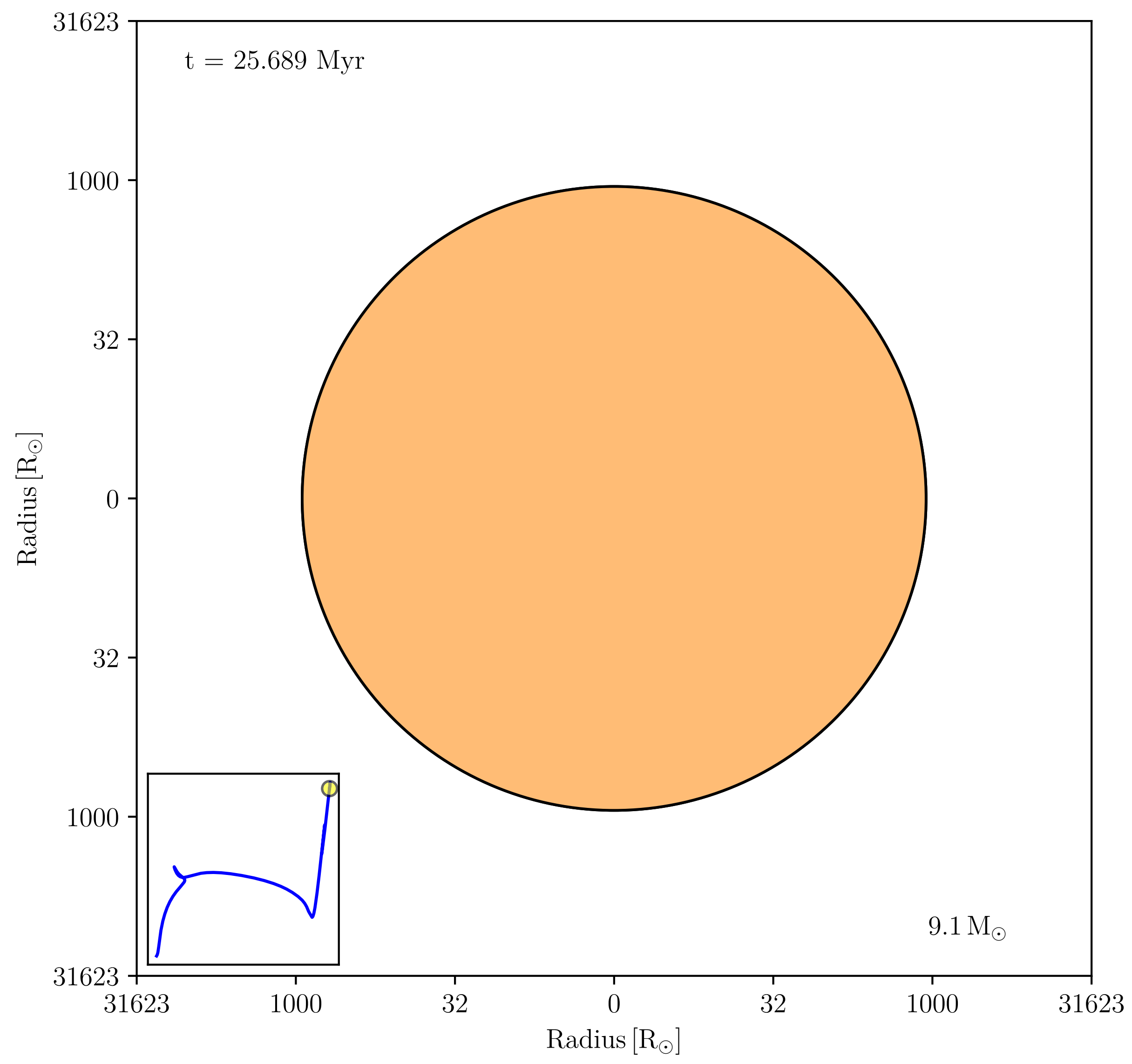
1.3.3 Create an animation

To see the star evolving, and observe its changing radius and color, we can also create an animation. To do this, we need to modify the `time_ind` property. Here, we want to plot from the first (0) to last (-1) time index, with steps of 10. For more information about the `time_ind` property, look at the *Getting Started* page.

[4]:
```
tulips.perceived_color(m11, time_ind=(0, -1, 10), fps=30, output_fname="perceived_color",
↪ fig_size=(8, 6))
plt.show()
```

```
  0%|          | 0/492 [00:00<?, ?it/s]
```

```
Creating animation
```



[5]:
```
from IPython.display import Video

Video("perceived_color.mp4", embed=True, width=700, height=600)
```

```
[5]: <IPython.core.display.Video object>
```

1.3.4 Showing other properties

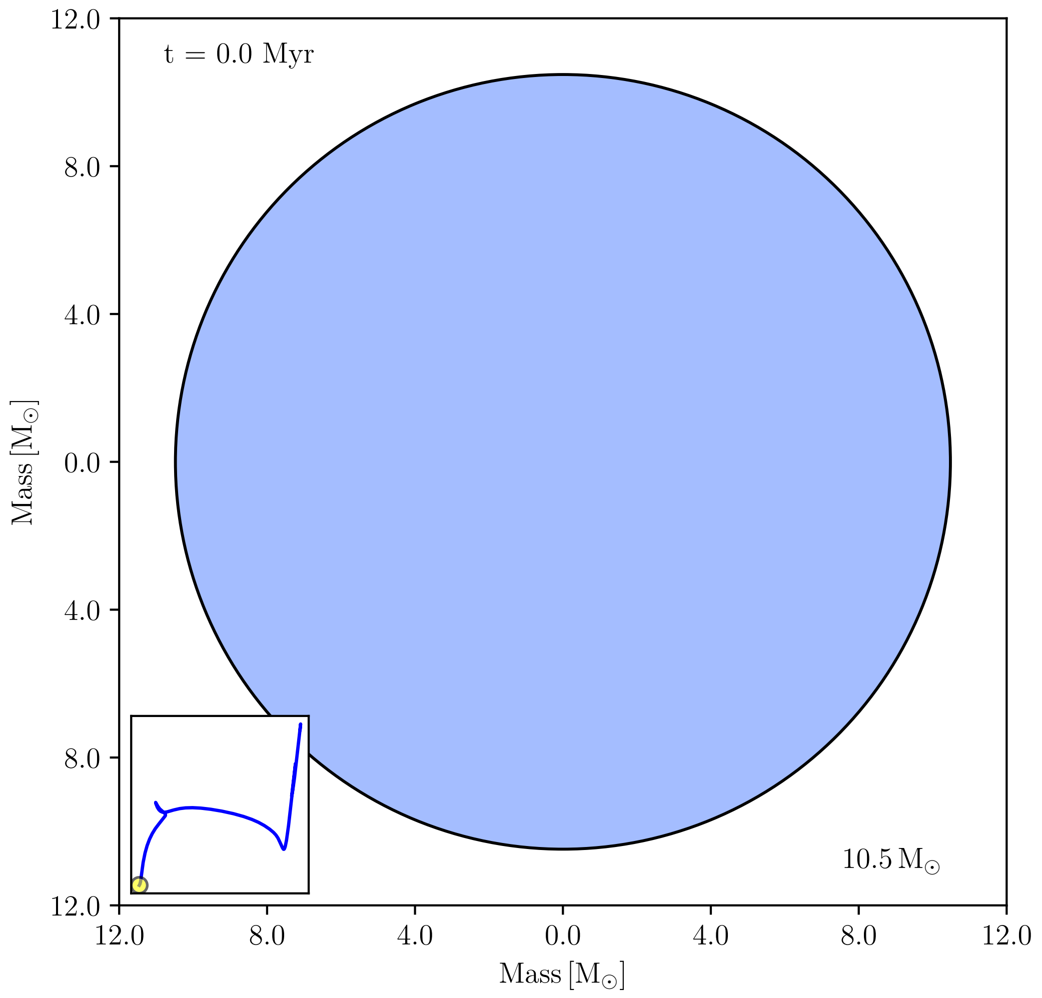
Next to showing the star's size, we can use this diagram to visualize other properties. It is possible to specify which quantity the radius is representing. This can be done by changing the `raxis` property to any other quantity that is stored in the MESA output file.

Tip

To access all the different properties, you can type `m11.hist.` and press `Tab`.

In the example below, we change this property to `star_mass`, to show the mass instead of radius on the x and y axis. A larger circle now represents a more massive star and vice versa. The color still shows the perceived color.

```
[6]: tulips.perceived_color(m11, time_ind=0, raxis='star_mass', fig_size=(8, 6))
     plt.show()
```



Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.4 Energy and mixing diagram

In this notebook we will discuss how to create one of the possible `tulips` diagrams. The energy and mixing diagrams gives an overview about the energy generation rates thoughout a the stellar model. Moreover, it provides information about the mixing processes occuring in the interior. In this diagram, `tulips` visualizes the stellar model as a circle, where its radius represents the total mass of the star.

1.4.1 Load example model

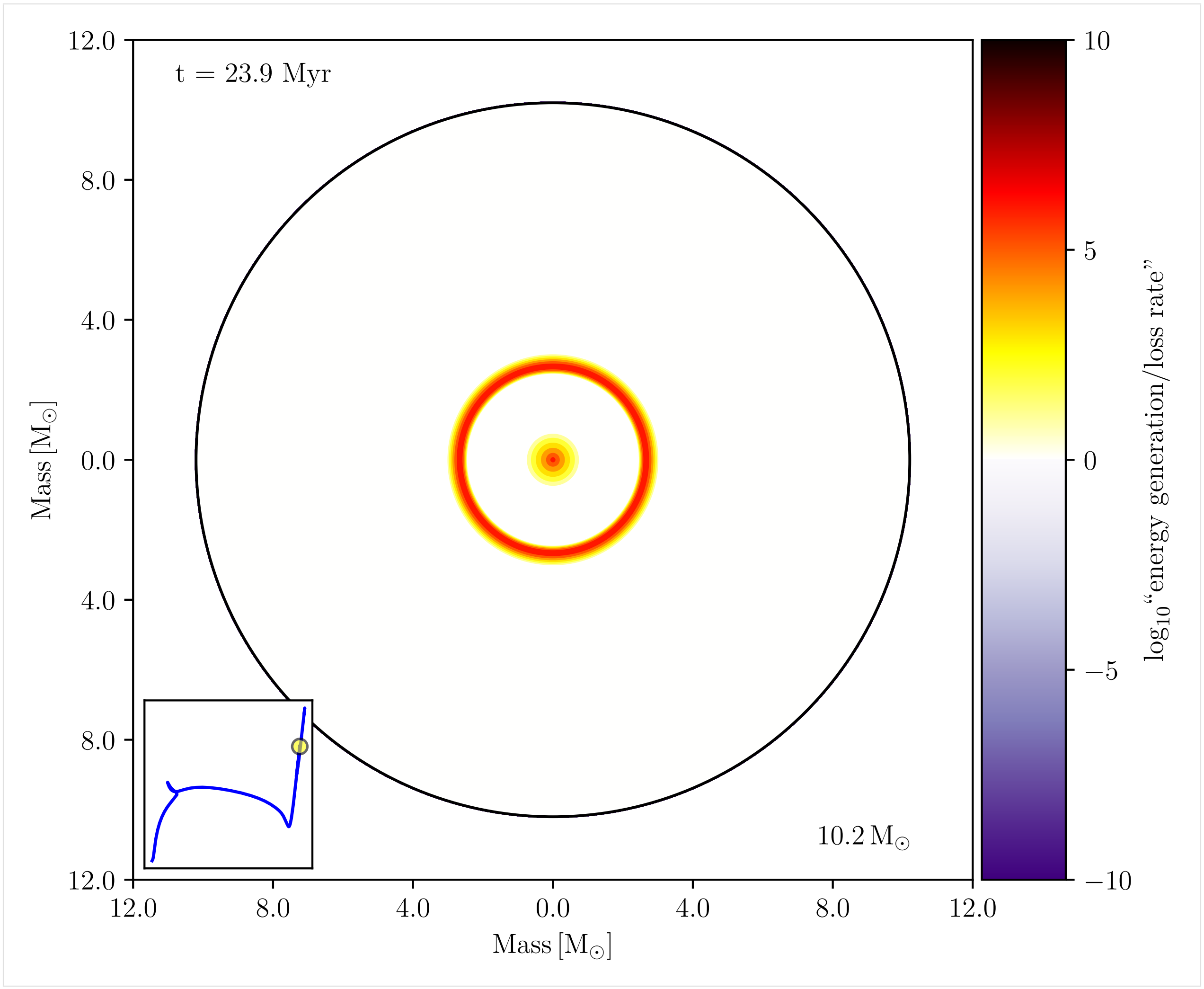
Let's plot an energy and mixing diagram of a $11 M_\odot$ star. As explained in the *Getting Started* page, we first need to load in the history output of a MESA model with `mesaPlot` and store it in an object called `m11`.

[1]:
```
# Interactive matplotlib plotting for jupyter lab
%matplotlib inline

# If you use jupyter notebook
# %matplotlib notebook

import matplotlib.pyplot as plt
import mesaPlot as mp
import tulips

# Specify directory of MESA model
SINGLE_M11_DIR = "../../tulips/MESA_DIR_EXAMPLE/"

m11 = mp.MESA()
m11.loadHistory(f=SINGLE_M11_DIR)
```

1.4.2 Create an energy diagram

Then we create an instance of the stellar model at `time_ind=2200` as follows:

[3]:
```
tulips.energy_and_mixing(m11, time_ind=2200, cmin=-10, cmax=10, show_total_mass=True,␣
↪fig_size=(8,6))
plt.show()
```



The stellar model is shown as a circle, whose radius represents the total mass (however, this can be changed by modifying the `raxis` property). The colors inside the star represents the energy generation rate. As indicated by the colorbar on the right, yellow to red colors are used for a positive energy generation rate, caused by nuclear burning processes. As we can see in the inset HR diagram, we are looking at a star in the later stages of its evolution. We can observe burning in a shell, as well as a burning region in its core. The colors can also show energy loss by neutrino emission, which are shown in purple. You can hide the colorbar by setting `show_colorbar` to `False`. To see how you can change the colormap, have a look at the *Customize options* page. The label of the colorbar can be changed with the `cbar_label` option. It is possible to put limits on the burning rates using the `cmin` and `cmax` property. In the example above we used the default values. However, if you are for example only interested in energy generation and not in cooling, you could change `cmin` to 0 to only include positive energy generation rates.

Note

The energy generation rate ϵ is calculated from the difference between nuclear burning rate and neutrino energy as follows:

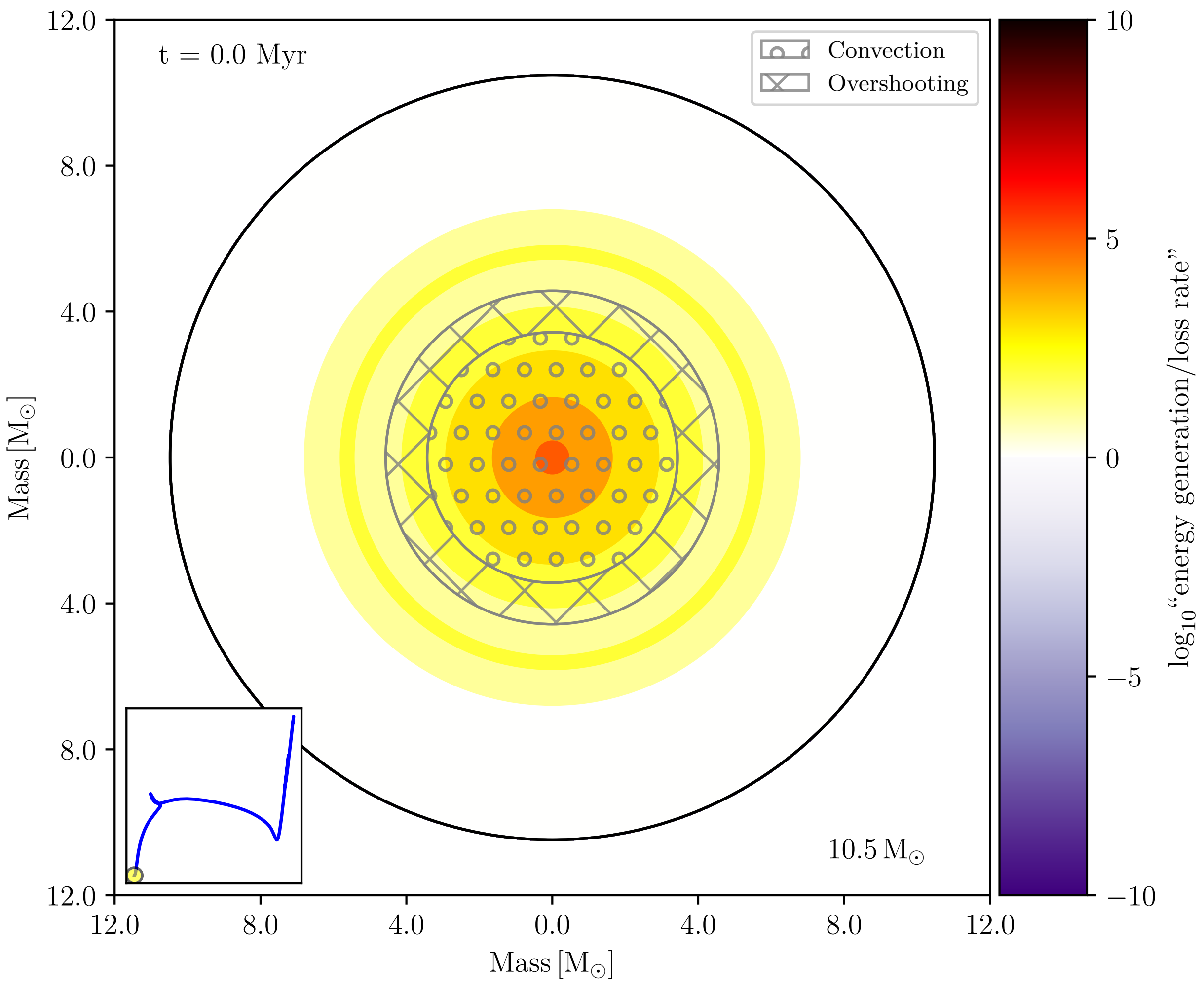
$\epsilon = \mathrm{sign}(\epsilon_{nuc} - \epsilon_{\nu})\log_{10}(\max(1.0, |\epsilon_{nuc} - \epsilon\nu|)/[\mathrm{erg\ g^{-1}s^{-1}}])$

Important here is to notice that this is a logarithmic quantity.

1.4.3 Create an energy and mixing diagram

Next to visualizing the regions where energy generation or losses occur, `tulips` can also show the dominant mixing process in the interior. We can add this information to the diagram by changing the `show_mix` and `show_mix_legend` variables to `True`:

```
[4]: tulips.energy_and_mixing(m11, time_ind=0, cmin=-10, cmax=10, show_total_mass=True,
                                  show_mix=True, show_mix_legend=True, fig_size=(8,6))
     plt.show()
```



In addition to the energy generation rates, we now also see areas with grey patterns indicating the mixing process. Which patterns correspond to the different types of mixing processes can be seen in the legend. The mixing hatches can be customized by changing the `tulips.MIX_HATCHES` variable. This stellar model has a convective core with overshooting at the edge.

1.4.4 Creating an animation

We can create an animation of our energy and mixing diagram by changing the `time_ind` property. You can specify the start index, end index and timestep. In the following example we choose to show the first (0) to last (-1) model, with steps of 10. `fps` (frames per second) controls the speed of the video.

```
[5]: tulips.energy_and_mixing(m11, time_ind=(0,-1, 10), fps=30, cmin=-10, cmax=10, fig_
     ↪size=(8,6)
                                  show_total_mass=True, show_mix=True, show_mix_legend=True,␣
     ↪output_fname="energy_and_mixing")
     plt.show()
```

```
  File "/tmp/ipykernel_307/3394379683.py", line 2
    show_total_mass=True, show_mix=True, show_mix_legend=True, output_fname="energy_and_
↪mixing")
                  ^
SyntaxError: invalid syntax
```

```
[6]: from IPython.display import Video

     Video("energy_and_mixing.mp4", embed=True, width=700, height=600)
```

```
[6]: <IPython.core.display.Video object>
```

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.5 Property profile diagram

In this notebook we demonstrate how to create a property profile diagram. The diagram can be used when we want to explore a certain property throughout the stellar interior, such as the density, at a given moment in time. It will show us the profile of that property of the star and helps to visualize how the property changes from the center to the surface of the stellar object. The property profile diagram is customizable and offers many options that we will demonstrate in this notebook. To generate these diagrams, `tulips` divides the star into rings and computes the physical property at the location of each ring. Each ring is assigned a color that represents the value of the property at this location within the star. In the following example we will show how to create a property diagram that represents the density of a $11M_\odot$ star.

1.5.1 Load example model

To create a `tulips` diagram, we first need to load in a MESA output model. We can load the output file in an object called `m11` as follows:

```
[5]: # Interactive matplotlib plotting for jupyter lab
     %matplotlib inline

     # If you use jupyter notebook
     # %matplotlib notebook

     import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import mesaPlot as mp
import tulips

# Specify directory of MESA model
SINGLE_M11_DIR = "../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS/"

m11 = mp.MESA()
m11.loadHistory(f=SINGLE_M11_DIR)
```

Warning

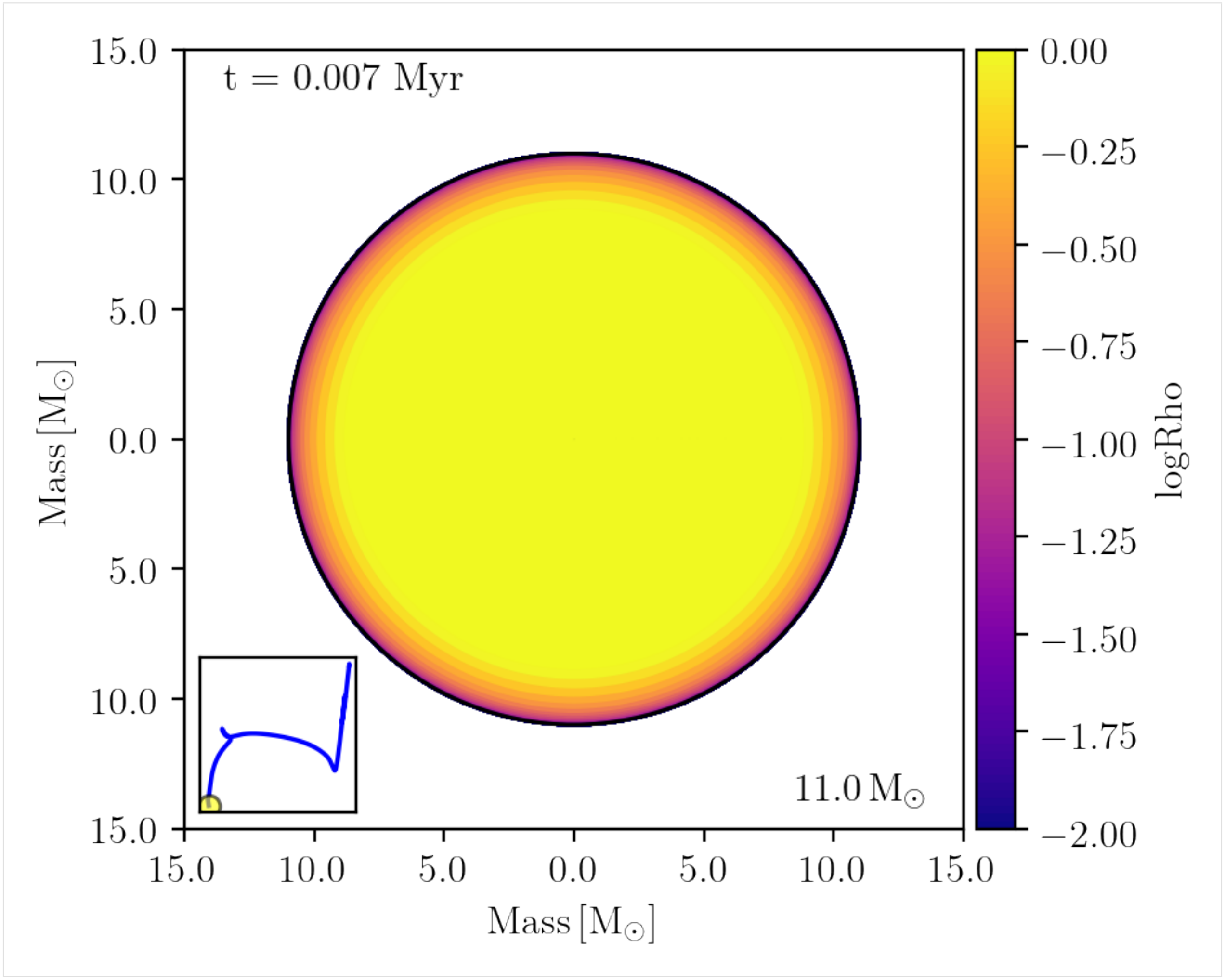
As explained in the *Getting Started* page, MESA produces two types of output: `history.data` and `profile.data` files. When making a property profile diagram, make sure that both of these types of files are stored in the MESA output folder.

1.5.2 Example 1: density

Let's try to plot the density of this stellar model in the diagram. To do this, we call the `tulips.property_profile` function, which requires our MESA object `m11` as the first argument. With the `raxis` keyword, we can specify what the radius of the circle represents. In this example the radius of the circle represents the mass of the star, but sometimes it might be useful to represent other quantities instead, such as the radius of the star. `property_name` indicates which physical property we want to show in the diagram, which in this case will be logarithm of the density. With `time_ind` we specify a time index. Here, we plot the first model. The time index is further explained in the *Getting Started* page.

```
[3]: tulips.property_profile(m11, time_ind=0, property_name='logRho', num_rings=100, cmin=-2,␣
     ↪cmax=0)
     plt.show()
```

```
../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS///profile1.data
```



The property diagram shows us a stellar model represented by a circle that is divided in `num_rings` layers. If `num_rings` is -1 (default), it shows the maximum amount of rings, but this will computationally be more heavy. Each layer has its own color, corresponding to a certain density, as indicated by the colorbar on the right. To see how you can change the colormap, have a look at the *Customize options* page. You can hide the colorbar with `show_colorbar=False` and you can specify a label with `cbar_label`. Here, we see that the density decreases when you move outward. With the `cmin` and `cmax` argument you can specify what range of values of the property you want to show. In this example, we showed log densities densities in the range of 10^{-2} to 1 $[\mathrm{g\,cm^{-3}}]$. By default, `cmin` and `cmax` correspond to the mininum and maximum value of that property in the complete `profile.data` file of the stellar object.

Note

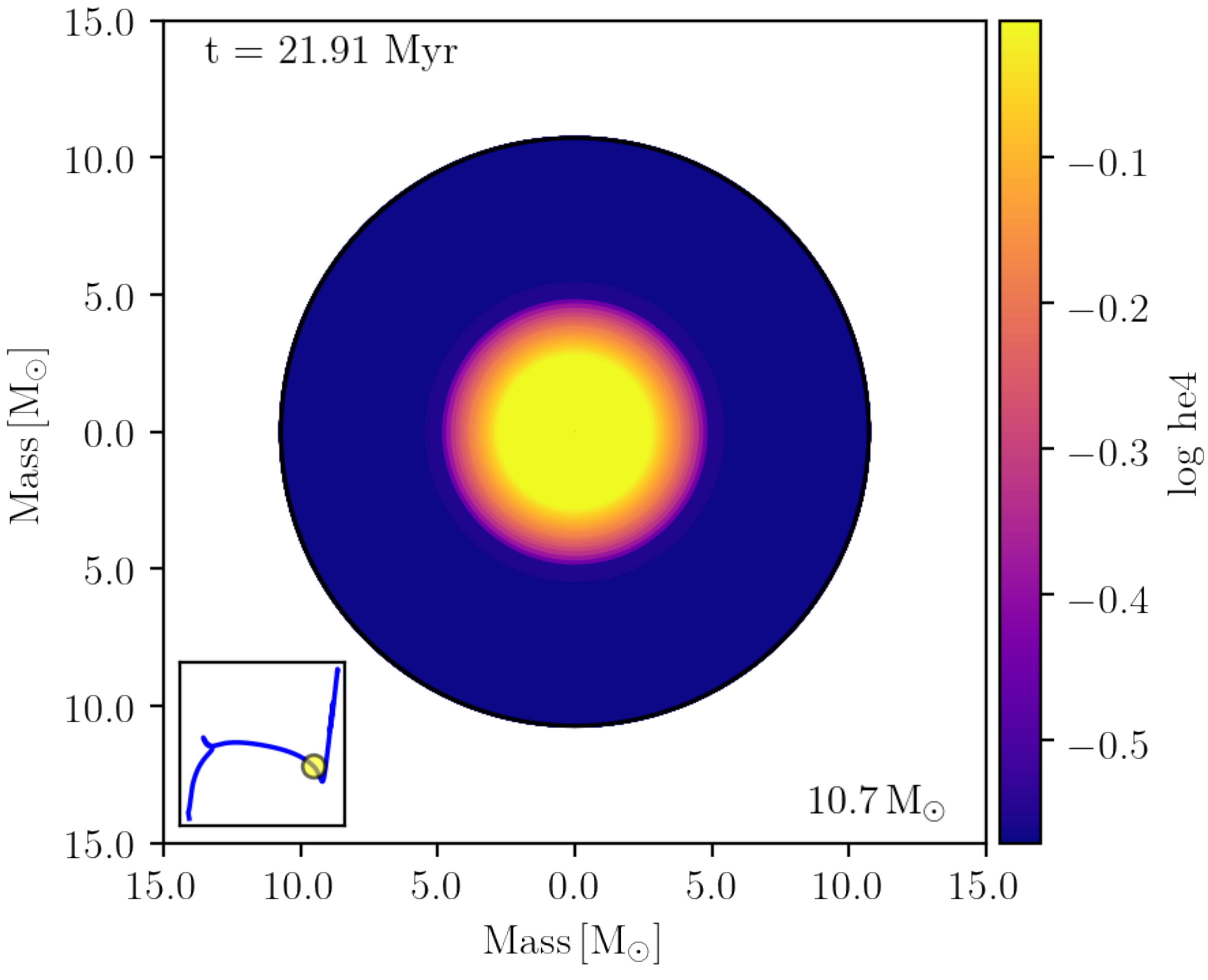
Make sure you are pointing to the correct folder that contains the profile data! MESA usually stores the `profiles.data` in a LOGS folder. If you get an error about profiles not being found, try `m11.log_fold='<your_directory>/LOGS'`. This will make sure that `tulips` can find the profile data.

1.5.3 Example 2: Helium-4

Not only the density, but many more properties of the star can be shown with this `tulips` diagram. Here, we will show the ^{4}He mass fraction throughout the stellar interior. If we want to see how this quantity changes from the center to the edge of the stellar object, we only have to change `property_name` to `'he4'`. If we then plot the diagram at a later time index, where the star is burning hydrogen in a shell, it looks as follows:

```
[4]: tulips.property_profile(m11, time_ind=1000, property_name='he4', num_rings=100, log=True)
     plt.show()
```

```
../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS///profile102.data
```



Here we didn't specify `cmin` and `cmax`, so the colorbar just shows the full range that is stored in the `profile.data` file. For this example, we also put `log=True`. With this option, we now see the log values of the property on the colorbar, which can be useful if you have quantities that span a wide range. In this case, when specifying `cmin` and `cmax`, you should also use the log values (for example:`cmin`=-0.4, `cmax`=0). With the `log_low_lim` keyword, you can define the value to adopt for negative and zero values of a property.

Tip

To explore the available properties, load a MESA profile with `m11.loadProfile(num=-1)`, then type `m11.prof.` + `Tab`, which will reveal a list with the options. Default units are in `cgs`. For more information about the properties in the output file, have a look at the MESA documentation.

1.5.4 Creating an animation

To create an animation to see how the property changes over time, we just have to change the `time_ind` variable. In the example below we chose to create an animation from the first (0) to the last (-1) time index. With `fps` (frames per second) you can adjust the speed of the video. To see the video, you need to save it first as a .mp4 file. You can specify the directory and the name with `output_fname`.

Warning

The amount of profiles stores in the `profile.data` file can be small. When making an animation, make sure that enough profile output has been saved to show a meaningful time evolution of the property of interest. You can check when profiles are saved with `m11.prof_ind['model']`. This returns a list with indices at which model number a profile is saved. You can use this to see how many profiles you have and decide if they cover the interesting stages in the evolution.

```
[ ]: tulips.property_profile(m11, time_ind=(0, -1), fps=30, property_name='logRho', num_
     ↪rings=100, output_fname='property_diagrams')
     plt.show()
```

```
[19]: from IPython.display import Video

      Video("property_diagrams.mp4", embed=True, width=700, height=600)
```

```
[19]: <IPython.core.display.Video object>
```

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.6 Chemical profile diagram

In this notebook we demonstrate how to create a chemical profile diagram with `tulips`. This diagram is useful when you are interested in the composition of a stellar model. It will not only show which isotopes are present, but also how the composition differs from the center to the edge of the stellar object. Similar to the other diagrams, `tulips` represents the stellar object as a circle, divided into rings. To show the interior composition, `tulips` uses nested pie-charts. For every ring, it shows the mass fraction of each isotope present in that layer by using different colors. Let's try and create an example chemical profile diagram.

1.6.1 Load an example model

At first, we have to load the history output of a MESA model. Here we load the `history.data` file of a $11M_\odot$ stellar model into object `m11`:

[1]:
```
# Interactive matplotlib plotting for jupyter lab
%matplotlib inline

# If you use jupyter notebook
# %matplotlib notebook

import matplotlib.pyplot as plt
import mesaPlot as mp

import tulips

# Specify directory of MESA model
SINGLE_M11_DIR = "../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS/"

m11 = mp.MESA()
m11.loadHistory(f=SINGLE_M11_DIR)
```

Warning

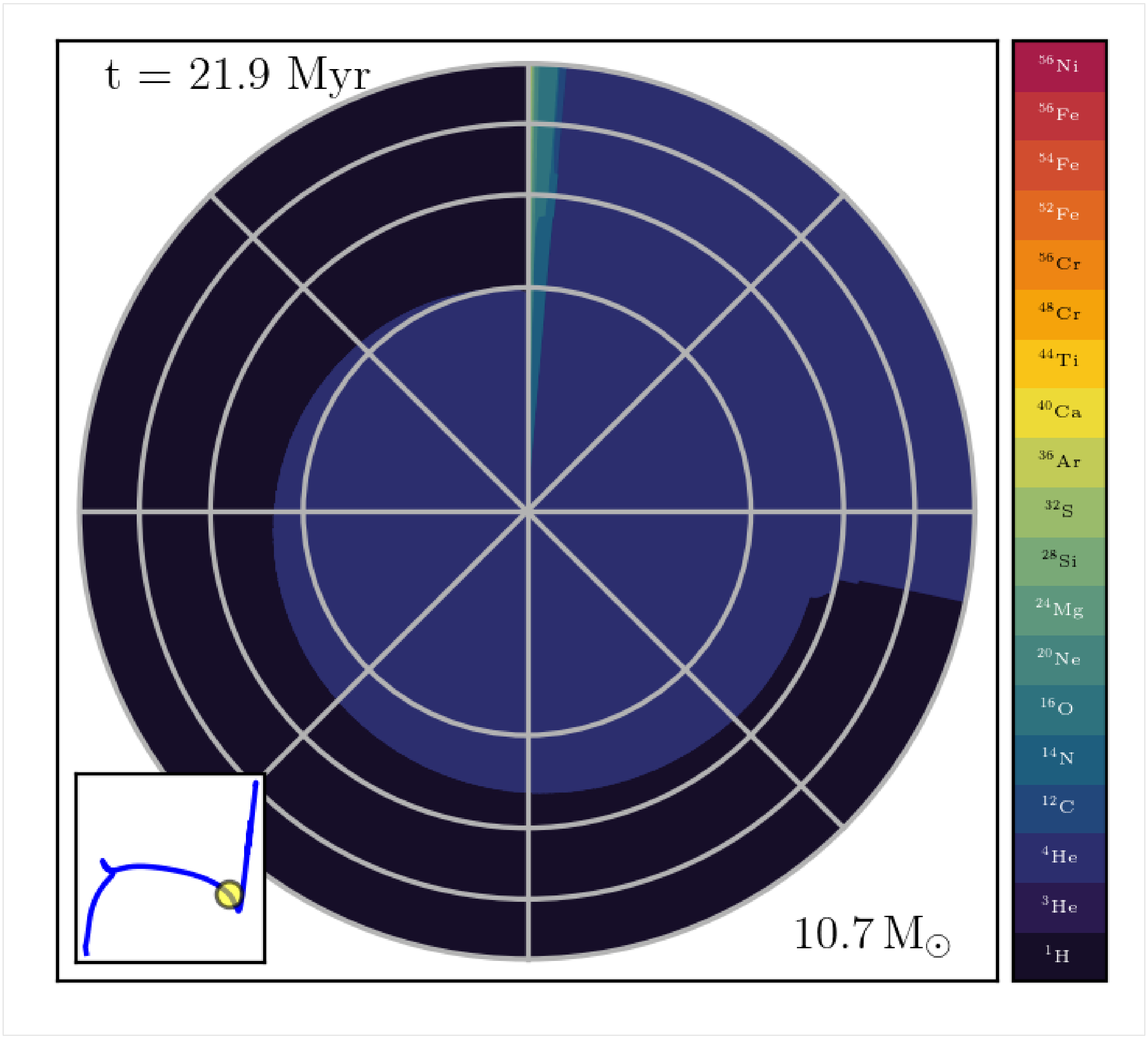
As explained in the *Getting Started* page, MESA produces two types of output: `history.data` and `profile.data` files. When making a chemical profile diagram, make sure that both of these types of files are stored in the model.

1.6.2 Plotting the chemical profile

We produce a chemical profile diagram of our stellar model with the `tulips.chemical_profile` function. As arguments it requires our `mesaPlot` object `m11`, and a time index. Here, we plot a model halfway its lifetime by setting `time_ind=1000`. Have a look at the *Getting Started* page for more information about how to use `time_ind`.

[3]:
```
tulips.chemical_profile(m11,time_ind=1000)
plt.show()
```

```
../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS///profile102.data
```



This diagram shows us a stellar model, represented by a circle that is divided into nested pie-charts. You can specify the minimum width of a nested pie-chart ring with the `width` keyword. The colors indicate which element is present in a certain region in its interior, as shown in the colorbar on the right. It is possible to choose which elements you want to include. To do this, you can specify a list with isotope names through the `isotope_list` option.

1.6.3 Reading the diagram

The radius of the cirle is proportional to the square root of the enclosed mass. Accordingly, the area covered by a color is proportional to the total mass of that element. The grey circles help to visualize this, since they indicate the regions that account for 25, 50, 75 and 100% of the total mass. You can hide these circles with `show_ring_annotations = False`. In this example, we see a star whose outer layers consists for almost 75% of hydrogen and 25% of helium. However, the inner layers contain almost only helium. There is also a small part with heavier elements that come from the initial composition of the star.

Note

It is possible to change what is represented by the radius with the `raxis` keyword. However, this is not recommended since it will make the diagram less intuitive and difficult to interpret. `tulips` automatically shows the square root of that quantity, which will make it hard to read the diagram.

1.6.4 Adjusting the diagram

By default, `tulips` creates pie-charts in a counterclockwise direction, beginning from an angle of 90 degrees from the horizontal axis. It is possible to change these settings by modifying the `counterclock` and `startangle` variables. Elements are plotted with a certain order. For each pie chart, `tulips` first plots the element that has the lowest mass number (hydrogen), and continues with heavier ones, ending with e.g. iron.

It is possible to scale the diagram. This can be useful if you are, for example, only interested in the composition in the core. You can do this by changing the `scale` variable, which gives the value to scale the circle to. The radius of the circle will be $\sqrt{M_{\mathrm{star}}}/\sqrt{\mathrm{scale}}$.

1.6.5 Creating your own colorbar

It is possible to create your own colorbar that is shown in the diagram. To do this, you can either change the colorbar settings in the `chemical_profile` function, or use the `tulips.create_elem_colorbars` function. Here, we demonstrate the latter. The function requires an isotope list that contains the elements that you want to include in the colorbar. To get all isotopes that are stored in a MESA model, you can use the `get_isotopes` and the `get_isotopes_from_prof` functions. The first one returns all elements in a MESA object, for example in our `m11` object:

[4]:
```python
isotope_list = tulips.get_isotopes(m11)
print(isotope_list)
```

```
['h1', 'he3', 'he4', 'c12', 'n14', 'o16', 'ne20', 'mg24', 'si28', 's32', 'ar36', 'ca40',
↪'ti44', 'cr48', 'cr56', 'fe52', 'fe54', 'fe56', 'ni56']
```

Alternatively, `get_isotopes_from_prof` only returns the elements in a single profile. This can be useful because loading all profiles is computationally expensive. To use this function, we load the first profile in the `m11` object with `m11.loadProfile(num=1)`. Then we use this profile, which can be accessed with `m11.prof`, as argument of the `get_isotopes_from_prof` function:

[5]:
```python
m11.loadProfile(num=1)
profile_isotope_list = tulips.get_isotopes_from_prof(m11.prof)
print(profile_isotope_list)
```

```
../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS///profile1.data
['h1', 'he3', 'he4', 'c12', 'n14', 'o16', 'ne20', 'mg24', 'si28', 's32', 'ar36', 'ca40',
↪'ti44', 'cr48', 'cr56', 'fe52', 'fe54', 'fe56', 'ni56']
```

Note

In the example above, you can see that the first profile contains the same list of isotopes as in the complete `m11` object. This is because MESA keeps track of isotopes during the entire calculations. Often, in the beginning of a star's life, small amounts of the heavier elements are already present (dependent on the star's metallicity). It is possible to check these amounts throughout the star at a particular moment in time, for example for C-12, with `m11.prof.c12` or, alternatively, with `m11.prof.data['c12']`.

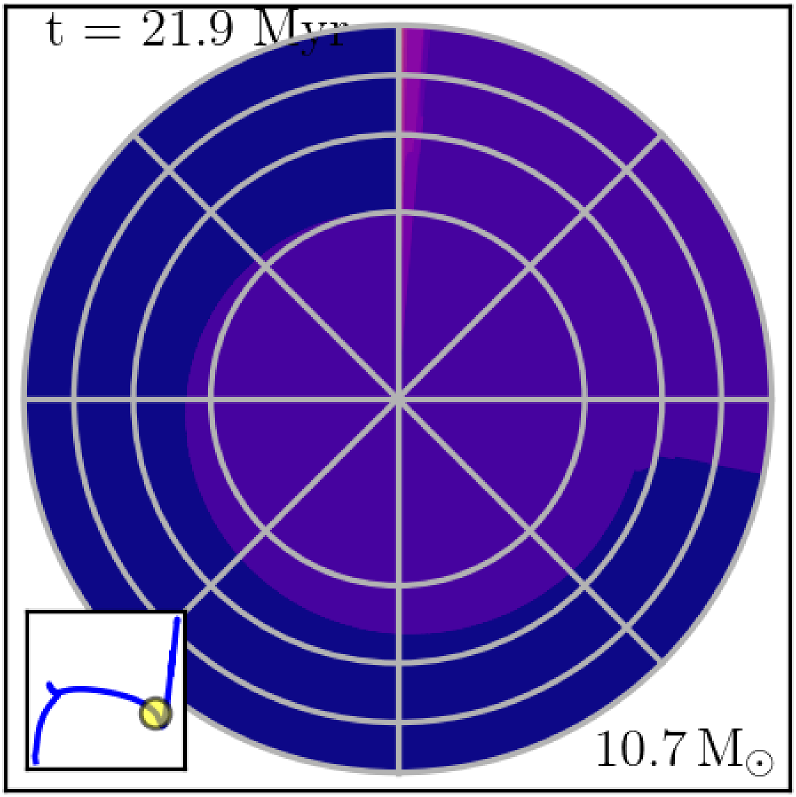
Now let's create a customized colorbar with these isotopes. To see how you can change the colormap, have a look at the *Customize options* page. It is also possible to specify how many elements you would like to show on the colorbar. With the `min_cbar_elem` and `max_cbar_elem` properties, you can manage the minimum and maximum amount of isotopes in one colorbar. To change the orientation of the colorbar, you can set `orientation` to `'horizontal'`. In the following example we first create a `chemical_profile` diagram, without a colorbar. Then we plot a customized colorbar with the `create_elem_colorbars` function on the existing `ax` object.

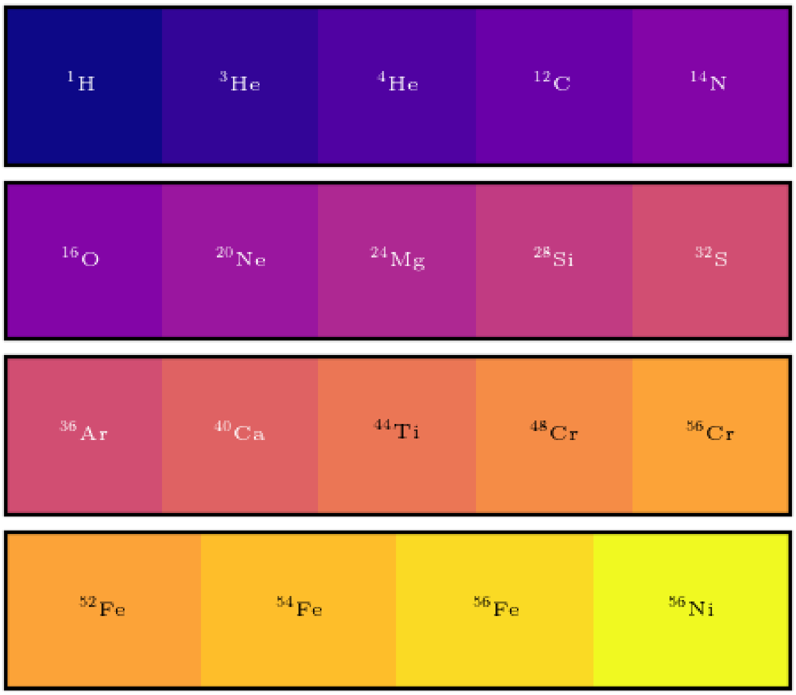
Note

Make sure to use the same colormap in your diagram and in your customized colorbar, otherwise the colors in the diagram will not correspond to the colorbar.

[6]:
```
fig, ax = tulips.chemical_profile(m11, time_ind=1000, show_colorbar=False, cmap='plasma',
↪ fig_size=(8,6))
tulips.create_elem_colorbars(isotope_list, ax, cmap='plasma', min_cbar_elem=4, max_cbar_
↪elem=5, orientation='horizontal')
plt.show()
```

```
../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS///profile102.data
```


t = 21.9 Myr
$10.7\,M_\odot$


^{1}H
^{3}He
^{4}He
^{12}C
^{14}N
^{16}O
^{20}Ne
^{24}Mg
^{28}Si
^{32}S
^{36}Ar
^{40}Ca
^{44}Ti
^{48}Cr
^{56}Cr
^{52}Fe
^{54}Fe
^{56}Fe
^{56}Ni

As expected, tulips produced a horizontal colorbar with a minimum of 4, and a maximum of 5 elements, with the specified matplotlib 'plasma' colormap.

1.6.6 Creating an animation

To create an animation of a chemical profile diagram and see how the composition of the stellar model changes over time, you have to change the time_ind option. In the following example, we produce an animation from the first(0) to the last(-1) MESA model, calculated for 300 layers. It is saved in a file called chemical_profile.mp4, as specified with the output_fname keyword. You can specify the speed of the video with the fps (frames per second) option. It might take some time to create the animation!

Warning

The amount of profiles stores in the profile.data file can be small. When making an animation, make sure that enough profile output has been saved to show a meaningful time evolution of the composition. You can check when profiles are saved with m11.prof_ind['model']. This returns a list of indices that indicate at which MESA model number a profile is saved. You can use this to see how many profiles you have and if they cover the evolutionary stages you are interested in.

```
[ ]: tulips.chemical_profile(m11, time_ind=(0,-1), fps=30, num_rings=300, output_fname=
     ↪'chemical_profile', cmap=tulips.CMAP_BASE)
     plt.show()
```

```
[7]: from IPython.display import Video

     Video("chemical_profile.mp4", embed=True, width=700, height=600)
```

```
[7]: <IPython.core.display.Video object>
```

During the first part of the animation, we clearly see the composition of the inner layers change from hydrogen (light blue) to helium (blue) because of nuclear burning. Then, after a the hydrogen shell burning phase where not much changes in the composition, the next stage of nuclear burning starts, which increases the carbon and oxygen mass in the core. In the same manner, the effect of all evolutionary stages on the composition can be observed in the animation.

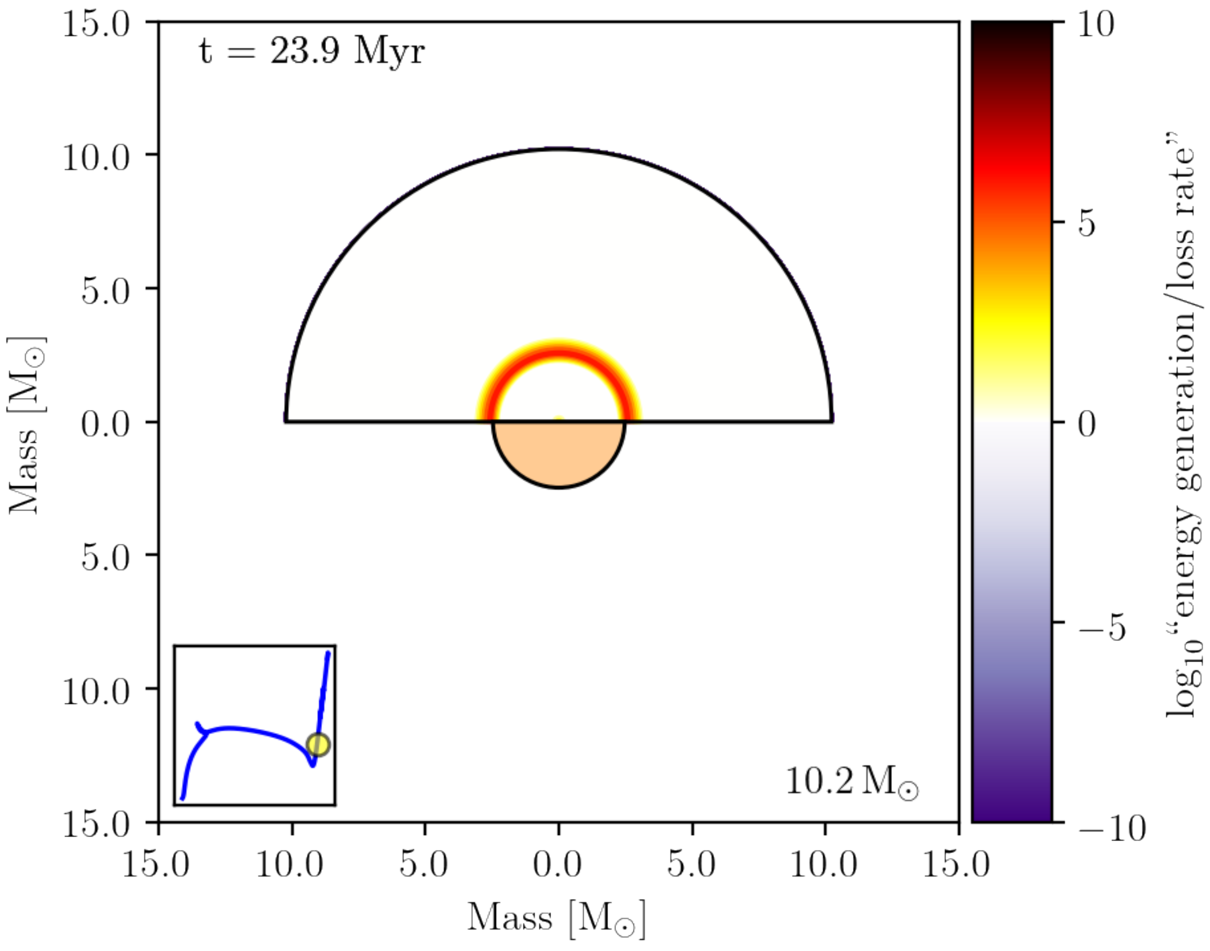
Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.7 Combining multiple static diagrams

It is possible to combine multiple tulips diagrams (at one particular time) in a single figure. This can provide additional insights about the properties of stellar models. In this notebook we demonstrate how you can do this and what you can learn from it. Let's plot an *energy and mixing diagram* together with a *perceived color diagram* of an 11 $M_\odot$ star. The perceived color diagram can show any property that is stored in the MESA output file of the stellar model. In our case we are interested in the helium core mass so we choose raxis=he_core_mass. To plot these diagrams in the same figure, we pass the same fig and ax that are returned from the energy_and_mixing function to the perceived_color function with fig=fig and ax=ax. In this way, tulips plots the perceived color diagram on the same figure and axis objects as the energy and mixing diagram. However, when we do this, the diagrams will be plotted on top of each other, which makes it difficult to interpret. Therefore, we choose different starting and ending angles so that each diagram covers only part of the circle. This can be done using the theta1 and theta2 properties. The angles are given in units of degrees and start counterclockwise from the horizontal axis. We choose to plot the stellar model at `time_ind=2000` and change the axis limit and label with the `axis_lim` and `axis_label` options.

[7]:
```
# Interactive matplotlib plotting for jupyter lab
%matplotlib inline

# If you use jupyter notebook
# %matplotlib notebook

import matplotlib.pyplot as plt
import mesaPlot as mp
import tulips

EXAMPLE_DIR = "../../tulips/MESA_DIR_EXAMPLE/"

m11 = mp.MESA() # Create MESA object
m11.loadHistory(filename_in=EXAMPLE_DIR + 'history.data')
```

[11]:
```
fig, ax = tulips.energy_and_mixing(m11, time_ind=2000, theta1=0, theta2=180)
tulips.perceived_color(m11, time_ind=2000, raxis='he_core_mass', fig=fig, ax=ax,␣
↪theta1=180, theta2=360, axis_label='Mass $[\mathrm{M}_{\odot}]$', axis_lim=15)
plt.show()
```



Note

To properly compare the diagrams, make sure they have the same axis coordinate for the radius of the circle, such as mass in our case.

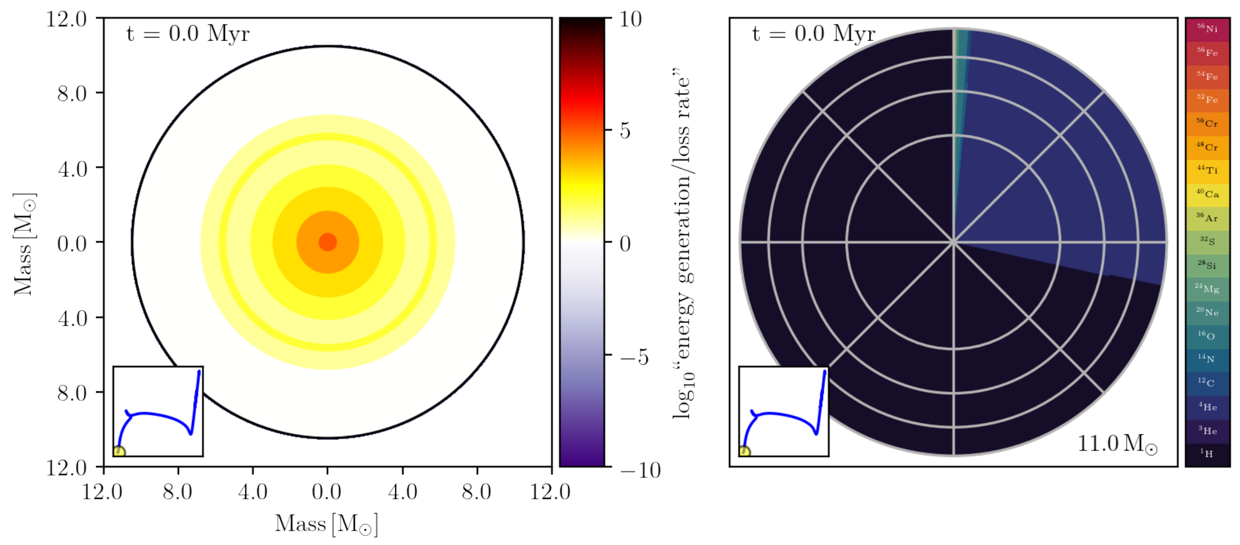
The upper half of the circle shows the energy generation rate, indicated by the colorbar on the right. The radius of this half-circle represents the total mass. The radius of the lower half of the circle shows the helium core mass, and its color indicates the perceived color of the star. From this plot, we can clearly see the hydrogen burning shell right above the helium rich core. This is just an example, but these combined diagrams can be created with all types of `tulips` diagrams, also with more than two. However, this feature is designed only for static diagrams that represent the stellar model at one moment in time.

1.7.1 Create diagrams in subplots

It is also possible to show different `tulips` diagrams in subplots. To do so, you first create a figure with multiple subplots. In the following example, we choose to create a figure with `plt.subplots`, where the first argument represents the amount of rows and the second the amount of columns. In our case we choose 1 row and 2 columns. We want to show an energy and mixing diagram together with a chemical profile diagram. With the `ax` property you can specify on which subplot the diagram should be shown. Since we want to show a *chemical profile diagram*, we first need to specify the folder where the `profile.data` file is stored.

[12]:
```
SINGLE_M11_DIR = "../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS/"
m11.log_fold = SINGLE_M11_DIR

fig, ax = plt.subplots(1,2,figsize=(8,5))
tulips.energy_and_mixing(m11, time_ind=0, fig=fig, ax=ax[0])
tulips.chemical_profile(m11, time_ind=0, fig=fig, ax=ax[1])
fig.tight_layout() # To make sure the axis labels don't overlap
plt.show()
```

```
../../../../old_tulips_project/tulips/tulips/MESA_DIR_EXAMPLE/LOGS//profile1.data
```



Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.8 Customize options

All tulips diagrams are customizable and contain many options that allow you to change the plots to your liking. These options are managed by a set of keywords that can be specified when calling a tulips function. In this notebook we mention the main possibilities.

1.8.1 Options available for all TULIPS diagrams

Changing the axis and label

The raxis property indicates what is represented by the radius of the circle. This is the quantity that is specified by the x and y axis labels. You can change the label of the x and y axis with the axis_label option.

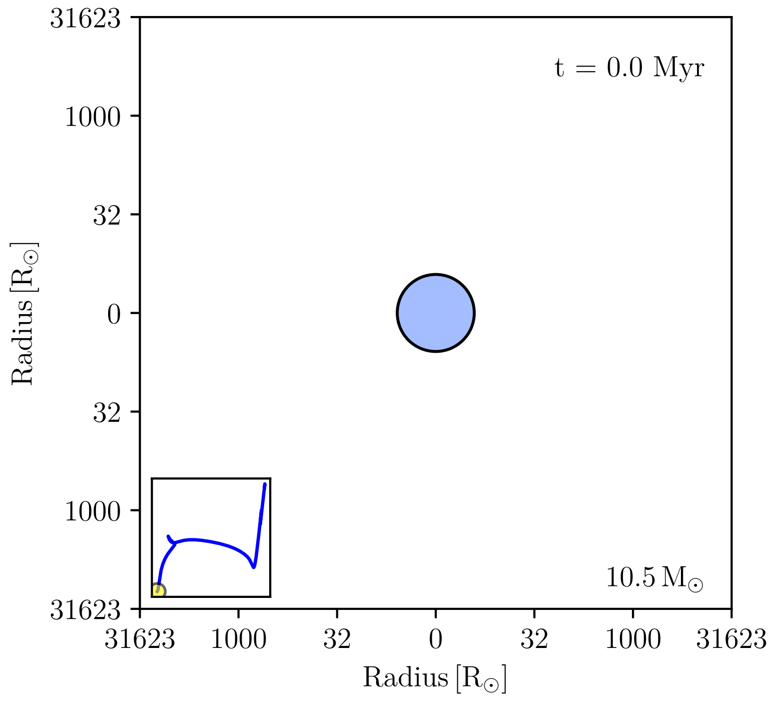
Figure options

It is possible to plot tulips diagrams on already existing matplotlib fig and ax objects. This can be useful if you want to create subplots with multiple diagrams. You can pass these objects to the function with the fig and ax keywords. For an example, look at the *Combining multiple static diagrams* page. Moreover, you can choose the size of the figure with the fig_size option. You can set this parameter with a tuple containing the size in inches.

Time label

tulips diagrams show stellar models at a particular moment in time. By default, the age of the star at that moment is shown in the upper left corner of the plot in units of Myr. You can hide this time label by setting show_time_label to False. It is also possible to change the location of the time label with time_label_loc. You can specify the location using a tuple of values that represent fractions of the maximum X and Y axis values. We can move the time label to be in the upper right corner as follows:

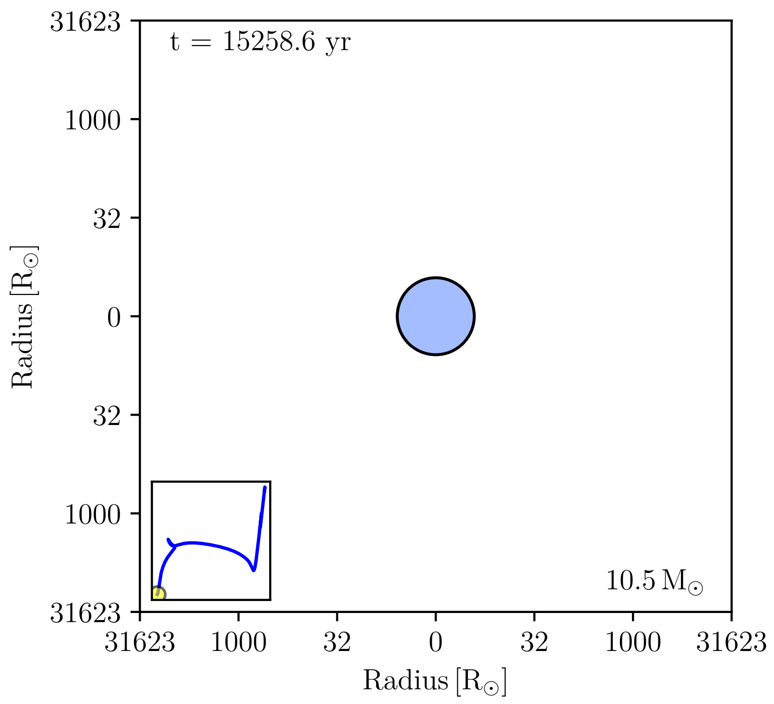
[3]:
```
tulips.perceived_color(m11, time_ind=0, time_label_loc=(0.7,0.9))
plt.show()
```



Time units

By default, the timelabel is shown in units of Myr. You can change this to another unit with the `time_unit` option. To convert the units, `tulips` makes use of `astropy.units` functionalities. Hence, the chosen unit should be a valid `astropy` unit. You can check this with `import astropy.units as u` and then press `u.+Tab`. This will reveal a lists with the valid units. For example, we can show the timelabel in year:

```
[4]: tulips.perceived_color(m11,time_ind=0,time_unit='yr')
     plt.show()
```



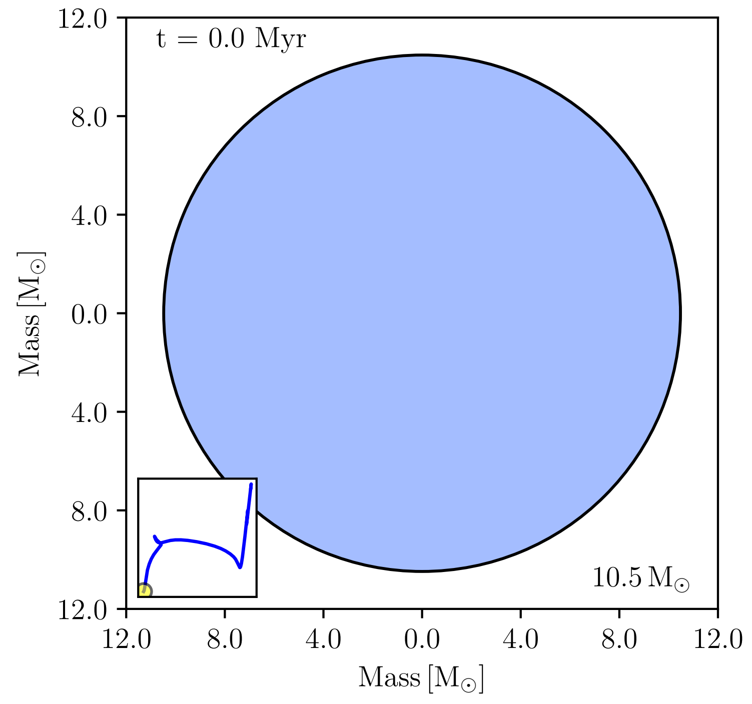
Note

For information about the `time_scale_type` option, look at the *Timestep options* page.

Setting axis limits

It is possible to specify plot axis limits, to zoom in or out on the stellar model. In the following example, we see a circle corresponding to a stellar model whose radius represents the total mass of the star. By default, the (identical) X and Y axis limits are computed as 1.5 times the maximum of the `raxis` quantity. You can change `axis_lim` to specify a different upper limit for the X and Y axis. For example, if we set `axis_lim=10`, we are zooming in and only see the stellar model up to 10 $M_\odot$.

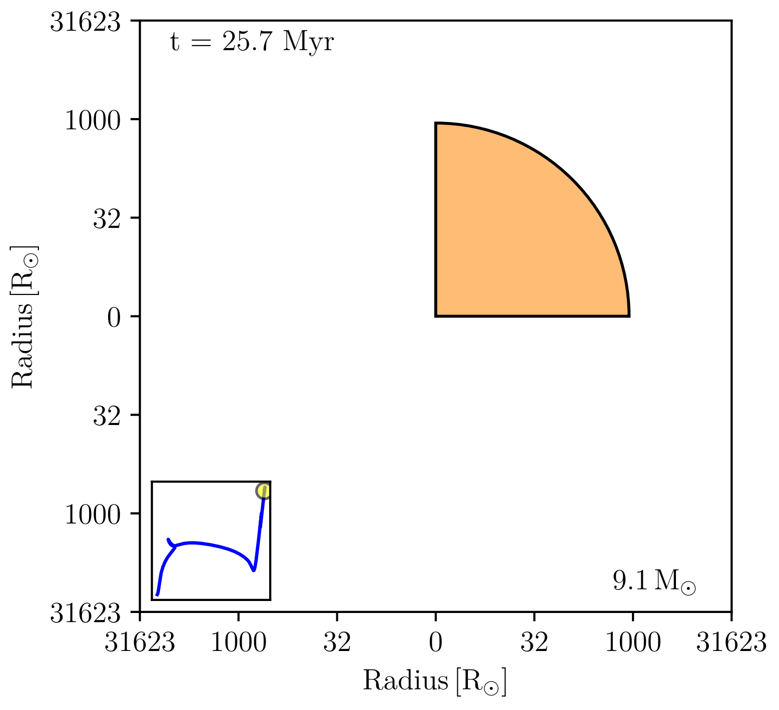
```
[5]: tulips.perceived_color(m11, time_ind=0, axis_lim=10, raxis='star_mass')
plt.show()
```



Plotting wedges

Instead of plotting a full circle, you can also just plot one wedge. To do this, you can specify `theta1` and `theta2` as the start and end angle (in degrees) of the wedge. This is useful when you want to combine multiple diagrams in one plot (as at the *Combining multiple static diagrams* page). For example, we decide to plot only the first quarter as follows:

[6]:
```
tulips.perceived_color(m11, time_ind=-1, theta1=0, theta2=90)
plt.show()
```



Hertzsprung-Russell diagram

All `tulips` diagrams show (by default) an inset Hertzsprung-Russell diagram in the lower left corner of the plot. A blue line shows the track that the stellar model will follow during its lifetime, and a yellow circle indicates where the model is located on this track at the current time index. To hide this inset HR diagram, set `hrd_inset` to `False`. You can also show labels on the axis (log:math:*_{10}(T_{textrm{eff}}/textrm{K})* vs. $\log_{10}(L/L_\odot)$) of the HR diagram by setting `show_hrd_ticks_and_labels` to `True`.

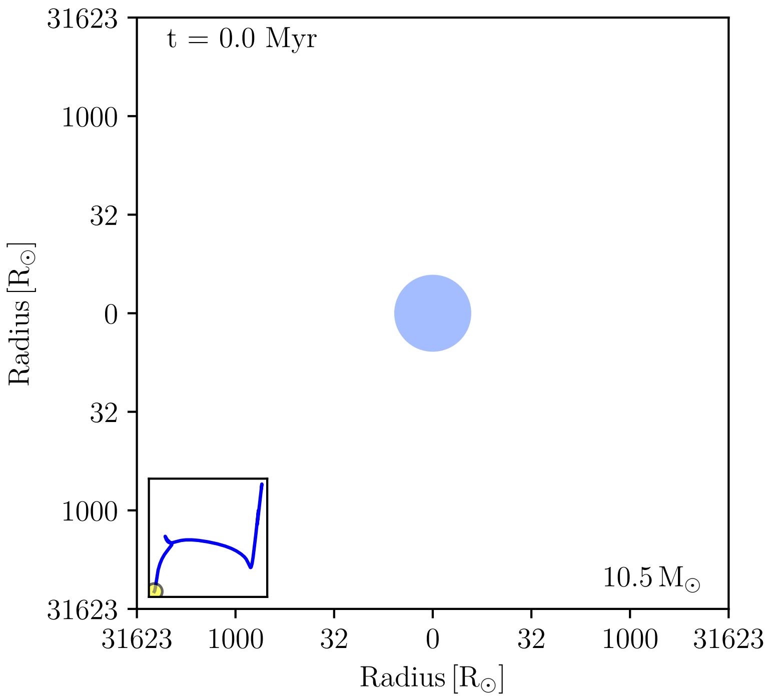
Total mass label

By default, `tulips` shows the total mass of the stellar object (in $M_\odot$) in the bottom right corner. To hide this, you can set `show_total_mass` to `False`.

Show stellar surface

In a `tulips` diagram, the outer boundary of the star is shown by a black solid line. If you don't want to show this line, set `show_surface` to `False`.

```
[7]: tulips.perceived_color(m11, time_ind=0, show_surface=False)
     plt.show()
```



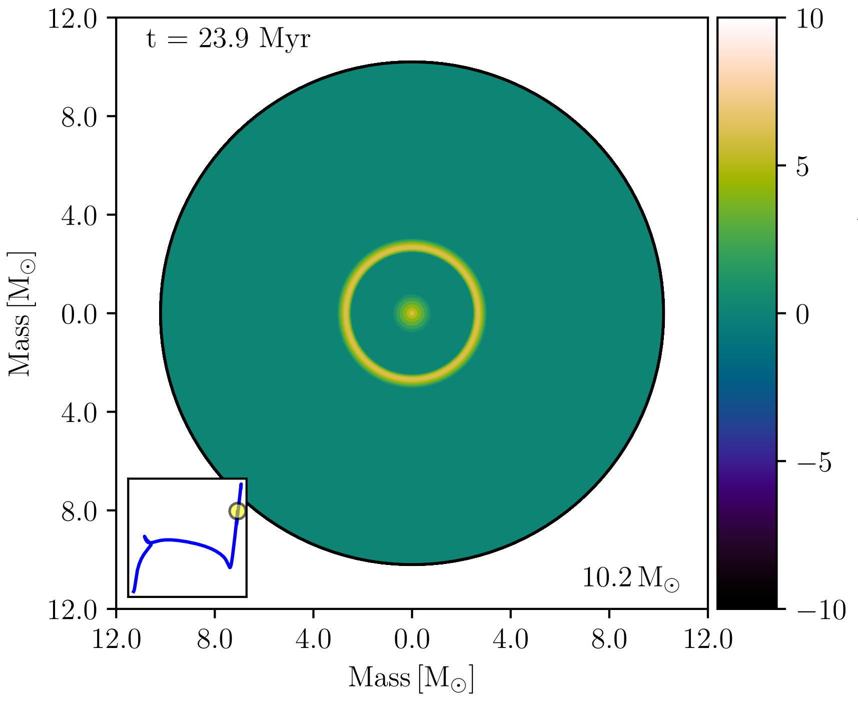
Output options

If you make an animation, it is automatically saved in a default filename in .mp4 format. However, you can change these settings. With the `output_fname` option, you can specify how you want to call the output file. Additionally, you can change the `anim_fmt` option to one of the other supported formats.

Colormap

For all `tulips` diagrams except the perceived color diagram, it is possible to specify the colormap with the `cmap` option. The colormap can be changed to any matplotlib colormap, but also to cmasher colormaps, which are color-vision deficiency friendly. In the following example we load the `cmasher` library and plot an `energy_and_mixing` diagram with another colormap:

[8]:
```
import cmasher as cmr
tulips.energy_and_mixing(m11, time_ind=2200, cmin=-10, cmax=10, show_total_mass=True,
                         cmap='cmr.rainforest')
plt.show()
```



1.8.2 Options available for specific TULIPS diagrams

In addition to these general options, there are more possibilities specific for each `tulips` diagram. These are explained in the page of the corresponding diagram, which you can access via the navigation menu.

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

1.9 Timestep options

When creating an animation, `tulips` offers multiple options to rescale the time in order to capture all evolutionary phases. This is controlled by the `time_scale_type` argument, for which three different options exist: `model_number`, `linear` and `log_to_end`. In this notebook we demonstrate these possibilities.

Note

The `time_scale_type` option is only available for the `perceived_color` and `energy_and_mixing` diagrams. It requires many output models to be stored, which is not always the case for diagrams that rely on MESA profile output.

1.9.1 model_number

The default option `model_number` is used for the animations in the tutorials. In this case, the timesteps follow the moments when a new MESA model was saved. This is a non-physical arbitrary timescale that depends on various resolution settings in MESA. We create a `energy_and_mixing` animation with this setting for a 11 $M_\odot$ star:

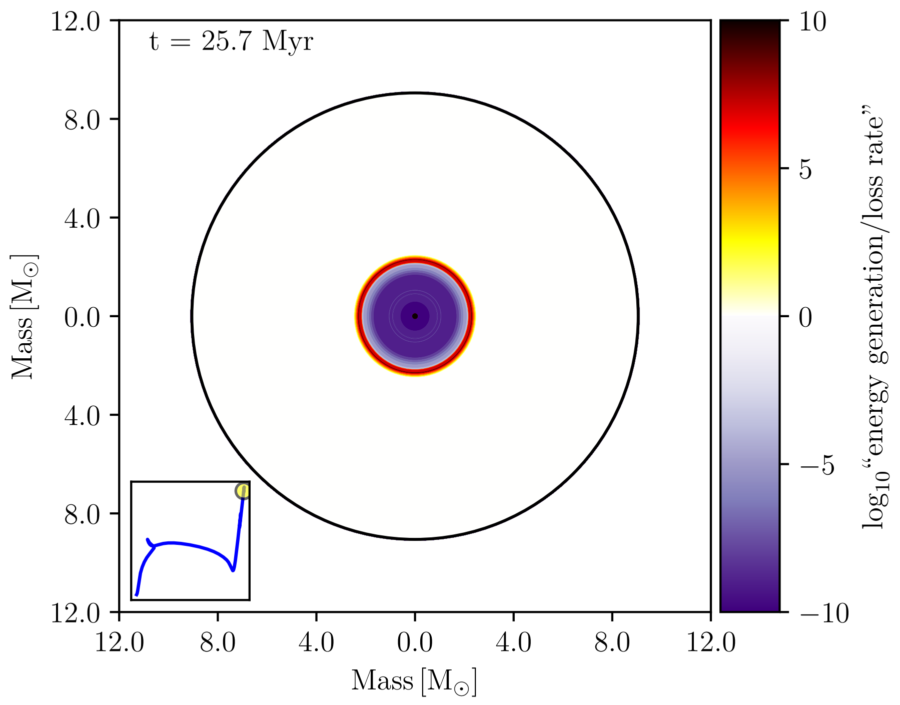
```
[1]: # Interactive matplotlib plotting for jupyter lab
%matplotlib inline

# If you use jupyter notebook
# %matplotlib notebook

import matplotlib as plt
import mesaPlot as mp
import tulips

EXAMPLE_DIR = "../../tulips/MESA_DIR_EXAMPLE/"

m = mp.MESA() # Create MESA object
m.loadHistory(filename_in=EXAMPLE_DIR + 'history.data')
```

```
[3]: tulips.energy_and_mixing(m, time_ind=(0,-1,10), fps=30, time_scale_type='model_number',␣
↪output_fname='em_model_number')
```

```
  0%|          | 0/492 [00:00<?, ?it/s]
```

```
Creating movie...
```

```
[3]: (<Figure size 1100x800 with 3 Axes>,
 <AxesSubplot:xlabel='Mass$\\,[\\rm{M}_\\odot]$', ylabel='Mass$\\,[\\rm{M}_\\odot]$'>)
```



```
[4]: from IPython.display import Video

     Video("em_model_number.mp4", embed=True, width=700, height=600)
```

```
[4]: <IPython.core.display.Video object>
```

1.9.2 linear

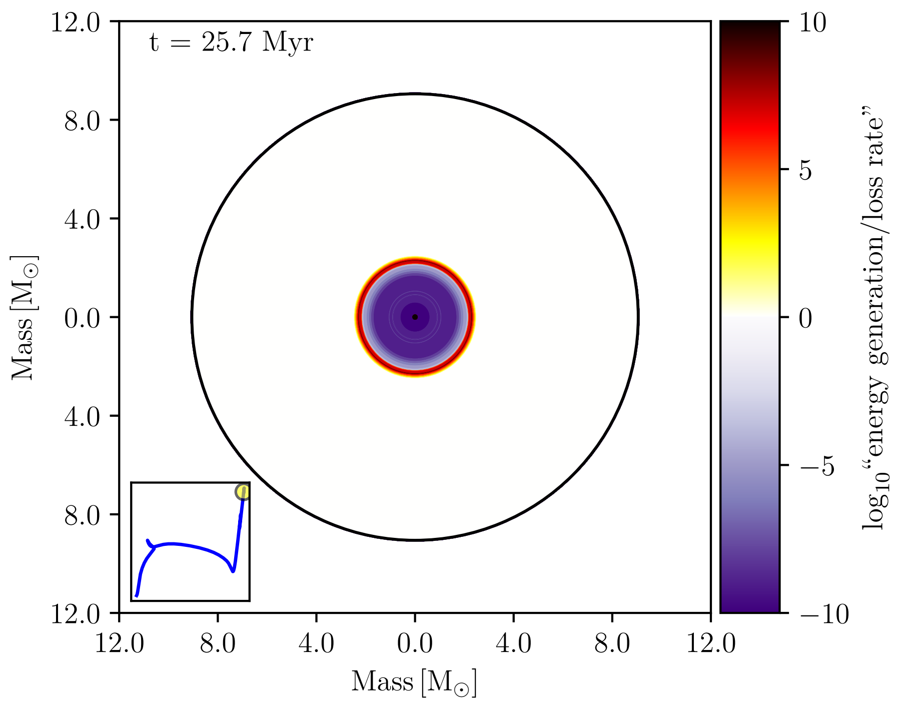
Alternatively, when you set `time_scale_time='linear'`, the timesteps follow the actual physical age of the star (linearly). During the lifetime of a star, it passes through different stages of stellar evolution. These stages can have vastly different timescales. Stars spend the longest part of their life on the main sequence, whereas the last stages of nuclear burning can last for only a day. When we produce an animation with this setting, the majority of the resulting video therefore shows the star slowly evolving on the main sequence:

```
[5]: tulips.energy_and_mixing(m, time_ind=(0,-1,10), fps=30, time_scale_type='linear', output_
     ↪fname='em_linear')
```

```
  0%|          | 0/492 [00:00<?, ?it/s]
```

```
Creating movie...
```

```
[5]: (<Figure size 1100x800 with 3 Axes>,
      <AxesSubplot:xlabel='Mass$\\,[\\rm{M}_\\odot]$', ylabel='Mass$\\,[\\rm{M}_\\odot]$'>)
```



```
[6]: from IPython.display import Video

     Video("em_linear.mp4", embed=True, width=700, height=600)
```

```
[6]: <IPython.core.display.Video object>
```

1.9.3 log_to_end

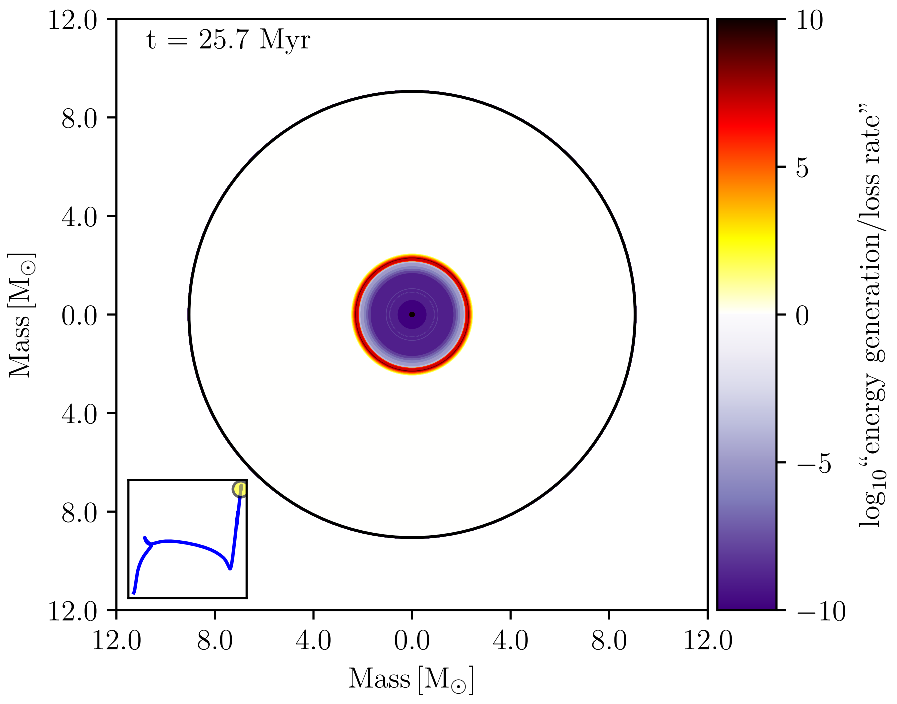
To capture the shorter phases at the end of the evolution of a stellar model, you can rescale the time in such a way that all evolution stages have more similar durations. You can do this by setting `time_scale_type='log_to_end'`. When this is set, the time follows $\log_{10}(t_{\rm end} - t)$, where $t_{\rm end}$ is the final age of the stellar model and t is the age of the star. As a result, the earlier phases shorten whereas the later phases lengthen. This allows us to distinguish the different evolutionary stages and also capture the fast transitions. Especially for the `energy_and_mixing` diagram this is useful since it reveals the different stages of nuclear burning in more detail.

```
[7]: tulips.energy_and_mixing(m, time_ind=(0,-1,10), fps=15, time_scale_type='log_to_end',␣
     ↪output_fname='em_log_to_end')
```

```
  0%|          | 0/492 [00:00<?, ?it/s]
```

```
Creating movie...
```

```
[7]: (<Figure size 1100x800 with 3 Axes>,
      <AxesSubplot:xlabel='Mass$\\,[\\rm{M}_\\odot]$', ylabel='Mass$\\,[\\rm{M}_\\odot]$'>)
```



```
[8]: from IPython.display import Video

     Video("em_log_to_end.mp4", embed=True, width=700, height=600)
```

```
[8]: <IPython.core.display.Video object>
```

1.10 Functions

1.10.1 tulips.tulips Module

Functions

add_inset_hrd(m[, time_index, ax, axins, ...])	Add inset HRD.
add_ring_annotations(ax, rmax[, ...])	Add concentric circles.
add_time_label(age, ax[, time_label_loc, ...])	Add time label.
animated_hist_comp(m1, m2[, raxis, label1, ...])	
animated_hist_comp_test(m1, m2[, fig, ax, ...])	Plot two models together.
check_time_indeces(time_ind, star_age)	Check time indices.
chem_elem_notation(elem_list)	Make LateX formatted elements
chemical_profile(m[, time_ind, ...])	Creates chemical profile diagram.
colorbar(mappable[, ax, size, pad, cax, ...])	
create_elem_colorbars(isotope_list, ax[, ...])	Create colorbar for isotopes
energy_and_mixing(m[, time_ind, show_mix, ...])	Create energy and mixing diagram.
find_closest(ary, value)	
find_profile(m[, time_ind])	Load closest MESA profile.
get_ffmpeg_exe()	Get the ffmpeg executable file.
get_isotopes(m)	Get list of isotopes
get_isotopes_from_prof(prof)	Get list of isotopes
inset_axes(parent_axes, width, height[, ...])	Create an inset axes with a given width and height.
make_axes_locatable(axes)	
make_pie_composition_plot(ax, prof[, ...])	Create composition plot.
make_property_plot(ax, prof[, ...])	Create property plot.
makedirs(name [[, mode, exist_ok])	Super-mkdir; create a leaf directory and all intermediate ones.
num_cbar_elem(num_elems[, min_cbar_elem, ...])	Calculate number of colorbars
perceived_color(m[, time_ind, raxis, fps, ...])	Create perceived color diagram of a stellar model.
property_profile(m[, time_ind, ...])	Create property profile diagram.
rescale_time(indices, m[, time_scale_type])	Rescale the time.
set_axis_ticks_and_labels(ax[, raxis, ...])	Format axis ticks.
teff2rgb(t_ary)	Convert effective temperature to rgb colors.
too_dark(color_ary)	Check if text too dark

add_inset_hrd

tulips.tulips.**add_inset_hrd**(*m*, *time_index=100*, *ax=None*, *axins=None*, *fraction='20%'*, *indices=None*, *loc='lower left'*, *bbox_to_anchor=None*, *show_hrd_ticks_and_labels=False*)

Add inset HRD.

Add an inset HRD to a plot that highlights the current time-step.

Parameters

m [mesaPlot object]

ax [axis object]

indices [None, list or ndarray] Selected indices for plotting

fraction [string] Fraction of the parent axis used for setting the size of the inset

axins [None or Axes] If provided, inset axis to use.

time_index [int] Time index to highlight on inset plot.

loc [str or int] Matplotlib location to put the inset axis on the parent axis.

bbox_to_anchor [tuple (x, y, width, height)] Bounding box for the axis

show_hrd_ticks_and_labels [Boolean] If set, display the axis ticks and labels of the inset HRD

Returns

——-

(axins, point): Axes Created inset axis, and the point moving on the plot.

add_ring_annotations

`tulips.tulips.`**`add_ring_annotations`**`(`*ax*, *rmax*, *fraction_list=None*, *show_fraction=True*, *show_fraction_labels=True*, *use_actual_mass_fraction=True*, *percentage_list=None*, *show_percentage=False*, *show_percentage_labels=False*, *startangle=90*, *counterclock=True*, *loc=1.25*, ***kwargs*`)`

Add concentric circles.

Add concentric circles on an axis that indicate the fraction of the maximum radius given. Optionally, indications of percentages can also be given.

Parameters

ax [matplotlib axis object]

rmax [float] Value of the maximum radius of the circle to compare to as a reference

fraction_list: list Default None, list of floats giving the fractions of the total mass.

show_fraction [boolean] Default True, whether or not to plot circles that have radii of a fraction of the reference circle radius.

show_fraction_labels: boolean Default True, whether or not to add labels indication the fractions.

use_actual_mass_fraction: boolean Default True, whether or not to locate the fraction circles at the location where a circle contains this mass or at a fraction of *rmax*.

percentage_list [list] Default None, list of floats giving the percentages to add.

show_percentage [boolean] Default False, whether to plot radial lines indicating a certain percentage.

show_percentage_labels [boolean] Default False, whether or not to add labels indicating percentages.

startangle [float] Value of the angle to start with for the percentages.

counterclock: bool, optional Default: True, specify percentage direction, clockwise or counterclockwise.

loc: float Default 1.25, location of percentage labels in units of fraction of *rmax*.

Returns

List of matplotlib artists created

add_time_label

`tulips.tulips.`**`add_time_label`**`(`*age*, *ax*, *time_label_loc=None*, *time_unit='Myr'*`)`

Add time label.

Add a time label in the upper left corner of a diagram.

Parameters

age [float] Age of the stellar object.

ax [matplotlib Axes object]

time_label_loc [None or tuple] Default None, the custom location of the time label in units of the Axes coordinate; (0, 0) is bottom left of the axes, and (1, 1) is top right of the axes

time_unit: str Unit of the time.

Returns

matplotlib Artist object

animated_hist_comp

`tulips.tulips.`**`animated_hist_comp`**`(`*m1*, *m2*, *raxis='log_R'*, *label1=''*, *label2=''*, *time_index_start1=0*, *time_index_start2=0*, *time_index_end1=- 1*, *time_index_end2=- 1*, *time_indices=None*, *frames=200*, *fps=10*, *plot_name_base='mesarings_comp_'*, *plot_dir='.'*, *hrd_inset=True*, *fig_size=(11, 9)*`)`

animated_hist_comp_test

`tulips.tulips.`**`animated_hist_comp_test`**`(`*m1*, *m2*, *fig=None*, *ax=None*, *raxis='log_R'*, *label1=''*, *label2=''*, *time_index1=0*, *time_index2=0*, *hrd_inset=True*`)`

Plot two models together.

Plot two MESA models at the same time as half-circles with radius r over the same evolutionary time.

Parameters

label2 :

label1 :

time_index2 :

m1 [mesaPlot object] Already loaded a history file.

m2 [second mesaPlot object] Already loaded a history file.

fig [Figure object] If set, plot on existing figure.

ax [Axes object] If set, plot on provided axis.

raxis: str Valid column name for a MESA history file. This sets the outer radius of the star.

time_label_loc [tuple] Location of the time label on the plot as fraction of the maximal size.

time_index [int] Contains time index of moment to plot.

hrd_inset [boolean] If set, add an inset HRD that indicates the current time index to each plot.

Returns

fig, ax

check_time_indeces

`tulips.tulips.`**`check_time_indeces`**`(`*time_ind*, *star_age*`)`

Check time indices.

Check if time indices are correctly set and define start_ind and end_ind.

Parameters

time_ind: int or tuple (start_index, end_index, step=1) or (start_index, end_index, step)
If int: create the plot at the index *time_ind*.

star_age: array Ages of star from MESA history file

Returns

start_ind, end_ind, ind_step

chem_elem_notation

`tulips.tulips.`**`chem_elem_notation`**`(`*elem_list*`)`

Make LateX formatted elements

Change list of isotopes into LateX formatted list of isotopes

Parameters

elem_list [np.array or list of RGBA color arrays]

Returns

new_list [list] New list of str that are laTeX formatted.

chemical_profile

`tulips.tulips.`**`chemical_profile`**`(`*m*, *time_ind=-1*, *isotope_list=None*, *num_rings=-1*, *scale=-1*, *width=0.03*, *raxis='mass'*, *show_ring_annotations=True*, *fps=10*, *fig=None*, *ax=None*, *show_time_label=True*, *time_label_loc=()*, *time_unit='Myr'*, *fig_size=(5.5, 4)*, *axis_lim=1.05*, *axis_label=''*, *show_colorbar=True*, *cmap=<matplotlib.colors.ListedColormap object>*, *min_cbar_elem=5*, *max_cbar_elem=25*, *show_legend=False*, *counterclock=True*, *startangle=90*, *hrd_inset=True*, *show_hrd_ticks_and_labels=False*, *show_total_mass=True*, *show_surface=False*, *output_fname='chemical_profile'*, *anim_fmt='.mp4'*, *cbar_orientation='vertical'*`)`

Creates chemical profile diagram.

Represent the model of a stellar object as a circle with radius raxis. The circle contains nested pie charts that show the composition of the star at a certain mass or radius coordinate. Requires MESA profile files that contain the profiles of certain isotopes, such as "h1" and the corresponding MESA history file.

Parameters

m [mesaPlot object] Already loaded a history file

time_ind [int or tuple (start_index, end_index, step=1) or (start_index, end_index, step)] If int: create the plot at the index time_ind. If tuple: create an animation from start index to end index with intervals of step.

isotope_list [None or list or np.array of str] Containing the names of isotopes to be included.

num_rings [int] Default -1, if greater than -1, limit the number of nested pie charts to this number

scale [float] Value to scale the circle to. If negative, use the maximum value of the raxis.

width [float] Minimum width of a nested pie chart ring.

raxis [str] Default axis to use as radius of the circle.

show_ring_annotations: boolean If set, add concentric rings on top of the nested pie charts.

fps [int] Number of frames per second for the animation.

fig: Figure object If set, plot on existing figure.

ax [Axes object] If set, plot on provided axis.

show_time_label [boolean] If set, insert a label that gives the age of the stellar object (in Myr).

time_label_loc [tuple] Location of the time label on the plot as fraction of the maximal size.

time_unit [str] Valid astropy time unit, default Myr.

fig_size [tuple] Size of the figure in inches.

axis_lim [float] Value to set for the maximum limit of the x and y axis.

axis_label [str] Label of the x and y axis.

show_colorbar [boolean] If set, add a colorbar that gives the isotopes and corresponding colors.

cmap [str or matplotlib.colors.ListedColormap] colormap to use for the isotopes

min_cbar_elem [int] Minimum number of isotopes in a colorbar

max_cbar_elem [int] Maximum number of isotopes per colorbar

cbar_orientation [string, one of "vertical" or "horizontal"] Orientation of the colorbar. If horizontal, it is placed below the figure, if vertical, it is placed to the right.

counterclock [boolean] If set, plot the isotopes counterclockwise.

startangle [int or float] Angle in degrees from the horizontal line at which to start plotting the isotopes.

hrd_inset [boolean] If set, add an inset HRD where the location of the current model is indicated with a circle.

show_hrd_ticks_and_labels [Boolean] If set, display the axis ticks and labels of the inset HRD

show_total_mass [boolean] Default False, display the value of the total mass of the model below the circle.

show_surface [boolean] Default True, if set, show the outer boundary of the stellar object.

show_grid [boolean] Default False, if set, add additional axes in crosshair form.

output_fname [str] Name of the output file.

anim_fmt [str] Format to use for saving an animation.

Returns

fig, ax

colorbar

`tulips.tulips.`**`colorbar`**`(`*mappable*, *ax=None*, *size=None*, *pad=None*, *cax=None*, *orientation='vertical'*, ***kwargs*`)`

create_elem_colorbars

`tulips.tulips.`**`create_elem_colorbars`**`(`*isotope_list*, *ax*, *cmap=<matplotlib.colors.ListedColormap object>*, *min_cbar_elem=5*, *max_cbar_elem=25*, *cbar_orientation='vertical'*`)`

Create colorbar for isotopes

Create colorbars containing labels of isotopes for chemical_profile plots

Parameters

isotope_list [list] List of isotopes to plot

ax [matplotlib.Axes object] The current axis to plot on

cmap [str or matplotlib.colors.ListedColormap] colormap to use for the isotopes

min_cbar_elem [int] Minimum number of isotopes in a colorbar

max_cbar_elem [int] Maximum number of isotopes per colorbar

cbar_orientation [string, one of "vertical" or "horizontal"] Orientation of the colorbar. If horizontal, it is placed below the figure, if vertical, it is placed to the right.

Returns

——-

cbar_list [list] List of colorbar objects

energy_and_mixing

`tulips.tulips.`**`energy_and_mixing`**`(`*m*, *time_ind=-1*, *show_mix=False*, *show_mix_legend=True*, *raxis='star_mass'*, *fps=10*, *fig=None*, *ax=None*, *show_time_label=True*, *time_label_loc=()*, *time_unit='Myr'*, *fig_size=(5.5, 4)*, *axis_lim=-99*, *axis_label=''*, *axis_units=''*, *show_colorbar=True*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*, *cmin=-10*, *cmax=10*, *cbar_label=''*, *theta1=0*, *theta2=360*, *hrd_inset=True*, *show_hrd_ticks_and_labels=False*, *show_total_mass=False*, *show_surface=True*, *show_grid=False*, *output_fname='energy_and_mixing'*, *anim_fmt='.mp4'*, *time_scale_type='model_number'*`)`

Create energy and mixing diagram.

Represent the model of a stellar object as circle with radius raxis. The circle is divided into rings whose color reflect how much energy is generated or lost from the star. Optionally, hashed areas representing mixing regions can be added. This corresponds to a "2D Kippenhahn plot". Requires MESA history files that contain burning_regions and optionally mixing_regions.

Parameters

m [mesaPlot object] Already loaded a history file.

time_ind [int or tuple (start_index, end_index, step=1) or (start_index, end_index, step)] If int: create the plot at the index time_ind. If tuple: create an animation from start index to end index with

intervals of step.

show_mix: boolean If set, add hatches for convection and overshooting zones in the star.

show_mix_legend: boolean If set, show a legend for the mixing types.

raxis [str] Default axis to use as radius of the circle.

fps: int Number of frames per second for the animation

fig [Figure object] If set, plot on existing figure.

ax [Axes object] If set, plot on provided axis.

show_time_label [boolean] If set, insert a label that gives the age of the stellar object.

time_label_loc [tuple] Location of the time label on the plot as fraction of the maximal size.

time_unit [str] Valid astropy time unit, default Myr.

fig_size [tuple] Size of the figure in inches.

axis_lim [float] Value to set for the maximum limit of the x and y axis.

axis_label [str] Label of the x and y axis.

axis_units [str] Astropy unit for the x and y axis.

show_colorbar [boolean] If set, add a colorbar corresponding to the property shown.

cmap [str or matplotlib.colors.ListedColormap] Colormap to use for the property

cmin [float] Minimum value to set for the colorbar.

cmax [float] Maximum value to set for the colorbar, if smaller or equal to cmin, use the minimum and maximum values of property_name instead.

cbar_label [str] Label to set for the colorbar.

theta1 [int or float] Start angle for the wedge.

theta2: int or float End angle for the wedge.

hrd_inset [boolean] If set, add an inset HRD where the location of the current model is indicated with a circle.

show_hrd_ticks_and_labels [Boolean] If set, display the axis ticks and labels of the inset HRD

show_total_mass [boolean] Default False, display the value of the total mass of the model below the circle.

show_surface [boolean] Default True, if set, show the outer boundary of the stellar object.

show_grid: boolean Default False, if set, add additional axes in crosshair form.

output_fname [str] Name of the output file.

anim_fmt [str] Format to use for saving an animation.

time_scale_type [str] One of *model_number*, *linear*, or *log_to_end*. For *model_number*, the time follows the moment when a new MESA model was saved. For *linear*, the time follows linear steps in star_age. For *log_to_end*, the time axis is tau = log10(t_final - t), where t_final is the final star_age of the model.

Returns

fig, ax

find_closest

`tulips.tulips.`**`find_closest`**`(`*ary*, *value*`)`

find_profile

`tulips.tulips.`**`find_profile`**`(`*m*, *time_ind=0*`)`

Load closest MESA profile.

Loads the MESA profil closest to the time_ind given.

Parameters

m [object] mesaPlot object

time_ind: int index of mesaPlot history

Returns

m.prof [profile] MESA profile

get_isotopes

`tulips.tulips.`**`get_isotopes`**`(`*m*`)`

Get list of isotopes

Get the complete list of isotopes in a MESA model. Requires MESA profiles to be available.

Parameters

m: mesaPlot object The current MESA model

get_isotopes_from_prof

`tulips.tulips.`**`get_isotopes_from_prof`**`(`*prof*`)`

Get list of isotopes

Get the complete list of isotopes in a MESA model. Requires MESA profiles to be available.

Parameters

prof: mesaPlot.prof object The current MESA profile

make_pie_composition_plot

`tulips.tulips.`**`make_pie_composition_plot`**`(`*ax*, *prof*, *num_rings=0*, *scale=11.0*, *width=0.01*, *startangle=90*, *isotope_list=None*, *cmap=None*, *show_colorbar=True*, *min_cbar_elem=5*, *max_cbar_elem=25*, *counterclock=True*, *raxis='mass'*, *boundary=0*, *log_low_lim=- 2.1*, *cbar_orientation='vertical'*`)`

Create composition plot.

Create a composition plot showing the percentage of composition for each element in each layer of the star.

Parameters

ax [axis object]

prof: mesaPlot profile object

num_rings [int] Number of rings to plot.

scale [float] Value to scale the plot to.

width [float] Width of each nested piechart.

startangle [float] Value of the angle to start with for the piecharts.

isotope_list [None or list] List of isotopes to take into account.

counterclock [bool, optional] Default: True, specify fractions direction, clockwise or counter-clockwise.

show_colorbar [boolean] If set, add a colorbar that gives the isotopes and corresponding colors.

cmap [str or matplotlib.colors.ListedColormap] colormap to use for the isotopes

min_cbar_elem [int] Minimum number of isotopes in a colorbar

max_cbar_elem [int] Maximum number of isotopes per colorbar

cbar_orientation [string, one of "vertical" or "horizontal"] Orientation of the colorbar. If horizontal, it is placed below the figure, if vertical, it is placed to the right.

Returns

——-

elem_list, artists

make_property_plot

`tulips.tulips.`**`make_property_plot`**`(`*ax*, *prof*, *property_name='logRho'*, *raxis='mass'*, *num_rings=- 1*, *cmin=-5*, *cmax=10*, *log=False*, *cmap='plasma'*, *theta1=0*, *theta2=360*, *log_low_lim=1e-20*`)`

Create property plot.

Plot a circle containing concentric rings with a color that corresponds to the evolution of a property.

Parameters

ax [axis object]

prof: mesaPlot profile object

property_name: string Default logRho, existing quantity in the MESA profile.

raxis [string] Default axis to use as radius representation. Any linear property can be used instead (for example radius).

num_rings [int] Default -1, if greater than -1, limit the number of rings to this number.

theta1 [int or float] Start angle for the wedge (in degrees).

theta2 [int or float] End angle for the wedge (in degrees).

cmap [str or matplotlib.colors.ListedColormap] Colormap to use for the property

cmin [float] Minimum value to set for the colorbar.

cmax [float] Maximum value to set for the colorbar, if smaller or equal to cmin, use the minimum and maximum values of property_name instead.

log [boolean] If set, show the natural logarithm of the property.

log_low_lim [float] Value to replace zero and negative values in a property with

Returns

——-

List of artists created in the plot

num_cbar_elem

`tulips.tulips.`**`num_cbar_elem`**(*num_elems*, *min_cbar_elem=5*, *max_cbar_elem=25*)

Calculate number of colorbars

Based on the maximum and minum number of elements per colorbar, compute the number of colorbars and elements per colorbar that will be displayed

Parameters

num_elems [int] Number of isotopes to add to the colorbar

min_cbar_elem [int] Minimum number of isotopes in a colorbar

max_cbar_elem [int] Maximum number of isotopes per colorbar

Returns

num_cbar [int] Number of colorbars needed

elem_per_cbar [list] List of the number of isotopes per colorbar

perceived_color

`tulips.tulips.`**`perceived_color`**(*m*, *time_ind=- 1*, *raxis='log_R'*, *fps=10*, *fig=None*, *ax=None*, *show_time_label=True*, *time_label_loc=()*, *time_unit='Myr'*, *fig_size=(5.5, 4)*, *axis_lim=- 99*, *axis_label=''*, *theta1=0*, *theta2=360*, *hrd_inset=True*, *show_hrd_ticks_and_labels=False*, *show_total_mass=True*, *show_surface=True*, *output_fname='perceived_color'*, *anim_fmt='.mp4'*, *time_scale_type='model_number'*)

Create perceived color diagram of a stellar model.

Represent the model of a stellar object as circle with radius *raxis*. The color of the circle corresponds to its color as perceived by the human eye by assuming black body radiation at a certain effective temperature. Requires a MESA history file.

Parameters

m [mesaPlot object] Already loaded a history file.

time_ind [int or tuple (start_index, end_index, step=1) or (start_index, end_index, step)] If int: create the plot at the index *time_ind*. If tuple: create an animation from start index to end index with intervals of step.

raxis [str] Default axis to use as radius of the circle.

fps [int] Number of frames per second for the animation.

fig [Figure object] If set, plot on existing figure.

ax [Axes object] If set, plot on provided axis.

show_time_label [boolean] If set, insert a label that gives the age of the stellar object (in Myr).

time_label_loc [tuple] Location of the time label on the plot as fraction of the maximal size.

time_unit [str] Valid astropy time unit.

fig_size [tuple] Size of the figure in inches.

axis_lim [float] Value to set for the maximum limit of the x and y axis.

axis_label [str] Label of the x and y axis.

theta1: int or float Start angle for the wedge.

theta2 [int or float] End angle for the wedge.

hrd_inset [boolean] If set, add an inset HRD where the location of the current model is indicated with a circle.

show_hrd_ticks_and_labels [Boolean] If set, display the axis ticks and labels of the inset HRD

show_total_mass [boolean] If set, display the value of the total mass of the model in the bottom right corner.

show_surface [boolean,] If set, show the outer boundary of the stellar object.

output_fname [str] Name of the output file.

anim_fmt [str] Format to use for saving an animation.

time_scale_type [str] One of *model_number*, *linear*, or *log_to_end*. For *model_number*, the time follows the moment when a new MESA model was saved. For *linear*, the time follows linear steps in star_age. For *log_to_end*, the time axis is tau = log10(t_final - t), where t_final is the final star_age of the model.

Returns

fig, ax

property_profile

`tulips.tulips.`**`property_profile`**`(`*m*, *time_ind=- 1*, *property_name='logRho'*, *num_rings=- 1*, *raxis='mass'*, *log=False*, *log_low_lim=1e-20*, *fps=10*, *fig=None*, *ax=None*, *show_time_label=True*, *time_label_loc=()*, *time_unit='Myr'*, *fig_size=(5.5, 4)*, *axis_lim=- 99*, *axis_label=''*, *show_colorbar=True*, *cmap='plasma'*, *cmin=0*, *cmax=0*, *cbar_label=''*, *theta1=0*, *theta2=360*, *hrd_inset=True*, *show_hrd_ticks_and_labels=False*, *show_total_mass=True*, *show_surface=True*, *show_grid=False*, *output_fname='property_profile'*, *anim_fmt='.mp4'*`)`

Create property profile diagram.

Represent the model of a stellar object as circle with radius raxis. The circle is divided into rings whose color reflect the values of a physical property of the model. Requires MESA profile files that contain this property and the corresponding MESA history file.

Parameters

m [mesaPlot object] Already loaded a history file

time_ind [int or tuple (start_index, end_index, step=1) or (start_index, end_index, step)] If int: create the plot at the index time_ind. If tuple: create an animation from start index to end index with intervals of step.

property_name [string] Property of a MESA profile to be shown as colors.

num_rings [int] Default -1, if greater than -1, limit the number of rings to this number.

raxis [str] Default axis to use as radius of the circle.

log [boolean] If set, show the natural logarithm of the property.

log_low_lim [float] Value to replace zero and negative values in a property with

fps [int] Number of frames per second for the animation.

fig [Figure object] If set, plot on existing figure.

ax [Axes object] If set, plot on provided axis.

show_time_label [boolean] If set, insert a label that gives the age of the stellar object (in Myr).

time_label_loc [tuple] Location of the time label on the plot as fraction of the maximal size.

time_unit [str] Valid astropy time unit, default Myr.

fig_size [tuple] Size of the figure in inches.

axis_lim [float] Value to set for the maximum limit of the x and y axis.

axis_label [str] Label of the x and y axis.

show_colorbar [boolean] If set, add a colorbar corresponding to the property shown.

cmin [float] Minimum value to set for the colorbar.

cmax [float] Maximum value to set for the colorbar, if smaller or equal to cmin, use the minimum and maximum values of property_name instead.

cmap [str or matplotlib.colors.ListedColormap] Colormap to use for the property

cbar_label [str] Label to set for the colorbar.

theta1 [int or float] Start angle for the wedge.

theta2 [int or float] End angle for the wedge.

hrd_inset [boolean] If set, add an inset HRD where the location of the current model is indicated with a circle.

show_hrd_ticks_and_labels [Boolean] If set, display the axis ticks and labels of the inset HRD

show_total_mass: boolean Default False, display the value of the total mass of the model below the circle

show_surface [boolean] Default True, if set, show the outer boundary of the stellar object.

show_grid [boolean] Default False, if set, add additional axes in crosshair form.

output_fname [str] Name of the output file.

anim_fmt [str] Format to use for saving an animation.

Returns

fig, ax

rescale_time

`tulips.tulips.`**`rescale_time`**`(`*indices*, *m*, *time_scale_type='model_number'*`)`

Rescale the time.

Rescale time indices depending on the time_type.

Parameters

indices [np.array or list of int] Containing selected indices.

m [mesa Object]

time_scale_type [str] One of *model_number*, *linear*, or *log_to_end*. For *model_number*, the time follows the moment when a new MESA model was saved. For *linear*, the time follows linear steps in star_age. For *log_to_end*, the time axis is tau = log10(t_final - t), where t_final is the final star_age of the model.

Returns

ind_select [list] New list of indices that reflect the rescaling in time.

set_axis_ticks_and_labels

`tulips.tulips.`**`set_axis_ticks_and_labels`**`(`*ax*, *raxis='star_mass'*, *axis_label=''*`)`

Format axis ticks.

Format the axis ticks such that no negative values are shown.

Parameters

ax [matplotlib axis object]

raxis: str Valid column name for a MESA history file. This sets the value used for the outer radius of the star.

axis_label [str] User defined axis label.

Returns

None

teff2rgb

`tulips.tulips.teff2rgb(t_ary)`

Convert effective temperature to rgb colors.

Convert effective temperature to rgb colors using colorpy.

Parameters

t_ary [array] Temperature array in K.

Returns

rgb_list: array Array with rgb colors.

too_dark

`tulips.tulips.too_dark(color_ary)`

Check if text too dark

Check whether dark text can be overlayed on colors in an array of RGBA colors.

Parameters

color_ary [np.array or list of RGBA color arrays]

Returns

final_ary [list] New list of Booleans.

CHAPTER

TWO

TULIPS DIAGRAMS

CHAPTER

THREE

ADDITIONAL OPTIONS

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

FIVE

CONTRIBUTING

If you wish to submit a new feature or bug fix please send a pull request.

CHAPTER

SIX

ACKNOWLEDGMENTS

tulips makes use of the open-source python modules mesaPlot (Farmer, 2019), matplotlib (Hunter, 2007), numpy (van der Walt et al., 2011), Colorpy (Mark Kness), astropy (Astropy Collaboration et al., 2013, 2018), CMasher (van der Velden, 2020), and ipython/jupyter (Perez and Granger, 2007). Logo design: A. Faber. Documentation: I. de Langen.

PYTHON MODULE INDEX

t

`tulips.tulips`, 38

INDEX

A

add_inset_hrd() (*in module tulips.tulips*), 38
add_ring_annotations() (*in module tulips.tulips*), 39
add_time_label() (*in module tulips.tulips*), 40
animated_hist_comp() (*in module tulips.tulips*), 40
animated_hist_comp_test() (*in module tulips.tulips*), 40

C

check_time_indeces() (*in module tulips.tulips*), 41
chem_elem_notation() (*in module tulips.tulips*), 41
chemical_profile() (*in module tulips.tulips*), 41
colorbar() (*in module tulips.tulips*), 43
create_elem_colorbars() (*in module tulips.tulips*), 43

E

energy_and_mixing() (*in module tulips.tulips*), 43

F

find_closest() (*in module tulips.tulips*), 45
find_profile() (*in module tulips.tulips*), 45

G

get_isotopes() (*in module tulips.tulips*), 45
get_isotopes_from_prof() (*in module tulips.tulips*), 45

M

make_pie_composition_plot() (*in module tulips.tulips*), 46
make_property_plot() (*in module tulips.tulips*), 46
module
    tulips.tulips, 38

N

num_cbar_elem() (*in module tulips.tulips*), 47

P

perceived_color() (*in module tulips.tulips*), 47
property_profile() (*in module tulips.tulips*), 49

R

rescale_time() (*in module tulips.tulips*), 50

S

set_axis_ticks_and_labels() (*in module tulips.tulips*), 50

T

teff2rgb() (*in module tulips.tulips*), 51
too_dark() (*in module tulips.tulips*), 51
tulips.tulips
    module, 38