stardate Documentation

Release 0.0.1

Ruth Angus

Sep 16, 2019

Contents

stardate currently only works with python3.

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), 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.

Note – if you are running the example below for the first time, the isochrones will be downloaded and this could take a while. This will only happen once though!

CHAPTER 1

Example usage

```python
import stardate as sd

# Create a dictionary of observables
iso_params = {"teff": (4386, 50),     # Teff with uncertainty.
              "logg": (4.66, .05),    # logg with uncertainty.
              "feh": (0.0, .02),      # Metallicity with uncertainty.
              "parallax": (1.48, .1),  # Parallax in milliarcseconds.
              "maxAV": .1}            # Maximum extinction

prot, prot_err = 29, 3

# 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(max_n=1000)

# max_n is the maximum number of MCMC samples. I recommend setting this
# much higher when running for real, or using the default value of 100000.

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

>> stellar age = 2.97 + 0.60 + 0.55
```

If you want to just use a simple gyrochronology model without running MCMC, you can predict a stellar age from a rotation period like this:

```python
import numpy as np
from stardate.lhf import age_model

bprp = .82  # Gaia BP - RP color.
```

(continues on next page)

(continued from previous page)

```
log10_period = np.log10(26)
log10_age_yrs = age_model(log10_period, bprp)
print((10**log10_age_yrs)*1e-9, "Gyr")
>> 4.565055357152765 Gyr
```

Or a rotation period from an age like this:

```
from stardate.lhf import gyro_model_praesepe

bprp = .82  # Gaia BP - RP color.
log10_age_yrs = np.log10(4.56*1e9)
log10_period = gyro_model_praesepe(log10_age_yrs, bprp)
print(10**log10_period, "days")
>> 25.98136488222407 days
```

BUT be aware that these simple relations are only applicable to FGK and early M dwarfs on the main sequence, older than a few hundred Myrs. If you're not sure if gyrochronology is applicable to your star, want the best age possible, or would like proper uncertainty estimates, I recommend using the full MCMC approach.

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, numba, emcee3, tqdm and isochrones.

These can be installed using pip:

```
pip install numpy pandas h5py numba "emcee==3.0rc2" tqdm isochrones
```

You can check out the isochrones documentation if you run into difficulties installing that.

2.2 Running tests

You can test *stardate* from within the base directory using pytest. Install pytest using pip if you don't already have it installed, then navigate to the *stardate* base directory and type:

```
pytest
```

2.3 API documentation

class `stardate.`**`Star`**(*iso_params*, *prot=None*, *prot_err=None*, *Av=None*, *Av_err=None*, *savedir='.'*, *filename='samples'*)

The star object.

Creates the star object which will be set up with stellar parameters and instructions for saving the MCMC results.

Parameters

- **`iso_params`** (*`dict`*) – A dictionary containing all available photometric and spectroscopic parameters for a star, as well as its parallax. All parameters should also have associated uncertainties. This dictionary should be similar to the standard one created for isochrones.py.
- **`prot`** (*`Optional[float]`*) – The rotation period of the star in days.
- **`prot_err`** (*`Optional[float]`*) – The uncertainty on the stellar rotation period in days.
- **`Av`** (*`Optional[float]`*) – The v-band extinction (if known).
- **`Av_err`** (*`Optional[float]`*) – The v-band extinction uncertainty (if known).
- **`savedir`** (*`Optional[str]`*) – The name of the directory where the samples will be saved. Default is the current working directory.
- **`filename`** (*`Optional[str]`*) – The name of the h5 file which the posterior samples will be saved in.

`Av_results`(*burnin=0*)

The Av samples.

The posterior samples for V-band extinction, optionally with a specified number of burn in steps thrown away.

Parameters **`burnin`** (*`Optional[int]`*) – The number of samples to throw away. Default is 0.

Returns The median Av, its 16th and 84th percentile lower and upper uncertainties and the Av samples.

`age_results`(*burnin=0*)

The age samples.

The posterior samples for age, optionally with a specified number of burn in steps thrown away.

Parameters **`burnin`** (*`Optional[int]`*) – The number of samples to throw away. Default is 0.

Returns The median age, its 16th and 84th percentile lower and upper uncertainties and the age samples. Age is log10(Age/yrs).

`distance_results`(*burnin=0*)

The ln(distance) samples.

The posterior samples for distance (in natural log, parsecs), optionally with a specified number of burn in steps thrown away.

Parameters **`burnin`** (*`Optional[int]`*) – The number of samples to throw away. Default is 0.

Returns The median ln(distance), its 16th and 84th percentile lower and upper uncertainties and the ln(distance) samples.

eep_results(*burnin=0*)
The EEP samples.

The posterior samples for Equivalent Evolutionary Point, optionally with a specified number of burn in steps thrown away.

Parameters **burnin** (*Optional[int]*) – The number of samples to throw away. Default is 0.

Returns The median EEP, its 16th and 84th percentile lower and upper uncertainties and the EEP samples.

feh_results(*burnin=0*)
The metallicity samples.

The posterior samples for metallicity, optionally with a specified number of burn in steps thrown away.

Parameters **burnin** (*Optional[int]*) – The number of samples to throw away. Default is 0.

Returns The median metallicity, its 16th and 84th percentile lower and upper uncertainties and the metallicity samples.

fit(*inits=[329.58, 9.5596, -0.0478, 260, 0.0045], nwalkers=50, max_n=100000, thin_by=100, burnin=0, iso_only=False, gyro_only=False, optimize=False, rossby=True, model='praesepe', seed=None, save_samples=False*)
Run MCMC on a star using emcee.

Explore the posterior probability density function of the stellar parameters using MCMC (via emcee).

Parameters

- **inits** (*Optional[array-like]*) – A list of initial values to use for EEP, age (in log10[yrs]), feh, distance (in pc) and Av.
- **nwalkers** (*Optional[int]*) – The number of walkers to use with emcee. The default is 50.
- **max_n** (*Optional[int]*) – The maximum number of samples to obtain (although not necessarily to save – see thin_by). The default is 100000.
- **thin_by** (*Optional[int]*) – Only one in every thin_by samples will be saved. The default is 100. Set = 1 to save every sample (note this substantially slows down the MCMC process because of the additional I/O time.
- **burnin** (*Optional[int]*) – The number of SAVED samples to throw away when accessing the results. This number cannot exceed the number of saved samples (which is max_n/thin_by). Default = 0.
- **iso_only** (*Optional[bool]*) – If true only the isochronal likelihood function will be used.
- **gyro_only** (*Optional[bool]*) – If true only the gyrochronal likelihood function will be used. Beware: this may not do what you might assume it does… In general this setting is not currently very useful!
- **optimize** (*Optional[bool]*) – If True, initial parameters will be found via optimization. Default is False.
- **rossby** (*Optional[bool]*) – If True, magnetic braking will cease after Ro = 2. Default is True.

- **model** (*Optional[bool]*) – The gyrochronology model to use. The default is “praesepe” (the Praesepe-based model). Can also be “angus15” for the Angus et al. (2015) model.
- **seed** (*Optional[int]*) – The random number seed. Set this if you want to regenerate exactly the same samples each time.
- **save_samples** (*Optional[bool]*) – saving samples is the computational bottleneck. If you want to save time and don’t need to save the samples using the HDF5 backend, set this to False.

mass_results(*burnin=0*)

The mass samples.

The posterior samples for mass, calculated from the EEP, age and feh samples, optionally with a specified number of burn in steps thrown away.

Parameters **burnin** (*Optional[int]*) – The number of samples to throw away. Default is 0.

Returns The median mass, its 16th and 84th percentile lower and upper uncertainties and the mass ‘samples’ in units of Solar masses.

run_mcmc()

Runs the MCMC.

Runs the MCMC using emcee. Saves progress to the file specified as <filename.h5> in the <savedir> directory. Samples are saved to this file after every <thin_by> samples are obtained. And only one sample in every <thin_by> is saved. Sampling continues until either max_n samples are obtained or convergence is acheived (this usually takes much more than the default 100,000 maximum number of samples. Convergence is achieved if both 1) the change in autocorrelation time is extremely small (less than 0.01) and 2) if 100 or more independent samples have been obtained.

Returns the emcee sampler object.

Return type sampler

stardate.lhf.**lnprob**(*lnparams*, **args*)

The ln-probability function.

Calculates the logarithmic posterior probability (likelihood times prior) of the model given the data.

Parameters

- **lnparams** (*array*) – The parameter array containing Equivalent Evolutionary Point (EEP), age in log10(yrs), metallicity, distance in ln(pc) and V-band extinction. [EEP, log10(age [yrs]), [Fe/H], ln(distance [pc]), A_v].
- ***args** – The arguments – mod, period, period_err, iso_only, gyro_only, rossby and model. mod is the isochrones starmodel object which is set up in stardate.py. period and period_err are the rotation period and rotation period uncertainty (in days). iso_only should be true if you want to use ONLY isochrone fitting and not gyrochronology. rossby is true if you want to use the van Saders + (2016) weakened magnetic braking law. Set to false to turn this off. model is “angus15” for the Angus + (2015) gyro model or “praesepe” for the Praesepe gyro model.

Returns

The log-posterior probability of the model given the data and the log-prior.

stardate.lhf.**gyro_model_rossby**(*params*, *Ro_cutoff=2*, *rossby=True*, *model='praesepe'*)

Predict a rotation period from parameters EEP, age, feh, distance, Av.

Parameters

- **`params`** (*`array`*) – The stellar parameters: EEP, log10(age), [Fe/H], distance and Av.
- **`Ro_cutoff`** (*`float, optional`*) – The critical Rossby number after which stars retain their rotation period. This is 2.16 in van Saders et al. (2016) and 2.08 in van Saders et al. (2018). We adopt a default value of 2.
- **`rossby`** (*`Optional[bool]`*) – If True (default), the van Saders (2016) weakened magnetic braking law will be implemented. If false, the gyrochronology relation will be used unmodified.
- **`model`** (*`Optional[str]`*) – The gyrochronology model. If "praesepe", the Praesepe-based gyro model will be used (default) and if "angus15", the Angus et al. (2015) model will be used.

Returns The log10(rotation period) and the period standard deviation in dex.

`stardate.lhf.`**`calc_bv`**(*mag_pars*)

Calculate a B-V colour from stellar parameters.

Calculate B-V colour from stellar parameters [EEP, log10(age, yrs), feh, distance (in parsecs) and extinction] using MIST isochrones.

Parameters **`mag_pars`** (*`list`*) – A list containing EEP, log10(age) in years, metallicity, distance in parsecs and V-band extinction, Av, for a star.

Returns B-V color.

`stardate.lhf.`**`convective_overturn_time`**(**args*)

Estimate the convective overturn time.

Estimate the convective overturn time using equation 11 in Wright et al. (2011): https://arxiv.org/abs/1109.4634 log tau = 1.16 - 1.49log(M/M) - 0.54log^2(M/M)

Parameters **`args`** – EITHER mass (float): Mass in Solar units OR eep (float): The Equivalent evolutionary point of a star (355 for the Sun), age (float): The age of a star in log_10(years) and feh (float): the metallicity of a star.

Returns The convective overturn time in days.

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.

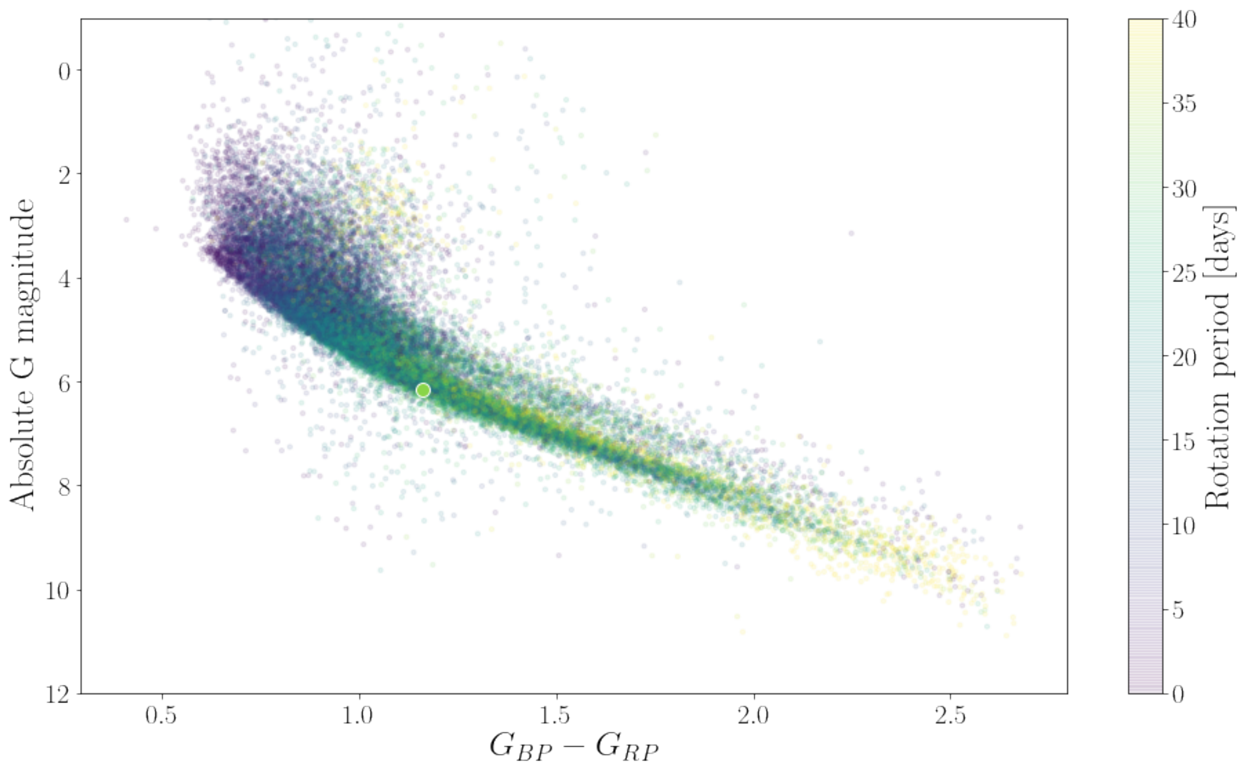
```
import astropy.utils as au
from astropy.io import fits

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, dtype="float64")
```

Now let's merge these two data frames to make one data frame containing rotation periods, Gaia parallaxes, colours, etc.

```
df = pd.merge(mc, 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.phot_g_mean_mag, (1./df.parallax)*1e3)

plt.figure(figsize=(16, 9))
plt.scatter(df.phot_bp_mean_mag - df.phot_rp_mean_mag, 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.phot_bp_mean_mag[ind] - df.phot_rp_mean_mag[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.phot_g_mean_mag[ind], .05),  # We'll just estimate the␣
↪uncertainties for now.
              "bp": (df.phot_bp_mean_mag[ind], .05),
              "rp": (df.phot_rp_mean_mag[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.
```

(continues on next page)

(continued from previous page)

```
```

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).

```
# If you don't want to automatically save samples to disk (much faster):
star = sd.Star(iso_params, prot=df.Prot[ind], prot_err=df.Prot_err[ind])

# If you do want to save samples to disk (slower but assesses convergence,
# although this feature is still in beta), you can provide a directory and filename␣
↪for the file location.
star = sd.Star(iso_params, prot=df.Prot[ind], prot_err=df.Prot_err[ind],
               savedir=".", filename="{}".format(df.KID[ind]));
```

If you want to save samples automatically, 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.

If you want to run without saving samples, you can still save the samples later manually using h5py or similar.

If you know the v-band extinction you can pass that to the Star object too (I recommend using the dustmaps package to calculate this), e.g.:

```
star = sd.Star(iso_params, prot=df.Prot[ind], prot_err=df.Prot_err[ind], Av=.2, Av_
↪err=.01,
               savedir=".", filename="{}".format(df.KID[ind]));
```

Now all we need to do is run the MCMC and wait.

```
nsteps = 10000
star.fit(max_n=nsteps)  # max_n is the number of MCMC steps to take.
```

```
100%|| 10000/10000 [02:23<00:00, 69.50it/s]
```

```
nsteps 10000 burnin 0
```

If you want to save the samples automatically this becomes the following (but note this takes a lot more time for the same number of samples because of the I/O overhead):

```
star.fit(max_n=nsteps, save_samples=True)  # 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%|| 10000/10000 [02:24<00:00, 70.78it/s]
```

```
nsteps 100 burnin 0
```

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()
print("Age = {0:.2f} + {1:.2f} - {2:.2f} Gyr".format(median_age, age_errp, age_errm))
```

```
Age = 3.95 + 0.26 - 3.41 Gyr
```

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

```
burnin = 10  # If you saved samples automatically, burnin is actually thin_by x␣
↪larger than this.
             # If you didn't save samples, your probably want your burnin to be␣
↪higher than this.

mass, mass_errp, mass_errm, mass_samples = star.mass_results(burnin=burnin)
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=burnin)
print("feh = {0:.2f} + {1:.2f} - {2:.2f}".format(feh, feh_errp, feh_errm))

lndistance, lndistance_errp, lndistance_errm, lndistance_samples = star.distance_
↪results(burnin=burnin)
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=burnin)
print("Av = {0:.2f} + {1:.2f} - {2:.2f}".format(Av, Av_errp, Av_errm))

# You can load the flattened samples with
samples = star.samples
print("Sample array shape = ", np.shape(samples))
```

```
Mass = 0.79 + 0.04 - 0.02 M_sun
feh = -0.12 + 0.08 - 0.11
ln(distance) = 6.21 + 0.01 - 0.01
Av = 0.20 + 0.01 - 0.01
Sample array shape =  (5000, 5)
```

If you want to load samples from an H5 file that you automatically saved earlier, you can do the following.

```
from stardate import load_samples, read_samples

# Load the samples.
flatsamples, _3Dsamples, posterior_samples, prior_samples = load_samples(
    "{}.h5".format(df.KID[ind]), burnin=0)

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

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

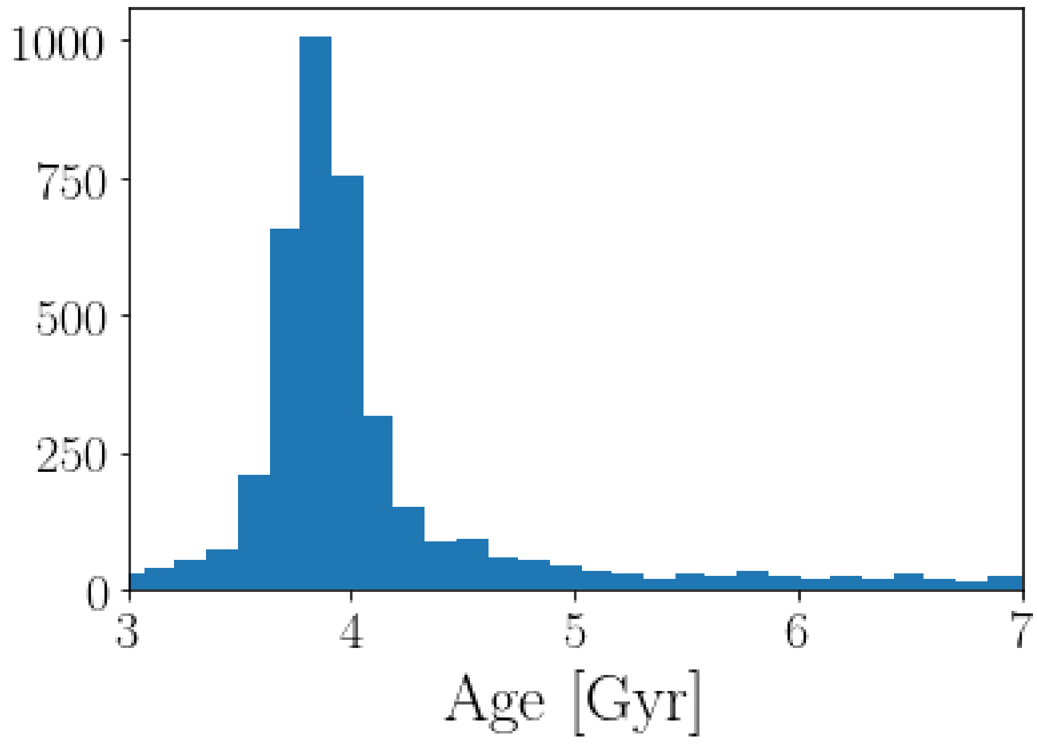
flatsamples is a 2D array of samples, useful for making corner plots. samples is a 3D array of samples, useful for plotting chains. Both flatsamples and samples have an extra dimension containing the log-probability.

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.

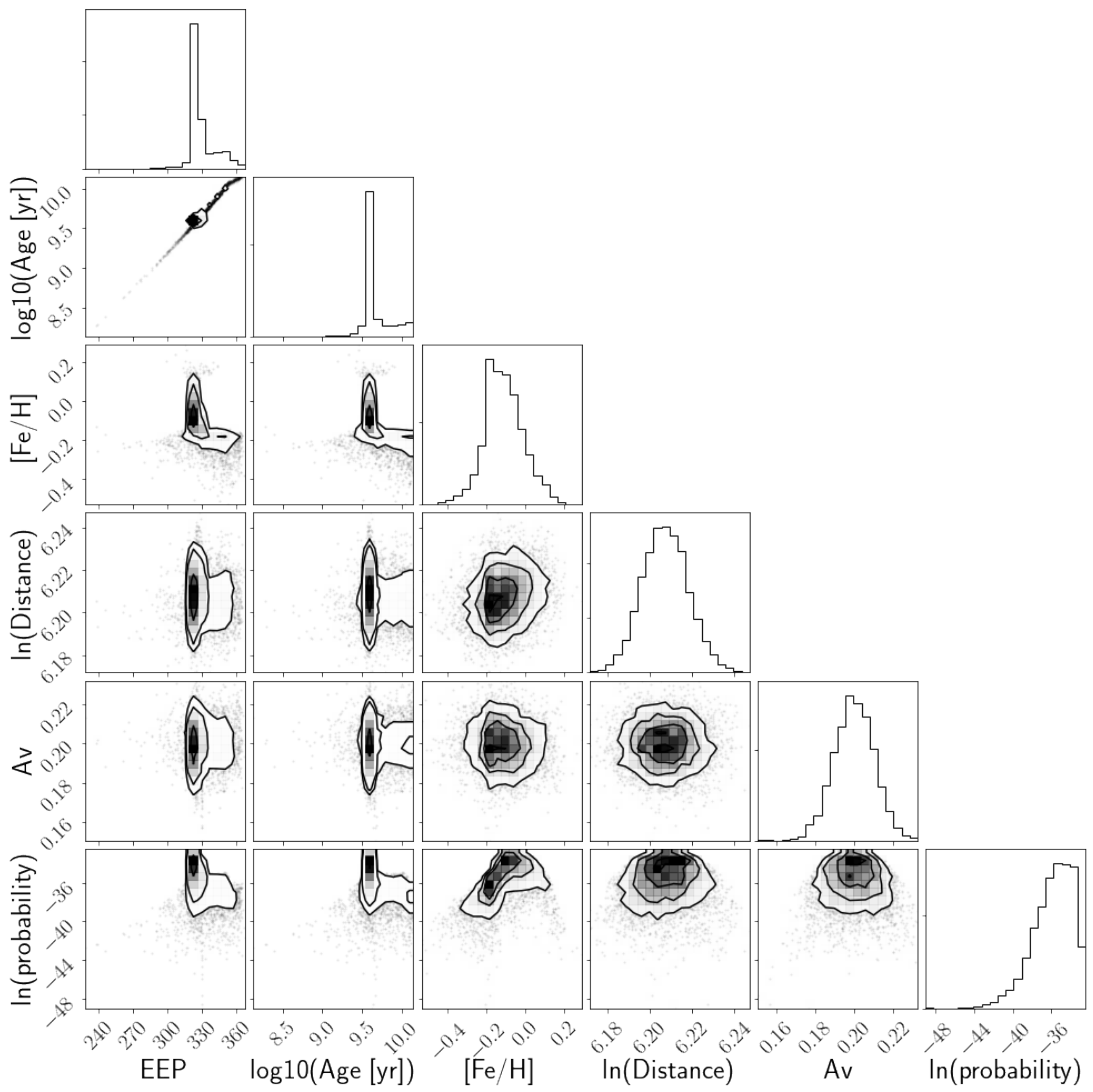
```
plt.hist((10**age_samples)*1e-9, 100);
plt.xlabel("$\mathrm{Age~[Gyr]}$")
plt.xlim(3, 7);
```



And let's make a corner plot.

```
import corner

labels = ["EEP", "log10(Age [yr])", "[Fe/H]", "ln(Distance)", "Av", "ln(probability)"]

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

```
WARNING:root:Too few points to create valid contours
```



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
ndim = 5  # There are 5 dimensions: EEP, age, feh, distance and Av
results = np.empty((N, ndim))  # An array to store summary statistics
ages, masses = [np.empty(N) for i in range(2)]
nsteps = 100   # You should set max_n much higher than this when you run for real.
nwalkers = 50  # stardate uses 50 walkers
sample_array = np.zeros((nsteps*nwalkers, ndim, N))
```

(continues on next page)

(continued from previous page)

```
for i in range(N):

    # You have to set up the parameter dictionary for every star.
    iso_params = {"G": (df.phot_g_mean_mag[i], .05),
                  "bp": (df.phot_bp_mean_mag[i], .004),
                  "rp": (df.phot_rp_mean_mag[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=nsteps)

    sample_array[:, :, i] = star.samples
    results[i, :] = np.median(star.samples, 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]
```

```
100%|| 100/100 [00:01<00:00, 70.12it/s]
  7%|          | 7/100 [00:00<00:01, 67.40it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 73.77it/s]
 14%|         | 14/100 [00:00<00:01, 66.27it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 67.51it/s]
 14%|         | 14/100 [00:00<00:01, 67.04it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 72.05it/s]
 14%|         | 14/100 [00:00<00:01, 64.71it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 70.91it/s]
  7%|          | 7/100 [00:00<00:01, 67.59it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 72.93it/s]
  7%|          | 7/100 [00:00<00:01, 69.26it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 72.45it/s]
 14%|         | 14/100 [00:00<00:01, 65.61it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 72.29it/s]
  7%|          | 7/100 [00:00<00:01, 64.77it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:01<00:00, 76.66it/s]
 14%|         | 14/100 [00:00<00:01, 66.91it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
100%|| 100/100 [00:02<00:00, 45.75it/s]
```

```
nsteps 100 burnin 0
(5000, 5) (5000, 5, 10)
```

```
print("Ages = ", results[:, 1], "\n")  # These ages are in log10(yrs)

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

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

print("Distances = ", results[:, 3], "\n")  # These are in ln(pc)

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

```
Ages =  [9.51112504 9.90461703 9.56008607 9.55981386 9.73172604 9.62600584
 9.58434138 9.66127084 9.57374104 9.5595891 ]

Equivalent Evolutionary Points =  [329.57897528 329.52641737 329.57626038 329.
↪35732305 329.7935348
 329.3695938  329.57982176 329.57296482 330.14254528 329.57980668]

Metallicities =  [-0.13635481 -0.20219718 -0.1029566  -0.0784935   0.04638792 -0.
↪25294648
 -0.19802885 -0.19361006 -0.02608147 -0.04899971]

Distances =  [5.56174874 5.71289638 5.572407   5.77557608 5.74169493 5.56985369
 5.71452286 5.61047538 5.71280237 5.56961465]
```

(continues on next page)

(continued from previous page)

```
A_vs =  [0.03380475 0.18007945 0.03253148 0.08023092 0.228755   0.09026418
 0.38374834 0.01566177 0.51577132 0.00850142]
```

3.5 Calculating other physical stellar parameters.

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

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

# Flatten sample array for the first star, computed above
flatsamples = np.reshape(sample_array[:, :, 0], (nsteps*nwalkers, ndim))

eep_samples = flatsamples[:, 0]
log_age_samples = flatsamples[:, 1]
feh_samples = flatsamples[:, 2]

mass_samps = mist.interp_value([eep_samples, log_age_samples, feh_samples], ["mass"])
print("mass = {0:.2f} M_sun".format(np.median(mass_samps)))
```

```
mass = 0.87 M_sun
```

You can use this same format to calculate other physical parameters too, e.g.:

```
radius_samps = mist.interp_value([eep_samples, log_age_samples, feh_samples], ["radius
↪"])
print("radius = {0:.2f} R_sun".format(np.median(radius_samps)))
```

```
radius = 0.81 R_sun
```

```
teff_samps = mist.interp_value([eep_samples, log_age_samples, feh_samples], ["Teff"])
print("teff = {0:.0f} K".format(np.median(teff_samps)))
```

```
teff = 5485 K
```

```
logg_samps = mist.interp_value([eep_samples, log_age_samples, feh_samples], ["logg"])
print("logg = {0:.2f}".format(np.median(logg_samps)))
```

```
logg = 4.56
```

Take a look at the column headers of the MIST tables in the isochrones documentation for more physical parameters that are supported.

3.6 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, or if you'd like to compare between gyro on and gyro off. 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, or nothing at all will also work. You can 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 [00:06<00:00, 160.22it/s]
```

```
nsteps 1000 burnin 0
```

3.7 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

Index