wdwarfdate

Release 0.1.0

Rocio Kiman

Jun 10, 2022

USER GUIDE

`wdwarfdate` is a Python open source code which derives a Bayesian age of a white dwarf from an effective temperature and a surface gravity. `wdwarfdate` runs a chain of models assuming single star evolution and estimate the following parameters and their uncertainties: total age of the object, mass and cooling age of the white dwarf and mass and lifetime of the progenitor star.

CHAPTER

ONE

BASIC USAGE

Let's set up two white dwarfs with their effective temperature and surface gravity. These two white dwarfs parameters come from Cummings,J.D. et al. (2018). To run `wdwarfdate` we just need to initialize the object `WhiteDwarf()` with the parameters indicating the models we want to use. For details on these parameters go to the tutorials in this documentation.

```
import wdwarfdate
import numpy as np

#Define data for the white dwarfs
teffs = np.array([19250,20250])
teffs_err = np.array([500,850])
loggs = np.array([8.16,8.526])
loggs_err = np.array([0.084,0.126])

WD = wdwarfdate.WhiteDwarf(teffs,teffs_err,loggs,loggs_err,
                           model_wd='DA',
                           feh='p0.00',
                           vvcrit='0.0',
                           model_ifmr = 'Cummings_2018_MIST',
                           high_perc = 84,
                           low_perc = 16,
                           datatype='yr',
                           save_plots=False,
                           display_plots=True)
WD.calc_wd_age()
```

The function `WD.calc_wd_age()` will take the effective temperature ($T_{\rm eff}$) and surface gravity ($\log g$) given, and will sample the posterior distribution defined by the chosen models using the grid method. The parameters which are being sampled are the mass of the progenitor star, the cooling age of the white dwarf and a $\Delta_{\rm m}$ parameter to model the scatter in the initial-to-final mass relation. For more information on how to run `wdwarfdate` see the tutorials and Kiman et al. in prep.

`wdwarfdate` allows us to select which models are going to go into the chain of models for the age estimation: 1. Cooling tracks for a DA or non-DA white dwarf. 2. The initial-to-final mass relation. 3. The parameter for the progenitor star . For details on the models available see the page *Models included* and Kiman et al. in prep.

The output is saved on a table, and can be accessed by doing

```
WD.results
```

This is an astropy Table with one row for each $T_{\rm eff}$ and $\log g$ given. The columns of the Table correspond to median, the difference between median and low percentile (16th unless indicated otherwise), and the difference between high

percentile and median (84th unless indicated otherwise) for the following parameters:

- `ms_age` Median values of main sequence age distribution of the progenitor of the white dwarf
- `cooling_age` Median values of cooling age distribution of the white dwarf
- `total_age` Median values of total age distribution of the white dwarf
- `initial_mass` Median values of initial mass distribution, meaning the mass of the progenitor
- `final_mass` Median values of final mass distribution, meaning the mass of the white dwarf

1.1 Installation

To install `wdwarfdate` you can do it from the github source:

```
git clone https://github.com/rkiman/wdwarfdate.git
cd wdwarfdate
python setup.py install
```

1.1.1 Dependencies

To run `wdwarfdate` the following packages are needed: NumPy, astropy, matplotlib and SciPy

These can be installed with the following line:

```
pip install numpy astropy matplotlib scipy
```

1.2 Models included

In order to estimate an age for a white dwarf a chain of models have to be used. Here are the models included in `wdwarfdate`.

wdwarfdate		
Models Included	Cooling Models	**Cooling tracks from the Montreal White Dwarf Group available online (Berdard et al. 2020):** • Thick H layer (DA) • Thin H layer (non-DA)
	IFMR	• Marigo et al. (2020) • Cummings et al. (2018) MIST and PARSEC based. • Salaris et al. (2009) • Williams et al. (2009)
	Stellar evolution models	**MESA Isochrones available online (Choi et al. 2016 and Dotter 2016):** • [Fe/H] $= -4.00, -1.00, 0.00, 0.50$ • v/vcrit $= 0.0, 0.4$ • $[\alpha/\mathrm{Fe}] = 0$

1.3 Constraints

We summarize here the most important constraints when using `wdwarfdate`. For a complete discussion see Kiman et al. 2022.

- Combining the limitation of the cooling tracks with the restrictions of the IFMR, the values for which `wdwarfdate` can estimate a total age are $1,500 \lesssim T_{\rm eff} \lesssim 90,000\,\mathrm{K}$ and $7.9 \lesssim \log g \lesssim 9.3$.
- Given that the cooling tracks assume single star evolution and C/O core for the white dwarfs, the best range of final masses to use `wdwarfdate` is $0.45 - 1.1\,\mathrm{M}_\odot$, because outside this range objects are not likely to have evolved as a single star.

1.4 Citation

If you use `wdwarfdate` in your research please cite: Kiman et al. 2022. Also, please cite the IFMR you decide to use (See *Models included*), the cooling models (Berdard et al 2020) and the stellar evolution models (Choi et al 2016 and Dotter 2016).

1.5 Use `wdwarfdate` to derive white dwarf ages

In this tutorial we are going to estimate the ages of two white dwarfs using `wdwarfdate`. We are going to start by importing all the packages we are going to need

[1]:
```
import wdwarfdate
import numpy as np
import time
```

We are going to set up the white dwarfs's effective temperatures and surface gravity obtained from Cummings,J.D. et al. 2018.

[2]:
```
#Define data for the white dwarf
teffs = np.array([19250,20250])
teffs_err = np.array([500,850])
loggs = np.array([8.16,8.526])
loggs_err = np.array([0.084,0.126])
```

Now we run the age estimation using `wdwarfdate` and record the time it takes to run. We are going to explicitly indicate the models we want to use:

1) `model_wd = 'DA'`, to indicate we want to use the thick outermost hydrogen layer model. This could be non-DA to use the thin layer model.

2) `model_ifmr = 'Cummings_2018_MIST'`, for the initial-to-final mass relation.

3) `feh = 'p0.00'` and `vvcrit='0.0'`, for the isochrone of the progenitor star.

These are the default models so we do not need to do it in this case, but we will write them down to be clear. For a list of the models that can be used with `wdwarfdate`, go the page Models included in the documentation. The rest of the parameters we are going to specify:

1) `high_perc = 84, low_perc = 16`: high and low percentiles we want to use to calculate the errors for each parameter.

2) `data_type = 'Gyr'`: units of the resulting ages. This can be yr, Gyr or log.

3) `save_plots = False`: With this parameter we decide to not save the resulting plots.

4) `display_plots=True`: With this parameter we decide to display the resulting plots in jupyter notebook.

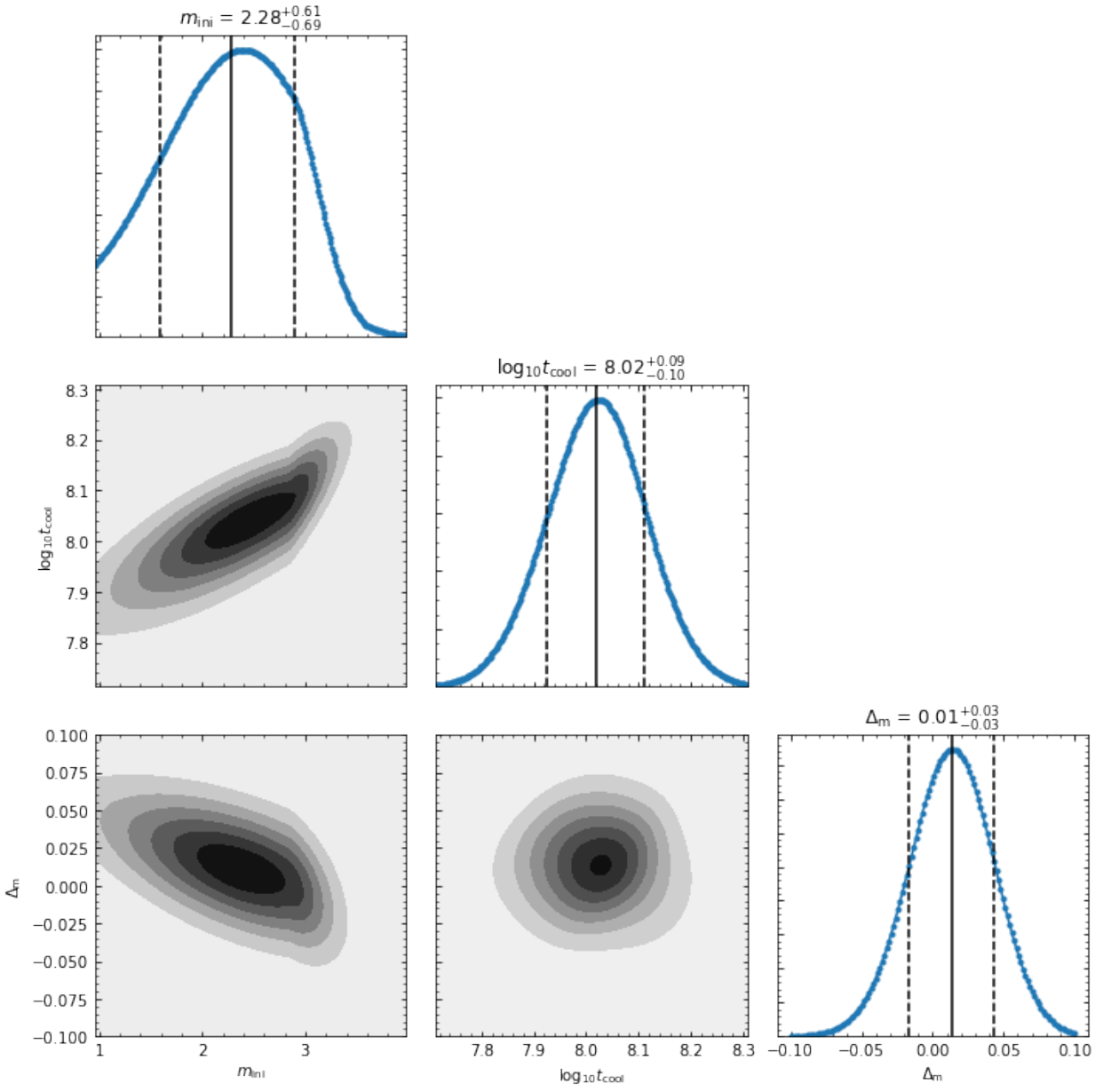
[3]:
```
start_time = time.time()
WD = wdwarfdate.WhiteDwarf(teffs,teffs_err,loggs,loggs_err,
                           model_wd='DA',feh='p0.00',vvcrit='0.0',
                           model_ifmr = 'Cummings_2018_MIST',
                           high_perc = 84, low_perc = 16,
                           datatype='Gyr',
                           save_plots=True, display_plots=True)
WD.calc_wd_age()
```

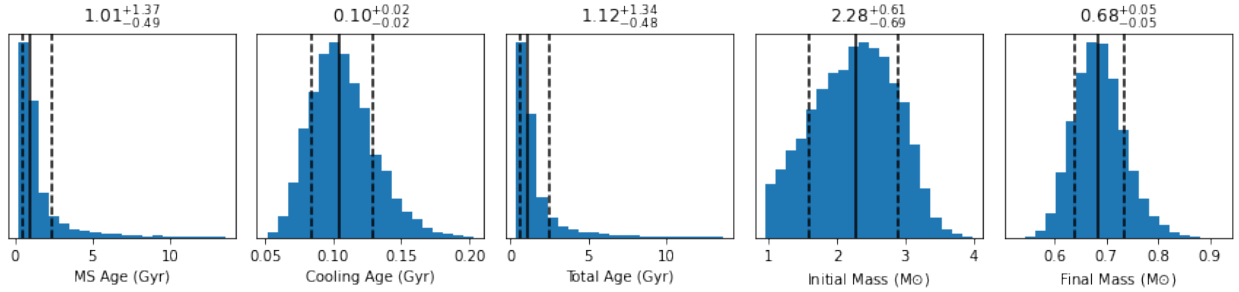
(continues on next page)

(continued from previous page)

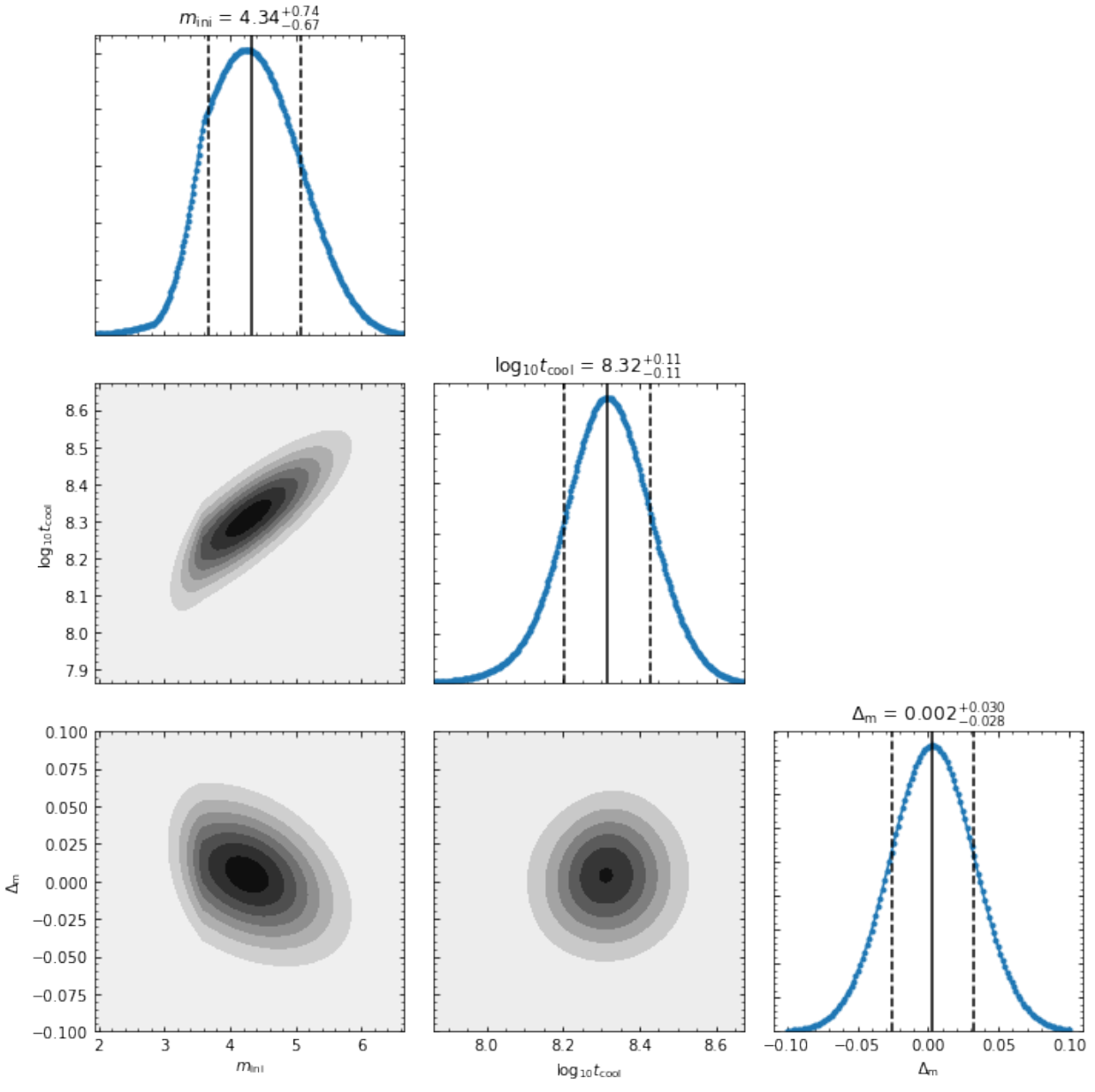
```
end_time = time.time()

print(f'Time to run two white dwarfs: {np.round(end_time-start_time,2)}s')
```

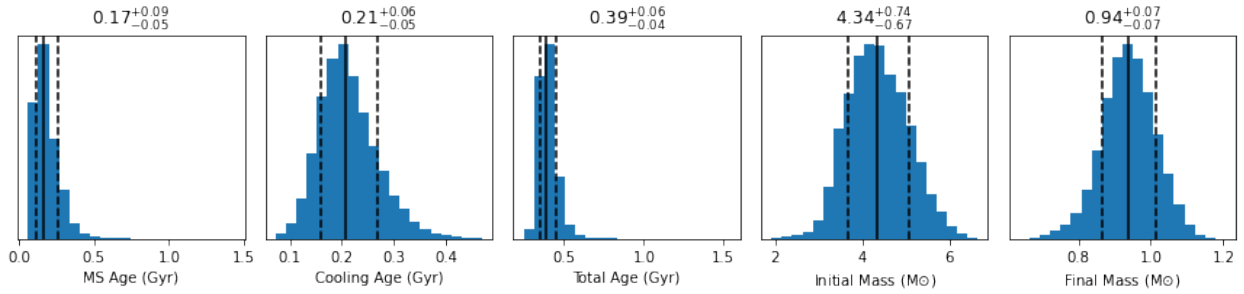
```
Running Teff = 19250 +/- 500 K, logg = 8.16 +/- 0.08
Grid limits used to evaluate the posterior: mi = 0.95-3.98 Msun, log10_tcool = 7.71-8.31
```






```
Running Teff = 20250 +/- 850 K, logg = 8.53 +/- 0.13
Grid limits used to evaluate the posterior: mi = 1.92-6.66 Msun, log10_tcool = 7.86-8.67
```






```
Time to run two white dwarfs: 40.97s
```

[4]:
```
WD.results
```

[4]:
```
<Table length=2>
   ms_age_median       ms_age_err_low    ... final_mass_err_high
      float64             float64        ...       float64
------------------- -------------------- ... --------------------
 1.0085847493888629   0.4949423725309816 ... 0.049727065362504375
0.16553841920328524 0.054311876463780356 ...  0.07383056622074491
```

The option `save_plots = True` will save several files to a folder called 'results', or any path we choose using the `path` parameter. Below we describe each of the files *wdwarfdate* method creates. In order of appearence (above):

1) "teff_19250_logg_8.16_feh_p0.00_vvcrit_0.0_DA_Cummings_2018_MIST_gridplot.png": grid plot for the three independent variables: initial mass, cooling age and delta m.

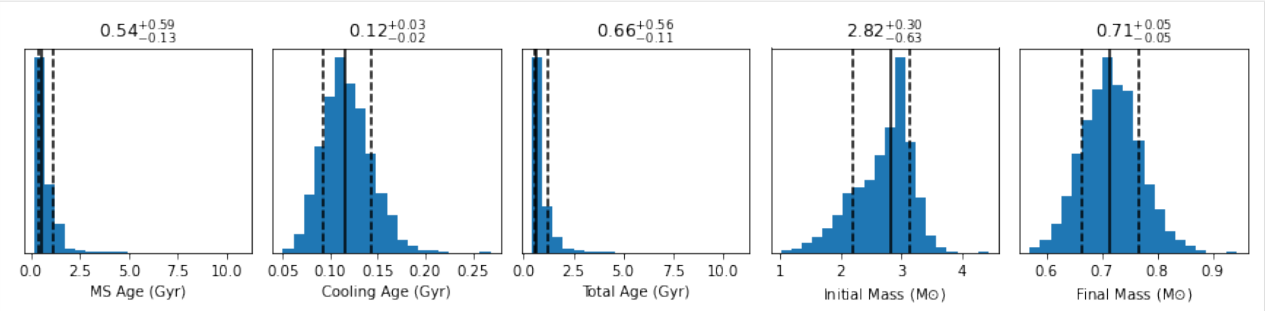
2) "teff_19250_logg_8.16_feh_p0.00_vvcrit_0.0_DA_Cummings_2018_MIST_distributions.png": Parameter distribution for the final mass and cooling age of the white dwarf, mass and main sequence age of the progenitor star and total age of the object.

The fast-test method

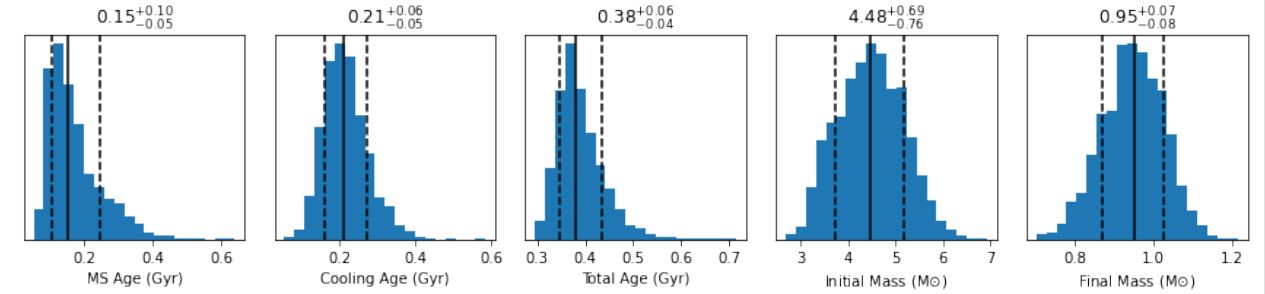
The fast-test method included in `wdwardate` generates a gaussian distribution for each pair of *teff* and *logg* using the uncertainties as standard deviation, and runs the full distribution through a chain of models to calculate the total age of the white dwarf, and the rest of the parameters described above. Below we show an example of the fast-test method.

[5]:
```
WD2 = wdwarfdate.WhiteDwarf(teffs,teffs_err,loggs,loggs_err,
                            method='fast_test',
                            model_wd='DA',feh='p0.00',vvcrit='0.0',
                            model_ifmr = 'Cummings_2018_MIST',
                            high_perc = 84, low_perc = 16,
                            datatype='Gyr', return_distributions=True,
                            save_plots=False, display_plots=True)
WD2.calc_wd_age()
```

```
Running teff:19250 logg:8.16
```




```
Running teff:20250 logg:8.526
```



The parameters used in the `wdwarfdate.WhiteDwarf` for the fast-test method are similar to the Bayesian method. In this case we also set `return_distributions = True`, which will generate that one of the comlumns of the resulting Table of results will have the distributions for each parameter in the case of . For a detailed comparison of the two methods, go to the paper describing *wdwarfdate* (Kiman et al. in prep). Also, from the output plots described above, in this method we only get the distributions of the parameters.

[6]:
```
WD2.results_fast_test
```

[6]:
```
<Table length=2>
   ms_age_median    ...
      float64       ...
------------------- ...
 0.5449890150641449 ...
0.15242746710347288 ...
```

1.6 Customize the output plots

In this tutorial we will show explicitly how to make the plots made by `wdwarfdate` and we are going to customize them. The goal of this tutorial is to provide code that is easy to modify to make our own versions of the plots.

[1]:
```
import wdwarfdate
import numpy as np
import time
import matplotlib.pyplot as plt
```

Then we set up the effective temperature and surface gravity that *wdwarfdate* needs to run. The values used in this tutorial are from Cummings, J.D. et al. 2018. We are going to add a list called `comparison` with the results of the parameter estimation from that paper too,

[2]:
```
#Define data for the white dwarf

teffs = 14500
teffs_err = 300
loggs = 8.325
loggs_err = 0.042

comparison = [0.705-0.364, #Main sequence age (Gyr)
              0.364, # Cooling age (Gyr)
              0.705, # Total age (Gyr)
              3.40, # Initial mass (Msun)
              0.813, # Final mass (Msun)
              ]
```

[3]:
```
start = time.time()
WD = wdwarfdate.WhiteDwarf(teffs,teffs_err,loggs,loggs_err,
                           model_wd='DA',feh='p0.00',vvcrit='0.0',
                           model_ifmr = 'Cummings_2018_MIST',
                           high_perc = 84, low_perc = 16,
                           datatype='yr',
                           save_plots=False, display_plots=False)
WD.calc_wd_age()
end = time.time()
print(f'{np.round(end - start,2)}s')
```

```
Running Teff = 14500 +/- 300 K, logg = 8.32 +/- 0.04
Grid limits used to evaluate the posterior: mi = 2.35-4.33 Msun, log10_tcool = 8.46-8.7
12.0s
```

Now we are going to make separatelly each plot that the code outputs automatically, so we can modify them.

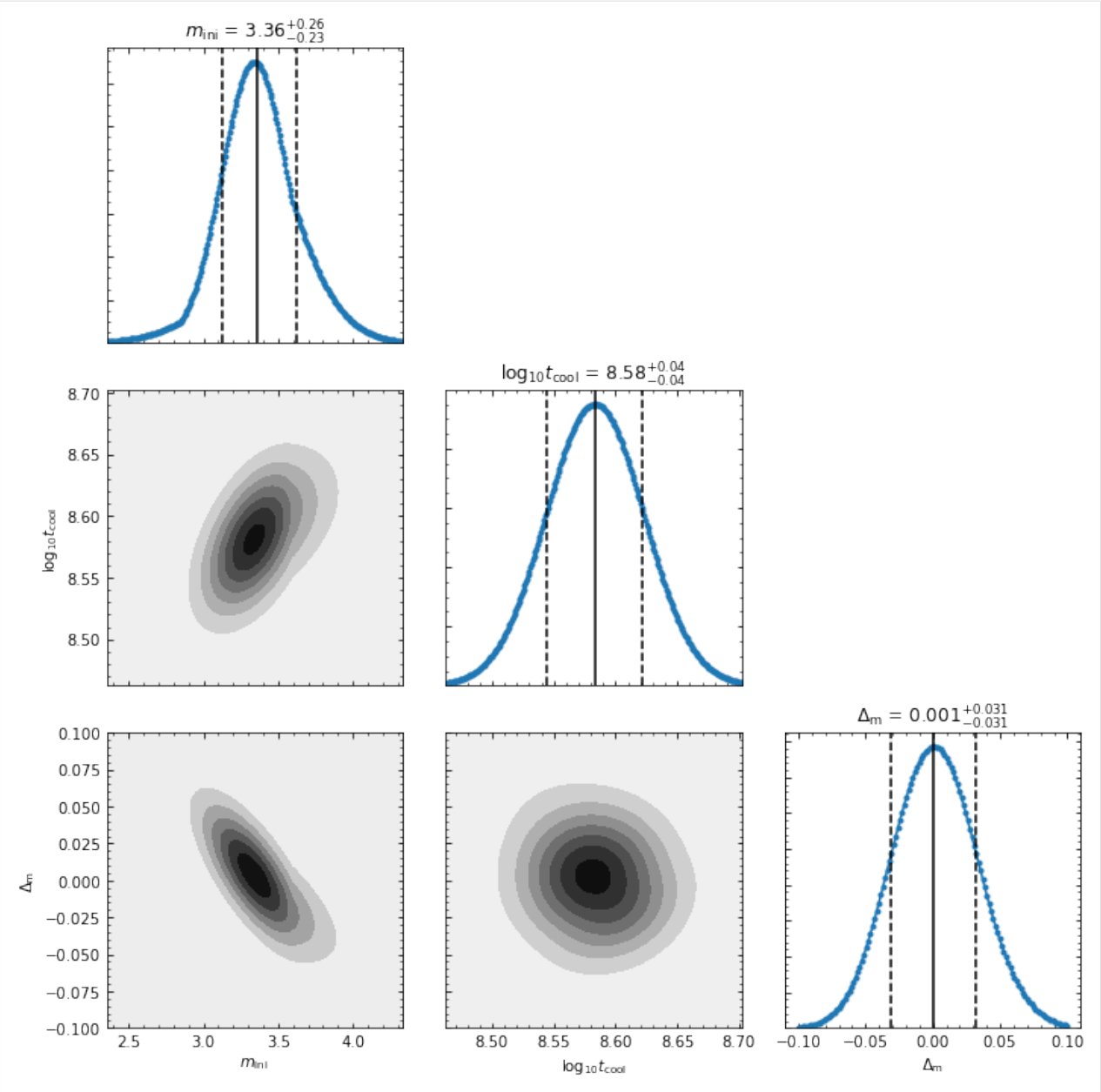
1) "teff_14500_logg_8.325_feh_p0.00_vvcrit_0.0_DA_Cummings_2018_MIST_gridplot.png"

[4]:
```
params_label = [r'$m_{\rm ini}$', r'$\log _{10} t_{\rm cool}$',
                        r'$\Delta _{\rm m}$']
res = [WD.mi_median, WD.log10_tcool_median, WD.delta_m_median]
res_err_low = [WD.mi_err_low, WD.log10_tcool_err_low, WD.delta_m_err_low]
res_err_high = [WD.mi_err_high, WD.log10_tcool_err_high, WD.delta_m_err_high]
title = r"${{{0:.2f}}}_{{-{1:.2f}}}^{{+{2:.2f}}}$"
f, axs = plt.subplots(3, 3, figsize=(10, 10), sharex='col')

for i in range(3):
    for j in range(3):
        if i == j:
            # Diagonal plots
            axs[i, j].plot(WD.params[i], WD.params_prob[i], '.-')
            axs[i, j].axvline(x=res[i], color='k')
            axs[i, j].axvline(x=res[i] - res_err_low[i], color='k', linestyle='--')
            axs[i, j].axvline(x=res[i] + res_err_high[i], color='k', linestyle='--')
            axs[i, j].set_yticklabels([])
            axs[i, j].set_ylim(0,)
            if any(np.array([np.round(res[i], 2), np.round(res_err_low[i], 2),
                             np.round(res_err_high[i], 2)]) == 0):
                dec_num = 2
```

(continues on next page)

(continued from previous page)

```
                while any(np.array([np.round(res[i], dec_num),
                                    np.round(res_err_low[i], dec_num),
                                    np.round(res_err_high[i], dec_num)]) == 0):
                    dec_num += 1
                    title2 = r"${{{0:." + str(dec_num) + "f}}}_{{-{1:." + str(dec_num) +
↪"f}}}^{{+{2:." + str(
                        dec_num) + "f}}}$"
                    axs[i, j].set_title(params_label[i] + ' = ' + title2.format(np.
↪round(res[i], dec_num),
                                                                               np.
↪round(res_err_low[i],
                                                                                  ␣
↪dec_num),
                                                                               np.
↪round(res_err_high[i],
                                                                                  ␣
↪dec_num)))
            else:
                axs[i, j].set_title(params_label[i] + ' = ' + title.format(np.
↪round(res[i], 2),
                                                                          np.round(res_
↪err_low[i], 2),
                                                                          np.round(res_
↪err_high[i], 2)))
        elif i > j:
            # Out of diagonal plots
            options = np.array([0, 1, 2])
            mask = np.array([x not in [i, j] for x in options])
            axis_sum = options[mask][0]
            axs[i, j].contourf(WD.params[j], WD.params[i], np.nansum(WD.posterior,␣
↪axis=(axis_sum)).transpose(),
                               cmap='gist_yarg')
            if j == 1:
                axs[i, j].set_yticklabels([])
        else:
            f.delaxes(axs[i, j])

        if i == 2:
            axs[i, j].set_xlabel(params_label[j])

        if j == 0 and i != 0:
            axs[i, j].set_ylabel(params_label[i])

        axs[i, j].tick_params('both', direction='in', top=True, right=True)
        axs[i, j].minorticks_on()
        axs[i, j].tick_params('both', which='minor', direction='in', top=True,␣
↪right=True)
plt.tight_layout()
plt.show()
```



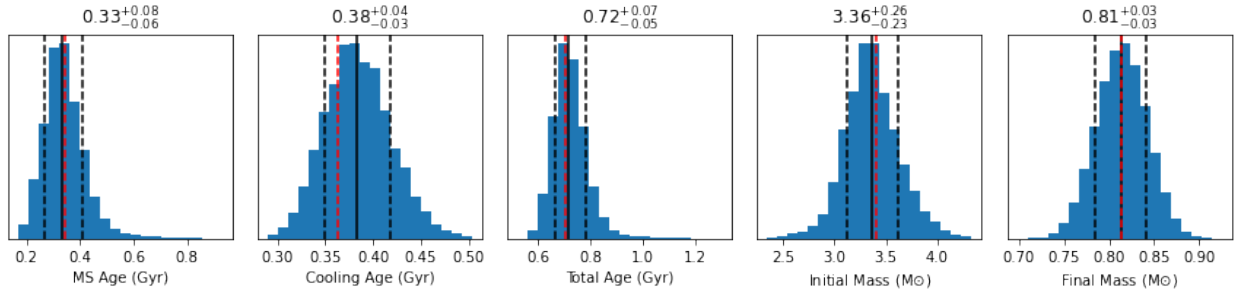
2) "teff_19250_logg_8.16_feh_p0.00_vvcrit_0.0_DA_Cummings_2018_MIST_distributions.png"

We calculate the median, low and high percentiles because, for example, we want to change the units of the output parameters, which we originally set to be yr. The default output of the code will provide the sample of the ages in log(yr). So we can change the units from log(yr) to Gyr.

[5]:
```
results = wdwarfdate.calc_percentiles(10**WD.log10_tms_sample/1e9,
                                      10**WD.log10_tcool_sample/1e9,
                                      10**WD.log10_ttot_sample/1e9,
                                      WD.mi_sample, WD.mf_sample,
                                      WD.high_perc, WD.low_perc)
```

And we have everything we need to plot the distributions, adding the results from Cummings et al. 2018 to compare

[6]:
```
title = r"${{{0:.2f}}}_{{-{1:.2f}}}^{{+{2:.2f}}}$"
f, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(12, 3))

axs = [ax1, ax2, ax3, ax4, ax5]

labels = ['MS Age (Gyr)', 'Cooling Age (Gyr)', 'Total Age (Gyr)',
          r'Initial Mass (M$\odot$)', r'Final Mass (M$\odot$)']

distributions = [10**WD.log10_tms_sample/1e9, 10**WD.log10_tcool_sample/1e9,
                 10**WD.log10_ttot_sample/1e9, WD.mi_sample, WD.mf_sample]

for ax, label, dist, i, comparison_i in zip(axs, labels, distributions, np.arange(0, 15,␣
↪3), comparison):
    ax.hist(dist[~np.isnan(dist)], bins=20)
    ax.axvline(x=results[i], color='k')
    ax.axvline(x=results[i] - results[i+1], color='k', linestyle='--')
    ax.axvline(x=results[i] + results[i+2], color='k', linestyle='--')
    ax.axvline(x=comparison_i, color='r', linestyle='--')
    ax.set_xlabel(label)
    ax.yaxis.set_visible(False)
    ax.set_title(title.format(np.round(results[i], 2),
                              np.round(results[i+1], 2),
                              np.round(results[i+2], 2)))
plt.tight_layout()
plt.show()
```



Fast-test method

We can do the same for the results from the fast-test method. In this case we need to set `return_distributions=True`, so we can plot the distributions after.

[7]:
```
start = time.time()
WD_2 = wdwarfdate.WhiteDwarf(teffs,teffs_err,loggs,loggs_err,
                             method='fast_test',
                             model_wd='DA',feh='p0.00',vvcrit='0.0',
                             model_ifmr = 'Cummings_2018_MIST',
                             high_perc = 84, low_perc = 16,
                             datatype='yr', return_distributions=True,
                             save_plots=False, display_plots=False)
WD_2.calc_wd_age()
end = time.time()
print(f'{np.round(end - start,2)}s')
```

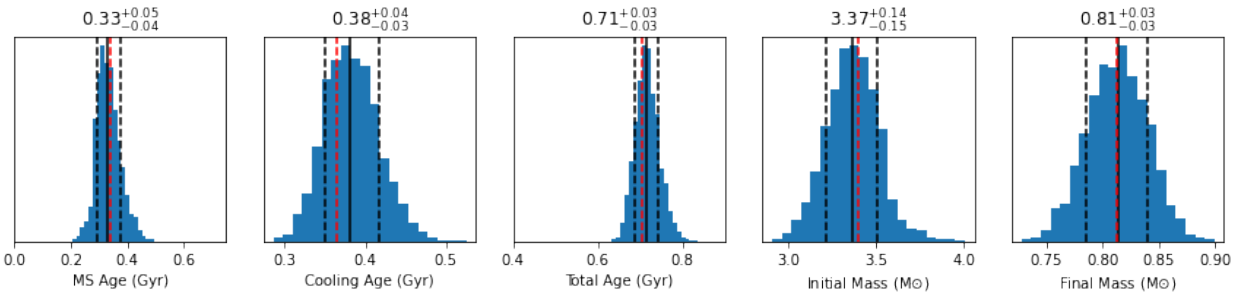
```
0.46s
```

In a similar way as before, we are going to transform the ages to Gyr. In this case the distributions are in yr.

[8]:
```
res_ms_age2 = np.array(WD_2.distributions[0][0])/1e9 # Main sequence age
res_cool_age2 = np.array(WD_2.distributions[1][0])/1e9 # Cooling age
res_tot_age2 = np.array(WD_2.distributions[2][0])/1e9 # Total age
initial_mass2 = np.array(WD_2.distributions[3][0])     # Initial mass
final_mass2 = np.array(WD_2.distributions[4][0])     # Final mass
results2 = wdwarfdate.calc_percentiles(res_ms_age2, res_cool_age2, res_tot_age2,
                                       initial_mass2, final_mass2,
                                       WD_2.high_perc, WD_2.low_perc)
```

Once we have the distributions, the plots work in the same way as above. We are going to add the comparisons in this plot too.

[9]:
```
title = r"${{{0:.2f}}}_{{-{1:.2f}}}^{{+{2:.2f}}}$"
f, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(12, 3))

axs = [ax1, ax2, ax3, ax4, ax5]

labels = ['MS Age (Gyr)', 'Cooling Age (Gyr)', 'Total Age (Gyr)',
          r'Initial Mass (M$\odot$)', r'Final Mass (M$\odot$)']

distributions2 = [res_ms_age2, res_cool_age2, res_tot_age2,initial_mass2, final_mass2]

for ax, label, dist, i, comparison_i in zip(axs, labels, distributions2, np.arange(0, 15,
↪ 3), comparison):
    ax.hist(dist[~np.isnan(dist)], bins=20)
    ax.axvline(x=results2[i], color='k')
    ax.axvline(x=results2[i] - results2[i+1], color='k', linestyle='--')
    ax.axvline(x=results2[i] + results2[i+2], color='k', linestyle='--')
    ax.axvline(x=comparison_i, color='r', linestyle='--')
    ax.set_xlabel(label)
    ax.yaxis.set_visible(False)
    ax.set_title(title.format(np.round(results2[i], 2),
                              np.round(results2[i+1], 2),
                              np.round(results2[i+2], 2)))
ax1.set_xlim(0,0.75)
ax3.set_xlim(0.4,0.9)
plt.tight_layout()
plt.show()
```



CHAPTER

TWO

LICENSE