vespa Documentation

Release 0.5.1

Timothy D. Morton

Jul 17, 2018

Contents

vespa is a Python package built to enable automated false positive probability (FPP) analysis of transiting planet signals (and so much more!). It implements the latest version of the general procedure described in detail in Morton (2012), and is being actively developed on GitHub. It also makes use of the isochrones package. Please raise an issue with any questions or comments you may have about this code.

CHAPTER 1

Overview

A false positive probability calculation in `vespa` is built of two basic components: a *`TransitSignal`* and a *`PopulationSet`*, joined together in a *`FPPCalculation`* object. The *`TransitSignal`* holds the data about the transit signal photometry, and the *`PopulationSet`* contains a set of simulated populations, one `EclipsePopulation` for each astrophysical model that is considered as a possible origin for the observed transit-like signal. By default, the populations included will be `PlanetPopulation` and three astrophysical false positive scenarios: an `EBPopulation`, an `HEBPopulation`, and a `BEBPopulation`.

The `EclipsePopulation` object derives from the more general *`vespa.stars.StarPopulation`*, which is useful beyond false positive calculations, such as for generating a hypothetical population of binary companions for a given star in order to help quantify completeness to stellar companions of an imaging survey.

1.1 Installation

To install, you can get the most recently released version from PyPI:

```
pip install vespa [--user]
```

Or you can clone the repository:

```
git clone https://github.com/timothydmorton/vespa.git
cd vespa
python setup.py install [--user]
```

The `--user` argument may be necessary if you don't have root privileges.

1.2 Basic Usage

The simplest way to run an FPP calculation straight out of the box is as follows.

1. Make a text file containing the transit photometry in three columns: `t_from_midtransit` [days], `flux` [relative, where out-of-transit is normalized to unity], and `flux_err`. The file should not have

a header row (no titles); and can be either whitespace or comma-delimited (will be ingested by `np.loadtxt()`).

2. Make a `star.ini` file that contains the observed properties of the target star (photometric and/or spectroscopic, whatever is available):

```
#provide spectroscopic properties if available
#Teff = 3503, 80  #value, uncertainty
#feh = 0.09, 0.09
#logg = 4.89, 0.1

#observed magnitudes of target star
# If uncertainty provided, will be used to fit StarModel
J = 9.763, 0.03
H = 9.135, 0.03
K = 8.899, 0.02
Kepler = 12.473
```

3. Make a `fpp.ini` file containing the following information:

```
name = k2oi #anything
ra = 11:30:14.510 #can be decimal form too
dec = +07:35:18.21

period = 32.988 #days
rprs = 0.0534   #Rp/Rstar
photfile = lc_k2oi.csv #contains transit photometry

[constraints]
maxrad = 12  # aperture radius [arcsec]
secthresh = 1e-4 # Maximum allowed depth of potential secondary eclipse
```

4. Run the following from the command line (from within the same folder that has `star.ini` and `fpp.ini`):

```
$  calcfpp -n 1000
```

This will take a few minutes the first time you run it (note the default simulation size is `n=20000`, which would take longer but be more reliable), and will output the FPP to the command line, as well as producing diagnostic plots and a `results.txt` file with the quantitative summary of the calculation. In addition, this will produce a number of data files in the same directory as your `fpp.ini` file:

- `trsig.pkl`: the pickled *vespa.TransitSignal* object.
- `starfield.h5`: the TRILEGAL field star simulation
- `starmodel.h5`: the `isochrones.StarModel` fit
- `popset.h5`: the *vespa.PopulationSet* object representing the model population simulations.

It will also generate the following diagnostic plots:

- `trsig.png`: A plot of the transit signal
- `eb.png`, `heb.png`, `beb.png`, `pl.png`: plots illustrating the likelihood of each model.
- `FPPsummary.png`: A summary figure of the FPP results.
- Summary plots of the `isochrones.StarModel` fits.

Once these files have been created, it is faster to re-run the calculation again, even if you change the constraints.

CHAPTER 2

False Positive Probability Calculation

`vespa` calculates the false positive probability for a transit signal as follows:

$$\mathrm{FPP} = 1 - P_{\mathrm{pl}},$$

where

$$P_{\mathrm{pl}} = \frac{\mathcal{L}_{\mathrm{pl}}\pi_{\mathrm{pl}}}{\mathcal{L}_{\mathrm{pl}}\pi_{\mathrm{pl}} + \mathcal{L}_{\mathrm{FP}}\pi_{\mathrm{FP}}}.$$

The $\mathcal{L}_i$ here represent the "model likelihood" factors and the π_i represent the "model priors," with the FP subscript representing the sum of $\mathcal{L}_i\pi_i$ for each of the false positive scenarios.

2.1 Likelihoods

Each `EclipsePopulation` contains a large number of simulated instances of the particular physical scenario, each of which has a simulated eclipse shape and a corresponding trapezoidal fit. This enables each population to define a 3-dimensional probability distribution function (PDF) for these trapezoid parameters, $p_{\mathrm{mod}}(\log_{10}(\delta), T, T/\tau)$. As the *`TransitSignal`* object provides an MCMC sampling of the trapezoid parameters for the observed transit signal, the likelihood of the transit signal under a given model can thus be approximated as a sum over the model PDF evaluated at the K samples:

$$\mathcal{L} = \sum_{k=1}^{K} p_{\mathrm{mod}}\left(\log_{10}(\delta_k), T_k, (T/\tau)_k\right)$$

This is implemented in `EclipsePopulation.lhood()`.

2.2 Priors

Each `EclipsePopulation` also has a `EclipsePopulation.prior` attribute, the value of which represents the probability of that particular astrophysical scenario existing. For a `BEBPopulation`, for example, the prior is `(star density) * (sky area) * (binary fraction) * (eclipse probability)`. If observational constraints are applied to a population, then an additional `selectfrac` factor will be multiplied into the prior, representing the fraction of scenarios that are still allowed to exist, given the constraints.

CHAPTER 3

High-level API

This page details the top-level classes that provide access to the `vespa` module. The simplest entry point into these calculations is the `calcfpp` command-line script, which creates a *FPPCalculation* object using *FPPCalculation.from_ini()*, and creates a bunch of data files/diagnostic plots. A *FPPCalculation* is made up of a *PopulationSet* and a *TransitSignal*.

For more details on the guts of the objects that make up a *PopulationSet*, please see the documentation on *Eclipse Populations*, *Star Populations*, and *Orbits*.

3.1 FPPCalculation

class vespa.**FPPCalculation**(*trsig*, *popset*, *folder='.'*)

An object to organize an FPP calculation.

May be created in one of three ways:

- Manually building a *TransitSignal* and a *PopulationSet* and then calling the constructor,
- Loading from a folder in which the correct data files have been saved, using *FPPCalculation.load()*, or
- Reading from a config file, using *FPPCalculation.from_ini()*

Parameters

- **trsig** – *TransitSignal* object representing the signal being modeled.
- **popset** – *PopulationSet* object representing the set of models being considered as an explanation for the signal.
- **folder** – (optional) Folder where likelihood cache, results file, plots, etc. are written by default.

FPP(*skipmodels=None*)

Return the false positive probability (FPP)

FPPplots (*folder=None*, *format='png'*, *tag=None*, ***kwargs*)
Make FPP diagnostic plots

Makes likelihood "fuzz plot" for each model, a FPP summary figure, a plot of the *TransitSignal*, and writes a `results.txt` file.

Parameters

- **folder** – (optional) Destination folder for plots/`results.txt`. Default is `self.folder`.
- **format** – (optional) Desired format of figures. e.g. `png`, `pdf`...
- **tag** – (optional) If this is provided (string), then filenames will have `_[tag]` appended to the filename, before the extension.
- ****kwargs** – Additional keyword arguments passed to `PopulationSet.lhoodplots()`.

FPPsummary (*fig=None*, *figsize=(10, 8)*, *saveplot=False*, *folder='.'*, *starinfo=True*, *siginfo=True*, *priorinfo=True*, *constraintinfo=True*, *tag=None*, *simple=False*, *figformat='png'*)
Makes FPP summary plot

Note: This is due for updates/improvements.

Parameters

- **figsize** (*fig,*) – (optional) Arguments for *plotutils.setfig()*.
- **saveplot** – (optional) Whether to save figure. Default is `False`.
- **folder** – (optional) Folder to which to save plot; default is current working dir.
- **figformat** – (optional) Desired format of saved figure.

classmethod from_ini (*folder*, *ini_file='fpp.ini'*, *ichrone='mist'*, *recalc=False*, *refit_trap=False*, ***kwargs*)
To enable simple usage, initializes a FPPCalculation from a .ini file

By default, a file called `fpp.ini` will be looked for in the current folder. Also present must be a `star.ini` file that contains the observed properties of the target star.

`fpp.ini` must be of the following form:

```
name = k2oi
ra = 11:30:14.510
dec = +07:35:18.21

period = 32.988 #days
rprs = 0.0534   #Rp/Rstar
photfile = lc_k2oi.csv

[constraints]
maxrad = 10 #exclusion radius [arcsec]
secthresh = 0.001 #maximum allowed secondary signal depth

#This variable defines contrast curves
#ccfiles = Keck_J.cc, Lick_J.cc
```

Photfile must be a text file with columns `(days_from_midtransit, flux, flux_err)`. Both whitespace- and comma-delimited will be tried, using `np.loadtxt`. Photfile need not be there if there is

a pickled *TransitSignal* saved in the same directory as `ini_file`, named `trsig.pkl` (or another name as defined by `trsig` keyword in `.ini` file).

`star.ini` should look something like the following:

```
B = 15.005, 0.06
V = 13.496, 0.05
g = 14.223, 0.05
r = 12.858, 0.04
i = 11.661, 0.08
J = 9.763, 0.03
H = 9.135, 0.03
K = 8.899, 0.02
W1 = 8.769, 0.023
W2 = 8.668, 0.02
W3 = 8.552, 0.025
Kepler = 12.473

#Teff = 3503, 80
#feh = 0.09, 0.09
#logg = 4.89, 0.1
```

Any star properties can be defined; if errors are included then they will be used in the `isochrones.StarModel` MCMC fit. Spectroscopic parameters (`Teff, feh, logg`) are optional. If included, then they will also be included in `isochrones.StarModel` fit. A magnitude for the band in which the transit signal is observed (e.g., `Kepler`) is required, though need not have associated uncertainty.

Parameters

- **`folder`** – Folder to find configuration files.
- **`ini_file`** – Input configuration file.
- **`star_ini_file`** – Input config file for `isochrones.StarModel` fits.
- **`recalc`** – Whether to re-calculate *PopulationSet*, if a `popset.h5` file is already present
- **`**kwargs`** – Keyword arguments passed to *PopulationSet*.

Creates:

- `trsig.pkl`: the pickled *vespa.TransitSignal* object.
- `starfield.h5`: the TRILEGAL field star simulation
- `starmodel.h5`: the `isochrones.StarModel` fit
- `popset.h5`: the *vespa.PopulationSet* object representing the model population simulations.

RuntimeError : If single, double, and triple starmodels are not computed, then raises with admonition to run *starfit –all*.

AttributeError : If *trsig.pkl* not present in folder, and *photfile* is not defined in config file.

`lhood`(*model*, ***kwargs*)
Return the likelihood for a given model.

`lhoodplot`(*model*, *suptitle=''*, ***kwargs*)
Make a plot of the likelihood for a given model.

`lhoodplots`(*folder='.'*, *tag=None*, *figformat='png'*, *recalc_lhood=False*, ***kwargs*)
Make a plot of the likelihood for each model in PopulationSet

classmethod load(*folder*)
Loads PopulationSet from folder

`popset.h5` and `trsig.pkl` must exist in folder.

Parameters **folder** – Folder from which to load.

plotsignal(*fig=None*, *saveplot=True*, *folder=None*, *figformat='png'*, ***kwargs*)
Plots TransitSignal

Calls *TransitSignal.plot()*, saves to provided folder.

Parameters

- **fig** – (optional) Argument for *plotutils.setfig()*.
- **saveplot** – (optional) Whether to save figure.
- **folder** – (optional) Folder to which to save plot
- **figformat** – (optional) Desired format for figure.
- ****kwargs** – Additional keyword arguments passed to *TransitSignal.plot()*.

prior(*model*)
Return the prior for a given model.

save(*overwrite=True*)
Saves PopulationSet and TransitSignal.

Shouldn't need to use this if you're using *FPPCalculation.from_ini()*.

Saves :class'PopulationSet' to `[folder]/popset.h5` and *TransitSignal* to `[folder]/trsig.pkl`.

Parameters **overwrite** – (optional) Whether to overwrite existing files.

save_popset(*filename='popset.h5'*, ***kwargs*)
Saves the PopulationSet

Calls *PopulationSet.save_hdf()*.

save_signal(*filename=None*)
Saves TransitSignal.

Calls *TransitSignal.save()*; default filename is `trsig.pkl` in `self.folder`.

write_results(*folder=None*, *filename='results.txt'*, *to_file=True*)
Writes text file of calculation summary.

Parameters

- **folder** – (optional) Folder to which to write `results.txt`.
- **filename** – Filename to write. Default=''results.txt''.
- **to_file** – If True, then writes file. Otherwise just return header, line.

Returns Header string, line

3.2 PopulationSet

This object is essentially an organized list of `EclipsePopulation` objects.

class vespa.**PopulationSet**(*poplist=None*, *period=None*, *mags=None*, *n=20000.0*, *ra=None*, *dec=None*, *trilegal_filename=None*, *Teff=None*, *logg=None*, *feh=None*, *starmodel=None*, *binary_starmodel=None*, *triple_starmodel=None*, *rprs=None*, *MAfn=None*, *savefile=None*, *heb_kws=None*, *eb_kws=None*, *beb_kws=None*, *pl_kws=None*, *hide_exceptions=False*, *fit_trap=True*, *do_only=None*)

A set of EclipsePopulations used to calculate a transit signal FPP

This can be initialized with a list of `EclipsePopulation` objects that have been pre-generated, or it can be passed the arguments required to generate the default list of :class:'EclipsePopulation's.

Parameters

- **poplist** – Can be either a list of `EclipsePopulation` objects, a filename (in which case a saved *PopulationSet* will be loaded), or `None`, in which case the populations will be generated.
- **period** – Orbital period of signal.
- **mags** (`dict`) – Observed magnitudes of target star.
- **n** – Size of simulations. Default is 2e4.
- **dec** (*ra,*) – (optional) Target star position; passed to `BEBPopulation`.
- **trilegal_filename** – Passed to `BEBPopulation`.
- **age, feh, radius** (*mass,*) – (optional) Properties of target star. Either in `(value, error)` form or as `simpledist.Distribution` objects. Not necessary if `starmodel` is passed.
- **starmodel** (`isochrones.StarModel`) – (optional) The preferred way to define the properties of the host star. If MCMC has been run on this model, then samples are just read off; if it hasn't, then it will run it.
- **rprs** – R_planet/R_star. Single-value estimate.
- **MAfn** – (optional) `transit_basic.MAInterpolationFunction` object. If not passed, then one with default parameters will be created.
- **colors** – (optional) Colors to use to constrain multiple star populations; passed to `EBPopulation` and `HEBPopulation`. Default will be ['JK', 'HK']
- **logg** (*Teff,*) – (optional) If `starmodel` not provided, then these can be used (single values only) in order for `PlanetPopulation` to use the right limb darkening parameters.
- **savefile** – (optional) HDF file in which to save *PopulationSet*.
- **eb_kws, beb_kws, pl_kws** (*heb_kws,*) – (optional) Keyword arguments to pass on to respective `EclipsePopulation` constructors.
- **hide_exceptions** – (optional) If `True`, then exceptions generated during population simulations will be passed, not raised.
- **fit_trap** – (optional) If `True`, then population generation will also call `EclipsePopulation.fit_trapezoids()` for each model population.
- **do_only** – (optional) Can be defined in order to make only a subset of populations. List or tuple should contain modelname shortcuts (e.g., 'beb', 'heb', 'eb', or 'pl').

add_population(*pop*)

Adds population to PopulationSet

apply_cc(*cc*, ***kwargs*)
Applies contrast curve constraint to each population

See *vespa.stars.StarPopulation.apply_cc()*; all arguments passed to that function for each population.

apply_dmaglim(*dmaglim=None*)
Applies a constraint that sets the maximum brightness for non-target star

`stars.StarPopulation.set_dmaglim()` not yet implemented.

apply_multicolor_transit(*band*, *depth*)
Applies constraint corresponding to measuring transit in different band

This is not implemented yet.

apply_secthresh(*secthresh*, ***kwargs*)
Applies secondary depth constraint to each population

See `EclipsePopulation.apply_secthresh()`; all arguments passed to that function for each population.

apply_trend_constraint(*limit*, *dt*, ***kwargs*)
Applies constraint corresponding to RV trend non-detection to each population

See *stars.StarPopulation.apply_trend_constraint()*; all arguments passed to that function for each population.

apply_vcc(*vcc*)
Applies velocity contrast curve constraint to each population

See *vespa.stars.StarPopulation.apply_vcc()*; all arguments passed to that function for each population.

change_prior(***kwargs*)
Changes prior factor(s) in all populations

colordict
Dictionary holding colors that correspond to constraints.

constrain_oddeven(*diff*, ***kwargs*)
Constrains the difference b/w primary and secondary to be < diff

constrain_property(*prop*, ***kwargs*)
Constrains property for each population

See *vespa.stars.StarPopulation.constrain_property()*; all arguments passed to that function for each population.

constraints
Unique list of constraints among all populations in set.

generate(*ra*, *dec*, *period*, *mags*, *n=20000.0*, *Teff=None*, *logg=None*, *feh=None*, *MAfn=None*, *rprs=None*, *trilegal_filename=None*, *starmodel=None*, *binary_starmodel=None*, *triple_starmodel=None*, *heb_kws=None*, *eb_kws=None*, *beb_kws=None*, *pl_kws=None*, *savefile=None*, *hide_exceptions=False*, *fit_trap=True*, *do_only=None*)
Generates PopulationSet.

classmethod load_hdf(*filename*, *path=''*)
Loads PopulationSet from file

modelnames
List of model names

priorfactors
Combinartion of priorfactors from all populations

remove_constraint (**names*)
Removes constraint from each population

See :func:‘vespa.stars.StarPopulation.remove_constraint

remove_population (*pop*)
Removes population from PopulationSet

replace_constraint (*name*, ***kwargs*)
Replaces removed constraint in each population.

See *vespa.stars.StarPopulation.replace_constraint()*

save_hdf (*filename*, *path=''*, *overwrite=False*)
Saves PopulationSet to HDF file.

set_maxrad (*newrad*)
Sets max allowed radius in populations.

Doesn't operate via the `stars.Constraint` protocol; rather just rescales the sky positions for the background objects and recalculates sky area, etc.

shortmodelnames
List of short modelnames.

3.3 TransitSignal

class vespa.**TransitSignal** (*ts*, *fs*, *dfs=None*, *P=None*, *p0=None*, *name=''*, *maxslope=None*)
A phased-folded transit signal.

Epoch of the transit at 0, 'continuum' set at 1.

Parameters

- **fs, dfs** (*ts,*) – Times (days from mid-transit), fluxes (relative to 1), flux uncertainties. dfs optional
- **P** – Orbital period.
- **p0** – (optional) Initial guess for least-squares trapezoid fit. If not provided, then some decent guess will be made (which is better on made-up data than real...)
- **name** – (optional) Name of the signal.
- **maxslope** – (optional) Upper limit to use for "slope" parameter (T/tau) in the MCMC fitting of signal. Default is 15.

Note: The implementation of this object can use some refactoring; as it is directly translated from some older code. As such, not all methods/attributes are well documented.

MCMC (*niter=500*, *nburn=200*, *nwalkers=200*, *threads=1*, *fit_partial=False*, *width=3*, *savedir=None*, *refit=False*, *thin=10*, *conf=0.95*, *maxslope=None*, *debug=False*, *p0=None*)
Fit transit signal to trapezoid model using MCMC

Note: As currently implemented, this method creates a bunch of attributes relevant to the MCMC fit; I plan to refactor this to define those attributes as properties so as not to have their creation hidden away here. I plan to refactor how this works.

plot (*fig=None*, *plot_trap=False*, *name=False*, *trap_color='g'*, *trap_kwargs=None*, ***kwargs*)
Makes a simple plot of signal

Parameters

- **fig** – (optional) Argument for *plotutils.setfig()*.
- **plot_trap** – (optional) Whether to plot the (best-fit least-sq) trapezoid fit.
- **name** – (optional) Whether to annotate plot with the name of the signal; can be `True` (in which case `self.name` will be used), or any arbitrary string.
- **trap_color** – (optional) Color of trapezoid fit line.
- **trap_kwargs** – (optional) Keyword arguments to pass to trapezoid fit line.
- ****kwargs** – (optional) Additional keyword arguments passed to `plt.plot`.

save (*filename*)
Calls save_pkl function.

save_hdf (*filename*, *path=''*)
Save transitsignal info using HDF. . . not yet implemented.

Note: Refactoring plan is to re-write saving to use HDF instead of pickle.

save_pkl (*filename*)
Pickles TransitSignal.

CHAPTER 4

Eclipse Populations

All physical eclipse models proposed as potential explanations for an obseved transit signal are defined as *EclipsePopulation* objects. Currently implemented within vespa are *EBPopulation*, *HEBPopulation*, *BEBPopulation*, and *PlanetPopulation*.

Note: More subclasses are under development for other scenarios, in particular eclipses around *specific* observed stars.

Also see the documentation for *vespa.stars.StarPopulation*, from which *EclipsePopulation* derives.

class vespa.populations.**EclipsePopulation**(*stars=None*, *period=None*, *model=''*, *priorfactors=None*, *lhoodcachefile=None*, *orbpop=None*, *prob=None*, *cadence=0.018819444444444444*, ***kwargs*)

Base class for populations of eclipsing things.

This is the base class for populations of various scenarios that could explain a tranist signal; that is, astrophysical false positives or transiting planets.

Once set up properly, *EclipsePopulation.fit_trapezoids()* can be used to fit the trapezoidal shape parameters, after which the likelihood of a transit signal under the model may be calculated.

Subclasses *vespa.stars.StarPopulation*, which enables all the functionality of observational constraints.

if prob is not passed; should be able to calculated from given star/orbit properties.

As with *vespa.stars.StarPopulation*, any subclass must be able to be initialized with no arguments passed, in order for *vespa.stars.StarPopulation.load_hdf()* to work properly.

Parameters

- **stars** – DataFrame with star properties. Must contain M_1, M_2, R_1, R_2, u1_1, u1_2, u2_1, u2_2. Also, either the period keyword argument must be

provided or a `period` column should be in `stars`. `stars` must also have the eclipse parameters: *'inc, ecc, w, dpri, dsec, b_sec, b_pri, fluxfrac_1, fluxfrac_2'*.

- **`period`** – (optional) Orbital period. If not provided, then `stars` must have period column.
- **`model`** – (optional) Name of the model.
- **`priorfactors`** – (optional) Multiplicative factors that quantify the model prior for this particular model; e.g. `f_binary`, etc.
- **`lhoodcachefile`** – (optional) File where likelihood calculation cache is written.
- **`orbpop`** (`orbits.OrbitPopulation` or `orbits.TripleOrbitPopulation`) – (optional) Orbit population.
- **`prob`** – (optional) Averaged eclipse probability of scenario instances. If not provided, this should be calculated, though this is not implemented yet.
- **`cadence`** – (optional) Observing cadence, in days. Defaults to *Kepler* value.
- **`**kwargs`** – Additional keyword arguments passed to *vespa.stars.StarPopulation*.

`add_priorfactor`(***kwargs*)
Adds given values to priorfactors

If given keyword exists already, error will be raised to use *EclipsePopulation.change_prior()* instead.

`apply_secthresh`(**args*, ***kwargs*)
Another name for constrain_secdepth

`change_prior`(***kwargs*)
Changes existing priorfactors.

If given keyword isn't already in priorfactors, then will be ignored.

`constrain_secdepth`(*thresh*)
Constrain the observed secondary depth to be less than a given value

Parameters **`thresh`** – Maximum allowed fractional depth for diluted secondary eclipse depth

`depth`
Observed primary depth (fitted undiluted depth * dilution factor)

`depth_in_band`(*band*)
Stub for future multicolor transit implementation

`dilution_factor`
Multiplicative factor (<1) that converts true depth to diluted depth.

`eclipse_new`(*i*, *secondary=False*, *npoints=200*, *width=3*, *texp=None*)
Returns times and fluxes of eclipse i (centered at t=0)

`eclipseprob`
Array of eclipse probabilities.

`fit_trapezoids`(*MAfn=None*, *msg=None*, *use_pbar=True*, ***kwargs*)
Fit trapezoid shape to each eclipse in population

For each instance in the population, first the correct, physical Mandel-Agol transit shape is simulated, and then this curve is fit with a trapezoid model

Parameters

- **MAfn** – `transit_basic.MAInterpolationFunction` object. If not passed, then one with default parameters will be created.
- **msg** – Message to be displayed for progressbar output.
- ****kwargs** – Additional keyword arguments passed to `fitebs.fitebs()`.

fluxfrac_eclipsing(*band=None*)
Stub for future multicolor transit implementation

lhood(*trsig*, *recalc=False*, *cachefile=None*)
Returns likelihood of transit signal

Returns sum of `trsig` MCMC samples evaluated at `self.kde`.

Parameters

- **trsig** – *vespa.TransitSignal* object.
- **recalc** – (optional) Whether to recalculate likelihood (if calculation is cached).
- **cachefile** – (optional) File that holds likelihood calculation cache.

lhoodplot(*trsig=None*, *fig=None*, *piechart=True*, *figsize=None*, *logscale=True*, *constraints='all'*, *suptitle=None*, *Ltot=None*, *maxdur=None*, *maxslope=None*, *inverse=False*, *colordict=None*, *cachefile=None*, *nbins=20*, *dur_range=None*, *slope_range=None*, *depth_range=None*, *recalc=False*, ***kwargs*)
Makes plot of likelihood density function, optionally with transit signal

If `trsig` not passed, then just density plot of the likelidhoo will be made; if it is passed, then it will be plotted over the density plot.

Parameters

- **trsig** – (optional) *vespa.TransitSignal* object.
- **fig** – (optional) Argument for `plotutils.setfig()`.
- **piechart** – (optional) Whether to include a plot of the piechart that describes the effect of the constraints on the population.
- **figsize** – (optional) Passed to `plotutils.setfig()`.
- **logscale** – (optional) If `True`, then shading will be based on the log-histogram (thus showing more detail at low density). Passed to *vespa.stars.StarPopulation.prophist2d()*.
- **constraints** – (`'all'`, `'none'` or `list`; optional) Which constraints to apply in making plot. Picking specific constraints allows you to visualize in more detail what the effect of a constraint is.
- **suptitle** – (optional) Title for the figure.
- **Ltot** – (optional) Total of `prior * likelihood` for all models. If this is passed, then "Probability of scenario" gets a text box in the middle.
- **inverse** – (optional) Intended to allow showing only the instances that are ruled out, rather than those that remain. Not sure if this works anymore.
- **colordict** – (optional) Dictionary to define colors of constraints to be used in pie chart. Intended to unify constraint colors among different models.
- **cachefile** – (optional) Likelihood calculation cache file.
- **nbins** – (optional) Number of bins with which to make the 2D histogram plot; passed to *vespa.stars.StarPopulation.prophist2d()*.

- **slope_range, depth_range** (*dur_range,*) – (optional) Define ranges of plots.
- ****kwargs** – Additional keyword arguments passed to *vespa.stars.StarPopulation.prophist2d()*.

classmethod load_hdf(*filename*, *path=''*)
Loads EclipsePopulation from HDF file

Also runs `EclipsePopulation._make_kde()` if it can.

Parameters

- **filename** – HDF file
- **path** – (optional) Path within HDF file

mean_eclipseprob
Mean eclipse probability for population

modelshort
Short version of model name

Dictionary defined in `populations.py`:

```
SHORT_MODELNAMES = {'Planets':'pl',
        'EBs':'eb',
        'HEBs':'heb',
        'BEBs':'beb',
        'Blended Planets':'bpl',
        'Specific BEB':'sbeb',
        'Specific HEB':'sheb'}
```

prior
Model prior for particular model.

Product of eclipse probability (`self.prob`), the fraction of scenario that is allowed by the various constraints (`self.selectfrac`), and all additional factors in `self.priorfactors`.

resample()
Returns a copy of population with stars resampled (with replacement).

Used in bootstrap estimate of FPP uncertainty.

TODO: check to make sure constraints properly copied!

secondary_depth
Observed secondary depth (fitted undiluted sec. depth * dilution factor)

4.1 Undiluted Eclipsing Binary

class vespa.populations.**EBPopulation**(*period=None*, *mags=None*, *mag_errs=None*, *Teff=None*, *logg=None*, *feh=None*, *starmodel=None*, *band='Kepler'*, *model='EBs'*, *f_binary=0.4*, *n=20000.0*, *MAfn=None*, *lhoodcachefile=None*, ***kwargs*)
Population of Eclipsing Binaries (undiluted)

Eclipsing Binary (EB) population is generated by fitting a two-star model to the observed properties of the system (photometric and/or spectroscopic), using `isochrones.starmodel.BinaryStarModel`.

Inherits from *EclipsePopulation* and `stars.Observed_BinaryPopulation`.

Parameters

- **period** – Orbital period
- **mags** (dict) – Observed apparent magnitudes. Won't work if this is None, which is the default.
- **Teff,logg,feh** – Spectroscopic properties of primary, if measured, in (value, err) format.
- **starmodel** (isochrones.BinaryStarModel) – (optional) Must be a BinaryStarModel. If MCMC has been run on this model, then samples are just read off; if it hasn't, then it will run it.
- **band** – (optional) Photometric bandpass in which transit signal is observed.
- **model** – (optional) Name of model.
- **f_binary** – (optional) Binary fraction to be assumed. Will be one of the priorfactors.
- **n** – (optional) Number of instances to simulate. Default = 2e4.
- **MAfn** – (optional) transit_basic.MAInterpolationFunction object. If not passed, then one with default parameters will be created.
- **lhoodcachefile** – (optional) Likelihood calculation cache file.

generate(*mags*, *n=20000.0*, *mag_errs=None*, *Teff=None*, *logg=None*, *feh=None*, *MAfn=None*, *f_binary=0.4*, *starmodel=None*, ***kwargs*)

Generates stars and eclipses

All arguments previously defined.

4.2 Hierarchical Elipsing Binary

class vespa.populations.**HEBPopulation**(*period=None*, *mags=None*, *mag_errs=None*, *Teff=None*, *logg=None*, *feh=None*, *starmodel=None*, *band='Kepler'*, *model='HEBs'*, *f_triple=0.12*, *n=20000.0*, *MAfn=None*, *lhoodcachefile=None*, ***kwargs*)

Population of Hierarchical Eclipsing Binaries

Hierarchical Eclipsing Binary (HEB) population is generated by fitting a two-star model to the observed properties of the system (photometric and/or spectroscopic), using isochrones.starmodel.BinaryStarModel.

by

Inherits from *EclipsePopulation* and stars.Observed_TriplePopulation.

Parameters

- **period** – Orbital period
- **mags,mag_errs** – Observed apparent magnitudes; uncertainties optional. If uncertainties not provided, Observed_TriplePopulation will default to uncertainties in all bands of 0.05 mag.
- **Teff,logg,feh** – Spectroscopic properties of primary, if measured, in (value, err) format.

- **starmodel** (isochrones.BinaryStarModel) – (optional) Must be a BinaryStarModel. If MCMC has been run on this model, then samples are just read off; if it hasn't, then it will run it.
- **band** – (optional) Photometric bandpass in which transit signal is observed.
- **model** – (optional) Name of model.
- **f_binary** – (optional) Binary fraction to be assumed. Will be one of the priorfactors.
- **n** – (optional) Number of instances to simulate. Default = 2e4.
- **MAfn** – (optional) transit_basic.MAInterpolationFunction object. If not passed, then one with default parameters will be created.
- **lhoodcachefile** – (optional) Likelihood calculation cache file.

generate(*mags*, *n=20000.0*, *mag_errs=None*, *Teff=None*, *logg=None*, *feh=None*, *MAfn=None*, *f_triple=0.12*, *starmodel=None*, ***kwargs*)

Generates stars and eclipses

All arguments previously defined.

4.3 Background Eclipsing Binary

class vespa.populations.**BEBPopulation**(*period=None*, *mags=None*, *ra=None*, *dec=None*, *trilegal_filename=None*, *n=20000.0*, *ichrone='dartmouth'*, *band='Kepler'*, *maxrad=10*, *f_binary=0.4*, *model='BEBs'*, *MAfn=None*, *lhoodcachefile=None*, ***kwargs*)

Population of "Background" eclipsing binaries (BEBs)

Parameters

- **period** – Orbital period.
- **mags** (dict) – Observed apparent magnitudes of target (foreground) star. Must have at least magnitude in band that eclipse is measured in (band argument).
- **ra,dec** – (optional) Coordinates of star (to simulate field star population). If trilegal_filename not provided, then TRILEGAL simulation will be generated.
- **trilegal_filename** – Name of file that contains TRILEGAL field star simulation to use. Should always be provided if population is to be generated. If file does not exist, then TRILEGAL simulation will be saved as this filename (use .h5 extension).
- **n** – (optional) Size of simulation. Default is 2e4.
- **ichrone** – (optional) isochrones.Isochrone object to use to generate stellar models.
- **band** – (optional) Photometric bandpass in which eclipse signal is observed.
- **maxrad** – (optional) Maximum radius [arcsec] from target star to assign to BG stars.
- **f_binary** – (optional) Assumed binary fraction. Will be part of priorfactors.
- **model** – (optional) Model name.
- **MAfn** – (optional) transit_basic.MAInterpolationFunction object. If not passed, then one with default parameters will be created.

- **lhoodcachefile** – (optional) Likelihood calculation cache file.
- ****kwargs** – Additional keyword arguments passed to `stars.BGStarPopulation_TRILEGAL`.

generate(*trilegal_filename*, *ra=None*, *dec=None*, *n=20000.0*, *ichrone='dartmouth'*, *MAfn=None*, *mags=None*, *maxrad=None*, *f_binary=0.4*, ***kwargs*)
Generate population.

4.4 Transiting Planet

class vespa.populations.**PlanetPopulation**(*period=None*, *rprs=None*, *mass=None*, *radius=None*, *Teff=None*, *logg=None*, *starmodel=None*, *band='Kepler'*, *model='Planets'*, *n=20000.0*, *fp_specific=None*, *u1=None*, *u2=None*, *rbin_width=0.3*, *MAfn=None*, *lhoodcachefile=None*)
Population of Transiting Planets

Subclass of *EclipsePopulation*. This is mostly a copy of *EBPopulation*, with small modifications.

Star properties may be defined either with either a `isochrones.StarModel` or by defining just its `mass` and `radius` (and `Teff` and `logg` if desired to set limb darkening coefficients appropriately).

Parameters

- **period** – Period of signal.
- **rprs** – Point-estimate of Rp/Rs radius ratio.
- **radius** (*mass,*) – (optional) Mass and radius of host star. If defined, must be either tuples of form `(value, error)` or `simpledist.Distribution` objects.
- **logg** (*Teff,*) – (optional) Teff and logg point estimates for host star. These are used only for calculating limb darkening coefficients.
- **starmodel** (`isochrones.StarModel`) – (optional) The preferred way to define the properties of the host star. If MCMC has been run on this model, then samples are just read off; if it hasn't, then it will run it.
- **band** – (optional) Photometric band in which eclipse is detected.
- **model** – (optional) Name of the model.
- **n** – (optional) Number of instances to simulate. Default = `2e4`.
- **fp_specific** – (optional) "Specific occurrence rate" for this type of planets; that is, the planet occurrence rate integrated from `(1-rbin_width)x` to `(1+rbin_width)x` this planet radius. This goes into the `priorfactor` for this model.
- **u2** (*u1,*) – (optional) Limb darkening parameters. If not provided, then calculated based on `Teff, logg` or just defaulted to solar values.
- **rbin_width** – (optional) Fractional width of rbin for `fp_specific`.
- **MAfn** – (optional) `transit_basic.MAInterpolationFunction` object. If not passed, then one with default parameters will be created.
- **lhoodcachefile** – (optional) Likelihood calculation cache file.

generate(*rprs=None*, *mass=None*, *radius=None*, *n=20000.0*, *fp_specific=0.01*, *u1=None*, *u2=None*, *starmodel=None*, *Teff=None*, *logg=None*, *rbin_width=0.3*, *MAfn=None*, *lhoodcachefile=None*)

Generates Population

All arguments defined in `__init__`.

save_hdf(*filename*, *path=''*, ***kwargs*)

Saves to HDF5 file.

Subclasses should be sure to define `_properties` attribute to ensure that all correct attributes get saved. Load a saved population with `StarPopulation.load_hdf()`.

Example usage:

```
>>> from vespa.stars import Raghavan_BinaryPopulation, StarPopulation
>>> pop = Raghavan_BinaryPopulation(1., n=1000)
>>> pop.save_hdf('test.h5')
>>> pop2 = StarPopulation.load_hdf('test.h5')
>>> pop == pop2
    True
>>> pop3 = Ragahavan_BinaryPopulation.load_hdf('test.h5')
>>> pop3 == pop2
    True
```

Parameters

- **filename** – Name of HDF file.
- **path** – (optional) Path within HDF file to save object.
- **properties** – (optional) Names of any properties (in addition to those defined in `_properties` attribute) that you wish to save. (This is an old keyword, and should probably be removed. Feel free to ignore it.)
- **overwrite** – (optional) Whether to overwrite file if it already exists. If `True`, then any existing file will be deleted before object is saved. Use `append` if you don't wish this to happen.
- **append** – (optional) If `True`, then if the file exists, then only the particular path in the file will get written/overwritten. If `False` and both file and path exist, then an `IOError` will be raised. If `False` and file exists but not path, then no error will be raised.

CHAPTER 5

Transit Utilities

In order to enable fast simulation of large numbers of eclipses, `vespa` makes use of the Mandel-Agol (2002) transit model implemented by the batman module.

`vespa.transit_basic.`**`ldcoeffs`**(*teff*, *logg=4.5*, *feh=0*)
Returns limb-darkening coefficients in Kepler band.

`vespa.transit_basic.`**`impact_parameter`**(*a*, *R*, *inc*, *ecc=0*, *w=0*, *return_occ=False*)
a in AU, R in Rsun, inc & w in radians

`vespa.transit_basic.`**`transit_T14`**(*P*, *Rp*, *Rs=1*, *b=0*, *Ms=1*, *ecc=0*, *w=0*)
P in days, Rp in Earth radii, Rs in Solar radii, b=impact parameter, Ms Solar masses. Returns T14 in hours. w in deg.

`vespa.transit_basic.`**`minimum_inclination`**(*P*, *M1*, *M2*, *R1*, *R2*)
Returns the minimum inclination at which two bodies from two given sets eclipse

Only counts systems not within each other's Roche radius

Parameters

- **`P`** – Orbital periods.
- **`M1,M2,R1,R2`** – Masses and radii of primary and secondary stars.

`vespa.transit_basic.`**`a_over_Rs`**(*P*, *R2*, *M2*, *M1=1*, *R1=1*, *planet=True*)
Returns a/Rs for given parameters.

`vespa.transit_basic.`**`eclipse_pars`**(*P*, *M1*, *M2*, *R1*, *R2*, *ecc=0*, *inc=90*, *w=0*, *sec=False*)
retuns p,b,aR from P,M1,M2,R1,R2,ecc,inc,w

`vespa.transit_basic.`**`eclipse_tt`**(*p0*, *b*, *aR*, *P=1*, *ecc=0*, *w=0*, *npts=100*, *u1=0.394*, *u2=0.261*, *conv=True*, *cadence=0.018819444444444444*, *frac=1*, *sec=False*, *pars0=None*, *tol=0.0001*, *width=3*)
Trapezoidal parameters for simulated orbit.

All arguments passed to `eclipse()` except the following:

Parameters **`pars0`** – (optional) Initial guess for least-sq optimization for trapezoid parameters.

Return dur,dep,slope Best-fit duration, depth, and T/tau for eclipse shape.

vespa.transit_basic.**occultquad**(*z*, *u1*, *u2*, *p0*, *return_components=False*)

Mandel-Agol code: # Python translation of IDL code. # This routine computes the lightcurve for occultation of a # quadratically limb-darkened source without microlensing. Please # cite Mandel & Agol (2002) and Eastman & Agol (2008) if you make use # of this routine in your research. Please report errors or bugs to # jdeast@astronomy.ohio-state.edu

---

Note: Should probably wrap the Fortran code at some point. (This particular part of the code was put together awhile ago.)

---

class vespa.transit_basic.**TraptransitModel**(*ts*, *fs*, *sigs=0.0001*, *maxslope=30*)

Model to enable MCMC fitting of trapezoidal shape.

vespa.transit_basic.**traptransit_MCMC**(*ts, fs, dfs=1e-05, nwalkers=200, nburn=300, niter=1000, threads=1, p0=[0.1, 0.1, 3, 0], return_sampler=False, maxslope=30*)

Fit trapezoidal model to provided ts, fs, [dfs] using MCMC.

Standard emcee usage.

CHAPTER 6

Star Populations

The fundamental population unit within `vespa` is a *StarPopulation*, from which `EclipsePopulation` inherits. This is the basic object which keeps track of the properties of a population of stars and enables application of various observational constraints to rule out portions of the population.

For the built-in false positive populations, `EBPopulation` inherits from *Observed_BinaryPopulation*, and `HEBPopulation` inherits from *Observed_TriplePopulation*. `BEBPopulation` inherits from *BGStarPopulation* through *BGStarPopulation_TRILEGAL*.

class vespa.stars.**StarPopulation**(*stars=None*, *distance=None*, *max_distance=1000*, *convert_absmags=True*, *name=''*, *orbpop=None*, *mags=None*)

A population of stars.

This object contains information of a simulated population of stars. It has a flexible purpose– it could represent many random realizations of a single system, or it could also represent many different random systems. This is the general base class; subclasses include, e.g., `MultipleStarPopulation` and *BGStarPopulation_TRILEGAL*.

The `StarPopulation.stars` attribute is a `pandas.DataFrame` containing all the information about all the random realizations, such as the physical star properties (mass, radius, etc.) and observational characteristics (magnitudes in different bands).

The `StarPopulation.orbpop` attribute stores information about the orbits of the random stars, if such a thing is relevant for the population in question (such as, e.g., a `MultipleStarPopulation`). If orbits are relevant, then attributes such as *StarPopulation.Rsky*, *StarPopulation.RV*, and *StarPopulation.dmag()* are defined as well.

Importantly, you can apply constraints to a *StarPopulation*, implemented via the `Constraint` class. You can constrain properties of the stars to be within a given range, you can apply a `ContrastCurveConstraint`, simulating the exclusion curve of an imaging observation, and many others.

You can save and re-load *StarPopulation* objects using *StarPopulation.save_hdf()* and *StarPopulation.load_hdf()*.

Warning: Support for saving constraints is planned and partially implemented but untested.

Any subclass must be able to be initialized with no arguments, with no calculations being done; this enables the way that *StarPopulation.load_hdf()* is implemented to work properly.

Parameters

- **stars** – (pandas.DataFrame, optional) Table containing properties of stars. Magnitude properties end with "_mag". Default is that these magnitudes are absolute, and get converted to apparent magnitudes based on distance, which is either provided or randomly assigned.
- **distance** (astropy.units.Quantity, float, or array-like, optional) – If None, then distances of stars are assigned randomly out to max_distance, or by comparing to mags. If float, then assumed to be in parsec. Or, if stars already has a distance column, this is ignored.
- **max_distance** (astropy.units.Quantity or float, optional) – Quantity or float, optional Max distance out to which distances will be simulated, according to random placements in volume ($p(d)simd^2$). Ignored if stars already has a distance column.
- **convert_absmags** – (bool, optional) If True, then magnitudes in stars will be converted to apparent magnitudes based on distance. If False, then magnitudes will be kept as-is. Ignored if stars already has a distance column.
- **orbpop** (orbits.OrbitPopulation) – Describes the orbits of the stars.

RV
Radial velocity difference between "primary" and "secondary" (exact meaning varies)

Rsky
Projected angular distance between "primary" and "secondary" (exact meaning varies)

append(*other*)
Appends stars from another StarPopulations, in place.

Parameters **other** – Another *StarPopulation*; must have same columns as self.

apply_cc(*cc*, *distribution_skip=False*, ***kwargs*)
Apply contrast-curve constraint to population.

Only works if object has Rsky, dmag attributes

Parameters

- **cc** (ContrastCurveConstraint) – Contrast curve.
- **distribution_skip** – This is by default True. *To be honest, I'm not exactly sure why. Might be important, might not (don't remember).*
- ****kwargs** – Additional keyword arguments passed to *StarPopulation.apply_constraint()*.

apply_constraint(*constraint*, *selectfrac_skip=False*, *distribution_skip=False*, *overwrite=False*)
Apply a constraint to the population

Parameters

- **constraint** (Constraint) – Constraint to apply.
- **selectfrac_skip** – (optional) If True, then this constraint will not be considered towards diminishing the

apply_trend_constraint (*limit*, *dt*, *distribution_skip=False*, ***kwargs*)
Constrains change in RV to be less than limit over time dt.

Only works if `dRV` and `Plong` attributes are defined for population.

Parameters

- **limit** – Radial velocity limit on trend. Must be `astropy.units.Quantity` object, or else interpreted as m/s.
- **dt** – Time baseline of RV observations. Must be `astropy.units.Quantity` object; else interpreted as days.
- **distribution_skip** – This is by default `True`. *To be honest, I'm not exactly sure why. Might be important, might not (don't remember).*
- ****kwargs** – Additional keyword arguments passed to *StarPopulation.apply_constraint()*.

apply_vcc (*vcc*, *distribution_skip=False*, ***kwargs*)
Applies "velocity contrast curve" to population.

That is, the constraint that comes from not seeing two sets of spectral lines in a high resolution spectrum.

Only works if population has `dmag` and `RV` attributes.

Parameters

- **vcc** – Velocity contrast curve; dmag vs. delta-RV.
- **distribution_skip** – This is by default `True`. *To be honest, I'm not exactly sure why. Might be important, might not (don't remember).*
- ****kwargs** – Additional keyword arguments passed to *StarPopulation.apply_constraint()*.

bands
Bandpasses for which StarPopulation has magnitude data

constrain_property (*prop*, *lo=-1*, *hi=1*, *measurement=None*, *thresh=3*, *selectfrac_skip=False*, *distribution_skip=False*)
Apply constraint that constrains property.

Parameters

- **prop** (`str`) – Name of property. Must be column in `self.stars`.
- **lo,hi** – (optional) Low and high allowed values for `prop`. Defaults to `-np.inf` and `np.inf` to allow for defining only lower or upper limits if desired.
- **measurement** – (optional) Value and error of measurement in form `(value, error)`.
- **thresh** – (optional) Number of "sigma" to allow for measurement constraint.
- **selectfrac_skip,distribution_skip** – Passed to *StarPopulation.apply_constraint()*.

constraint_df
A DataFrame representing all constraints, hidden or not

constraint_piechart (*primarylist=None*, *fig=None*, *title=''*, *colordict=None*, *legend=True*, *nolabels=False*)
Makes piechart illustrating constraints on population

Parameters

- **primarylist** – (optional) List of most import constraints to show (see *StarPopulation.constraint_stats()*)
- **fig** – (optional) Passed to `plotutils.setfig()`.
- **title** – (optional) Title for pie chart
- **colordict** – (optional) Dictionary describing colors (keys are constraint names).
- **legend** – (optional) `bool` indicating whether to display a legend.
- **nolabels** – (optional) If `True`, then leave out legend labels.

constraint_stats (*primarylist=None*)
Returns information about effect of constraints on population.

Parameters **primarylist** – List of constraint names that you want specific information on (i.e., not blended within "multiple constraints".)

Returns `dict` of what percentage of population is ruled out by each constraint, including a "multiple constraints" entry.

constraints
Constraints applied to the population.

countok
Boolean array showing which stars pass all count constraints.

A "count constraint" is a constraint that affects the number of stars.

dRV (*dt*)
Change in RV between two epochs separated by dt

Parameters **dt** – Time difference between two epochs, either `astropy.units.Quantity` or days.

Returns Change in RV.

distance
Distance to stars.

distok
Boolean array showing which stars pass all distribution constraints.

A "distribution constraint" is a constraint that affects the distribution of stars, rather than just the number.

distribution_skip
Names of constraints that should *not* be considered for distribution purposes

dmag (*band*)
Magnitude difference between "primary" and "secondary" in given band

Exact definition will depend on context. Only legit if `self.mags` is defined (i.e., not `None`).

Parameters **band** – (`string`) Desired photometric bandpass.

generate (**args*, ***kwargs*)
Function that generates population.

hidden_constraints
Constraints applied to the population, but temporarily removed.

is_ruled_out
Will be `True` if contraints rule out all (or all but one) instances

classmethod load_hdf(*filename*, *path=''*)
Loads StarPopulation from .h5 file

Correct properties should be restored to object, and object will be original type that was saved. Complement to *StarPopulation.save_hdf()*.

Example usage:

```
>>> from vespa.stars import Raghavan_BinaryPopulation, StarPopulation
>>> pop = Raghavan_BinaryPopulation(1., n=1000)
>>> pop.save_hdf('test.h5')
>>> pop2 = StarPopulation.load_hdf('test.h5')
>>> pop == pop2
    True
>>> pop3 = Ragahavan_BinaryPopulation.load_hdf('test.h5')
>>> pop3 == pop2
    True
```

Parameters

- **filename** – HDF file with saved *StarPopulation*.
- **path** – Path within HDF file.

Returns *StarPopulation* or appropriate subclass; whatever was saved with *StarPopulation.save_hdf()*.

prophist(*prop*, *fig=None*, *log=False*, *mask=None*, *selected=False*, ***kwargs*)
Plots a 1-d histogram of desired property.

Parameters

- **prop** – Name of property to plot. Must be column of `self.stars`.
- **fig** – (optional) Argument for `plotutils.setfig()`
- **log** – (optional) Whether to plot the histogram of log10 of the property.
- **mask** – (optional) Boolean array (length of `self.stars`) to say which indices to plot (`True` is good).
- **selected** – (optional) If `True`, then only the "selected" stars (that is, stars obeying all distribution constraints attached to this object) will be plotted. In this case, `mask` will be ignored.
- ****kwargs** – Additional keyword arguments passed to `plt.hist()`.

prophist2d(*propx*, *propy*, *mask=None*, *logx=False*, *logy=False*, *fig=None*, *selected=False*, ***kwargs*)
Makes a 2d density histogram of two given properties

Parameters

- **propx,propy** – Names of properties to histogram. Must be names of columns in `self.stars` table.
- **mask** – (optional) Boolean mask (`True` is good) to say which indices to plot. Must be same length as `self.stars`.
- **logx,logy** – (optional) Whether to plot the log10 of x and/or y properties.
- **fig** – (optional) Argument passed to `plotutils.setfig()`.

- **selected** – (optional) If `True`, then only the "selected" stars (that is, stars obeying all distribution constraints attached to this object) will be plotted. In this case, `mask` will be ignored.
- **kwargs** – Additional keyword arguments passed to `plotutils.plot2dhist()`.

remove_constraint (*name*)
Remove a constraint (make it "hidden")

Parameters **name** – Name of constraint.

replace_constraint (*name*, *selectfrac_skip=False*, *distribution_skip=False*)
Re-apply constraint that had been removed

Parameters

- **name** – Name of constraint to replace
- **selectfrac_skip,distribution_skip** – (optional) Same as *StarPopulation.apply_constraint()*

save_hdf (*filename*, *path=''*, *properties=None*, *overwrite=False*, *append=False*)
Saves to HDF5 file.

Subclasses should be sure to define `_properties` attribute to ensure that all correct attributes get saved. Load a saved population with *StarPopulation.load_hdf()*.

Example usage:

```
>>> from vespa.stars import Raghavan_BinaryPopulation, StarPopulation
>>> pop = Raghavan_BinaryPopulation(1., n=1000)
>>> pop.save_hdf('test.h5')
>>> pop2 = StarPopulation.load_hdf('test.h5')
>>> pop == pop2
    True
>>> pop3 = Ragahavan_BinaryPopulation.load_hdf('test.h5')
>>> pop3 == pop2
    True
```

Parameters

- **filename** – Name of HDF file.
- **path** – (optional) Path within HDF file to save object.
- **properties** – (optional) Names of any properties (in addition to those defined in `_properties` attribute) that you wish to save. (This is an old keyword, and should probably be removed. Feel free to ignore it.)
- **overwrite** – (optional) Whether to overwrite file if it already exists. If `True`, then any existing file will be deleted before object is saved. Use `append` if you don't wish this to happen.
- **append** – (optional) If `True`, then if the file exists, then only the particular path in the file will get written/overwritten. If `False` and both file and path exist, then an `IOError` will be raised. If `False` and file exists but not path, then no error will be raised.

selected
All stars that pass all distribution constraints.

selectfrac
Fraction of stars that pass count constraints.

selectfrac_skip
Names of constraints that should *not* be considered for counting purposes

set_maxrad(*maxrad*, *distribution_skip=True*)
Adds a constraint that rejects everything with Rsky > maxrad

Requires `Rsky` attribute, which should always have units.

Parameters

- **maxrad** (`astropy.units.Quantity`) – The maximum angular value of Rsky.
- **distribution_skip** – This is by default `True`. *To be honest, I'm not exactly sure why. Might be important, might not (don't remember).*

6.1 Observationally Constrained Star Populations

`EBPopulation` and `HEBPopulation` inherit from very similar star population classes: *`Observed_BinaryPopulation`* and *`Observed_TriplePopulation`*. Both of these take either photometric or spectroscopic observed properties of a star and generate binary or triple populations consistent with those observations.

class vespa.stars.**Observed_BinaryPopulation**(*mags=None, mag_errs=None, Teff=None, logg=None, feh=None, starmodel=None, n=20000.0, ichrone='dartmouth', bands=['g', 'r', 'i', 'z', 'J', 'H', 'K', 'Kepler'], period=None, ecc=None, orbpop=None, stars=None, **kwargs*)
A population of binary stars matching observed constraints.

Parameters

- **mags** (`dict`) – Observed apparent magnitudes
- **Teff,logg,feh** – Observed spectroscopic properties of primary star, if available. Format: `(value, err)`.
- **starmodel** – `isochrones.BinaryStarModel`. If not passed, it will be generated.

generate(*mags=None, mag_errs=None, n=10000.0, ichrone='dartmouth', starmodel=None, Teff=None, logg=None, feh=None, bands=['g', 'r', 'i', 'z', 'J', 'H', 'K', 'Kepler'], orbpop=None, period=None, ecc=None, **kwargs*)
Function that generates population.

classmethod load_hdf(*filename*, *path=''*)
Loads StarPopulation from .h5 file

Correct properties should be restored to object, and object will be original type that was saved. Complement to *`StarPopulation.save_hdf()`*.

Example usage:

```
>>> from vespa.stars import Raghavan_BinaryPopulation, StarPopulation
>>> pop = Raghavan_BinaryPopulation(1., n=1000)
>>> pop.save_hdf('test.h5')
>>> pop2 = StarPopulation.load_hdf('test.h5')
>>> pop == pop2
    True
>>> pop3 = Ragahavan_BinaryPopulation.load_hdf('test.h5')
```

(continues on next page)

(continued from previous page)

```
>>> pop3 == pop2
    True
```

Parameters

- **filename** – HDF file with saved *StarPopulation*.
- **path** – Path within HDF file.

Returns *StarPopulation* or appropriate subclass; whatever was saved with *StarPopulation.save_hdf()*.

save_hdf (*filename*, *path=''*, ***kwargs*)

Saves to HDF5 file.

Subclasses should be sure to define `_properties` attribute to ensure that all correct attributes get saved. Load a saved population with *StarPopulation.load_hdf()*.

Example usage:

```
>>> from vespa.stars import Raghavan_BinaryPopulation, StarPopulation
>>> pop = Raghavan_BinaryPopulation(1., n=1000)
>>> pop.save_hdf('test.h5')
>>> pop2 = StarPopulation.load_hdf('test.h5')
>>> pop == pop2
    True
>>> pop3 = Ragahavan_BinaryPopulation.load_hdf('test.h5')
>>> pop3 == pop2
    True
```

Parameters

- **filename** – Name of HDF file.
- **path** – (optional) Path within HDF file to save object.
- **properties** – (optional) Names of any properties (in addition to those defined in `_properties` attribute) that you wish to save. (This is an old keyword, and should probably be removed. Feel free to ignore it.)
- **overwrite** – (optional) Whether to overwrite file if it already exists. If `True`, then any existing file will be deleted before object is saved. Use `append` if you don't wish this to happen.
- **append** – (optional) If `True`, then if the file exists, then only the particular path in the file will get written/overwritten. If `False` and both file and path exist, then an `IOError` will be raised. If `False` and file exists but not path, then no error will be raised.

starmodel_props

Default mag_err is 0.05, arbitrarily

class vespa.stars.**Observed_TriplePopulation** (*mags=None, mag_errs=None, Teff=None, logg=None, feh=None, starmodel=None, n=20000.0, ichrone='dartmouth', bands=['g', 'r', 'i', 'z', 'J', 'H', 'K', 'Kepler'], period=None, ecc=None, orbpop=None, stars=None, **kwargs*)

A population of triple stars matching observed constraints.

Parameters

- **mags** (dict) – Observed apparent magnitudes
- **Teff,logg,feh** – Observed spectroscopic properties of primary star, if available. Format: (value, err).
- **starmodel** – isochrones.TripleStarModel. If not passed, it will be generated.

generate (*mags=None, mag_errs=None, n=10000.0, ichrone='dartmouth', starmodel=None, Teff=None, logg=None, feh=None, bands=['g', 'r', 'i', 'z', 'J', 'H', 'K', 'Kepler'], orbpop=None, period=None, ecc=None, **kwargs*)

Function that generates population.

classmethod load_hdf (*filename*, *path=''*)

Loads StarPopulation from .h5 file

Correct properties should be restored to object, and object will be original type that was saved. Complement to *StarPopulation.save_hdf()*.

Example usage:

```
>>> from vespa.stars import Raghavan_BinaryPopulation, StarPopulation
>>> pop = Raghavan_BinaryPopulation(1., n=1000)
>>> pop.save_hdf('test.h5')
>>> pop2 = StarPopulation.load_hdf('test.h5')
>>> pop == pop2
    True
>>> pop3 = Ragahavan_BinaryPopulation.load_hdf('test.h5')
>>> pop3 == pop2
    True
```

Parameters

- **filename** – HDF file with saved *StarPopulation*.
- **path** – Path within HDF file.

Returns *StarPopulation* or appropriate subclass; whatever was saved with *StarPopulation.save_hdf()*.

save_hdf (*filename*, *path=''*, ***kwargs*)

Saves to HDF5 file.

Subclasses should be sure to define _properties attribute to ensure that all correct attributes get saved. Load a saved population with *StarPopulation.load_hdf()*.

Example usage:

```
>>> from vespa.stars import Raghavan_BinaryPopulation, StarPopulation
>>> pop = Raghavan_BinaryPopulation(1., n=1000)
>>> pop.save_hdf('test.h5')
>>> pop2 = StarPopulation.load_hdf('test.h5')
>>> pop == pop2
    True
>>> pop3 = Ragahavan_BinaryPopulation.load_hdf('test.h5')
>>> pop3 == pop2
    True
```

Parameters

- **filename** – Name of HDF file.
- **path** – (optional) Path within HDF file to save object.
- **properties** – (optional) Names of any properties (in addition to those defined in `_properties` attribute) that you wish to save. (This is an old keyword, and should probably be removed. Feel free to ignore it.)
- **overwrite** – (optional) Whether to overwrite file if it already exists. If `True`, then any existing file will be deleted before object is saved. Use `append` if you don't wish this to happen.
- **append** – (optional) If `True`, then if the file exists, then only the particular path in the file will get written/overwritten. If `False` and both file and path exist, then an `IOError` will be raised. If `False` and file exists but not path, then no error will be raised.

starmodel_props
Default mag_err is 0.05, arbitrarily

6.2 Background Star Population

`BEBPopulation` inherits from *BGStarPopulation* through *BGStarPopulation_TRILEGAL*.

class vespa.stars.**BGStarPopulation_TRILEGAL**(*filename=None*, *ra=None*, *dec=None*, *mags=None*, *maxrad=1800*, ***kwargs*)

Creates TRILEGAL simulation for ra,dec; loads as BGStarPopulation

Parameters

- **filename** – Desired name of the TRILEGAL simulation. Can either have '.h5' extension or not. If filename (or 'filename.h5') exists locally, it will be loaded; otherwise, TRILEGAL will be called via the `get_trilegal` perl script, and the file will be generated.
- **ra,dec** – (optional) Sky coordinates of TRILEGAL simulation. Must be passed if generating TRILEGAL simulation and not just reading from existing file.
- **mags** ((optional) `dict`) – (optional) Dictionary of primary star magnitudes (if this is being used to generate a background population behind a particular foreground star). This must be set in order to use the `dmag` attribute.
- **maxrad** – (optional) Maximum distance (arcsec) out to which to place simulated stars.
- ****kwargs** – Additional keyword arguments passed to `stars.trilegal.get_trilegal()`

class vespa.stars.**BGStarPopulation**(*stars=None*, *mags=None*, *maxrad=1800*, *density=None*, ***kwargs*)

Background star population

This should usually be accessed via the *BGStarPopulation_TRILEGAL* subclass.

Parameters

- **stars** – (`pandas.DataFrame`, optional) Properties of stars. Must have 'distance' column defined.
- **mags** – (optional) Magnitudes of primary (foreground) stars.
- **maxrad** – (optional) Maximum distance (arcseconds) of BG stars from foreground primary star.
- **density** – (optional) Density in arcsec^{-2} for BG star population.

- ****kwargs** – Additional keyword arguments passed to *StarPopulation*.

Rsky
Project on-sky separation between primary star and BG stars

dmag(*band*)
Magnitude difference between primary star and BG stars

6.3 Other Star Populations

These are the other *StarPopulation* classes defined in `vespa`. *Raghavan_BinaryPopulation* is particularly useful, which produces a population according to the binary distribution described by the Raghavan (2010) survey.

class vespa.stars.**BinaryPopulation**(*stars=None*, *primary=None*, *secondary=None*, *orbpop=None*, *period=None*, *ecc=None*, *is_single=None*, ***kwargs*)

A population of binary stars.

If `vespa.orbits.OrbitPopulation` provided via `orbpop` keyword, that will describe the orbits; if not, then orbit population will be generated. Single stars may be indicated if desired by having their mass set to zero and all magnitudes set to `inf`.

This will usually be used via, e.g., the *Raghavan_BinaryPopulation* subclass, rather than instantiated directly.

Parameters

- **primary, secondary** – (`pandas.DataFrame`) Properties of primary and secondary stars, respectively. These get merged into new `stars` attribute, with "_A" and "_B" tags.
- **orbpop** – (`vespa.orbits.OrbitPopulation`, optional) Object describing orbits of stars. If not provided, then `period` and `ecc` keywords must be provided, or else they will be randomly generated (see below).
- **period, ecc** – Periods and eccentricities of orbits. If `orbpop` not passed, and these are not provided, then periods and eccs will be randomly generated according to the empirical distributions of the Raghavan (2010) and Multiple Star Catalog distributions using *utils.draw_raghavan_periods()* and *utils.draw_eccs()*.

Plong
Orbital period.

Called "Plong" to be consistent with hierarchical populations that have this attribute mean the longer of two periods.

binaries
Subset of stars that are binaries.

binary_fraction(*query='mass_A >= 0'*)
Binary fraction of stars passing given query

Parameters **query** – Query to pass to stars `DataFrame`.

dmag(*band*)
Difference in magnitude between primary and secondary stars

Parameters **band** – Photometric bandpass.

rsky_distribution(*rmax=None*, *smooth=0.1*, *nbins=100*)
Distribution of projected separations

Returns a `simpledists.Hist_Distribution` object.

Parameters

- **rmax** – (optional) Maximum radius to calculate distribution.
- **dr** – (optional) Bin width for histogram
- **smooth** – (optional) Smoothing parameter for `simpledists.Hist_Distribution`
- **nbins** – (optional) Number of bins for histogram

Returns `simpledists.Hist_Distribution` describing Rsky distribution

rsky_lhood(*rsky*, ***kwargs*)
Evaluates Rsky likelihood at provided position(s)

Parameters

- **rsky** – position
- ****kwargs** – Keyword arguments passed to *BinaryPopulation.rsky_distribution()*

singles
Subset of stars that are single.

class vespa.stars.**Simulated_BinaryPopulation**(*M=None, q_fn=None, P_fn=None, ecc_fn=None, n=10000.0, ichrone='dartmouth', qmin=0.1, bands=['g', 'r', 'i', 'z', 'J', 'H', 'K', 'Kepler'], age=9.6, feh=0.0, minmass=0.12, **kwargs*)
Simulates BinaryPopulation according to provide primary mass(es), generating functions, and stellar isochrone models.

Parameters

- **M** (`float` or array-like) – Primary mass(es).
- **q_fn** (*Callable function.*) – (optional) Mass ratio generating function. Must return 'n' mass ratios, and be called as follows:

```
qs = q_fn(n)
```

- **P_fn** (*Callable function.*) – (optional) Orbital period generating function. Must return `n` orbital periods, and be called as follows:

```
Ps = P_fn(n)
```

- **ecc_fn** (*Callable function.*) – (optional) Orbital eccentricity generating function. Must return `n` orbital eccentricities generated according to provided period(s):

```
eccs = ecc_fn(n,Ps)
```

- **n** – (optional) Number of instances to simulate.
- **ichrone** (`isochrones.Isochrone`) – (optional) Stellar model object from which to simulate stellar properties. Default is the default Dartmouth isochrone.

- **bands** – (optional) Photometric bands to simulate via `ichrone`.
- **age,feh** – (optional) log(age) and metallicity at which to simulate population. Can be `float` or array-like
- **minmass** – (optional) Minimum mass to simulate. Default = 0.12.

generate(*M*, *age=9.6*, *feh=0.0*, *ichrone='dartmouth'*, *n=10000.0*, *bands=None*, ***kwargs*)
Function that generates population.

Called by `__init__` if `M` is passed.

class vespa.stars.**Raghavan_BinaryPopulation**(*M=None*, *e_M=0*, *n=10000.0*, *ichrone='dartmouth'*, *age=9.5*, *feh=0.0*, *q_fn=None*, *qmin=0.1*, *minmass=0.12*, ***kwargs*)
A Simulated_BinaryPopulation with empirical default distributions.

Default mass ratio distribution is flat down to chosen minimum mass, default period distribution is from Raghavan (2010), default eccentricity/period relation comes from data from the Multiple Star Catalog (Tokovinin, xxxx).

Parameters

- **M** – Primary mass(es) in solar masses.
- **e_M** – (optional) 1-sigma uncertainty in primary mass.
- **n** – (optional) Number of simulated instances to create.
- **ichrone** (`isochrones.Isochrone`) – (optional) Stellar models from which to generate binary companions.
- **age,feh** – (optional) Age and metallicity of system.
- **name** – (optional) Name of population.
- **q_fn** – (optional) A function that returns random mass ratios. Defaults to flat down to provided minimum mass. Must be able to be called as follows:

```
qs = q_fn(n, qmin, qmax)
```

to provide `n` random mass ratios.

class vespa.stars.**TriplePopulation**(*stars=None*, *primary=None*, *secondary=None*, *tertiary=None*, *orbpop=None*, *period_short=None*, *period_long=None*, *ecc_short=0*, *ecc_long=0*, ***kwargs*)
A population of triple stars.

(Primary) orbits (secondary + tertiary) in a long orbit; secondary and tertiary orbit each other with a shorter orbit. Single or double stars may be indicated if desired by having the masses of secondary or tertiary set to zero, and all magnitudes to `inf`.

Parameters

- **stars** – (optional) Full stars `DataFrame`. If not passed, then primary, secondary, and tertiary must be.
- **primary,secondary,tertiary** – (optional) Properties of primary, secondary, and tertiary stars, in `pandas.DataFrame` form. These will get merged into a new `stars` attribute, with "_A", "_B", and "_C" tags.
- **orbpop** (`TripleOrbitPopulation`) – (optional) Object describing orbits of stars. If not provided, then the period and eccentricity keywords must be provided, or else they will be randomly generated (see below).

- **period_short,period_long,ecc_short,ecc_long** – (array-like, optional) Orbital periods and eccentricities of short and long-period orbits. "Short" describes the close pair of the hierarchical system; "long" describes the separation between the two major components. Randomly generated if not provided.

A_brighter(*band='g'*)
Instances where star A is brighter than (B+C)

BC_brighter(*band='g'*)
Instances where stars (B+C) are brighter than star A

Plong
Longer of two orbital periods in Triple system

binary_fraction(*query='mass_A > 0'*, *unc=False*)
Binary fraction of stars following given query

dRV(*dt*, *band='g'*)
Returns dRV of star A, if A is brighter than B+C, or of star B if B+C is brighter

dmag(*band*)
Difference in magnitudes between fainter and brighter components in band.

Parameters **band** – Photometric bandpass.

triple_fraction(*query='mass_A > 0'*, *unc=False*)
Triple fraction of stars following given query

CHAPTER 7

Observational Constraints

The mechanism for incorporating observational constraints into the `vespa` calculations is via the `Constraint` object. The way this is currently implemented is that a `Constraint` is essentially a boolean array of the same length as a `EclipsePopulation` (or `StarPopulation`, more generally), where simulated instances that would not have been detected by the observation in question remain `True`, and any instances that would have been observed become `False`.

7.1 Contrast Curve Constraint

One of the most common kinds of follow-up observation for false positive identification/ analysis is a high-resolution imaging observation. The output of such an observation is a "contrast curve": the detectable brightness contrast as a function of angular separation from the central source. As every false positive `EclipsePopulation` simulation includes simulated magnitudes in many different bands as well as simulated sky-positions relative to the central target star, it is very easy to implement a contrast curve in this way: any instances that would have been detected by the observation get ruled out, and thus the "prior" factor diminishes for that scenario (this is kept track of by the `EclipsePopulation.countok` attribute).

class vespa.stars.contrastcurve.**ContrastCurve**(*rs*, *dmags*, *band*, *mag=None*, *name=None*)

Object representing an imaging contrast curve

Usually accessed via *ContrastCurveFromFile* and then applied using *ContrastCurveConstraint*, e.g., through `StarPopulation.apply_cc()`.

Parameters

- **rs** – Angular separation from target star, in arcsec.
- **dmags** – Magnitude contrast.
- **band** – Photometric bandpass in which observation is taken.
- **mag** – Magnitude of central star (rarely used?)
- **name** – Name; e.g., "PHARO J-band", "Keck AO", etc. Should be a decent label.

class vespa.stars.contrastcurve.**ContrastCurveFromFile**(*filename*, *band*, *mag=None*, *mas=False*, ***kwargs*)

A contrast curve derived from a two-column file

Parameters

- **filename** – Filename of contrast curve; first column separation in arcsec, second column delta-mag.
- **band** – Bandpass of imaging observation.
- **mas** – Set to `True` if separation is in milliarcsec rather than arcsec.

class vespa.stars.contrastcurve.**ContrastCurveConstraint**(*rs*, *dmags*, *cc*, *name='CC'*, ***kwargs*)

CHAPTER 8

Star Utilities

The `vespa.stars` module provides several useful utilities in support of generating *`StarPopulation`* objects.

8.1 Extinction at Infinity

`vespa.stars.extinction.`**`get_AV_infinity`**(*ra*, *dec*, *frame='icrs'*)

Gets the A_V exctinction at infinity for a given line of sight.

Queries the NED database using `curl`.

Note: It would be desirable to rewrite this to avoid dependence on `curl`.

Parameters

- **`ra,dec`** – Desired coordinates, in degrees.
- **`frame`** – (optional) Frame of input coordinates (e.g., `'icrs', 'galactic'`)

8.2 TRILEGAL Simulations

`vespa.stars.trilegal.`**`get_trilegal`**(*filename*, *ra*, *dec*, *folder='.'*, *galactic=False*, *filterset='kepler_2mass'*, *area=1*, *maglim=27*, *binaries=False*, *trilegal_version='1.6'*, *sigma_AV=0.1*, *convert_h5=True*)

Runs get_trilegal perl script; optionally saves output into .h5 file

Depends on a perl script provided by L. Girardi; calls the web form simulation, downloads the file, and (optionally) converts to HDF format.

Uses A_V at infinity from `utils.get_AV_infinity()`.

Note: Would be desirable to re-write the get_trilegal script all in python.

Parameters

- **filename** – Desired output filename. If extension not provided, it will be added.
- **ra,dec** – Coordinates (ecliptic) for line-of-sight simulation.
- **folder** – (optional) Folder to which to save file. *Acknowledged, file control in this function is a bit wonky.*
- **filterset** – (optional) Filter set for which to call TRILEGAL.
- **area** – (optional) Area of TRILEGAL simulation [sq. deg]
- **maglim** – (optional) Limiting magnitude in first mag (by default will be Kepler band) If want to limit in different band, then you have to got directly to the `get_trilegal` perl script.
- **binaries** – (optional) Whether to have TRILEGAL include binary stars. Default `False`.
- **trilegal_version** – (optional) Default `'1.6'`.
- **sigma_AV** – (optional) Fractional spread in A_V along the line of sight.
- **convert_h5** – (optional) If true, text file downloaded from TRILEGAL will be converted into a `pandas.DataFrame` stored in an HDF file, with `'df'` path.

8.3 Other Utility Functions

Here is a grab bag of stuff that gets used (or maybe doesn't) when generating various *StarPopulation* objects.

vespa.stars.utils.**addmags**(**mags*)
"Adds" magnitudes. Yay astronomical units!

vespa.stars.utils.**dfromdm**(*dm*)
Returns distance given distance modulus.

vespa.stars.utils.**distancemodulus**(*d*)
Returns distance modulus given d in parsec.

vespa.stars.utils.**draw_eccs**(*n*, *per=10*, *binsize=0.1*, *fuzz=0.05*, *maxecc=0.97*)
draws eccentricities appropriate to given periods, generated according to empirical data from Multiple Star Catalog

vespa.stars.utils.**draw_msc_periods**(*n*)
Draw orbital periods according to Multiple Star Catalog

vespa.stars.utils.**draw_pers_eccs**(*n*, ***kwargs*)
Draw random periods and eccentricities according to empirical survey data.

vespa.stars.utils.**draw_raghavan_periods**(*n*)
Draw orbital periods according to Raghavan (2010)

vespa.stars.utils.**fluxfrac**(**mags*)
Returns fraction of total flux in first argument, assuming all are magnitudes.

vespa.stars.utils.**mult_masses**(*mA*, *f_binary=0.4*, *f_triple=0.12*, *minmass=0.11*, *qmin=0.1*, *n=100000.0*)

Returns m1, m2, and m3 appropriate for TripleStarPopulation, given "primary" mass (most massive of system) and binary/triple fractions.

star with m1 orbits (m2 + m3). This means that the primary mass mA will correspond either to m1 or m2. Any mass set to 0 means that component does not exist.

vespa.stars.utils.**rochelobe**(*q*)

returns r1/a; q = M1/M2

vespa.stars.utils.**semimajor**(*P*, *mstar=1*)

Returns semimajor axis in AU given P in days, mstar in solar masses.

vespa.stars.utils.**withinroche**(*semimajors*, *M1*, *R1*, *M2*, *R2*)

Returns boolean array that is True where two stars are within Roche lobe

CHAPTER 9

Orbits

If they represent binary or triple star systems, *vespa.stars.StarPopulation* objects are created with a large population of randomized orbits. This is done using the `OrbitPopulation` and `TripleOrbitPopulation` objects.

The engine that makes it possible to initialize large numbers of random orbital positions nearly instantaneously is the *kepler.Efn()* function (as used by *utils.orbit_posvel()*), which uses a precomputed grid to interpolate the solutions to Kepler's equation for a given mean anomaly and eccentricity (or arrays thereof).

The final coordinate system of these populations is "observer-oriented," with the `z` axis along the line of sight, and the `x-y` plane being the plane of the sky. Practically, this is accomplished by first simulating all the random orbits in the `x-y` plane, and then "observing" them from lines of sight randomly oriented on the unit sphere, and projecting appropriately.

Coordinates are handled using `astropy.coordinates.SkyCoord` objects.

9.1 Orbit Populations

class vespa.orbits.populations.**OrbitPopulation**(*M1*, *M2*, *P*, *ecc=0*, *n=None*, *mean_anomaly=None*, *obsx=None*, *obsy=None*, *obsz=None*, *obspos=None*)

Population of orbits.

Parameters

- **M1,M2** – Primary and secondary masses (if not `Quantity`, assumed to be in solar masses). Can be `float`, array-like or `Quantity`.
- **P** (`float`, array-like or `Quantity`.) – Orbital period(s) (if not `Quantity`, assumed to be in days)
- **ecc** – (`float` or array-like, optional) Eccentricities.
- **n** – (optional) Number of instances to simulate. If not provided, then this number will be the length of `M2` (or `P`) provided.

- **mean_anomaly** – (optional) Mean anomalies of orbits. Usually this will just be set randomly, but can be provided to initialize a particular state (e.g., when restoring an *OrbitPopulation* object from a saved state).
- **obsy, obsz** (*obsx,*) – (optional) "Observer" positions to define coordinates. Will be set randomly if not provided.
- **obspos** (astropy.coordinates.SkyCoord) – (optional) "Observer" positions may be set with a SkyCoord object (replaces obsx, obsy, obsz)

RV
Relative radial velocities of two stars

RV_com1
RVs of star 1 relative to center-of-mass

RV_com2
RVs of star 2 relative to center-of-mass

RV_timeseries(*ts*, *recalc=False*)
Radial Velocity time series for star 1 at given times ts.

Parameters

- **ts** (array-like or Quantity) – Times. If not Quantity, assumed to be in days.
- **recalc** – (optional) If False, then if called with the exact same ts as last call, it will return cached calculation.

Rsky
Sky separation of stars, in projected AU.

dRV(*dt*, *com=False*)
Change in RV of star 1 for time separation dt (default=days)

Parameters

- **dt** (float, array-like, or Quantity) – Time separation for which to compute RV change. If not a Quantity, then assumed to be in days.
- **com** – (bool, optional) If True, then return dRV of star 1 in center-of-mass frame.

Return dRV Change in radial velocity over time dt.

dataframe
Summary DataFrame of OrbitPopulation

Used to save/restore state.

classmethod from_df(*df*)
Creates an OrbitPopulation from a DataFrame.

Parameters **df** – pandas.DataFrame object. Must contain the following columns: ['M1','M2','P','ecc','mean_anomaly','obsx','obsy','obsz'], i.e., as what is accessed via *OrbitPopulation.dataframe*.

Returns *OrbitPopulation*.

classmethod load_hdf(*filename*, *path=''*)
Loads OrbitPopulation from HDF file.

Parameters

- **filename** – HDF file
- **path** – Path within HDF file store where *OrbitPopulation* is saved.

save_hdf (*filename*, *path=''*)
Saves all relevant data to .h5 file; so state can be restored.

scatterplot (*fig=None*, *figsize=(7, 7)*, *ms=0.5*, *rmax=None*, *log=False*, ***kwargs*)
Makes a scatter plot of projected X-Y sky separation

Parameters

- **fig** – (optional) Passed to `plotutils.setfig()`
- **figsize** – (optional) Size of figure (in).
- **ms** – (optional) Marker size
- **rmax** – (optional) Maximum projected radius to plot.
- **log** – (optional) Whether to plot with log scale.
- ****kwargs** – Additional keyword arguments passed to `plt.plot`.

class vespa.orbits.populations.**TripleOrbitPopulation** (*M1*, *M2*, *M3*, *Plong*, *Pshort*, *ecclong=0*, *eccshort=0*, *n=None*, *mean_anomaly_long=None*, *obsx_long=None*, *obsy_long=None*, *obsz_long=None*, *obspos_long=None*, *mean_anomaly_short=None*, *obsx_short=None*, *obsy_short=None*, *obsz_short=None*, *obspos_short=None*)
Stars 2 and 3 orbit each other (short orbit), far from star 1 (long orbit)

This object defines the orbits of a triple star system, with orbits calculated assuming the "long" orbit does not perturb the "short" orbit, which will not be true in the long run, but should be true over short timescales as long as `Plong >> Pshort`.

A *TripleOrbitPopulation* is essentially made up of two *OrbitPopulation* objects: one for the "long" orbit and one for the "short."

Parameters

- **M1,M2,M3** – Masses of stars. Stars 2 and 3 are in a short orbit, far away from star 1. If not `astropy.units.Quantity` objects, then assumed to be in solar mass units. May be single value or array-like.
- **Plong,Pshort** – Orbital Periods. Plong is orbital period of 2+3 and 1; Pshort is orbital period of 2 and 3. If not `astropy.units.Quantity` objects, assumed to be in days. Can be single value or array-like. N.B. If any item in Pshort happens to be longer than the corresponding item in Plong, they will be switched.
- **ecclong,eccshort** – (optional) Eccentricities. Same story (long vs. short). Default=0 (circular). Can be single value or array-like.
- **n** – (optional) Number of systems to simulate (if `M1`, `M2`, `M3` aren't arrays of size > 1 already).
- **mean_anomaly_short,mean_anomaly_long** – (optional) Mean anomalies. This is only passed if you need to restore a particular specific configuration (i.e., a particular saved simulation), e.g., as done by *TripleOrbitPopulation.from_df()*. If not provided, then randomized on (0, 2pi).

- **obsx_short,obsy_short,obsz_short** – (optional) "Observer" positions for the short orbit. Also only passed for purposes of restoring configuration.
- **obsx_long,obsy_long,obsz_long** – (optional) "Observer" positions for long orbit. Also only passed for purposes of restoring configuration.
- **obspos_short,obspos_long** – (optional) "Observer positions for short and long orbits, provided as `astropy.SkyCoord` objects. These will replace obsx_short/long, obsy_short/long, obsz_short/long parameters if present.

RV
Instantaneous RV of star 1 with respect to system center-of-mass

RV_1
Instantaneous RV of star 1 with respect to system center-of-mass

RV_2
Instantaneous RV of star 2 with respect to system center-of-mass

RV_3
Instantaneous RV of star 3 with respect to system center-of-mass

Rsky
Projected separation of star 2+3 pair from star 1 [projected AU]

dRV(*dt*)
Returns difference in RVs (separated by time dt) of star 1.

Parameters **dt** – Time separation for which to compute RV change. If not an `astropy.units.Quantity` object, then assumed to be in days.

dRV_1(*dt*)
Returns difference in RVs (separated by time dt) of star 1.

Parameters **dt** – Time separation for which to compute RV change. If not an `astropy.units.Quantity` object, then assumed to be in days.

dRV_2(*dt*)
Returns difference in RVs (separated by time dt) of star 2.

Parameters **dt** – Time separation for which to compute RV change. If not an `astropy.units.Quantity` object, then assumed to be in days.

dRV_3(*dt*)
Returns difference in RVs (separated by time dt) of star 3.

Parameters **dt** – Time separation for which to compute RV change. If not an `astropy.units.Quantity` object, then assumed to be in days.

classmethod from_df(*df_long*, *df_short*)
Builds TripleOrbitPopulation from DataFrame

`DataFrame` objects must be of appropriate form to pass to *`OrbitPopulation.from_df()`*.

Parameters **df_short** (*df_long,*) – `pandas.DataFrame` objects to pass to *`OrbitPopulation.from_df()`*.

classmethod load_hdf(*filename*, *path=''*)
Load TripleOrbitPopulation from saved .h5 file.

Parameters

- **filename** – HDF file name.
- **path** – Path within HDF file where data is stored.

save_hdf (*filename*, *path=''*)
Save to HDF5 file in desired path.

9.2 Utility Functions

The following functions are used in the creation of `OrbitPopulation` objects. *kepler.Efn()* is used for instanteous solution of Kepler's equation (via interpolation), and *utils.orbit_posvel()* does the projecting of random orbits into 3-d Cartesian coordinates, assisted by *utils.orbitproject()* and *utils.random_spherepos()*.

vespa.orbits.kepler.**Efn** (*Ms*, *eccs*)
Returns Eccentric anomaly, interpolated from pre-computed grid of M, ecc

Instantaneous solution of Kepler's equation!

Works for `-2*np.pi < Ms < 2*np.pi` and `eccs <= 0.97`

Parameters

- **Ms** – (`float` or array-like) Mean anomaly
- **eccs** – (`float` or array-like)

vespa.orbits.utils.**orbit_posvel** (*Ms*, *eccs*, *semimajors*, *mreds*, *obspos=None*)
Returns positions in projected AU and velocities in km/s for given mean anomalies.

Returns 3-D positions and velocities as SkyCoord objects, in "observer" reference frame. Uses `kepler.Efn()` to calculate eccentric anomalies using interpolation.

Parameters

- **Ms,eccs,semimajors,mreds** – (`float` or array-like) Mean anomalies, eccentricities, semimajor axes [AU], reduced masses [Msun].
- **obspos** – (`None`, `(x,y,z)` tuple or `SkyCoord` object) Locations of observers for which to return coordinates. If `None` then populate randomly on sphere. If `(x,y,z)` or `SkyCoord` object provided, then use those.

Returns pos,vel `SkyCoord` Objects representing the positions and velocities, the coordinates of which are `Quantity` objects that have units. Positions are in projected AU and velocities in km/s.

vespa.orbits.utils.**orbitproject** (*x*, *y*, *inc*, *phi=0*, *psi=0*)
Transform x,y planar coordinates into observer's coordinate frame.

`x,y` are coordinates in `z=0` plane (plane of the orbit)

observer is at `(inc, phi)` on celestial sphere (angles in radians); `psi` is orientation of final `x-y` axes about the `(inc,phi)` vector.

Returns `x,y,z` values in observer's coordinate frame, where `x,y` are now plane-of-sky coordinates and `z` is along the line of sight.

Parameters

- **x,y** – (`float` or array-like) Coordinates to transform.
- **inc** – (`float` or array-like) Polar angle(s) of observer (where `inc=0` corresponds to north pole of original `x-y` plane). This angle is the same as standard "inclination."
- **phi** – (`float` or array-like, optional) Azimuthal angle of observer around `z` -axis

- **psi** – (float or array-like, optional) Orientation of final observer coordinate frame (azimuthal around (inc,phi) vector.

Return x,y,z (ndarray) Coordinates in observers' frames. x,y in "plane of sky" and z along line of sight.

vespa.orbits.utils.**random_spherepos**(*n*)

Returns SkyCoord object with n positions randomly oriented on the unit sphere.

Parameters n – (int) Number of positions desired.

Return c astropy.coordinates.SkyCoord object with random positions

CHAPTER 10

Other Utilities

Here are documented (occasionally sparsely) a few other utilities used in the `vespa` package.

10.1 Plotting

`vespa.plotutils.`**`plot2dhist`**(*xdata*, *ydata*, *cmap='binary'*, *interpolation='nearest'*, *fig=None*, *logscale=True*, *xbins=None*, *ybins=None*, *nbins=50*, *pts_only=False*, ***kwargs*)

Plots a 2d density histogram of provided data

Parameters

- **`xdata,ydata`** – (array-like) Data to plot.
- **`cmap`** – (optional) Colormap to use for density plot.
- **`interpolation`** – (optional) Interpolation scheme for display (passed to `plt.imshow`).
- **`fig`** – (optional) Argument passed to *`setfig()`*.
- **`logscale`** – (optional) If `True` then the colormap will be based on a logarithmic scale, rather than linear.
- **`xbins,ybins`** – (optional) Bin edges to use (if `None`, then use `np.histogram2d` to find bins automatically).
- **`nbins`** – (optional) Number of bins to use (if `None`, then use `np.histogram2d` to find bins automatically).
- **`pts_only`** – (optional) If `True`, then just a scatter plot of the points is made, rather than the density plot.
- **`**kwargs`** – Keyword arguments passed either to `plt.plot` or `plt.imshow` depending upon whether `pts_only` is set to `True` or not.

vespa.plotutils.**setfig**(*fig=None*, ***kwargs*)
Sets figure to 'fig' and clears; if fig is 0, does nothing (e.g. for overplotting)

if fig is None (or anything else), creates new figure

I use this for basically every function I write to make a plot. I give the function a "fig=None" kw argument, so that it will by default create a new figure.

Note: There's most certainly a better, more object-oriented way of going about writing functions that make figures, but this was put together before I knew how to think that way, so this stays for now as a convenience.

10.2 Stats

vespa.statutils.**conf_interval**(*x*, *L*, *conf=0.683*, *shortest=True*, *conftol=0.001*, *return_max=False*)
Returns desired 1-d confidence interval for provided x, L[PDF]

vespa.statutils.**kdeconf**(*kde*, *conf=0.683*, *xmin=None*, *xmax=None*, *npts=500*, *shortest=True*, *conftol=0.001*, *return_max=False*)
Returns desired confidence interval for provided KDE object

vespa.statutils.**qstd**(*x*, *quant=0.05*, *top=False*, *bottom=False*)
returns std, ignoring outer 'quant' pctiles

10.3 Hashing

In order to be able to compare population objects, it's useful to define utility functions to hash `ndarrays` and `DataFrames` and to combine hashes in a legit way. This is generally useful and could be its own mini-package, but for now it's stashed here.

class vespa.hashutils.**hashable**(*wrapped*, *tight=False*)
Hashable wrapper for ndarray objects.

Instances of ndarray are not hashable, meaning they cannot be added to sets, nor used as keys in dictionaries. This is by design - ndarray objects are mutable, and therefore cannot reliably implement the __hash__() method.

The hashable class allows a way around this limitation. It implements the required methods for hashable objects in terms of an encapsulated ndarray object. This can be either a copied instance (which is safer) or the original object (which requires the user to be careful enough not to modify it).

This class taken from here; edited only slightly.

unwrap()
Returns the encapsulated ndarray.

If the wrapper is "tight", a copy of the encapsulated ndarray is returned. Otherwise, the encapsulated ndarray itself is returned.

vespa.hashutils.**hasharray**(*arr*)
Hashes array-like object (except DataFrame)

vespa.hashutils.**hashcombine**(**xs*)
Combines multiple hashes using xor

vespa.hashutils.**hashdf**(*df*)
hashes a pandas dataframe, forcing values to float

vespa.hashutils.**hashdict**(*d*)
Hash a dictionary

Python Module Index

v

Index

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

V

W