gyro-interp

Release 0.3

Luke Bouma et al

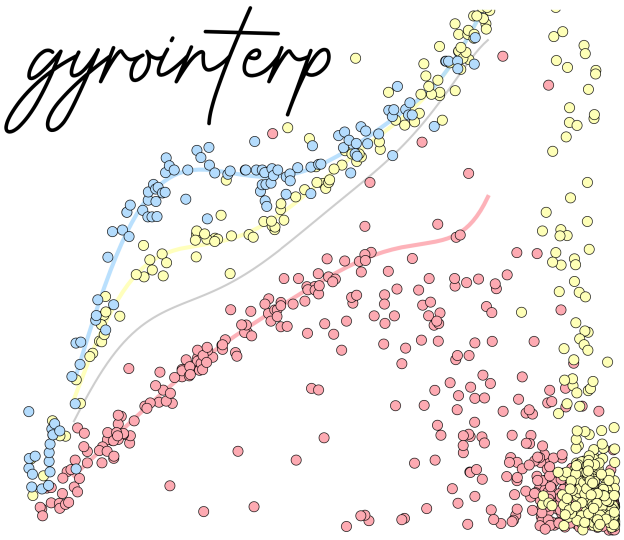
May 15, 2023

CONTENTS:

Hi! Welcome to the documentation for `gyrointerp`, a Python package that calculates gyrochronal ages by interpolating between open cluster rotation sequences.

`gyrointerp` packages the model from Bouma, Palumbo & Hillenbrand (2023) into a fast and easy-to-use framework. The documentation below will walk you through the most common use-cases. For brevity, we'll refer to that paper as BPH23.

This package is designed to meet the needs of working astronomers with interests in gyrochronal age measurement, and we encourage community involvement. If you find a bug or would like to request a feature, please do create an issue on Github.



CHAPTER

ONE

ATTRIBUTION:

The reference for both the software and method is Bouma, Palumbo & Hillenbrand (2023). The relevant bibtex entry is as follows.

```
@ARTICLE{2023ApJ...947L...3B,
       author = {{Bouma}, Luke G. and {Palumbo}, Elsa K. and {Hillenbrand}, Lynne A.},
        title = "{The Empirical Limits of Gyrochronology}",
      journal = {\apjl},
     keywords = {Stellar ages, Stellar rotation, Field stars, Bayesian statistics, 1581,␣
↪1629, 2103, 1900, Astrophysics - Solar and Stellar Astrophysics, Astrophysics -␣
↪Instrumentation and Methods for Astrophysics},
         year = 2023,
        month = apr,
       volume = {947},
       number = {1},
          eid = {L3},
        pages = {L3},
          doi = {10.3847/2041-8213/acc589},
archivePrefix = {arXiv},
       eprint = {2303.08830},
 primaryClass = {astro-ph.SR},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2023ApJ...947L...3B},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

If your result is particularly dependent on the rotation data from any one cluster, we also encourage you to refer to the relevant study:

- Per: Boyle & Bouma (2023)
- Pleiades: Rebull et al. (2016)
- Blanco-1: Gillen et al. (2020)
- Psc-Eri: Curtis et al. (2019a)
- NGC-3532: Fritzewski et al. (2022)
- Group-X: Messina et al. (2022)
- Praesepe: Rampalli et al. (2021)
- NGC-6811: Curtis et al. (2019b)
- NGC-6819: Meibom et al. (2015)
- Ruprecht-147 Curtis et al. (2020)

- M67: Barnes et al. (2016) and Dungee et al (2022).

CHAPTER

TWO

USER GUIDE:

2.1 Installation

`gyrointerp` works in Python>3.6. To use the code, first make sure you have the latest versions of the "standard astro stack" installed. With `pip`, you can do this with the command:

```
$ pip install numpy astropy pandas scipy --upgrade
```

Next, install `gyrointerp`:

```
$ pip install gyrointerp
```

We recommend installing and running `gyrointerp` in a conda virtual environment. You can install anaconda or miniconda here; the instructions here provide more information about conda virtual environments.

Issues?

If you run into any issues installing `gyrointerp`, please create an issue on Github.

2.2 Examples

2.2.1 Gyrochronal age for one star

Given a single star's rotation period, effective temperature, and uncertainties, what is the gyrochronological age posterior over a grid spanning 0 to 2.6 Gyr?

```
import numpy as np
from gyrointerp import gyro_age_posterior

# units: days
Prot, Prot_err = 11, 0.2

# units: kelvin
Teff, Teff_err = 4500, 100

# uniformly spaced grid between 0 and 2600 megayears
age_grid = np.linspace(0, 2600, 500)

# calculate the age posterior at each age in `age_grid`
age_posterior = gyro_age_posterior(
```

(continues on next page)

(continued from previous page)

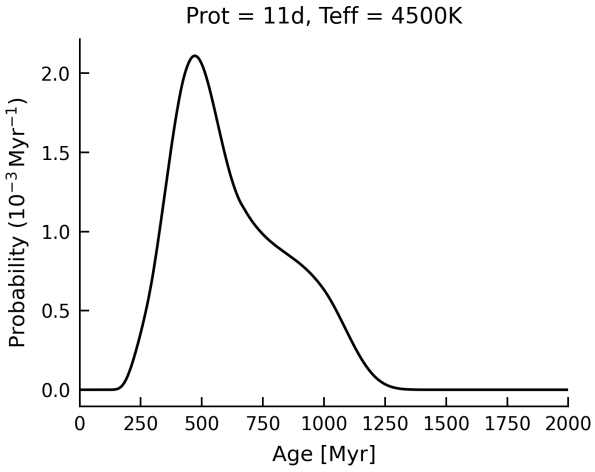
```
    Prot, Teff,
    Prot_err=Prot_err, Teff_err=Teff_err,
    age_grid=age_grid
)
```

This takes about 30 seconds to run on my laptop. You can then use a helper function to calculate summary statistics of interest:

```
# calculate dictionary of summary statistics
from gyrointerp import get_summary_statistics
result = get_summary_statistics(age_grid, age_posterior)

print(f"Age = {result['median']} +{result['+1sigma']} -{result['-1sigma']} Myr.")
```

You can also plot the posterior using matplotlib:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot(age_grid, 1e3*age_posterior, c='k', lw=1)
ax.update({
    'xlabel': 'Age [Myr]',
    'ylabel': 'Probability ($10^{-3}\,$Myr$^{-1}$)',
    'title': f'Prot = {Prot}d, Teff = {Teff}K',
    'xlim': [0,2000]
})
plt.show()
```



This age posterior is highly asymmetric because this particular rotation period and temperature overlap with the era of "stalled spin-down".

2.2.2 Gyrochronal ages for many stars

Given the rotation periods, temperatures, and uncertainties for many stars, what are the implied age posteriors?

In the example below, we will calculate age posteriors using the `gyro_age_posterior_list` function. We will do this for a number of stars equal to the number of cores on your computer, so that the runtime should be roughly the same as for the single-star example above.

```
import os
import numpy as np, pandas as pd
from gyrointerp import gyro_age_posterior_list, get_summary_statistics

def main():

    # Define temperatures, periods, and uncertainties for some stars.
    # At >~20 days, assume a few percent relative uncertainty on periods.
    N_stars = os.cpu_count()
    Teffs = np.linspace(4000, 5500, N_stars)
    Teff_errs = 100 * np.ones(N_stars)
    Prots = np.linspace(15, 22, N_stars)
    Prot_errs = 0.03 * Prots

    # The output posteriors will be cached at ~/.gyrointerp_cache/{cache_id}
    cache_id = 'my_awesome_stars'

    # A 5500 K star with Prot = 22 d will be near the Ruprecht-147 sequence.
    # Let's extend the age_grid up to 4000 Myr (4 Gyr); the extrapolation
    # past 2.6 Gyr will be based on the M67 data.
    age_grid = np.linspace(0, 4000, 500)

    # Let's pass optional star IDs to name the posterior csv files.
    star_ids = [f"FOO{ix}" for ix in range(N_stars)]

    # This function will compute the posteriors, and cache them to CSV files
    csvpaths = gyro_age_posterior_list(
        cache_id, Prots, Teffs, Prot_errs=Prot_errs, Teff_errs=Teff_errs,
        star_ids=star_ids, age_grid=age_grid, bounds_error="4gyrlimit",
        interp_method="pchip_m67"
    )

    # Read the posteriors and print their summary statistics.
    for csvpath, Prot, Teff in zip(sorted(csvpaths), Prots, Teffs):
        df = pd.read_csv(csvpath)
        r = get_summary_statistics(df.age_grid, df.age_post)
        msg = f"Age = {r['median']} +{r['+1sigma']} -{r['-1sigma']} Myr."
        print(f"Teff {int(Teff)} Prot {Prot:.2f} {msg}")

if __name__ == "__main__":
    main()
```

In this example we guarded the multiprocessing being executed in `gyro_age_posterior_list` in a `__main__` block, per the suggestion in the multiprocessing docs. This example also takes about 30 seconds to run on my laptop. Since this is the same runtime as the single-star case, this means that the multithreading is doing what we want.

2.2.3 Visual interpolation for a star's age

We sometimes might want to examine where a given star falls in the rotation-temperature plane in comparison to known reference clusters. If a star has a rotation period that corresponds to lots of possible ages, we should be sure that that this expectation is being mirrored in the age posteriors! Accounting for this type of intrinsic population level scatter is one of the main goals of the BPH23 model.

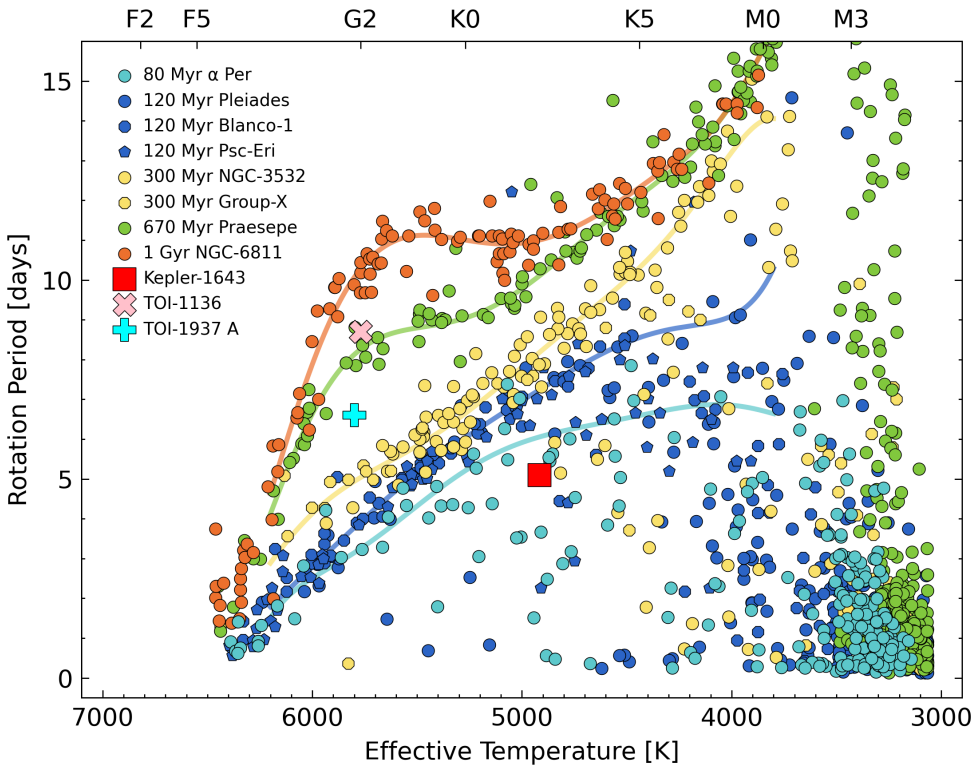
In this example, we will compare the rotation periods of a few stars that are known to have transiting planets against the reference cluster datasets. To make the plot, let's first install a package to automate the matplotlib style-setting:

```
$ pip install aesthetic
```

We can then use the `plot_prot_vs_teff` function under `gyrointerp.plotting`:

```
from gyrointerp.plotting import plot_prot_vs_teff

# write the results to the current working directory
outdir = "./"

# show these cluster Prot vs Teff datasets
reference_clusters = [
    ' Per', 'Pleiades', 'Blanco-1', 'Psc-Eri', 'NGC-3532', 'Group-X',
    'Praesepe', 'NGC-6811'
]

# underplot these polynomial fits
model_ids = [
    ' Per', '120-Myr', '300-Myr', 'Praesepe', 'NGC-6811'
]

# overplot these stars with big markers
custom_stardict = {
    "Kepler-1643": {"Prot":5.1, "Teff":4916, "m":"s", "c":"red"},
    "TOI-1136": {"Prot":8.7, "Teff":5770, "m":"X", "c":"pink"},
    "TOI-1937 A": {"Prot":6.6, "Teff":5798, "m":"P", "c":"aqua"},
}

# make the plot
plot_prot_vs_teff(
    outdir, reference_clusters=reference_clusters, model_ids=model_ids,
    custom_stardict=custom_stardict, writepdf=0
)
```

which yields the following plot:



Kepler-1643, TOI-1136, and TOI-1937 provide three interestingly different examples. Kepler-1643 is ~40 Myr old based on cluster membership, and it hosts a close-in mini-Neptune around twice the size of Earth. TOI-1136 is a field star with six known transiting planets, and rotation is currently the most constraining line of evidence for its ~700 Myr age. Finally, TOI-1937 is a system for which gyrochronology should probably not be applied. The reasons are that it is both a known binary, with a widely-separated companion, and the primary also hosts a hot Jupiter, which might spin up the primary through tides. This kind of interaction is exactly the kind of thing that we tried to avoid by cleaning out binaries in BPH23! While it is in principle possible to construct models that account for known tidal or other spin-up, the BPH23 model does not attempt to do this.

2.3 Caveats

2.3.1 Stellar Evolution

This code models the ensemble evolution of rotation periods for main-sequence stars with temperatures of 3800-6200 K (masses of 0.5-1.2 solar). The calibration data for the model span 0.08-2.6 Gyr. At younger ages, rotation periods are less predictive of age, and other age indicators may be more useful. At older ages, the effects of stellar evolution begin to become important, especially for the more massive stars.

If you have auxiliary data, such as stellar surface gravities derived from spectra, they can help determine whether your star might have undergone significant nuclear evolution – in other words whether it is a subgiant or even a giant. If it is, this model is not applicable.

2.3.2 Binarity

Before applying gyrochronology, it's worth asking whether your star might be a binary. If it is, you should proceed carefully. Binarity can observationally bias temperature and rotation period measurements. There is also a physical concern that even a widely-separated companion might influence a star's spin-down, by encouraging early dispersal of the protostellar disk.

A few ways to determine whether your star might be a binary include:

- Get high resolution images, and determine whether there are nearby companions. You can also check the Gaia point source catalog for such companions, though you may not achieve the same contrast limits close to the star.
- Check the Gaia point source catalog for whether the renormalized unit weight error (RUWE) exceeds some threshold. Additional astrometric scatter, which is what this quantity measures, can be caused by either a marginally resolved point source, or by astrometric binarity.
- Obtain spectra, and check if they are double-lined, or if they show significant radial variations over time. The *radial_velocity_error* column in Gaia DR3 can help diagnose the latter case, although care should be taken for stars that are faint and have low S/N spectra.
- Query the Gaia point source catalog in a local spatial volume around your star. With the resulting sample of stars, check whether your star is an outlier in the HR diagram. This can be an indication of photometric binarity.

As mentioned in the section on *Visual interpolation for a star's age*, the same types of considerations apply to hot Jupiter systems, or any kinds of systems in which tidal effects might alter the star's rotation period.

Generally speaking, the best approaches will differ based on your stars of interest. A few separate utilities that can help in assessing these types of utilties are available through astroquery.gaia, and cdips.utils.gaiaqueries. Both are pip-installable.

2.4 Frequently asked questions

2.4.1 What happens for stars cooler than 3800 K?

For stars with measured effective temperatures outside of our calibrated range (3800-6200 K), we have opted to force our model to return `NaN` age posteriors. For stars near the boundary (e.g., 3800+/-100 K) for which a portion of the likelihood is outside our nominal temperature range, the returned posterior includes that region through a local extrapolation of the mean polynomial models discussed in BPH23.

The same point applies for stars hotter than 6200 K.

2.4.2 What happens for stars younger than 80 Myr?

For stars younger than 0.08 Gyr, we fixed our "mean rotation period model" to equal the lowest reference polynomial rotation period values, as set by the Per cluster. This yields posterior distributions that are uniformly distributed at ages younger than 0.08 Gyr. So, the age posteriors for such systems will be upper limits, which may or may not be useful for you!

Once stars begin getting this young, there are other age-dating techniques that may be more constraining. For instance, you might search spectra for lithium absorption, check broadband photometry for near-infrared excesses, or analyze HR diagrams for evidence of pre-main-sequence evolution.

2.4.3 What happens for stars older than 2.6 Gyr?

For stars older than 2.6 Gyr, we have implemented a few possible extrapolation approaches. Our default adopted approach, the `pchip_m67` extrapolation, provides one plausible interpolation between 2.6 and 4 Gyr based on the M67 data, though it is subject to larger systematic errors than e.g., our model between 1 and 2.6 Gyr because the change of the slope in rotation period versus time is not as well-constrained. After 4 Gyr, as for Per we simply force the mean model's rotation period to equal the highest reference rotation period values, but now as set by M67. This yields posterior probability distributions that are uniformly distributed at ages older than 4 Gyr.

So, the age posteriors for such systems will be lower limits, and they may or may not be useful for you!

2.4.4 Are there movies?

Yes! This movie shows random draws from the model over the first two gigayears. This movie compares these random draws to available cluster data at fixed timesteps.

2.5 gyrointerp package

2.5.1 gyrointerp.gyro_posterior module

Main drivers:

```
gyro_age_posterior
gyro_age_posterior_list
gyro_age_posterior_mcmc
```

gyrointerp.gyro_posterior.**gyro_age_posterior**(*Prot*, *Teff*, *Prot_err=None*, *Teff_err=None*, *age_grid=np.linspace(0, 3000, 500)*, *interp_method='pchip_m67'*, *bounds_error='4gyrlimit'*, *n=None*, *N_grid='default'*, *age_scale='default'*, *popn_parameters='default'*, *verbose=False*)

Given a single star's rotation period and effective temperature, and (optionally) their uncertainties, what is the posterior probability distribution for its age?

The answer returned by this code assumes that the `gyrointerp.models.slow_sequence_residual` model holds, which is the probability distribution described in BPH23 for the distribution of rotation periods at any given age and temperature.

If Prot_err and Teff_err are not specified, they are assumed to be 1% relative and 100 K, respectively. Spectroscopic temperature are acceptable, though the preferred effective temperature scale is implemented in `gyrointerp.teff`, in the `given_dr2_BpmRp_AV_get_Teff_Curtis2020` function. This requires an accurate estimate for the reddening. Whatever your effective temperature scale, it should ideally be compared against that in Appendix A of Curtis+2020.

Parameters

- **Prot** (*int or float*) – Rotation period in units of days.
- **Prot_err** (*int or float*) – Rotation period uncertainty in units of days.
- **Teff** – (int or float): Effective temperature in units of degrees Kelvin. Must be between 3800 and 6200 K.
- **Teff_err** (*int or float*) – Effective temperature uncertainty in units of degrees Kelvin.

- **age_grid** (*np.ndarray*) – Grid over which the age posterior is evaluated; units here and throughout are fixed to be megayears (10^6 years). A fine choice is 500 points uniformly distributed between 0 and 3000 Myr: `np.linspace(0, 3000, 500)`, assuming that the default choices of `bounds_error == '4gyrlimit'` and `interp_method == 'pchip_m67'` are being used.
- **interp_method** (*str*) – How will you interpolate between the polynomial fits to the reference open clusters? "pchip_m67" is the suggested default method, which uses Piecewise Cubic Hermite Interpolating Polynomials (PCHIP) to interpolate over not only 0.08-2.6 Gyr, but also sets the gradient in Prot vs Time in the 1-2.6 Gyr interval based on the observations of M67 from Barnes+2016 and Dungee+2022. This yields an evolution of the rotation period envelope that is smooth and by design fits the cluster data from the age of alpha-Per through M67. Other available interpolation methods include "skumanich_vary_n", "alt", "diff", "skumanich_fix_n_0.XX", "1d_linear", "1d_slinear", "1d_quadratic", and "1d_pchip", some of which are described in Appendix A of BPH23. Unless you know what you are doing, "pchip_m67" is recommended.
- **bounds_error** (*str*) – How will you extrapolate at <0.08 Gyr and >2.6 Gyr? Available options are "nan", "limit" or "4gyrlimit". If "limit", then extrapolate by returning the fixed limiting rotation period at the oldest or youngest cluster given in `models.slow_sequence.reference_model_ids`. If "4gyrlimit" (the suggested default), extrapolate out to 4 Gyr by also including M67. Past 4Gyr, use the same behavior as "limit". If one is interested in obtaining unbiased ages near the recommended 2.6 Gyr limit of this code, use "4gyrlimit", otherwise "limit" will overestimate the probability density beyond 2.6 Gyr. Finally, if "nan", ages above or below the minimum reference age return NaNs.
- **n** (*None, int, or float*) – Power-law index analogous to the Skumanich braking index, but different in detail (read the source code to see how). This is used only if `interp_method == "alt"` or `interp_method == "diff"`, neither of which is recommended for most users. Default is None.
- **N_grid** (*str or int*) – The number of grid points in effective temperature and residual-period over which the integration is performed to evaluate the posterior (Equation 1 of BPH23). "default" sets it to `N_grid = (Prot_grid_range)/Prot_err`, where "Prot_grid_range" is set to 20 days, the range of the grid used in the integration. This default ensures convergence, because numerical tests show convergence down to ~0.7x smaller grid sizes. If an integer is passed, will use that instead. For most users, "default" is best.
- **age_scale** (*str*) – "default", "1sigmaolder", or "1sigmayounger". Shifts the entire age scale appropriately, based on the user's beliefs about what ages of reference clusters are correct. The scale is as described in the systematic uncertainty tests of BPH23, and defined in `gyrointerp.age_scale`.
- **popn_parameters** (*str*) – "default", or (dict) containing the population-level free parameters. Keys of "a0", "a1", "k0", "k1", "y_g", "l_hidden", and "k_hidden" must all be specified. Wrapped by `gyro_age_posterior_mcmc`, for users who wish to do the population-level hyperparameter sampling described by BPH23.

Returns

p_ages

Numpy array containing posterior probabilities at each point of the age_grid. Values are NaN if Teff outside of [3800, 6200] K.

Return type

np.ndarray

gyrointerp.gyro_posterior.**gyro_age_posterior_list**(*cache_id*, *Prots*, *Teffs*, *Prot_errs=None*, *Teff_errs=None*, *star_ids=None*, *age_grid=np.linspace(0, 3000, 500)*, *N_grid='default'*, *bounds_error='4gyrlimit'*, *interp_method='pchip_m67'*, *nworkers=None*)

Given rotation periods and effective temperatures for many stars, run them in parallel. This is a thin wrapper to `gyro_age_posterior` assuming default parameters.

Parameters

- **cache_id** (*str*) – The output posteriors will be cached to `~/.gyrointerp_cache/{cache_id}` (required).
- **Prots** (*np.ndarray*) – 1-D array of rotation periods
- **Teffs** (*np.ndarray*) – 1-D array of temperatures, same length as Prots
- **Prot_errs** (*np.ndarray*) – 1-D array of rotation period uncertainties. If None, assumes 1% relative uncertainties by default.
- **Teff_errs** (*np.ndarray*) – 1-D array of effective temperature uncertainties. If None, assumes 100K by default.
- **star_ids** (*np.ndarray of strings*) – Arbitrary strings naming your stars; optional. For example, if you give "TIC1234567", posteriors will be written to CSV files with a pattern matching `TIC1234567_ProtXX.XXXX_TeffYYYY.Y_limitgrid_defaultparameters.csv`. If None, then the identifier is omitted.
- **nworkers** (*int or None*) – Number of workers to thread over. By default, will be taken to be all available CPU cores.

Returns

List of paths to all available output posteriors at `~/.gyrointerp_cache/{cache_id}`. If you re-use your *cache_id*, this means you will get more than you asked for!

gyrointerp.gyro_posterior.**gyro_age_posterior_mcmc**(*Prot*, *Teff*, *Prot_err=None*, *Teff_err=None*, *age_grid=np.linspace(0, 3000, 500)*, *verbose=False*, *bounds_error='4gyrlimit'*, *interp_method='pchip_m67'*, *N_grid='default'*, *n=None*, *age_scale='default'*, *N_pop_samples=512*, *N_post_samples=10000*, *cachedir=None*)

Given the rotation period and effective temperature of a single star, sample over the population-level hyperparameters a1/a0, ln k0, ln k1, and y_g to determine the posterior probability distribution of the age. These are the dotted lines in the upper panel of Fig3 in BPH23.

Parallelization is done over the hyperparameters. However, the computational cost for a given star is about 1000x that of running the best-fit hyperparameters, as implemented in `gyro_age_posterior`. Use of this function is therefore generally not recommended, unless you have an understood need for doing things this way.

Arguments are as in `gyro_age_posterior`, but with four additions:

Parameters

- **cache_id** (*str*) – The output posteriors will be cached to `~/.gyrointerp_cache/{cache_id}` (required).
- **N_pop_samples** (*int*) – The number of draws from the posteriors for the aforementioned parameters to average over.

- **N_post_samples** (*int*) – For each of the above draws, the number of draws from the resulting age posterior to cache before concatenating them all together.
- **cachedir** (*str*) – Path to directory where individual posteriors will be cached (for every star, *N_post_samples* files will be written here!). It is highly recommended that you specify this.

Returns

p_ages

Numpy array containing posterior probabilities at each point of the age_grid. Values are NaN if Teff outside of [3800, 6200] K.

Return type
np.ndarray

2.5.2 gyrointerp.models module

Functions to fit rotation versus effective temperature sequences, or to quickly return the results of those fits (including their interpolations!)

Contents:

`reference_cluster_slow_sequence`
`slow_sequence`
`slow_sequence_residual`

Helper functions:

`teff_zams`
`g_lineardecay`

gyrointerp.models.**g_lineardecay**(*age*, *bounds_error='4gyrlimit'*, *y_g=1 / 2*)

Function *g(t)* from BPH23 defining the linear rate at which the uniform component amplitude from `models.slow_sequence_residual` decreases with age. Unity at <=120 Myr, decreasing linearly to *y_g* by 300 Myr (eg. 1/2, 1/4, 1/6). Decreases linearly thereafter, and once it reaches zero, it stays at zero.

gyrointerp.models.**reference_cluster_slow_sequence**(*Teff*, *model_id*, *poly_order=7*, *verbose=True*)

Given a set of temperatures, get the rotation periods implied by a polymial fit to the Prot-Teff slow sequence of a particular cluster between 3800 and 6200 K.

Parameters

- **Teff** (*np.ndarray / float / list-like iterable*) – Effective temperature in Kelvin. Curtis+2020 Gaia DR2 BP-RP scale, or spectroscopic effective temperatures, preferred above all other options.
- **model_id** (*str*) – String identifying the desired reference cluster. Can be any of `[' Per', 'Pleiades', 'Blanco-1', 'Psc-Eri', 'NGC-3532', 'Group-X', 'Praesepe', 'NGC-6811', '120-Myr', '300-Myr', 'NGC-6819', 'Ruprecht-147', '2.6-Gyr', 'M67']`, where '120-Myr' will concatenate of Pleiades, Blanco-1, and Psc-Eri into one polynomial fit, and '300-Myr' will concatenate NGC-3532 and Group-X.
- **poly_order** (*int*) – Integer order of the polynomial fit.

Returns

Prot_model

Numpy array containing rotation periods for each requested temperature.

Return type
np.ndarray

gyrointerp.models.**slow_sequence**(*Teff*, *age*, *poly_order=7*, *reference_model_ids=[' Per', '120-Myr', '300-Myr', 'Praesepe', 'NGC-6811', '2.6-Gyr']*, *reference_ages=[80, 120, 300, 670, 1000, 2600]*, *verbose=True*, *bounds_error='4gyrlimit'*, *interp_method='pchip_m67'*, *n=None*)

Given an age and a set of temperatures, return the implied slow sequence rotation periods, as derived from interpolation using the reference clusters with known ages. This function is the "mean gyrochronal model", i.e., it assumes slow sequence evolution.

Parameters

- **age** (*int or float*) – An integer or float corresponding to the age for which we want a rotation period. Units: Myr (=10^6 years).
- **Teff** (*float or iterable of floats*) – Effective temperature(s) of the sample to be dated. Units: Kelvin. Must be between 3800 and 6200 K.
- **reference_model_ids** (*list*) – This list can include any of `[' Per', 'Pleiades', 'Blanco-1', 'Psc-Eri', 'NGC-3532', 'Group-X', 'Praesepe', 'NGC-6811', '120-Myr', '300-Myr', '2.6-Gyr', 'NGC-6819', 'Ruprecht-147', 'M67']` The default is set as described in the manuscript, to enable gyro-age derivations between 0.08-2.6 Gyr. Note that "120-Myr" and "300-Myr" are concenations of the relevant clusters.
- **reference_ages** (*iterable of floats*) – Ages (units of Myr) corresponding to `reference_model_ids`.
- **verbose** (*bool*) – True or False to choose whether to print error messages. Default is False
- **interp_method** (*str*) – How will you interpolate between the polynomial fits to the reference open clusters? "pchip_m67" is the suggested default method, which uses Piecewise Cubic Hermite Interpolating Polynomials (PCHIP) to interpolate over not only 0.8-2.6 Gyr, but also sets the gradient in Prot vs Time in the 1-2.6 Gyr interval based on the observations of M67 from Barnes+2016 and Dungee+2022. This yields an evolution of the rotation period envelope that is smooth and by design fits the cluster data from the age of alpha-Per through M67. Other available interpolation methods include "skumanich_vary_n", "alt", "diff", "skumanich_fix_n_0.XX", "1d_linear", "1d_slinear", "1d_quadratic", and "1d_pchip", some of which are described in Appendix A of BPH23. Unless you know what you are doing, "pchip_m67" is recommended.
- **bounds_error** (*str*) – How will you extrapolate at <0.08 Gyr and >2.6 Gyr? Available options are "nan", "limit" or "4gyrlimit". If "limit", then extrapolate by returning the fixed limiting rotation period at the oldest or youngest cluster given in `models.slow_sequence.reference_model_ids`. If "4gyrlimit" (the suggested default), extrapolate out to 4 Gyr by also including M67. Past 4Gyr, use the same behavior as "limit". If one is interested in obtaining unbiased ages near the recommended 2.6 Gyr limit of this code, use "4gyrlimit". Finally, if "nan", ages above or below the minimum reference age return NaNs.
- **n** (*None, int, or float*) – Power-law index analogous to the Skumanich braking index, but different in detail (see the implementation). This is used only if `interp_method == "alt"` or `interp_method == "diff"`, neither of which is recommended for most users. Default is None.

Output:

np.ndarray : Prot_slow_sequence

Numpy array containing period estimate for the slow sequence at the given effective temperatures and age, in units of days.

gyrointerp.models.**slow_sequence_residual**(*age*, *y_grid=np.linspace(-14, 6, 1000)*, *teff_grid=np.linspace(3800, 6200, 1001)*, *poly_order=7*, *n=None*, *reference_model_ids=[' Per', '120-Myr', '300-Myr', 'Praesepe', 'NGC-6811', '2.6-Gyr']*, *reference_ages=[80, 120, 300, 670, 1000, 2600]*, *popn_parameters='default'*, *verbose=True*, *bounds_error='4gyrlimit'*, *interp_method='pchip_m67'*)

Given an effective temperature and an age, return the 2-D distribution of residuals around and underneath the slow sequence, sampled onto grids of *y_grid* X *teff_grid*, where *y_grid* is the residual of (rotation period data - mean gyrochronal model). This is Equation 7 of BPH23.

The two components of the residual are:

- a gaussian in *y_grid* , with an age-varying cutoff in Teff (`teff_zams`), imposed as a logistic taper.
- an age-varying and Teff-varying uniform distribution, multiplied by the inverse of the gaussian's taper (but with independent scale length), and then truncated to ensure that stars rotate faster than zero days, and to ensure that we model only the fast sequence. This uniform distribution is also tapered by a logistic function at the "slow" end to yield a smoother transition to the gaussian.

Parameters

- **teff_grid** (*np.ndarray*) – As in `models.slow_sequence`.
- **y_grid** (*np.ndarray*) – A grid over the residual of (rotation period - mean gyrochronal model).
- **poly_order** (*int*) – As in `models.slow_sequence`.
- **reference_model_ids** (*list*) – As in `models.slow_sequence`.
- **reference_ages** (*list*) – As in `models.slow_sequence`.
- **interp_method** (*str*) – As in `models.slow_sequence`.
- **bounds_error** (*str*) – As in `models.slow_sequence`.
- **popn_parameters** (*str or dict*) – "default", or a dict containing the population-level free parameters. Keys of "a0", "a1", "k0", "k1", "y_g", "l_hidden", and "k_hidden" must all be specified.

Returns

resid_y_Teff

2d array with dimension (N_y_grid x N_teff_grid), containing the probability distribution of the residual over the dimensions of *y* and *Teff*.

Return type

np.ndarray

gyrointerp.models.**teff_zams**(*age*, *bounds_error='limit'*)

Physics-informed MIST effective temperature for the effective temperature a star has when it arrives on the ZAMS. Changes over 80 to 1000 Myr (the floor beyond 1000 Myr is well below the 3800 K cool limit of BPH23). For a test, see `/tests/plot_teff_cuts.py`.

2.5.3 gyrointerp.getters module

These functions return the rotation periods and effective temperatures for single members of benchmark open clusters. These are the functions used to acquire and clean the data behind Figure 1 of BPH23. Generally, most users will want to just download the data from the paper, rather than using these functions. An additional dependency if you do want to use them is the `cdips` module, best installed via a setup.py install from https://github.com/lgbouma/cdips.

Get all available cluster data:

_get_cluster_Prot_Teff_data

Get and clean individual cluster data:

get_alphaPer
get_Pleiades
get_Blanco1
get_PscEri
get_NGC3532
get_GroupX
get_Praesepe_Rampalli_2021
get_NGC6811
get_NGC6819
get_Ruprecht147

gyrointerp.getters.**get_Blanco1**(*overwrite=0*)

Returns the Gillen+2020 (Table 1) rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_GroupX**(*overwrite=0*)

Returns the Messina+2022 rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_NGC3532**(*overwrite=0*)

Returns the Fritzewski+2021 rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_NGC6811**(*overwrite=0*)

Returns the Curtis+2019 (Table 1) rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_NGC6819**(*overwrite=0*)

Return NGC-6819 (Meibom+2015), as processed by Curtis+2020 (Table 5).

gyrointerp.getters.**get_Pleiades**(*overwrite=0*)

Returns the Rebull+2016 (Table 2) rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_Praesepe_Rampalli_2021**(*overwrite=0*)

Returns the Rampalli+2021 Praesepe rotation period dataframe with keys: "Prot", "Teff_Curtis20",

"flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_PscEri**(*overwrite=0*)

Returns the Curtis+2019 (Table 2) rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_Ruprecht147**(*overwrite=0*)

Return Curtis+2020 (Table 1) rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ the usual Gaia DR3 columns.

Note: The "flag_possible_binary" flag in this case is weaker than in the other comparison clusters, because there aren't very many good stars.

---

gyrointerp.getters.**get_alphaPer**(*overwrite=0*)

Returns the Boyle & Bouma 2023 rotation period dataframe with keys: "Prot", "Teff_Curtis20", "flag_benchmark_period" (for gyro calibration), _plus_ "flag_possible_binary", "flag_ruwe_outlier", "flag_camd_outlier", "flag_rverror_outlier", _plus_ the usual Gaia DR3 columns.

gyrointerp.getters.**get_alphaPer_construct**(*overwrite=0*)

Constructor for `getters.get_alphaPer`

2.5.4 gyrointerp.age_scale module

This module contains a dictionary defining the default cluster age scale, as well as the assumed +1sigma and -1sigma uncertainties on those ages. These data are the same as Table 1 in BPH23.

2.5.5 gyrointerp.extinctionpriors module

This module defines a dictionary of mean extinction A_V values adopted for the reference open clusters when calibrating the gyrochronal model.

2.5.6 gyrointerp.helpers module

This module contains reusable helper functions. The most generally useful one will be `get_summary_statistics`.

gyrointerp.helpers.**get_population_hyperparameter_posterior_samples**()

Access the posterior samples described in section 3.5 of BPH23. (These are generated by `drivers.run_emcee_fit_gyro_model`).

The returned numpy array is samples in the following parameters: a1/a0, y_g, logk0, logk1, log_f.

The notation follows Sections 3.3-3.5 of BPH23.

gyrointerp.helpers.**get_summary_statistics**(*age_grid*, *age_post*, *N=int(100000.0)*)

Given an age posterior probability density, `age_post`, over a grid over ages, `age_grid`, determine summary statistics for the posterior (its median, mean, +/-1 and 2-sigma intervals, etc). Do this by sampling *N* times from the posterior, with replacement, while weighting by the probability.

Parameters

- **age_grid** (*np.ndarray*) – Array-like of ages, in units of megayears. For instance, the default *age_grid* in `gyro_posterior.gyro_age_posterior` is `np.linspace(0, 3000, 500)`.
- **age_post** (*np.ndarray*) – Posterior probability distribution for ages; length should match *age_grid*. The posterior probabilities returned by``gyro_posterior.gyro_age_posterior`` and `gyro_posterior.gyro_age_posterior_list` are examples that would work. Generally, this helper function works for any grid and probability distribution.

Returns

summary_statistics

Dictionary containing keys and values for median, mean, peak (mode), +/-1sigma, +/-2sigma, +/-3sigma, and +/-1sigmapct. The units of all values are megayears, except for *+/-1sigmapct*, which is the relative +/-1-sigma uncertainty normalized by the median of the posterior, and is dimensionless.

Return type
dict

gyrointerp.helpers.**given_dr2_get_dr3_dataframes**(*dr2_source_ids*, *runid_dr2*, *runid_dr3*, *overwrite=False*)

gyrointerp.helpers.**left_merge**(*df0*, *df1*, *col0*, *col1*)

gyrointerp.helpers.**prepend_colstr**(*colstr*, *df*)

2.5.7 gyrointerp.plotting module

2.5.8 gyrointerp.teff module

This module contains functions for calculating photometric effective temperatures.

Contents:

given_dr2_BpmRp_AV_get_Teff_Curtis2020

gyrointerp.teff.**given_dr2_BpmRp_AV_get_Teff_Curtis2020**(*dr2_BpmRp*, *A_V*)

Empirical color-temperature relation from Appendix A of Curtis+2020. Visible in their Figure 11. Coefficients from their Table 4.

This relation was constructed using benchmark stars from Brewer+2016a, Boyajian+2012, and Mann+2015 (which should also be cited).

It is calibrated in the range 0.55 < (BP-RP)0 < 3.25, and has a scatter of about +/- 50 K.

It's important that the passed BP-RP colors are from Gaia DR2. For FGK stars in a few test clusters (eg. NGC-3532), the typical offset is +0.02 mag and color dependent. For late K dwarf and M-dwarfs, it flips, and gets down to -0.1 mag at BP-RP of >2 (SpType>M1V). This translates to a >100 K systematic error if you use the wrong Gaia data release.

See https://www.cosmos.esa.int/web/gaia/edr3-passbands for a description of why exactly the Gaia passbands changed between reductions.

Parameters

- **dr2_BpmRp** (*np.ndarray*) – *observed* Gaia DR2 BP-RP array.
- **A_V** (*float*) – mean reddening.

Returns
array of effective temperatures.

Return type
Teff (np.ndarray)

CHAPTER

THREE

CHANGELOG:

0.3 (2023-03-03)

- Bugfix a ModuleNotFoundError for calls to `gyrointerp.plotting`
- Add "posterior stacker" and cross-validation drivers.

0.2 (2023-02-21)

- Initial software release to PyPI and github.

0.1 (2023-02-21)

- Initial software release to github.

PYTHON MODULE INDEX

g

INDEX