hoki Documentation

Release 0.1.dev

H. F. Stevance

Aug 06, 2019

INTRODUCTION

CHAPTER

ONE

WHAT IS HOKI?

1.1 Bridging the gap between theory and observations

What is BPASS?

The **Binary Populations And Spectral Synthesis (BPASS)** code simulates stellar populations and follows their evolution until death. Including binary evolution is **crucial to correctly interpreting** astronomical observations. The detailed follow-up of the stellar evolution within the code allows the retreival of important information such as supernova and gravitational wave event rates, giving us the ability to understand the properties of the stellar populations that are the precursors of these events.

Historically, theoretical models have been released to the community with the expectation that users would create their own data handling scripts as they perform the analysis. This was in line with a research culture where most people had their own code, resulting in a duplication of effort and creating the potential for a reproducibility crisis, as most personal scripts are not shared with the community.

In order to facilitate the application of BPASS to a wide range of scientific investigations, we have developped the tools necessary for observers to take full advantage of our models in a stream-lined, intuitive manner.

> **Hoki is a dedicated python package designed to provide a user friendly interface to the BPASS models in order to bridge the gap between theory and observations.**

The varsatility of **Hoki** allows *[and then list all the fancy stuff it can do!]*

1.2 Built with Data Analysis in mind

MABE I NEED TO DO SOMETHING HERE

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

1.3 Sell that ish honey

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

CHAPTER

TWO

QUICK START

2.1 Set-up

2.1.1 Install Hoki

To install the most recent stable release of hoki you can use:

```
sudo pip3 install --user hoki
```

If you are feeling adventurous and want the most recent (in development) version of hoki, you can clone our GitHub repository

2.1.2 Download the BPASS models

BPASS models can be downloaded from the BPASS:

Download	Hoki Compatible	Release Date	Reference
BPASSv2.2	Yes	July 2018	Stanway & Eldridge (2018)
BPASSv2.1	?	October 2017	Eldridge, Stanway, Xiao, et al. (2017)
BPASSv2.0	?	–	–
BPASSv1	?	–	–

Note: `hoki` is dedicated to being an interface with the BPASS models, but given the substancial size of the entire set of models, they are not downloaded upon installation of `hoki`, and you should download the models you want to work on.

2.2 Loading in Data

2.2.1 Stellar Model Outputs

W.I.P

2.2.2 Stellar Population Outputs

A stellar output file can be loaded in using the `population_output()` function in the `load` module.

```python
from hoki import load

output = load.population_output('path')
```

The function will figure out based on the file name what data is being loaded in and will return the appropriate data format: `pandas.DataFrames` in most cases, apart from HR diagrams, which have their own `HRDiagrams` class – because they're quite a complex data structure.

Tip: The full details of the he stellar population outputs can be found in the BPASS manual.

Here we summarise the shape of the outputs (51 time bins) for a given metalicity and IMF.

Output	File Name Root	Shape
Massive star type numbers	numbers	51 x 21
Supernova Rates	supernova	51 x 18
Energy and elemental yields	yields	51 x 9
Stellar mass remaining at the end	starmass	51 x 3
HR diagrams	hrs	51 x 100 x 100 x 3 x 3

Note: These models are calculated for 9 IMFs and 13 metalicities.

2.2.3 Spectral Synthesis Outputs

See also:

For dedicated tutorials about specific aspecs of hoki and BPASS, check our Cook Book section in the side bar!

CHAPTER

THREE

1. TRANSIENT RATES

3.1 1.1 Initial imports

[1]:
```
from hoki import load
import pandas as pd
import matplotlib.pyplot as plt

# Feel free to use your own style sheet
plt.style.use('tuto.mplstyle')

%matplotlib inline
```

3.2 1.2 Loading in the data and initial set-up

Transient rates in BPASS are one of the several types of stellar population outputs. We can simply use the `hoki.load.population_ouput()` function to laod the data. The function will automatically know what type of data it is loading from the file name and create an appropriate data frame to put it in.

If you are not familiar with `pandas` and `pandas.DataFrame`, you should pick it up easily. If you want to look into them more check out this Data Camp article.

[2]:
```
# Loading the binary and single star population transient rates.
# Note we chose this particular IMF and metallicity in order to reproduce the plot
# shown on the left hand sife of Figure 1 in Eldridge et al. 2018

bin_rates = load.population_output('supernova-bin-imf135_300.z002.dat')
sin_rates = load.population_output('supernova-sin-imf135_300.z002.dat')
```

Let's have a look at one of our data frames

[9]:
```
bin_rates.head()
```

[9]:
```
   age_log   Ia  IIP   II        Ib   Ic  LGRB     PISNe  low_mass  e_Ia  \
0      6.0  0.0  0.0  0.0  0.000000  0.0   0.0  0.000000       0.0   0.0
1      6.1  0.0  0.0  0.0  0.000000  0.0   0.0  0.000000       0.0   0.0
2      6.2  0.0  0.0  0.0  0.000000  0.0   0.0  0.000000       0.0   0.0
3      6.3  0.0  0.0  0.0  0.000000  0.0   0.0  0.000000       0.0   0.0
4      6.4  0.0  0.0  0.0  3.847896  0.0   0.0  5.109734       0.0   0.0

   e_IIP  e_II      e_Ib  e_Ic  e_LGRB  e_PISNe  e_low_mass     age_yrs
0    0.0   0.0  0.000000   0.0     0.0  0.00000         0.0  1122019.00
```

(continues on next page)

(continued from previous page)

```
1    0.0   0.0  0.000000   0.0     0.0  0.00000          0.0   290520.12
2    0.0   0.0  0.000000   0.0     0.0  0.00000          0.0   365743.12
3    0.0   0.0  0.000000   0.0     0.0  0.00000          0.0   460443.62
4    0.0   0.0  0.496761   0.0     0.0  0.80792          0.0   579664.00
```

Most column names are pretty self-explanatory, appart from `low_mass`, which just detones the rate of low-mass supernovae (< 2M_sun). The `age_yrs` bin is th size of each time bin in years; indeed since BPASS works in log(time) space, each time bin has a different width in years.

If you need more detail on each of these rates, you should have a look at the BPASS user manual.

```
[4]: # The last time bin in BPASS does some weird stuff so it's better to just ignore it.
bin_rates = bin_rates[:-1]
sin_rates = sin_rates[:-1]
```

```
[10]: # We are going to use the log_age and the size of the bin in years a lot, so I'm just␣
↪renaming them for ease.
age = bin_rates.age_log.values
bin_size = bin_rates.age_yrs.values
```

3.2.1 Core Collapse Supernovae rates

Core collapse supernovae comprise the type IIP, II, Ib and Ic. To get the total rate we need to sum these columns as well as put the single star and binary populations together.

```
[27]: ccsne = ( bin_rates[['IIP', 'II', 'Ib', 'Ic']].sum(axis=1) +
          sin_rates[['IIP', 'II', 'Ib', 'Ic']].sum(axis=1) )
```

3.2.2 Other transients

For type Ia SNe, LGRBs and PISNe, all we need to do is combine single and binary star populations.

```
[28]: typeIa = (bin_rates.Ia.values + sin_rates.Ia.values)
lgrbs = (bin_rates.LGRB.values + sin_rates.LGRB.values)
pisne = bin_rates.PISNe.values + sin_rates.PISNe.values
```

3.3 1.3 Getting the Units right

We want to plot our rates as **events/M_sun/year**, this means we need to normalise by the total mass and the number of years in each time bin.

BPASS calulates stellar populations with 10 M_sun and we've already put the bin size in years in a convenient variable called `bin_size`.

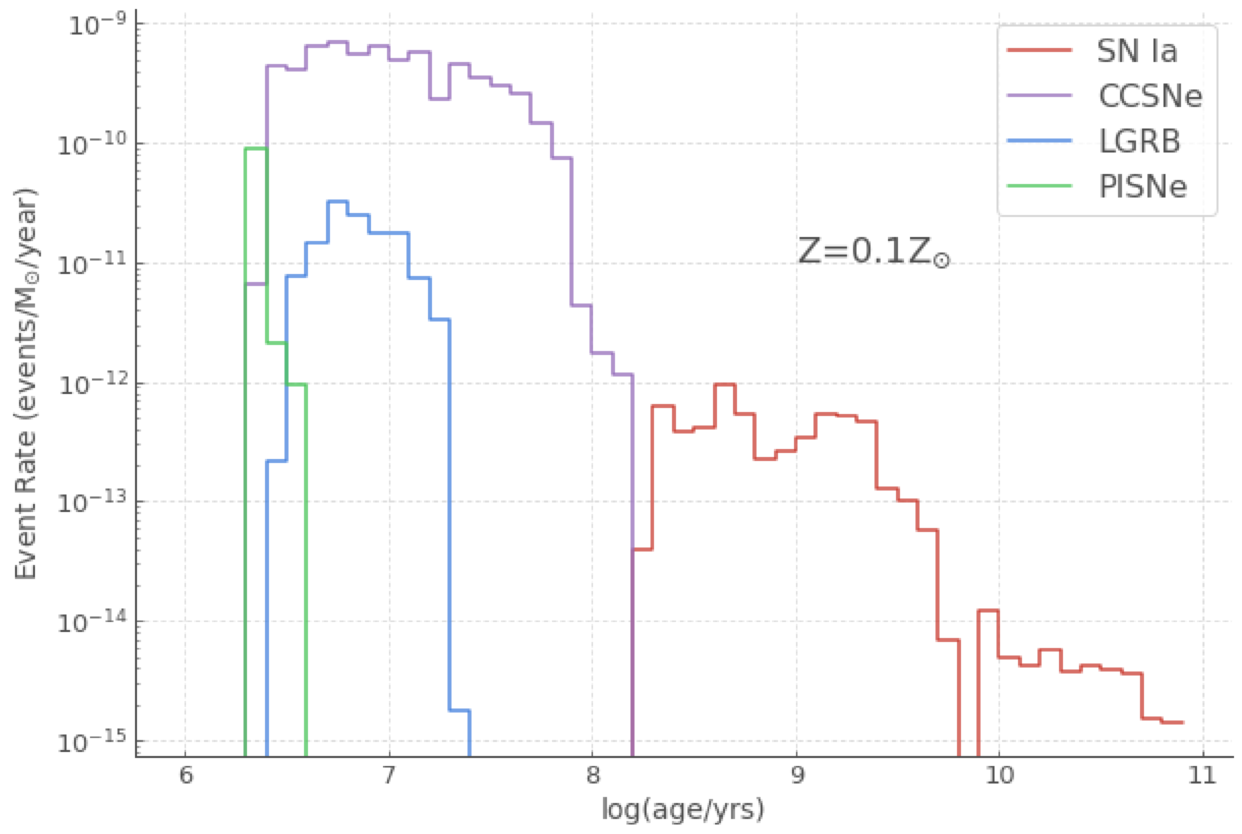
```
[36]: ccsne_norm = ccsne/bin_size/(10**6)
typeIa_norm = typeIa/bin_size/(10**6)
lgrbs_norm = lgrbs/bin_size/(10**6)
pisne_norm = pisne/bin_size/(10**6)
```

3.4 1.4 Plotting the transient rates

Now we have everything we need to plot our transient rates! The only trick here is to remember to **log rate axis** to be able to see everything.

[38]:
```
plt.figure(figsize = (10,7))

plt.step(age, typeIa_norm, label='SN Ia')
plt.step(age, ccsne_norm, label='CCSNe')
plt.step(age, lgrbs_norm, label='LGRB')
plt.step(age, pisne_norm, label='PISNe')

plt.yscale("log")

plt.text(9, 10**(-11), r"Z=0.1Z$_{\odot}$", fontsize=18)

plt.ylabel(r"Event Rate (events/M$_{\odot}$/year)")
plt.xlabel("log(age/yrs)")
plt.legend(fontsize=16)
```

[38]:
```
<matplotlib.legend.Legend at 0x7fbd6bffd4e0>
```



3.4.1 *That's not quite the same as figure 1 - PISNe and CCSNe in particular ??*

CHAPTER

FOUR

2. HR DIAGRAMS

4.1 2.1 Initial imports

[9]:
```
from hoki import load
import pandas as pd
import matplotlib.pyplot as plt


plt.style.use('tuto.mplstyle')
```

4.2 2.2 Loading in the data

4.2.1 Information about the tools you're about to use.

HR diagram data can be loaded using the `hoki.load.population_output()` function. This function will automatically know that you are trying to load an HR diagram from the name of the text file.

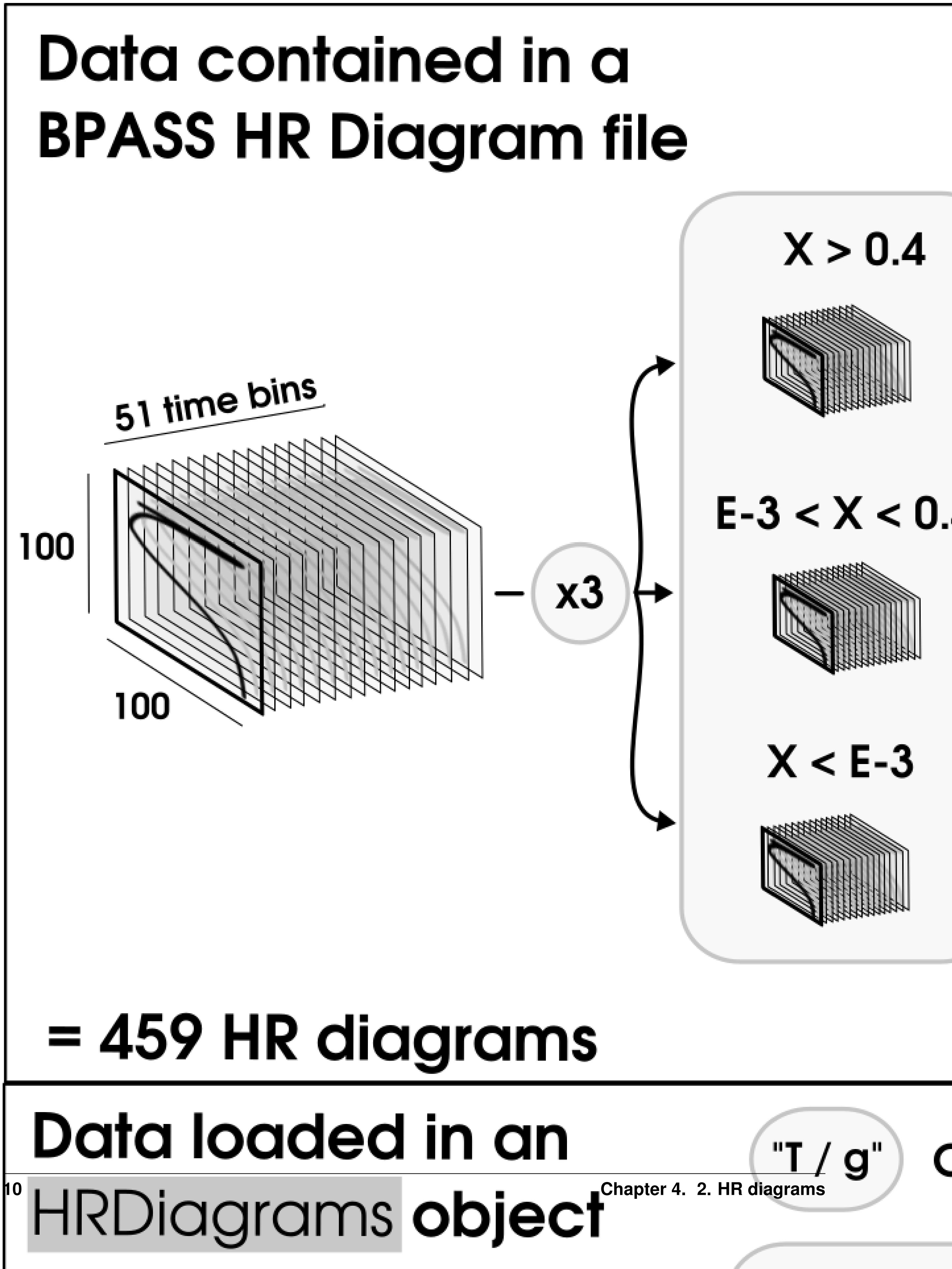
Important: You should note that when loading an HR diagram you need to specify what type of HR diagram you want to load form the file. The HR diagram files are large and quite complex - see left hand side of the diagam below - so we'll be loading in one type of HR diagram at a time.

You can choose from 3 options:

- `'TL'`: For a temperature/Luminosity HR diagram
- `'Tg'`: For a temperature/surface gravity HR diagram
- `'TTG'`: For a temperature/(temperature**4/surface gravity) HR diagram

These options are there for 2 reasons:

- 1. They tell the object which segment of the text file to load in
- 2. They tell the plotting function what grid to create in order to plot the contours (also picks the right axis labels).


Data contained in a
BPASS HR Diagram file
51 time bins
100
100
x3
X > 0.4
E-3 < X < 0.
X < E-3
= 459 HR diagrams
Data loaded in an
HRDiagrams object
"T / g"

4.2.2 Loading the HR diagrams

```
[10]: # Loading the HR diagrams for signle stars and binary star populations
sin_hr_diagram = load.population_output('hrs-sin-imf135_300.z002.dat', hr_type = 'TL')
bin_hr_diagram = load.population_output('hrs-bin-imf135_300.z002.dat', hr_type = 'TL')
```

4.2.3 Loading some observational data

We're going to want to plot some observational data to compare to our models. A nice way to do that is to use `pandas`.

```
[11]: # Loading the observational data of Upper Scorpio
usco = pd.read_csv('USco.dat', sep='\t', engine='python', names=['Temperature',
↪'Luminosity'])
```

```
[12]: usco.head()
```

```
[12]:    Temperature  Luminosity
0         3.756        0.38
1         3.742        0.34
2         3.742        0.36
3         3.750        0.08
4         3.728        0.27
```

4.3 2.3 Plotting HR diagrams

Now that we have loaded in our data, it's time to create some visualisation. The advantage of using the `HRDiagrams` object is that it comes with an **easy-to-use plotting method** that is highly **versatile** and designed to create **publication-ready figures**.

Let's run through a couple of ways you can use this tool and make it your own.

4.3.1 Making a single plot

First we are going to make just one HR diagram plot. The plotting method returns the plote object, which **allows you to customize you plot**.

For example you can **add observational data**, some **text**, a **title**, a **legend** and redefine the axis **limits** to create a more effective visualisation.

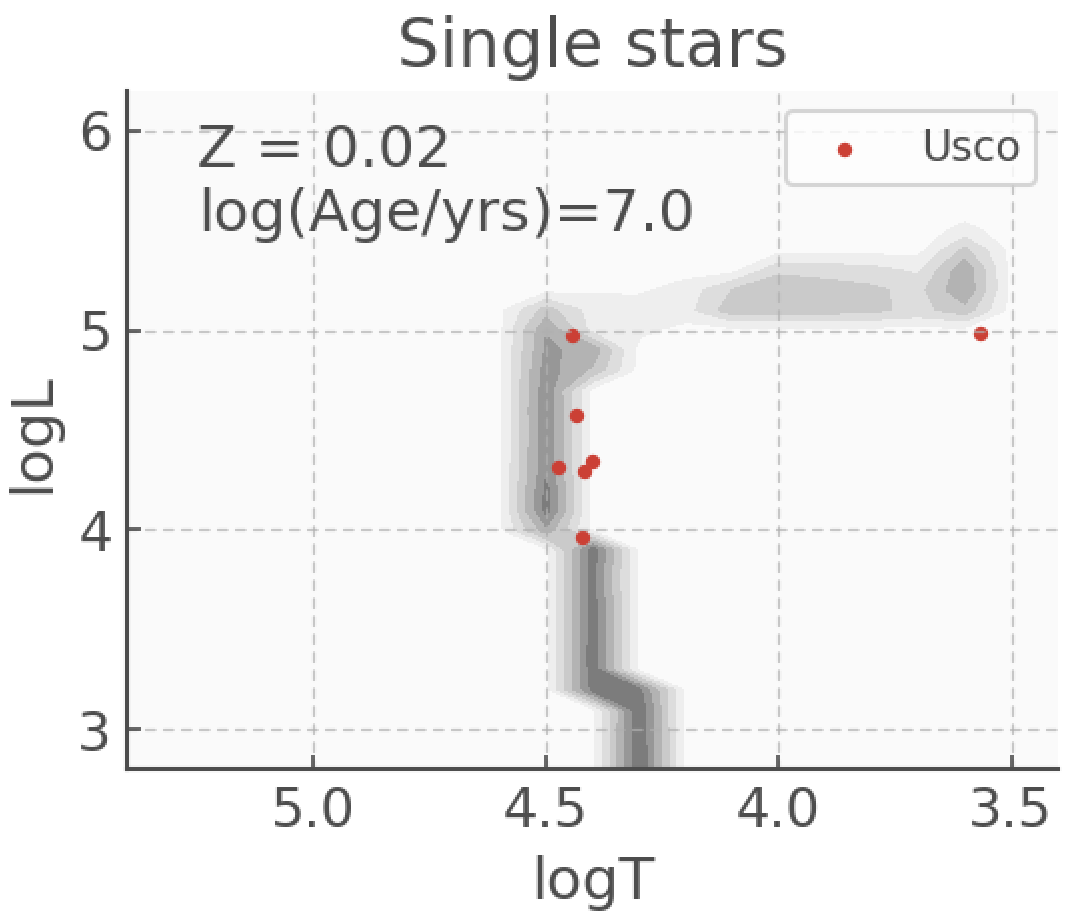
```
[13]: # Just maing sure teh figure size is sensible
plt.figure(figsize=(4,3))

# Plotting the hr_diagram (all hydrogen abundances added up) at the 10 million year␣
↪time bin
my_plot = sin_hr_diagram.plot(log_age=7)

# Addind the observational data of Upper Scorpio
my_plot.scatter(usco.Temperature, usco.Luminosity, label='Usco')

# Customizing the visualisation
my_plot.set_xlim([5.4, 3.4])
my_plot.set_ylim([2.8, 6.2]) # better limits
```

(continues on next page)

(continued from previous page)

```
my_plot.text(5.25, 5.5, "Z = 0.02 \nlog(Age/yrs)=7.0", fontsize = 14) # Informative␣
↪text
my_plot.legend() # a legend
my_plot.set_title('Single stars') # A title
```

```
[13]: Text(0.5, 1.0, 'Single stars')
```



4.3.2 2.3.2 Multiple plots

There might be times you want to put multiple plots on the same figure. The good news is the `HRDiagrams.plot()` allows you to do that intuitively! There are actually more than one way to do it, depending on how you like making your subplots in Python, either with `add_subplot()` or `subplots()`.

Here is the same HR diagram as presented above, but with the Single star population on the left and the Binary star population on the right.

a) Using ``subplots()`` syntax

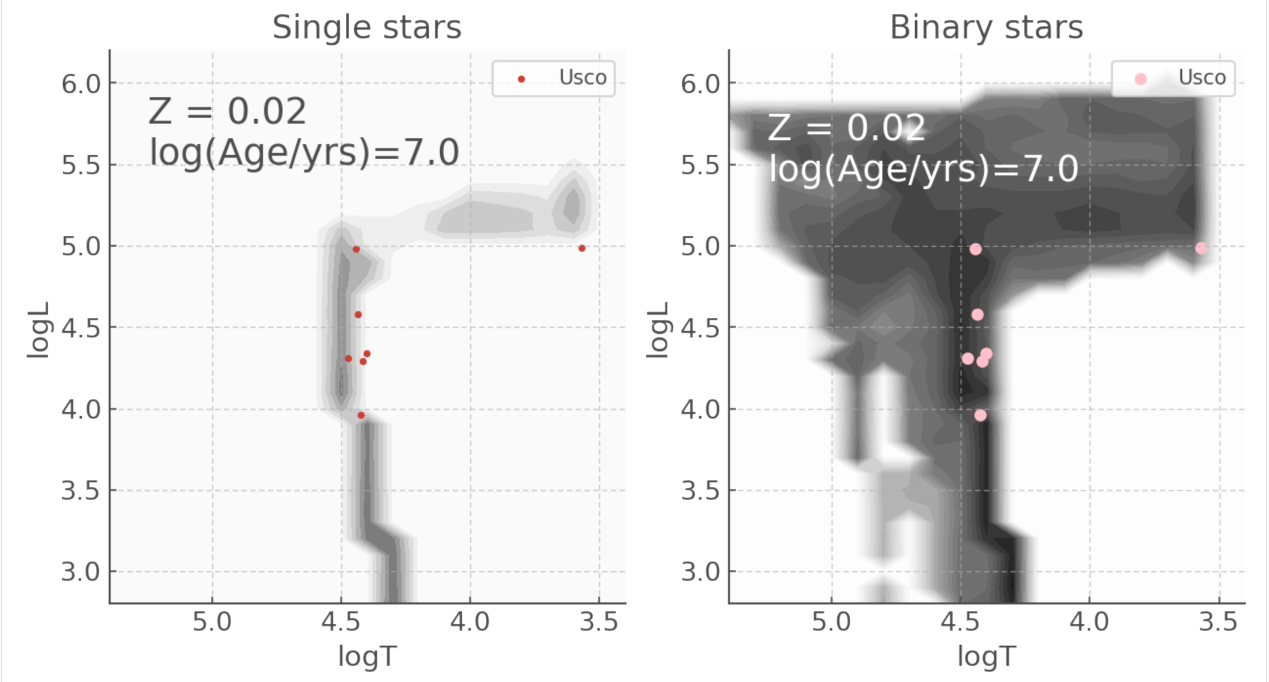
```
[19]: fig, ax = plt.subplots(1,2, figsize=(10,5))

sin_hr_diagram.plot(log_age=7, loc=ax[0])
ax[0].scatter(usco.Temperature, usco.Luminosity, label='Usco')
ax[0].set_xlim([5.4, 3.4])
```

(continues on next page)

(continued from previous page)

```
ax[0].set_ylim([2.8, 6.2])
ax[0].set_title('Single stars')
ax[0].text(5.25, 5.5, "Z = 0.02 \nlog(Age/yrs)=7.0", fontsize = 18)
ax[0].legend()


bin_hr_diagram.plot(log_age=7, loc=ax[1], levels=30) # levels (number of contours)␣
↪adjusted for visualisation
ax[1].scatter(usco.Temperature, usco.Luminosity, c='pink', s=100, label='Usco')
ax[1].set_xlim([5.4, 3.4])
ax[1].set_ylim([2.8, 6.2])
ax[1].set_title('Binary stars')
ax[1].text(5.25, 5.4, "Z = 0.02 \nlog(Age/yrs)=7.0", color='w', fontsize = 18)
ax[1].legend()
```

[19]:
```
<matplotlib.legend.Legend at 0x7fbe3c38de10>
```



b) Using ‘‘add_subplot()‘‘ syntax

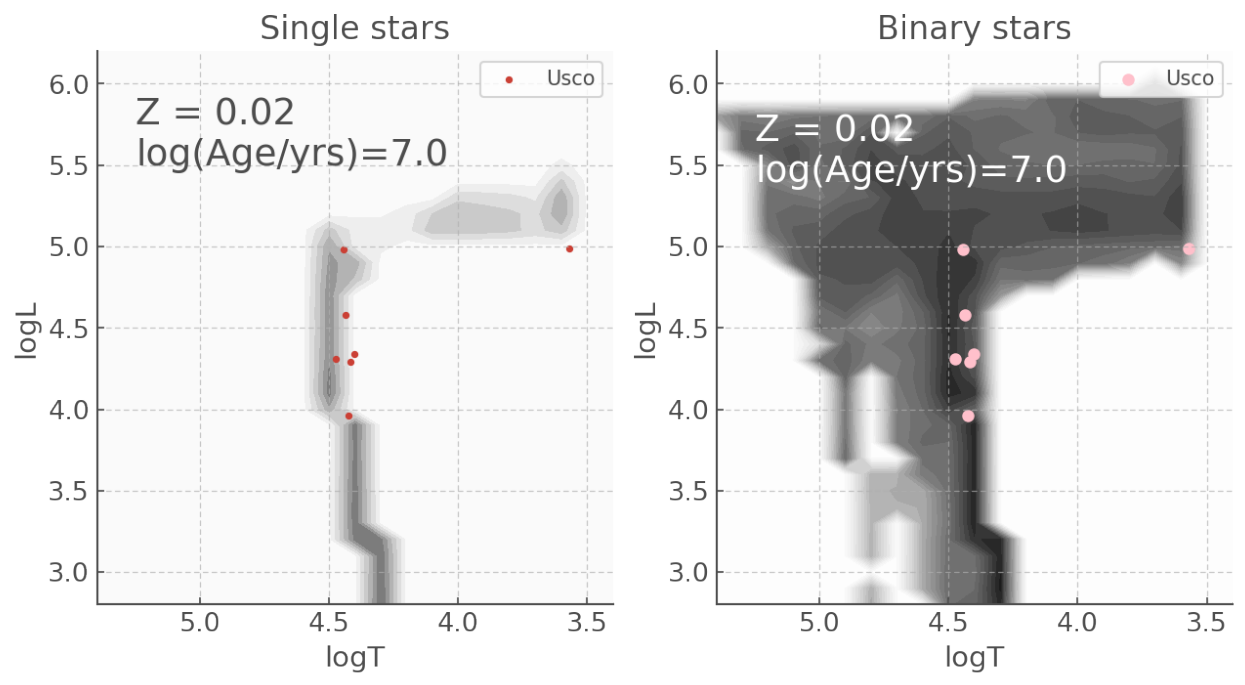
[20]:
```
plt.figure(figsize=(10,5))

sin_plot = sin_hr_diagram.plot(log_age=7, loc=121)
sin_plot .scatter(usco.Temperature, usco.Luminosity, label='Usco')
sin_plot .set_xlim([5.4, 3.4])
sin_plot .set_ylim([2.8, 6.2])
sin_plot .set_title('Single stars')
sin_plot .text(5.25, 5.5, "Z = 0.02 \nlog(Age/yrs)=7.0", fontsize = 18)
sin_plot .legend()


bin_plot = bin_hr_diagram.plot(log_age=7, loc=122, levels=30) # levels adjusted for␣
↪visualisation
bin_plot.scatter(usco.Temperature, usco.Luminosity, s=100, c='pink', label='Usco')
```

(continues on next page)

(continued from previous page)

```
bin_plot.set_xlim([5.4, 3.4])
bin_plot.set_ylim([2.8, 6.2])
bin_plot.set_title('Binary stars')
bin_plot.text(5.25, 5.4, "Z = 0.02 \nlog(Age/yrs)=7.0", color='w', fontsize = 18)
bin_plot.legend()
```

[20]:
```
<matplotlib.legend.Legend at 0x7fbe3c6eaac8>
```



4.4 2.4 Customizing your plots with matplotlib key word arguments

Althought the default parameters for the HR diagram pltos were chosen to allow the creating of publication-ready plots with as few modifications as possible, the plotting function of `HRDiagrams` objects are designed to take any subplementary key word arguments and pass them on to `matplotlib.pyplot.contour`.

This allows a much greater level of flexibility when making graphics. For example if you wish to cahnge the colormap you can do the following.

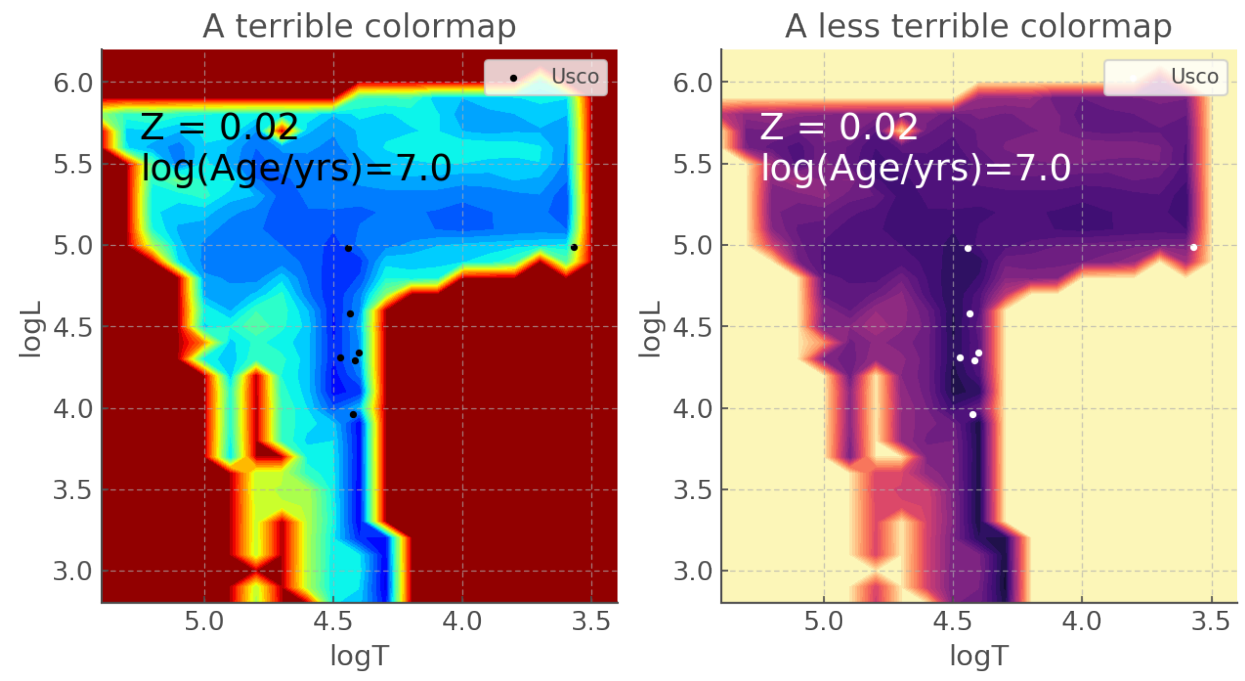
[17]:
```
fig, ax = plt.subplots(1,2, figsize=(10,5))

bin_hr_diagram.plot(log_age=7, levels=30, loc=ax[0], cmap='jet_r') # levels adjusted␣
↪for visualisation
ax[0].scatter(usco.Temperature, usco.Luminosity, c='k', label='Usco')
ax[0].set_xlim([5.4, 3.4])
ax[0].set_ylim([2.8, 6.2])
ax[0].set_title('A terrible colormap')
ax[0].text(5.25, 5.4, "Z = 0.02 \nlog(Age/yrs)=7.0", color='k', fontsize = 18)
ax[0].legend()


bin_hr_diagram.plot(log_age=7, levels=30, loc=ax[1], cmap='magma_r') # levels␣
↪adjusted for visualisation
```

(continues on next page)

(continued from previous page)

```
ax[1].scatter(usco.Temperature, usco.Luminosity, c='w', label='Usco')
ax[1].set_xlim([5.4, 3.4])
ax[1].set_ylim([2.8, 6.2])
ax[1].set_title('A less terrible colormap')
ax[1].text(5.25, 5.4, "Z = 0.02 \nlog(Age/yrs)=7.0", color='w', fontsize = 18)
ax[1].legend()
```

```
[17]: <matplotlib.legend.Legend at 0x7fbe34031160>
```



CHAPTER

FIVE

LOAD

This module implements the tools to easily load BPASS data.

`hoki.load.`**`population_output`**(*path*, *hr_type=None*)

Loads a population output file

Parameters

- **`path`** (*`str`*) – Path to the file containing the target data.
- **`hr_type`** (*`str, optional`*) – Type of HR diagram to load: 'TL', 'Tg' or 'TTG'.

Returns Output Data

Return type pandas.DataFrame or hoki.hrdiagrams.HRDiagrams object

CHAPTER

SIX

HRDIAGRAMS

This module implements the HR diagram infrastructure.

class hoki.hrdiagrams.**HRDiagram**(*high_H_input*, *medium_H_input*, *low_H_input*, *hr_type*)

A class containing the HR diagram data produced by BPASS.

This class is called by the functions hrTL(), hrTg() and hrTTG() in hoki.load and users should not need to create an HRDiagram object themselves.

For more details on the BPASS outputs - and therefore why the data structure is as it is - please refer to the manual: https://bpass.auckland.ac.nz/8/files/bpassv2_1_manual_accessible_version.pdf

Notes

HRDiagram supports indexing. The indexed array is a 51x100x100 np.array that stacked the time weighted arrays corresponding to the 3 different abundances.

self.**high_H**

HR diagrams for 51 time bins with a hydrogen abundance X > 0.4. Time weighted.

Type np.ndarray (51x100x100)

self.**medium_H**

HR diagrams for 51 time bins with a hydrogen abundance E-3 < X < 0.4. Time weighted.

Type np.ndarray (51x100x100)

self.**low_H**

HR diagrams for 51 time bins with a hydrogen abundance X < E-3. Time weighted.

Type np.ndarray (51x100x100)

self.**type**

Type of HR diagram: TL, Tg or TTG

Type str

self.**high_H_not_weighted**

HR diagrams for 51 time bins with a hydrogen abundance X > 0.4.

Type np.ndarray (51x100x100)

self.**medium_H_not_weighted**

HR diagrams for 51 time bins with a hydrogen abundance E-3 < X < 0.4.

Type np.ndarray (51x100x100)

self.**low_H_not_weighted**

HR diagrams for 51 time bins with a hydrogen abundance X < E-3.

Type np.ndarray (51x100x100)

`self.`**`_all_H`**
HR diagrams for 51 time bins - all hydrogen abundances stacked. This attribute is private because it can simply be called using the indexing capabilities of the class.

Type np.ndarray (51x100x100)

`self.`**`high_H_stacked`**
HR diagram stacked for a given age range - hydrogen abundance X > 0.4. None before calling self.stack()

Type np.ndarray (51x100x100)

`self.`**`medium_H_stacked`**
HR diagram stacked for a given age range - hydrogen abundance E-3 < X < 0.4. None before calling self.stack()

Type np.ndarray (51x100x100)

`self.`**`low_H_stacked`**
HR diagram stacked for a given age range - hydrogen abundance E-3 > X. None before calling self.stack()

Type np.ndarray (51x100x100)

`self.`**`all_stacked`**
HR diagram stacked for a given age range - all abundances added up. None before calling self.stack()

Type np.ndarray (51x100x100)

`self.`**`t`**
Class attribute - The time bins in BPASS - note they are in LOG SPACE

Type np.ndarray 1D

`self.`**`dt`**
Class attribute - Time intervals between bins NOT in log space

Type np.ndarray 1D

`at_log_age`(*log_age*)
Returns the HR diagrams at a specific age.

Parameters **`log_age`** (*`int or float`*) – The log(age) of choice.

Returns

- [0] : Stack of all the abundances
- [1] : High hydrogen abundance X>0.4
- [2] : Medium hydrogen abundance (E-3 < X < 0.4)
- [3] : Low hydrogen abundance (X < E-3)

Return type Tuple of 4 np.ndarrays (100x100)

`plot`(*log_age=None*, *age_range=None*, *abundances=(1, 1, 1)*, ***kwargs*)
Plots the HR Diagram - calls hoki.hrdiagrams.plot_hrdiagram()

Parameters

- **`log_age`** (*`int or float, optional`*) – Log(age) at which to plot the HRdiagram.
- **`age_range`** (*`tuple or list of 2 ints or floats, optional`*) – Age range within which you want to plot the HR diagram

- **abundances** (*tuple or list of 3 ints, zeros or ones, optional*) – This turns on or off the inclusion of the abundances. The corresponding abundances are: (X > 0.4, E-3 < X < 0.4, E-3>X). A 1 means a particular abundance should be included, a 0 means it will be ignored. Default is (1,1,1), meaning all abundances are plotted. Note that (0,0,0) is not valid and will return and assertion error.
- ****kwargs** (*matplotlib keyword arguments, optional*) –

Notes

If you give both an age and an age range, the age range will take precedent and be plotted. You will get a warning if that happens though.

Returns The plot created is returned, so you can add stuff to it, like text or extra data.

Return type matplotlib.axes._subplots.AxesSubplot

stack (*log_age_min=None*, *log_age_max=None*)

Creates a stack of HR diagrams within a range of ages

Parameters

- **log_age_min** (*int or float, optional*) – Minimum log(age) to stack
- **log_age_max** (*int or float, optional*) – Maximum log(age) to stack

Returns This method stores the stacked values in the class attributes self.high_H_stacked, self.medium_H_stacked, self.low_H_stacked and self.all_stacked.

Return type None

hoki.hrdiagrams.**plot_hrdiagram** (*single_hr_grid*, *kind='TL'*, *levels=10*, *loc=111*, *cmap='Greys'*, ***kwargs*)

Plots an HR diagram with a contour plot

Parameters

- **single_hr_grid** (*np.ndarray (100x100)*) – One HR diagram grid.
- **kind** (*str, optional*) – Type of HR diagram: 'TL', 'Tg', or 'TTG'. Default is 'TL'.
- **levels** (*int, optional*) – Number of contours to plot. Default is 10.
- **loc** (*int - 3 digits, optional*) – Location to parse plt.subplot(). The Default is 111, to make only one plot.
- **cmap** (*str, optional*) – The matplotlib colour map to use. Default is 'RdGy'.
- **kwargs** (*matplotlib key word arguments to parse*) –

Returns The plot created is returned, so you can add stuff to it, like text or extra data.

Return type matplotlib.axes._subplots.AxesSubplot

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX