stardate Documentation

Release 0.0.1

Ruth Angus

May 10, 2019

Contents

stardate is a tool for measuring precise stellar ages. It combines isochrone fitting with gyrochronology (rotation-based age inference) to increase the precision of stellar ages on the main sequence. The best possible ages provided by *stardate* will be for stars with rotation periods, although ages can be predicted for stars without rotation periods too. If you don't have rotation periods for any of your stars, you might consider using isochrones.py as *stardate* is simply an extension to *isochrones* that incorporates gyrochronology. *stardate* reverts back to *isochrones* when no rotation period is provided.

In order to get started you can create a dictionary containing the observables you have for your star. These could be atmospheric parameters (like those shown in the example below for the Sun), or just photometric colors, like those from *2MASS*, *SDSS* or *Gaia*. If you have a parallax, asteroseismic parameters, or an idea of the maximum V-band extinction you should throw those in too. Set up the star object and `stardate.Star.fit()` will run Markov Chain Monte Carlo (using *emcee*) in order to infer a Bayesian age for your star.

CHAPTER 1

Example usage

```
import stardate as sd

# Create a dictionary of observables
iso_params = {"teff": (5777, 10),      # Teff with uncertainty.
              "logg": (4.44, .05),     # logg with uncertainty.
              "feh": (0., .001),       # Metallicity with uncertainty.
              "parallax": (1., .01),   # Parallax in milliarcseconds.
              "B": (15.48, 0.02),      # You must provide at least one magnitude.
              "maxAV": .1}             # Maximum extinction

prot, prot_err = 26, 1

# Set up the star object.
star = sd.Star(iso_params, prot=prot, prot_err=prot_err)  # Here's where you add a␣
↪rotation period

# Run the MCMC
star.fit()

# Print the median age with the 16th and 84th percentile uncertainties.
age, errp, errm, samples = star.age_results()
print("stellar age = {0} + {1} + {2}".format(age, errp, errm))

>> stellar age = 4.5 + 2.1 - 1.3
```

CHAPTER 2

User Guide

2.1 Installation

Currently the best way to install *stardate* is from github.

From source:

```
git clone https://github.com/RuthAngus/stardate.git
cd stardate
python setup.py install
```

2.1.1 Dependencies

The dependencies of *stardate* are NumPy, pandas, h5py, and tqdm and isochrones.

The first four of these can be installed using conda or pip:

```
conda install numpy pandas h5py tqdm
```

or

```
pip install numpy pandas h5py tqdm
```

You'll also need to download isochrones and switch to the bolo branch:

```
git clone https://github.com/timothydmorton/isochrones
cd isochrones
git checkout bolo
python setup.py install
```

Note that the bolo branch is currently the development branch for the upcoming release of isochrones v2.0, so stay tuned for updates.

2.2 API documentation

CHAPTER 3

Tutorials

Note: This tutorial was generated from an IPython notebook that can be downloaded here.

3.1 A quick stardate tutorial: measuring the ages of rotating stars

In this tutorial we'll infer the ages of some Kepler targets with measured rotation periods.

We'll start by downloading the McQuillan *et al.* (2014) (https://arxiv.org/abs/1402.5694) catalogue of stellar rotation periods.

```
import pandas as pd

url = "https://arxiv.org/src/1402.5694v2/anc/Table_1_Periodic.txt"
mc = pd.read_csv(url)
```

In order to get some photometric colors for these stars (which is the minimum requirement to measure a *stardate* age), as well as some parallaxes, which will improve the age estimate, let's find these stars in the Gaia DR2 catalogue. Thankfully, we don't have to crossmatch targets ourselves because Megan Bedell has already done it for us (check out https://gaia-kepler.fun). Her table also contains useful information from the Kepler input catalogue. Let's download the Gaia-Kepler crossmatch.

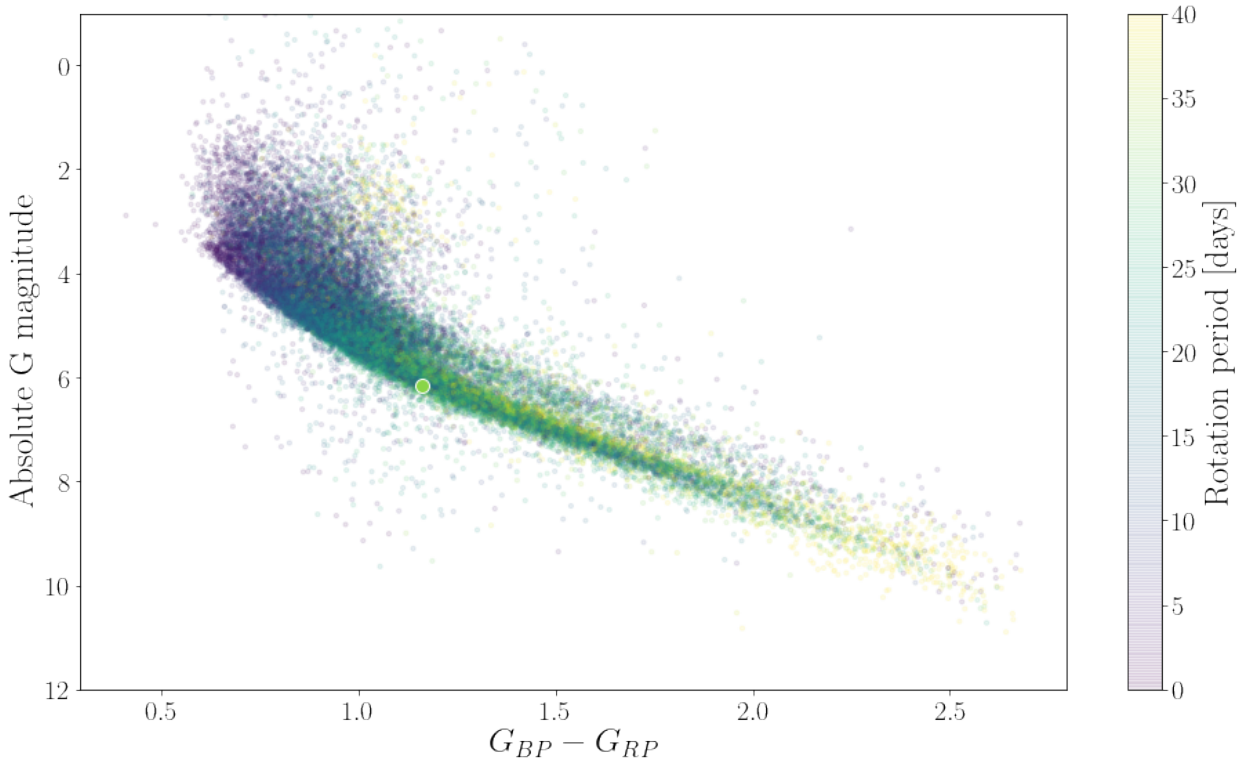
```
import astropy.utils as au
from astropy.io import fits

gaia_url = "https://dl.dropboxusercontent.com/s/xo1n12fxzgzybny/kepler_dr2_1arcsec.
↪fits?dl=0"

with fits.open(gaia_url) as data:
    gaia = pd.DataFrame(data[1].data)
```

Now let's merge these two data frames to make one data frame containing rotation periods, Gaia parallaxes, colours, etc. If we did this naively we'd run into memory problems because the gaia dataframe is too large (even when just restricted to the Kepler sample). So first we'll create a new database with only the columns we want.

```
mini_gaia = pd.DataFrame(dict({"source_id": gaia.source_id,
                               "kepid": gaia.kepid,
                               "ra": gaia.ra,
                               "dec": gaia.dec,
                               "parallax": gaia.parallax,
                               "parallax_error": gaia.parallax_error,
                               "G": gaia.phot_g_mean_mag,
                               "bp": gaia.phot_bp_mean_mag,
                               "rp": gaia.phot_rp_mean_mag,
                               "jmag": gaia.jmag,
                               "hmag": gaia.hmag,
                               "kmag": gaia.kmag,
                              }))

df = pd.merge(mc, mini_gaia, left_on="KID", right_on="kepid", how="inner")
```

Now we have everything we need (and more!) to start measuring ages. First, let's plot these stars on a (crude) colour-magnitude diagram and colour them by their rotation periods.

```
import matplotlib.pyplot as plt
%matplotlib inline

plotpar = {'axes.labelsize': 25,
           'xtick.labelsize': 20,
           'ytick.labelsize': 20,
           'text.usetex': True}
plt.rcParams.update(plotpar)

import numpy as np

def m_to_M(m, D):
    """
    Convert apparent magnitude to absolute magnitude.
    """
    return m - 5*np.log10(abs(D)) + 5

abs_G = m_to_M(df.G, (1./df.parallax)*1e3)

plt.figure(figsize=(16, 9))
plt.scatter(df.bp - df.rp, abs_G, c=df.Prot, s=10, alpha=.1, vmin=0, vmax=40);
plt.gca().invert_yaxis()
plt.ylim(12, -1)
plt.colorbar(label="$\mathrm{Rotation~period~[days]}$")
plt.xlabel("$G_{BP} - G_{RP}$")
plt.ylabel("$\mathrm{Absolute~G~magnitude}$");

np.random.seed(1234)
ind = np.random.randint(0, len(df))
plt.scatter([df.bp[ind] - df.rp[ind]], [abs_G[ind]], c=[df.Prot[ind]], s=100, vmin=0,␣
↪vmax=40, edgecolor="w");
```



In this figure you can see that the lowest mass stars (bottom right) are rotating slowly and stars rotate more and more rapidly as they increase in mass. On the main sequence, stars of the same mass rotate more slowly as they get older and move up the y-axis. You can also see the binary sequence. On average, binary stars spin more rapidly than single stars. In this version of stardate we do not account for the possibility that a star might be in a binary.

3.2 Inferring a stellar age

Now let's measure the age of a random star in this data set. We'll use the randomly chosen star plotted as a large circle on the CMD above.

We'll start by creating a dictionary of observables for this star. We'll use Gaia magnitudes *and* 2MASS JHK magnitudes (because they're available from the Kepler input catalogue) but one set of magnitudes would have worked too. These are the observables that provide information about the amount of hydrogen left in a star's core (i.e. placement on a stellar evolution track) which will be passed to the *isochrones* isochrone fitting algorithm. These observables could include photometric colours, atmospheric properties (effective temperature, surface gravity and metallicity), parallax, and even asteroseismic parameters. For most Kepler stars, only broad-band photometry is available so that's what we'll use here.

```
import stardate as sd

iso_params = {"G": (df.G[ind], .05),  # We'll just estimate the uncertainties for now.
              "bp": (df.bp[ind], .05),
              "rp": (df.rp[ind], .05),
              "J": (df.jmag[ind], .05),
              "H": (df.hmag[ind], .05),
              "K": (df.kmag[ind], .05),
              "parallax": (df.parallax[ind], df.parallax_error[ind])}  # Parallax in␣
↪milliarcseconds.
```

```
WARNING:root:Holoviews not imported. Some visualizations will not be available.
WARNING:root:PyMultiNest not imported.  MultiNest fits will not work.
WARNING:root:Emcee3 not imported; be advised.
```

This error message comes from the *isochrones* package and we're not going to use MultiNest so it's okay to ignore it! Pay attention to the exact format of the iso_params dictionary. The *isochrones* package requires it to be in exactly this format. Observables should be tuples containing values and uncertainties and this should **not be a pandas dictionary!** In addition, a new iso_params dictionary should be created for every star – *isochrones* will not currently accept arrays of multiple stars). I'll walk you through running *stardate* on multiple stars later in this tutorial.

Now let's set up the star object. This is where we'll add the rotation period (Prot) and rotation period uncertainty (Prot_err).

```
star = sd.Star(iso_params, prot=df.Prot[ind], prot_err=df.Prot_err[ind], savedir=".",
               filename="{}".format(df.KID[ind]));
```

The savedir argument should be the path to the directory you'd like the posterior samples to be saved in. The default is the current working directory. The filename is the name you'd like to give to the h5 file that will contain the saved posterior samples. The default is "samples". It's useful to set this to the name or id of the star if you're running *stardate* on multiple stars.

Now all we need to do is run the MCMC and wait (usually around 45 minutes for this many samples).

```
star.fit(max_n=200000)  # max_n is the number of MCMC steps to take but only 1/thin_
↪by of these will be saved
                        # to file. The default thin_by value is 100.
```

```
100%|| 200000/200000 [42:13:46<00:00,  1.32it/s]
```

3.3 Accessing and plotting the results.

Let's print the median value of age and it's 16th and 84th percentile uncertainties.

```
median_age, age_errp, age_errm, age_samples = star.age_results()
age_gyr = (10**median_age)*1e-9
print("log10(age) = {0:.2f} + {1:.2f} - {2:.2f}".format(median_age, age_errp, age_
↪errm))
print("Age = {0:.2f} + {1:.2f} - {2:.2f} gyr".format(age_gyr,
                                                    (10**(median_age + age_errp))*1e-
↪9 - age_gyr,
                                                    abs(age_gyr - (10**(median_age +␣
↪age_errm))*1e-9)))
```

```
log10(age) = 9.59 + 0.02 - 0.02
Age = 3.91 + 0.16 - 0.16 gyr
```

We can do the same thing for mass, metallicity, distance and extinction:

```
mass, mass_errp, mass_errm, mass_samples = star.mass_results(burnin=100)  # burnin is␣
↪thin_by x larger than this
print("Mass = {0:.2f} + {1:.2f} - {2:.2f} M_sun".format(mass, mass_errp, mass_errm))

feh, feh_errp, feh_errm, feh_samples = star.feh_results(burnin=100)
print("feh = {0:.2f} + {1:.2f} - {2:.2f}".format(feh, feh_errp, feh_errm))
```

(continues on next page)

(continued from previous page)

```
lndistance, lndistance_errp, lndistance_errm, lndistance_samples = star.distance_
↪results(burnin=100)
print("ln(distance) = {0:.2f} + {1:.2f} - {2:.2f} ".format(lndistance, lndistance_
↪errp, lndistance_errm))

Av, Av_errp, Av_errm, Av_samples = star.Av_results(burnin=100)
print("Av = {0:.2f} + {1:.2f} - {2:.2f}".format(Av, Av_errp, Av_errm))
```

```
Mass = 0.79 + 0.02 - 0.02 M_sun
feh = -0.11 + 0.15 - 0.13
ln(distance) = 6.21 + 0.01 - 0.01
Av = 0.19 + 0.13 - 0.20
```

If you want to load samples from an H5 file that you saved earlier (stardate automatically saves H5 files with posterior samples), you can do the following.

```
from stardate import load_samples, read_samples

# Load the samples.
flatsamples, _3Dsamples = load_samples("{}.h5".format(df.KID[ind]), burnin=1)

# Extract the median and maximum likelihood parameter estimates from the samples.
results = read_samples(flatsamples)

# Print the results as a pandas dataframe.
results
```

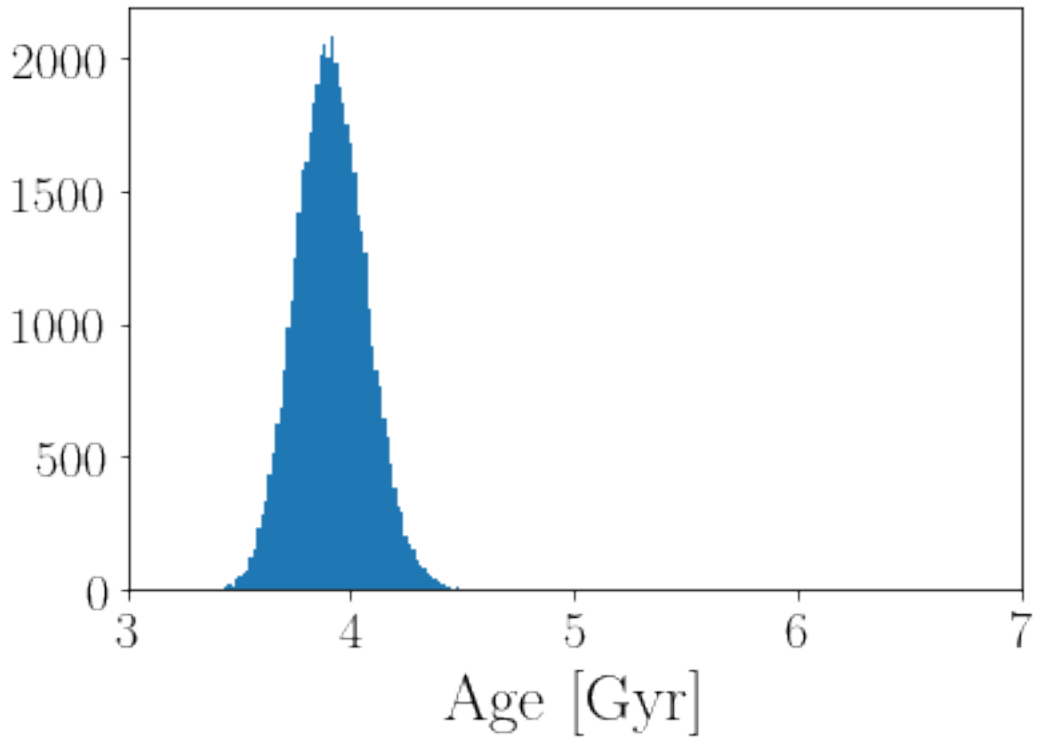
flatsamples is a 2D array of samples, useful for making corner plots. samples is a 3D array of samples, useful for plotting chains. Both flatsamples and samples have an extra dimension containing the log-probability.

results is a pandas dictionary containing best-fit stellar parameters. It has both the median and maximum-likelihood values. I recommend using maximum-likelihood (suffixed with 'ml') values for stars with Gaia G_BP - G_RP color < 1.3 and median values (suffixed with 'med') for everything else.

Age is in gyr and distance is in pc. EEP is dimensionless. feh and Av are in dex.

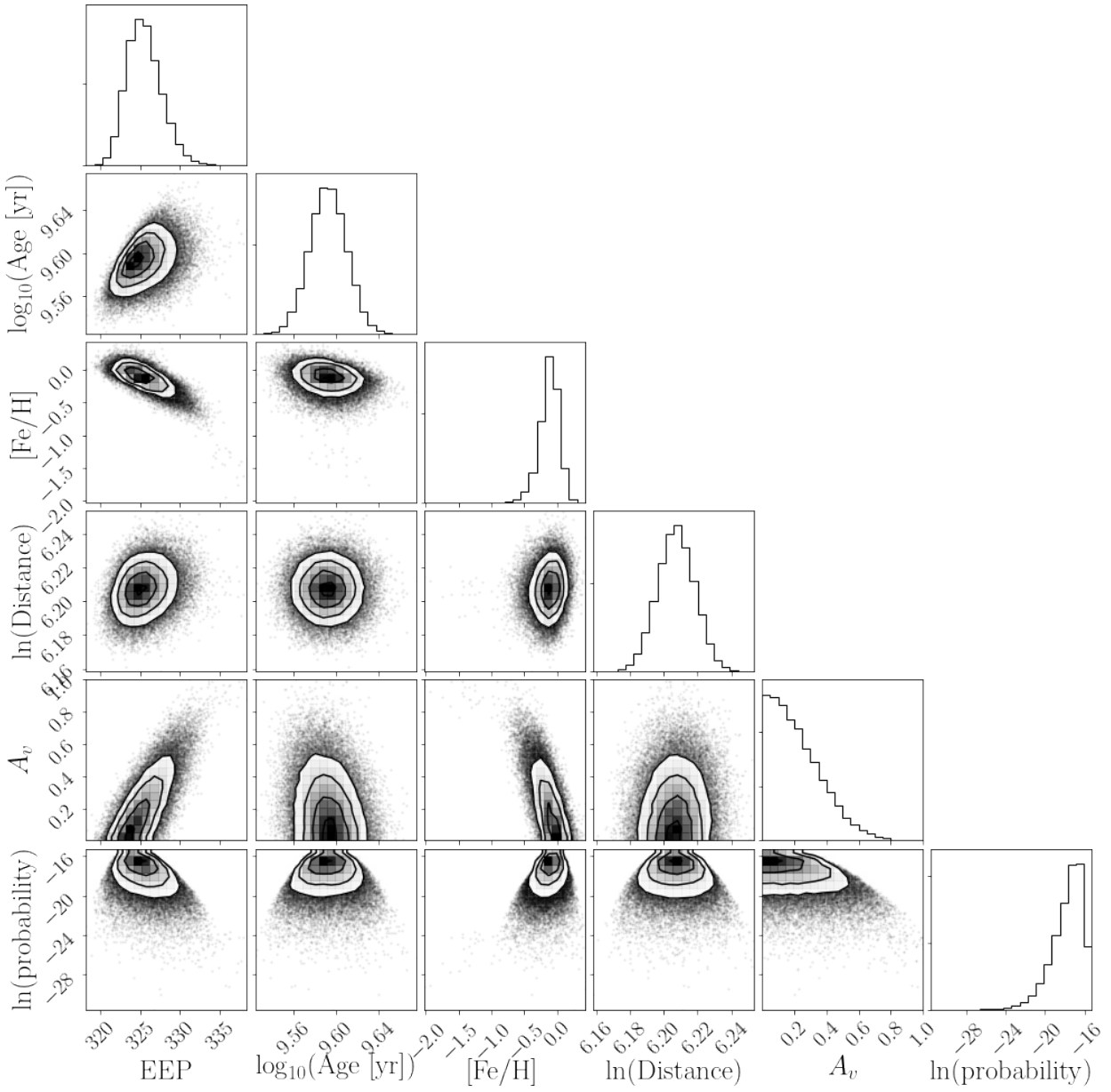
Now we'll plot a histogram of the marginalized posterior over stellar age.

```
plt.hist((10**age_samples)*1e-9, 100);
plt.xlabel("$\mathrm{Age~[Gyr]}$")
plt.xlim(3, 7);
```



And let's make a corner plot.

```
import corner

labels = ["$\mathrm{EEP}$", "$\log_{10}(\mathrm{Age~[yr]})$",
          "$\mathrm{[Fe/H]}$", "$\ln(\mathrm{Distance})$", "$A_v$",
          "$\ln(\mathrm{probability})$"]

corner.corner(flatsamples, labels=labels);
```



3.4 Multiple stars

Looping over multiple stars could look something like this:

```
N = 10  # Measure ages for the first 10 stars in the McQuillan catalog
results = np.empty((N, 5))  # There are 5 dimensions: EEP, age, feh, distance and Av
ages, masses = [np.empty(N) for i in range(2)]

for i in range(10):
    print(i)

    # You have to set up the parameter dictionary for every star.
```

(continues on next page)

(continued from previous page)

```
    iso_params = {"G": (df.G[i], .05),
                  "bp": (df.bp[i], .004),
                  "rp": (df.rp[i], .004),
                  "J": (df.jmag[i], .05),
                  "H": (df.hmag[i], .05),
                  "K": (df.kmag[i], .05),
                  "parallax": (df.parallax[i], df.parallax_error[i])}  # Parallax in␣
↪milliarcseconds.

    star = sd.Star(iso_params, df.Prot[i], df.Prot_err[i], filename="{}".format(df.
↪KID[i]));
    star.fit(max_n=100)  # You should set max_n much higher than this when you run␣
↪for real.

    samps = star.samples
    results[i, :] = np.median(samps, axis=0)

    # Or you could use the built-in functions to calculate medians.
    ages[i] = star.age_results()[0]
    masses[i] = star.mass_results()[0]
```

```
9%|         | 9/100 [00:00<00:01, 86.83it/s]
```

```
0
```

```
100%|| 100/100 [00:00<00:00, 108.64it/s]
 20%|        | 20/100 [00:00<00:00, 92.81it/s]
```

```
1
```

```
100%|| 100/100 [00:01<00:00, 92.95it/s]
 10%|         | 10/100 [00:00<00:00, 91.66it/s]
```

```
2
```

```
100%|| 100/100 [00:00<00:00, 103.48it/s]
 10%|         | 10/100 [00:00<00:00, 90.86it/s]
```

```
3
```

```
100%|| 100/100 [00:00<00:00, 104.43it/s]
 10%|         | 10/100 [00:00<00:00, 99.73it/s]
```

```
4
```

```
100%|| 100/100 [00:00<00:00, 113.67it/s]
 20%|        | 20/100 [00:00<00:00, 99.05it/s]
```

```
5
```

```
100%|| 100/100 [00:00<00:00, 102.35it/s]
 20%|        | 20/100 [00:00<00:00, 94.06it/s]
```

```
6
```

```
100%|| 100/100 [00:00<00:00, 113.04it/s]
 20%|        | 20/100 [00:00<00:00, 96.83it/s]
```

```
7
```

```
100%|| 100/100 [00:01<00:00, 99.99it/s]
 10%|         | 10/100 [00:00<00:00, 96.93it/s]
```

```
8
```

```
100%|| 100/100 [00:00<00:00, 100.72it/s]
  9%|         | 9/100 [00:00<00:01, 88.75it/s]
```

```
9
```

```
100%|| 100/100 [00:00<00:00, 100.42it/s]
```

```
print("Ages = ", results[:, 1], "\n")

print("Equivalent Evolutionary Points = ", results[:, 0], "\n")

print("Metallicities = ", results[:, 2], "\n")

print("Distances = ", results[:, 3], "\n")

print("A_vs = ", results[:, 4], "\n")
```

```
Ages =  [ 8.62244849  9.44316231  8.62296712  9.14169578  9.75353868 10.10893305
  9.60958922  9.4200721   9.75988753  9.18445985]

Equivalent Evolutionary Points =  [327.93635351 333.05156569 328.014169   328.
↪77598444 329.35379586
 330.34274869 330.71146838 330.63614242 329.54340351 328.84924634]

Metallicities =  [ 0.36172406 -2.73808717  0.46339695  0.42917958  0.3983381   0.
↪02608326
  0.28120972 -0.18165572  0.25252582  0.4119441 ]

Distances =  [6.55360207 6.39335019 6.42436376 6.40539686 5.9298473  5.65765442
 5.86783196 6.19545724 6.01413063 6.30991347]

A_vs =  [0.5795697  0.77254342 0.50904773 0.5554797  0.6849852  0.68050969
 0.93230603 0.94863904 0.83407094 0.49548167]
```

If you want to calculate a mass from a EEP, age and metallicity, you can do the following:

```
from isochrones.mist import MIST_Isochrone
mist = MIST_Isochrone()

EEP, age, feh, distance, av = results[0, :]
mass = mist.mass(EEP, age, feh)
print(mass)
```

```
1.9551719917662531
```

3.5 Isochrone fitting only

It's also possible to switch off gyrochronology and just infer an age using isochrones only. This is useful if you'd like to predict ages for a list of stars where only some of them have rotation periods. There are a couple of different ways to do this. The simplest way is just to pass 'None' instead of a period and period uncertainty:

```
star = sd.Star(iso_params, prot=None, prot_err=None)
```

Passing zeros instead of Nones will also work. It's best, but not crucial, to also set the 'isochrone fitting only' key word argument to be true when you run the MCMC:

```
star.fit(max_n=1000, iso_only=True)
```

```
100%|| 1000/1000 [00:05<00:00, 170.38it/s]
```

3.6 Incorporating asteroseismology

As well as apparant magnitudes, parallax and spectroscopic parameters, you can also provide asteroseismic parameters in the dictionary of observables. The dictionary below is the full set of parameters that can be provided. These parameters do not all need to be provided and can be given in any order. The precision and accurancy of inferred ages will improve as more information is provided. See the *isochrones* documentation for more information.

```
iso_params = {"G": (G, G_err),  # Gaia magnitudes
              "bp": (G_bp, G_bp_err),
              "rp": (G_rp, G_rp_err),
              "B": (B, B_err),
              "V": (V, V_err),
              "J": (J, J_err),
              "H": (H, H_err),
              "K": (K, K_err),
              "g": (g, g_err),  # SDSS colours -- these are lower case.
              "r": (r, r_err),
              "i": (i, i_err),
              "z": (z, z_err),
              "teff": (teff, teff_err),  # Spectroscopic properties
              "logg": (logg, logg_err),
              "feh": (feh, feh_err),
              "nu_max": (nu_max, nu_max_err),  # Asteroseismic parameters
              "delta_nu": (delta_nu, delta_nu_err),  # Asteroseismic parameters
              "parallax": (parallax, parallax_error)}  # Parallax in milliarcseconds
```

CHAPTER 4

License & attribution

If you make use of this code, please cite this package and its dependencies. You can find more information about how and what to cite in the citation documentation.

- search

Python Module Index

S

Index

S