SysSimPyPlots

Release 0.1.0

Matthias Yang He

Sep 20, 2022

CONTENTS

SysSimPyPlots is a codebase for loading, analyzing, and plotting catalogs generated from the SysSim models.

Tip: What's behind the name?

SysSim – a comprehensive forward modeling framework for studying planetary systems based on the *Kepler* mission. In particular, SysSimExClusters provides clustered planetary system models that characterize the underlying occurrence and intra-system correlations of multi-planet systems.

Py – this package is written almost entirely in Python 3. This is unlike the SysSim codebase which is written in Julia.

Plots – while the SysSim codebase provides the workhorse for simulating catalogs of planetary systems and the *Kepler* mission, this package allows you to plot, visualize, and explore those catalogs!

Note: Some parts of the documentation for this project (this site!) are still under development! Feel free to create an issue on Github if you find any problems.

CHAPTER

ONE

CONTENTS

1.1 Installation

1.1.1 Using pip

You can install the most recent stable version of SysSimPyPlots using pip:

```
python -m pip install syssimpyplots
```

1.1.2 From source

All of the code is publicly available on Github. Thus an alternative to pip is to download or clone the repository (make sure to fork it first if you might want to make your own changes) and install it from there:

```
git clone git@github.com:hematthi/SysSimPyPlots.git
python -m pip install .
```

1.1.3 Downloading simulated catalogs

After you have installed SysSimPyPlots, you will still need to download some simulated catalogs in order to explore any models. You can download a single, representative catalog generated from the latest model (the "maximum AMD model" described in He et al. 2020) using this link.

Note: This simulated catalog contains five times as many stars as the Kepler catalog in our sample, and is about 450 MB in size.

If you want to perform more robust analyses, you can download many more simulated catalogs from the SysSimExClusters Simulated Catalog folder.

Note: This folder contains 100 individual catalogs, each with the same number of stars as the Kepler catalog in our sample. The total file size is about 3.5 GB compressed, and roughly 8.2 GB when opened/uncompressed.

More details are provided in the SysSimExClusters repository. Check the READMEs of the individual branches for each paper.

That's it – you are now ready to use SysSimPyPlots!

1.1.4 Dependencies

SysSimPyPlots has been tested on Python >3.7 and uses:

- `numpy` (for almost everything)
- `matplotlib` (for making plots)
- `scipy` (for some miscellaneous functions)
- `corner` (for plotting multi-dimensional parameter spaces)

1.1.5 If you want to go further

If you want to simulate even more additional or new catalogs, you will need to download Julia and install SysSimExClusters which also requires installing ExoplanetsSysSim. Please check those pages for instructions.

1.2 Loading catalogs

The data you download from following the *"Downloading simulated catalogs" section* contains a number of catalog pairs, each consisting of:

- a "**physical catalog**": a set of intrinsic, physical planetary systems (before any observations; contains properties like the true orbital periods, planet radii, etc.)
- an "**observed catalog**": a set of transiting and detected planet candidates derived from a physical catalog (after a Kepler-like mission; contains properties like the measured orbital periods, transit depths, etc.)

1.2.1 Loading physical catalogs

First, you need to import some required packages:

```
from syssimpyplots.general import *
from syssimpyplots.load_sims import *
```

Then specify the path to where you saved your data and load it as follows, for example:

```
load_dir = '/path/to/a/simulated/catalog/' # replace with your path!

cat_phys = load_cat_phys(load_dir + 'physical_catalog.csv')
```

This returns a table (which we stored in `cat_phys`) with the properties of all the planets in the physical catalog, where each row corresponds to one planet. See the documentation for *load_cat_phys* for a detailed description of the table columns.

While you could directly work with the `cat_phys` table, it would be convenient if the data was processed in some more useful ways. For example, you may want to know which planets belong to the same system, their period ratios, etc. You don't have to compute these yourself – we've defined functions for computing these and *many* other summary statistics from the catalogs!

```
sssp_per_sys, sssp = compute_summary_stats_from_cat_phys(file_name_path=load_dir)
```

(This may take a minute or two, depending on the size of the catalog.) The function above outputs two dictionary objects, which we have stored in `sssp_per_sys` and `sssp`. They contain mostly the same information but summarized in different ways.

`sssp_per_sys` includes the detailed properties of each individual planetary system. Most of its data fields are two-dimensional arrays, with the first dimension (i.e. indexing rows) running through the different systems and the second dimension (i.e. indexing columns) running through the different planets in a system. For example, `sssp_per_sys['P_all']` gives a 2-d array of orbital periods.

Warning: Each row is padded with zeros, since different systems have different numbers of planets.

Some fields in `sssp_per_sys` are one-dimensional arrays, i.e. for system-level quantities such as the multiplicity of each system (`sssp_per_sys['Mtot_all']`).

On the other hand, `sssp` contains only one-dimensional arrays, such as `sssp['P_all']` for the orbital periods of all the planets in the catalog. This loses information about which planet(s) belong to which system, but is very convenient for plotting histograms, or performing simple calculations like computing the median period or the number of planets with periods less than 10 days.

For a complete list of all the data fields, see the documentation for the *compute_summary_stats_from_cat_phys* function.

Notice that we only passed the path to the simulated catalog, `load_dir`, to the function `compute_summary_stats_from_cat_phys()`. This tells it to load several ancillary files containing the same information as what's in "physical_catalog.csv", which is actually faster than loading the catalog itself using `load_cat_phys()`. The function *compute_summary_stats_from_cat_phys* also accepts a physical catalog table (i.e. the `cat_phys` object) as input, but we recommend using the load directory path.

1.2.2 Loading observed catalogs

The process for loading simulated observed catalogs is similar; after importing the packages and defining the path to the data as above, simply do:

```
cat_obs = load_cat_obs(load_dir + 'observed_catalog.csv')
```

for loading a table with all of the observed planets, or

```
sss_per_sys, sss = compute_summary_stats_from_cat_obs(file_name_path=load_dir)
```

for computing the summary statistics from the observed catalog.

Analogous to the dictionaries for the physical catalogs, `sss_per_sys` includes the detailed properties of each individual planetary system (mostly two-dimensional arrays), while `sss` includes only one-dimensional arrays. For example, `sss_per_sys['P_obs']` gives a 2-d array of the observed orbital periods, while `sss['P_obs']` gives the same periods as a 1-d array.

Warning: Again, each row in a 2-d array is padded with either zeros or negative ones, since different systems have different numbers of observed planets!

For a complete list of all the data fields, see the documentation for the *compute_summary_stats_from_cat_obs* function.

As before, we only passed the path to the simulated catalog to the function `compute_summary_stats_from_cat_obs()`, which loads several ancillary files containing the same information instead of "observed_catalog.csv". You can also pass the `cat_obs` object into the function but we recommend the load directory path approach.

1.2.3 Reading simulation parameters

You may want to read the number of simulated targets and the period and radius bounds for the simulated planets, without loading the full catalog (which may take several minutes for larger physical catalogs):

```
N_sim, cos_factor, P_min, P_max, radii_min, radii_max = read_targets_period_radius_
↪bounds(load_dir + 'periods.out')
```

You may also want to read the parameters of the model that went into the simulation:

```
param_vals_all = read_sim_params(load_dir + 'periods.out')
```

In these examples, you can replace the `periods.out` file with any of the other simulation files – they all have the same header information.

1.2.4 Loading the Kepler catalog

Analogous to the functions for loading and summarizing an observed catalog, there are also functions for loading and processing the real Kepler data:

```
from syssimpyplots.compare_kepler import *

koi_table = load_Kepler_planets_cleaned()

ssk_per_sys, ssk = compute_summary_stats_from_Kepler_catalog(P_min, P_max, radii_min,␣
↪radii_max)
```

The function *compute_summary_stats_from_Kepler_catalog* requires the arguments `P_min`, `P_max`, `radii_min`, and `radii_max` for selecting a sample of exoplanets that is restricted to a given orbital period and planet radius range, in order to be comparable to the simulated planets – parameters which are conveniently provided by the *read_targets_period_radius_bounds* function shown earlier.

The outputs stored in `ssk_per_sys` and `ssk` contain the same summary statistics as those in `sss_per_sys` and `sss`, respectively.

Tip: The variable names `sss` and `ssk` were chosen to stand for "summary statistics simulated" and "summary statistics Kepler", respectively (and `sssp` for "summary statistics simulated physical"). Of course, you are free to choose whatever variable names you prefer.

You are now ready to use the catalogs to explore the models!

1.3 Plotting histograms

1.3.1 A simple example

On the previous page, you learned how to load a catalog (physical and observed). These catalogs are in the form of dictionaries containing various planetary (and stellar) properties (sometimes referred to as *summary statistics*). One of the most basic and yet illuminating ways of visualizing a catalog is to plot histograms of the various properties. We provide several flexible functions for plotting histograms:

```
from syssimpyplots.general import *
from syssimpyplots.load_sims import *
from syssimpyplots.plot_catalogs import *
from syssimpyplots.compare_kepler import *

load_dir = '/path/to/a/simulated/catalog/' # replace with your path!

sss_per_sys, sss = compute_summary_stats_from_cat_obs(file_name_path=load_dir)

fig_size = (8,4) # size of each figure (width, height)

# To plot a histogram of the observed multiplicities (number of planets per system):
ax = plot_fig_counts_hist_simple(fig_size, [sss_per_sys['Mtot_obs']], [], x_min=0, x_
↪max=8, x_llim=0.5, log_y=True, xlabel_text='Observed multiplicity', ylabel_text=
↪'Number of systems')

# To plot a histogram of the observed orbital periods:
ax = plot_fig_pdf_simple(fig_size, [sss['P_obs']], [], x_min=3., x_max=300.,␣
↪normalize=False, log_x=True, log_y=True, xticks_custom=[3,10,30,100,300], xlabel_text=r
↪'$P$ (days)', ylabel_text='Number of planets')

plt.show()
```

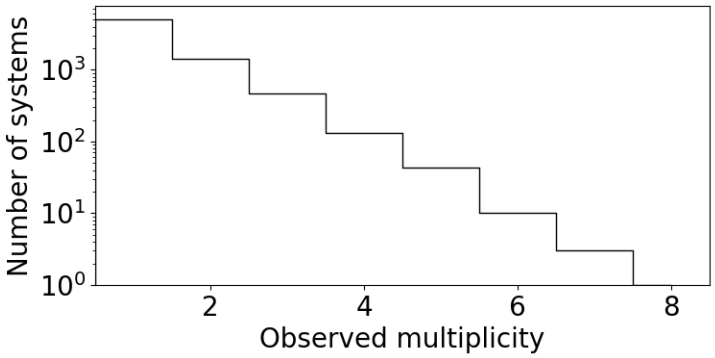


Fig. 1: The observed multiplicity distribution of a simulated catalog.

As demonstrated above, the *plot_fig_counts_hist_simple* function should be used for quantities taking on discrete, integer values, as it is designed to center each bin on an integer. The multiplicity distribution is a perfect example of this case!

For continuous distributions (such as the period distribution), the *plot_fig_pdf_simple* function should be used.

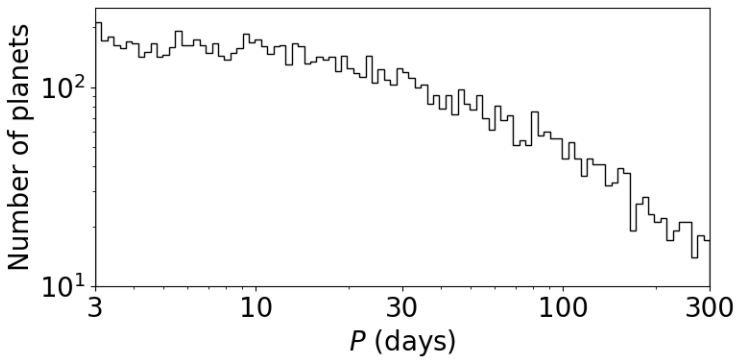


Fig. 2: The observed period distribution of a simulated catalog.

Tip: The two functions above are actually wrappers of the functions *plot_panel_counts_hist_simple* and *plot_panel_pdf_simple*, respectively, which do most of the work and create a single panel (requiring an axes subplot object to plot on) instead of a figure. These are useful for making multi-panel figures!

1.3.2 Plotting multiple catalogs

The third argument (empty list `[]` in the above examples) allows you to easily over-plot the Kepler catalog with a simulated observed catalog. Here is an example:

```
N_sim, cos_factor, P_min, P_max, radii_min, radii_max = read_targets_period_radius_
↪bounds(load_dir + 'periods.out')

# Load the Kepler catalog first:
ssk_per_sys, ssk = compute_summary_stats_from_Kepler_catalog(P_min, P_max, radii_min,␣
↪radii_max)

# To plot a histogram of the observed multiplicities (number of planets per system):
ax = plot_fig_counts_hist_simple(fig_size, [sss_per_sys['Mtot_obs']], [ssk_per_sys['Mtot_
↪obs']], x_min=0, x_max=9, y_max=1, x_llim=0.5, normalize=True, log_y=True, xlabel_text=
↪'Observed multiplicity', ylabel_text='Fraction', legend=True)

# To plot a histogram of the observed orbital periods:
ax = plot_fig_pdf_simple(fig_size, [sss['P_obs']], [ssk['P_obs']], x_min=3., x_max=300.,␣
↪log_x=True, log_y=True, xticks_custom=[3,10,30,100,300], xlabel_text=r'$P$ (days)',␣
↪legend=True)

plt.show()
```

Note that we've set `legend=True` to tell which is which! The `normalize=True` option is also useful when the catalogs have different numbers of systems (in this case, the simulated catalog has five times as many targets as the Kepler catalog).

You can also plot multiple simulated (and Kepler) catalogs simultaneously by simply adding them to the lists:

```
# Load two separate simulated-observed catalogs,
# both of which are in the same 'load_dir',
```

(continues on next page)

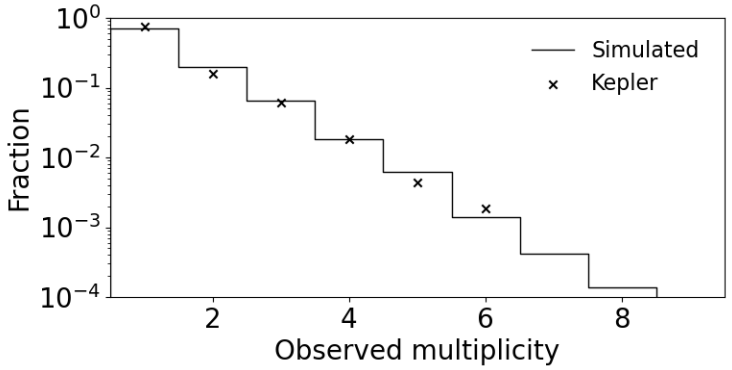


Fig. 3: The observed multiplicity distribution of a simulated catalog compared to the Kepler catalog.

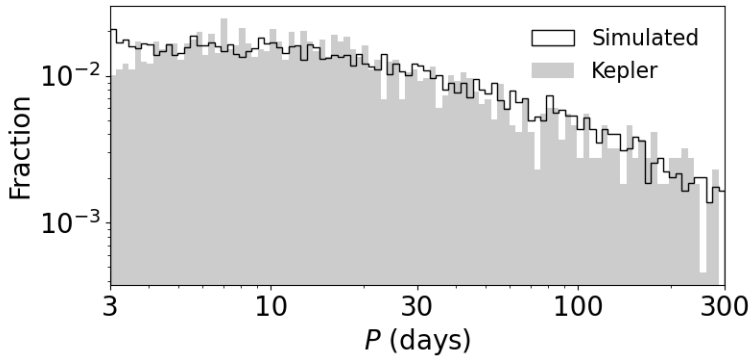


Fig. 4: The observed period distribution of a simulated catalog compared to the Kepler catalog.

(continued from previous page)

```
# with run numbers '1' and '2'.
sss_per_sys1, sss1 = compute_summary_stats_from_cat_obs(file_name_path=load_dir, run_
↪number='1')
sss_per_sys2, sss2 = compute_summary_stats_from_cat_obs(file_name_path=load_dir, run_
↪number='2')

# To plot histograms of the observed orbital periods:
ax = plot_fig_pdf_simple(fig_size, [sss1['P_obs'], sss2['P_obs']], [], x_min=3., x_
↪max=300., log_x=True, log_y=True, c_sim=['k','r'], ls_sim=['-','-'], labels_sim=[
↪'Catalog 1', 'Catalog 2'], xticks_custom=[3,10,30,100,300], xlabel_text=r'$P$ (days)',␣
↪legend=True)

plt.show()
```

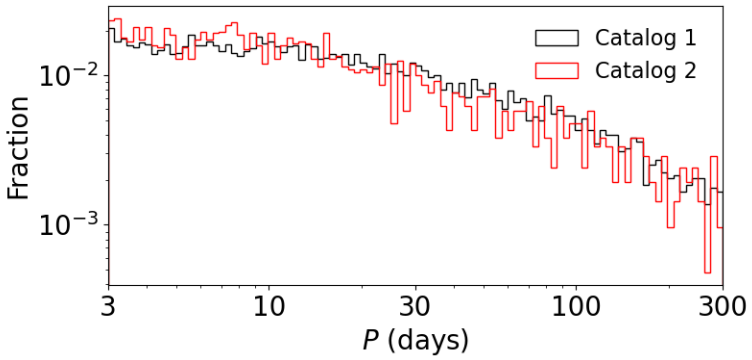


Fig. 5: The observed period distributions of two simulated catalogs.

Note: You also need to pass lists for the optional arguments `c_sim`, `ls_sim`, and `labels_sim` to define the color, line-style, and legend label, respectively, for each catalog that you are plotting!

1.3.3 Plotting CDFs

Similarly, we also provide the following functions for plotting cumulative distribution functions (CDFs):

```
# To plot a CDF of the observed multiplicities:
ax = plot_fig_mult_cdf_simple(fig_size, [sss_per_sys['Mtot_obs']], [ssk_per_sys['Mtot_obs
↪']], y_min=0.6, y_max=1., xlabel_text='Observed planets per system', legend=True)

# To plot a CDF of the observed orbital periods:
ax = plot_fig_cdf_simple(fig_size, [sss['P_obs']], [ssk['P_obs']], x_min=3., x_max=300.,␣
↪log_x=True, xticks_custom=[3,10,30,100,300], xlabel_text=r'$P$ (days)', legend=True)

plt.show()
```

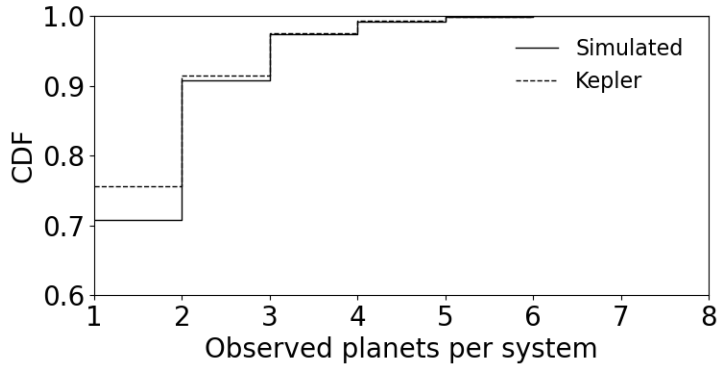


Fig. 6: The observed multiplicity CDFs for a simulated and the Kepler catalog.

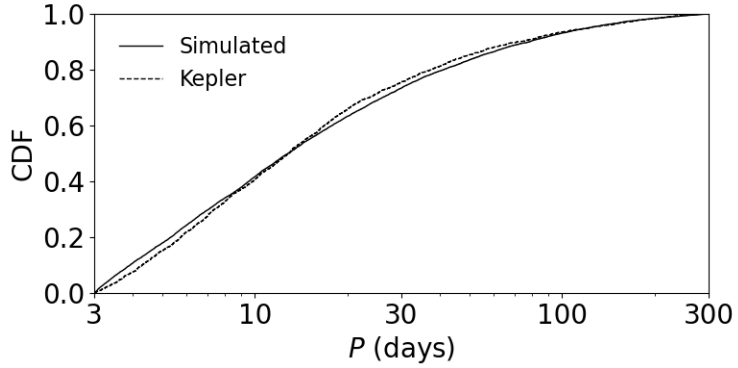


Fig. 7: The observed period CDFs for a simulated and the Kepler catalog.

1.4 Comparing catalogs

1.4.1 Computing KS distances

Plotting histograms and CDFs are a great way of visually comparing different models and the data. However, a more quantitative way of comparing distributions is possible using various distance measures. A widely used and intuitively simple distance function is the two-sample Kolmogorov-Smirnov (KS) distance, which is simply defined as the maximum difference between two CDFs. We have two functions that compute the KS distance, one for discrete distributions and one for continuous distributions:

```
# Load modules and catalogs as before:
from syssimpyplots.general import *
from syssimpyplots.load_sims import *
from syssimpyplots.plot_catalogs import *
from syssimpyplots.compare_kepler import *

load_dir = '/path/to/a/simulated/catalog/' # replace with your path!

N_sim, cos_factor, P_min, P_max, radii_min, radii_max = read_targets_period_radius_
↪bounds(load_dir + 'periods.out')

sss_per_sys, sss = compute_summary_stats_from_cat_obs(file_name_path=load_dir)
ssk_per_sys, ssk = compute_summary_stats_from_Kepler_catalog(P_min, P_max, radii_min,␣
↪radii_max)



# Compute the KS distance between two multiplicity distributions:
d_KS, x_KS = KS_dist_mult(sss_per_sys['Mtot_obs'], ssk_per_sys['Mtot_obs'])

# Compute the KS distance between two period distributions:
d_KS, x_KS = KS_dist(sss['P_obs'], ssk['P_obs'])
```

Both functions return the KS distance (`d_KS`) as well as the x-value corresponding to that distance (`x_KS`, i.e. where the difference in the CDFs is the greatest).

1.4.2 Computing AD distances

Another well known distance is the two-sample Anderson-Darling (AD) distance, which computes an integral over the difference of two CDFs (weighted towards the tails). This measure is more sensitive to differences in the extremes of the distributions. However, we found that samples with vastly different sizes (e.g., numbers of planets) can still produce low AD distances (see Section 2.4.2 of He et al. 2019 for further discussion). Thus, we also define a "modified" AD distance which re-normalizes by (divides out) the constant in front of the integral, $n*m/N$ where n and m are the sample sizes (and $N=n+m$):

```
# Compute the standard AD distance between two period distributions:
d_AD = AD_dist(sss['P_obs'], ssk['P_obs'])

# Compute the modified AD distance between two period distributions:
d_ADmod = AD_mod_dist(sss['P_obs'], ssk['P_obs'])
```

1.4.3 Labeling distances on CDF plots

The function for plotting the CDFs of continuous distributions shown on the previous page also allows for the optional parameter `label_dist`, which can be set to `True` to also compute and print the KS and (modified) AD distances on the figure:

```
# To plot a CDF of the observed orbital periods, with KS and AD (modified) distances␣
↪shown:
ax = plot_fig_cdf_simple(fig_size, [sss['P_obs']], [ssk['P_obs']], x_min=3., x_max=300.,␣
↪log_x=True, xticks_custom=[3,10,30,100,300], xlabel_text=r'$P$ (days)', legend=True,␣
↪label_dist=True)

plt.show()
```

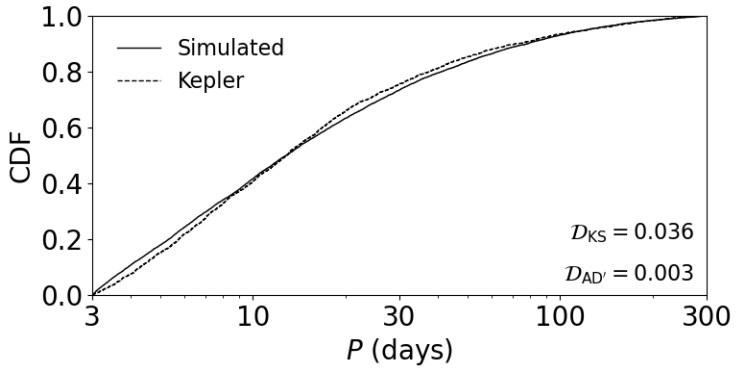


Fig. 8: The observed period CDFs for a simulated and the Kepler catalog.

1.4.4 Summing multiple distances

As we have shown, it is relatively easy to compare two distributions by computing distances between them. It is much harder to compare two *catalogs*, which can be characterized by many distributions and summary statistics, including but not limited to the total number of planets per star and the 1-d ("marginal") distributions of various planetary properties (not to mention correlations in 2-d and higher dimensions!). The simulations are also stochastic, so there are statistical variations in the simulated catalogs and thus the distances, even for the same model with the exact same parameters!

A reasonable approach is to sum the distances computed between multiple marginal distributions that we care about, for example the observed multiplicities, orbital periods, period ratios,... etc. using a "distance function" (see our *publications* for more details about the distance functions we have used). However, some distances are naturally larger or exhibit more variance compared to others, and so we cannot just sum the distance terms directly. For example, how do you compare the change in one distance term (e.g., for the orbital periods) with the change in another (e.g., for the transit duration ratios)? We must weight each term in some manner.

One idea is to simulate many iterations of catalogs of the same model, and compute the distances for each of the marginal distributions between each unique pair of catalogs (see e.g. Section 2.2.2 of He et al. 2021a for more details). The distances obtained in this way give us an idea of the typical distances for each summary statistic when you have a "perfect" model, and can be used as the weights for summing different distance terms. Below is an illustrative example of how you can do this using some pre-computed weights that we have provided.

```
# To load a file with pre-computed weights:
weights_all = load_split_stars_weights_only()
```

(continues on next page)

(continued from previous page)

```
# To pick a specific set of distance terms to include:
dists_include = ['delta_f',
                 'mult_CRPD_r',
                 'periods_KS',
                 'durations_KS',
                 'depths_KS']

# To compute many distance terms:
dists, dists_w = compute_distances_sim_Kepler(sss_per_sys, sss, ssk_per_sys, ssk,␣
↪weights_all['all'], dists_include, N_sim)
```

Here, 'delta_f' refers to the distance for the total number of observed planets relative to the number of stars, 'mult_CRPD_r' refers to the Cressie-Read Power Divergence (CRPD; see *CRPD_dist*) statistic for comparing the multiplicity distributions, and the remaining items refer to the KS distances for the distributions of periods, transit durations, and transit depths. The function *compute_distances_sim_Kepler* will print out the weights and distances for each of these terms, and also compute many other distances to be included in the outputs `dists` (raw distance terms) and `dists_w` (weighted distance terms)!

Caution: The function *compute_distances_sim_Kepler* also enables you to compute the unmodified AD distances if you pass `AD_mod=False` (default is `True`). However, the weights were computed for the `AD_mod=True` only, so the weighted distances should not be used in that case.

1.5 Visualizing catalogs

Another way to visualize the catalogs is to plot *galleries* of their individual systems, where the planets in each system are plotted as points on a line along the orbital period axis (or semi-major axis). This is especially useful for looking at the multi-planet systems and their architectures.

1.5.1 Plotting a simple gallery

Here is a simple example using our function *plot_figs_observed_systems_gallery_from_cat_obs* to plot a gallery of systems with three or more observed planets:

```
# Load modules and catalogs as before:
from syssimpyplots.general import *
from syssimpyplots.load_sims import *
from syssimpyplots.plot_catalogs import *
from syssimpyplots.compare_kepler import *

load_dir = '/path/to/a/simulated/catalog/' # replace with your path!

N_sim, cos_factor, P_min, P_max, radii_min, radii_max = read_targets_period_radius_
↪bounds(load_dir + 'periods.out')

# Load a simulated physical catalog:
sssp_per_sys, sssp = compute_summary_stats_from_cat_phys(file_name_path=load_dir, load_
↪full_tables=True, match_observed=True)
```

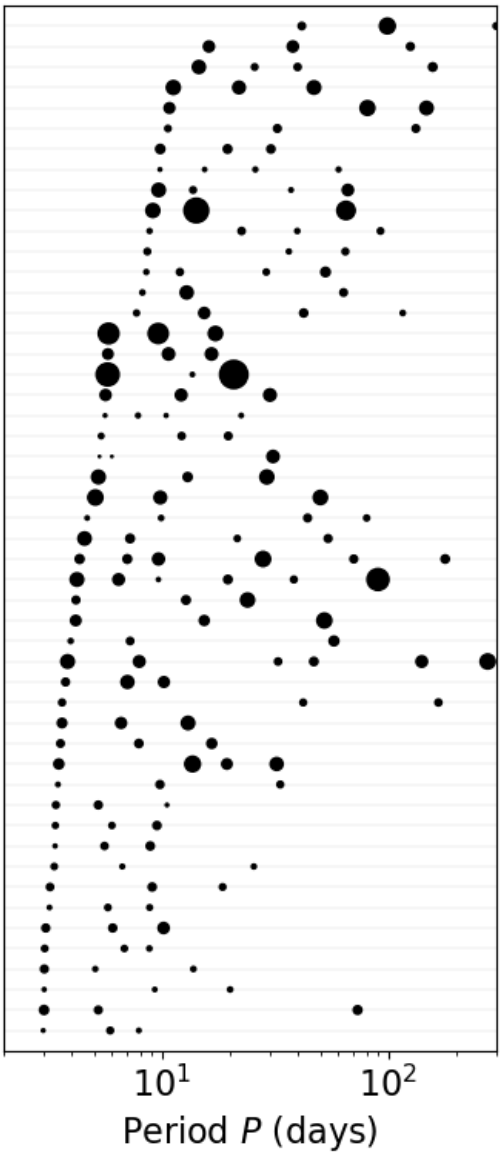
(continues on next page)

(continued from previous page)

```
# Load simulated and Kepler observed catalogs:
sss_per_sys, sss = compute_summary_stats_from_cat_obs(file_name_path=load_dir)
ssk_per_sys, ssk = compute_summary_stats_from_Kepler_catalog(P_min, P_max, radii_min,␣
↪radii_max)



# Plot a gallery of the simulated systems with 3+ observed planets:
plot_figs_observed_systems_gallery_from_cat_obs(sss_per_sys, sort_by='inner', n_min=3,␣
↪color_by='k', max_sys=50, sys_per_fig=50)
plt.show()
```



This function can be applied to both simulated observed (e.g. `sss_per_sys`) and Kepler-observed (e.g. `ssk_per_sys`)

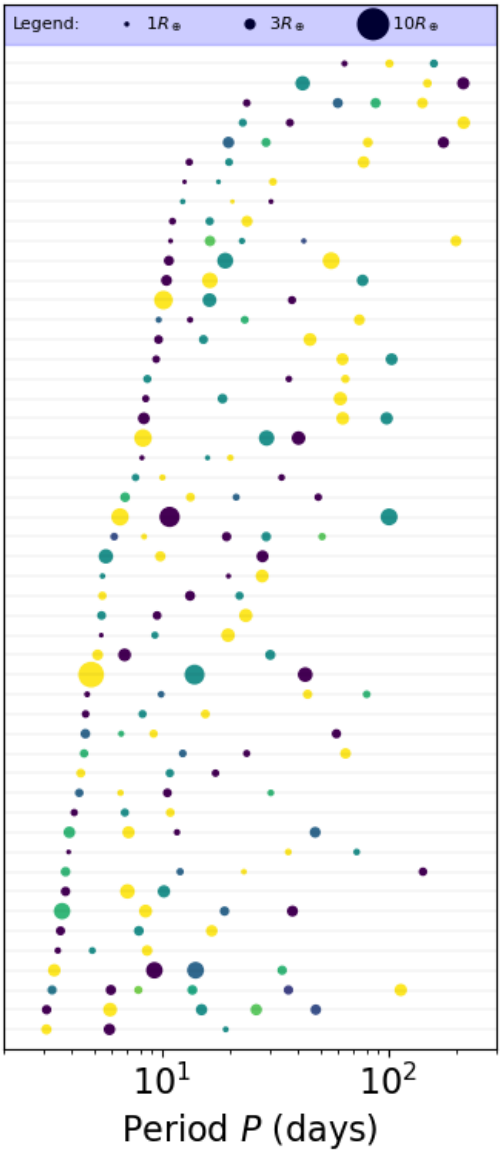
catalogs. In this example, we have selected systems with at least three observed planets (using `n_max=3`), sorted them by the period of their inner-most planet (using `sort_by='inner'`), and included 50 systems in a single figure.

Tip: You can change the number of systems, and/or the number of figures, by varying the `max_sys` and `sys_per_fig` parameters. For example, setting `max_sys=200` and `sys_per_fig=50` will make four figures each showing 50 different systems (if there are enough systems with the required multiplicity in the catalog).

If there are more than `max_sys` systems satisfying the given criteria, the function will randomly sample a subset of these systems.

The `color_by` parameter allows you to choose from a number of color schemes. For example, you can also color the planets by their relative size order in each system (so all the smallest planets are one color, all the 2nd smallest planets are another color, etc.):

```
# Plot a gallery with the planets colored by their size ordering:
plot_figs_observed_systems_gallery_from_cat_obs(sss_per_sys, sort_by='inner', n_min=3,␣
↪color_by='size_order', legend=True, max_sys=50, sys_per_fig=50)
plt.show()
```

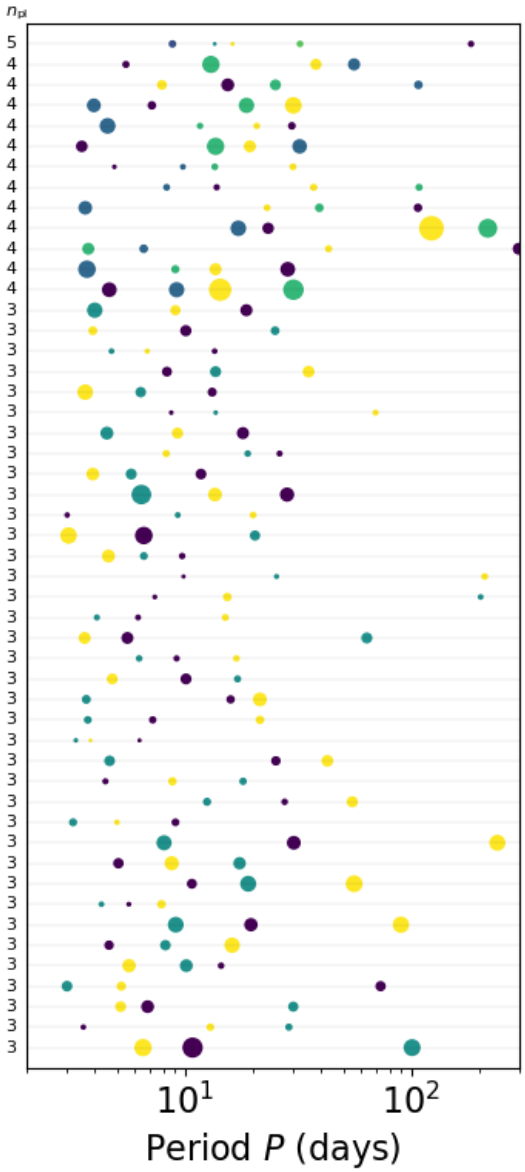


Here, we have also included a legend by setting `legend=True`. This gives us a reference for the relative sizes of the planets!

1.5.2 Sorting and labeling systems

You can also sort by planet multiplicity instead of inner-most period by setting `sort_by='multiplicity'`, and label each system by a given quantity by setting `llabel` and `llabel_text` such as in the following example:

```
# Plot a gallery with the systems sorted and labeled by multiplicity:
plot_figs_observed_systems_gallery_from_cat_obs(sss_per_sys, sort_by='multiplicity', n_
↪min=3, color_by='size_order', llabel='multiplicity', llabel_text=r'$n_{\rm pl}$', max_
↪sys=50, sys_per_fig=50)
plt.show()
```


n_{pl}
5
4
4
4
4
4
4
4
4
4
4
4
4
4
4
3
3
3

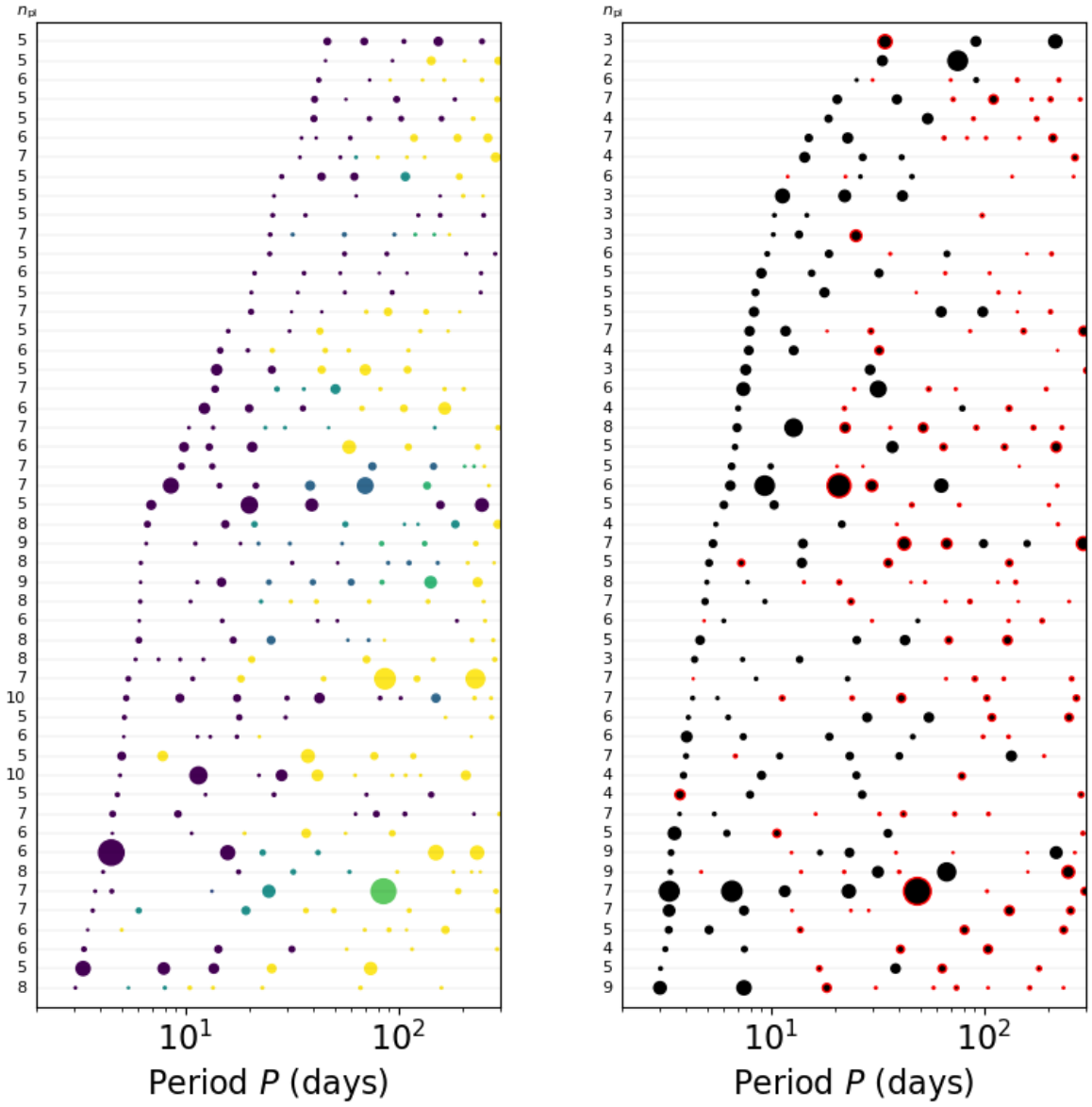
1.5.3 Plotting detected/undetected planets

There is a separate function for plotting galleries of physical systems, *plot_figs_physical_systems_gallery_from_cat_phys*. It provides much of the same functionality and uses mostly the same parameters, except it allows you to filter systems based on both the intrinsic multiplicity (using `n_min` and `n_max`) as well as the observed multiplicity (using `n_det_min` and `n_det_max`). It also contains more options for `color_by`, and has a `mark_det` boolean parameter for whether or not to indicate the detected and undetected planets. The following examples showcase some of these options:

```
# Plot a gallery of physical systems with at least 5 planets:
plot_figs_physical_systems_gallery_from_cat_phys(sssp_per_sys, sssp, sort_by='inner', n_
↪min=5, n_det_min=0, color_by='cluster', mark_det=False, llabel='multiplicity', llabel_
↪text=r'$n_{\rm pl}$', max_sys=50, sys_per_fig=50)

# Plot a gallery of physical systems with at least two detected planets:
plot_figs_physical_systems_gallery_from_cat_phys(sssp_per_sys, sssp, sort_by='inner', n_
↪det_min=2, color_by='k', mark_det=True, llabel='multiplicity', llabel_text=r'$n_{\rm␣
↪pl}$', max_sys=50, sys_per_fig=50)

plt.show()
```



In the left figure, we selected only systems with at least five planets (regardless of whether or not any planets are detected) and colored them by their cluster id's, so planets with the same color were drawn from the same "cluster".

In the right figure, we selected systems with at least two detected planets and marked all undetected planets with red outlines using the `mark_det=True` option.

1.5.4 More customizations

While the two functions demonstrated above (one for plotting observed systems, another for plotting physical systems) provide many useful options for plotting galleries, you may wish to make versions of these figures that are outside the scope of what can be accomplished by these two functions. You can achieve some additional flexibility by using the function `plot_figs_systems_gallery` directly, which is called by both of the previous functions. For example, you may wish to plot along semi-major axes instead of orbital period for the x-axis, use planet masses instead of radii for setting the relative sizes of the points, sort the systems in a special way, provide a custom set of systems, etc… the possibilities are endless!

1.5.5 Other ways of plotting catalogs

There are many other functions in the syssimpyplots.plot_catalogs module for visualizing catalogs, some of which have been used to characterize other correlations in the planetary systems generated by our models.

Warning: Many of these functions are currently undocumented (they do not show up in the detailed API) and are not meant for flexible use – use them at your own risk!

1.6 Detailed API

Here, we provide detailed documentation for *many* of the functions that you may find useful when using SysSimPyPlots. They are organized into the following modules:

- *general.py*

  Includes a collection of fundamental constants and functions, as well as some statistical and other miscellaneous functions.

- *load_sims.py*

  Contains the big functions for loading, processing, and manipulating the simulated catalogs.

- *compare_kepler.py*

  Contains the functions for loading and processing the Kepler catalogs, as well as functions for comparing catalogs (computing distances).

- *plot_catalogs.py*

  Includes many functions for plotting figures for visualizing the catalogs and their properties.

- *plot_params.py*

  Includes some functions for plotting model parameter spaces. Also includes functions for loading files containing many evaluations of model parameters.

- *compute_RVs.py*

  Contains functions for computing, simulating, and fitting radial velocity observations of the simulated planetary systems.

Note: Currently under construction!

1.6.1 general.py

a_from_P(*P*, *Mstar*)

Compute the semi-major axis using Kepler's third law.

Note: Assumes that the planet mass is negligible compared to the stellar mass.

Parameters

- **P** (*float or array[floats]*) – The orbital period (days).
- **Mstar** (*float or array[floats]*) – The stellar mass (solar masses).

Returns
a – The semi-major axis (AU).

Return type
float or array[floats]

P_from_a(*a*, *Mstar*)

Compute the orbital period using Kepler's third law.

Note: Assumes that the planet mass is negligible compared to the stellar mass.

Parameters

- **a** (*float or array[floats]*) – The semi-major axis (AU).
- **Mstar** (*float or array[floats]*) – The stellar mass (solar masses).

Returns
P – The orbital period (days).

Return type
float or array[floats]

M_from_R_rho(*R*, *rho=5.51*)

Compute the planet mass from the radius and constant mean density.

Parameters

- **R** (*float or array[floats]*) – The planet radius (Earth radii).
- **rho** (*float or array[floats], default=5.51*) – The planet density (g/cm^3). Default value is that of Earth.

Returns
M – The planet mass (Earth masses).

Return type
float or array[floats]

rho_from_M_R(*M*, *R*)

Compute the planet mean density (total mass divided by volume).

Parameters

- **M** (*float or array[floats]*) – The planet mass (Earth masses).

- **R** (*float or array[floats]*) – The planet radius (Earth radii).

Returns
rho – The planet mean density (g/cm^3).

Return type
float or array[floats]

tdur_circ(*P*, *Mstar*, *Rstar*)

Compute the transit duration assuming a circular orbit with zero impact parameter.

Parameters

- **P** (*float or array[floats]*) – The orbital period (days).
- **Mstar** (*float or array[floats]*) – The stellar mass (solar masses).
- **Rstar** (*float or array[floats]*) – The stellar radius (solar radii).

Returns
tdur – The transit duration (hrs).

Return type
float or array[floats]

AMD(*mu*, *a*, *e*, *im*)

Compute the AMD (angular momentum deficit) of a planet(s).

Note: Uses units of `G*M_star = 1`.

Parameters

- **mu** (*float or array[floats]*) – The planet/star mass ratio.
- **a** (*float or array[floats]*) – The semi-major axis (AU).
- **e** (*float or array[floats]*) – The orbital eccentricity.
- **im** (*float or array[floats]*) – The inclination relative to the system invariable plane (radians).

Returns
amd_pl – The AMD of the planet(s).

Return type
float or array[floats]

NAMD(*m*, *a*, *e*, *im*)

Compute the NAMD (normalized angular momentum deficit) of a planetary system.

First defined in Chambers 2001, the NAMD of a system is the AMD divided by the angular momentum of the same system with circular and coplanar orbits.

Parameters

- **m** (*float or array[floats]*) – The planet masses (Earth masses).
- **a** (*float or array[floats]*) – The semi-major axes (AU).
- **e** (*float or array[floats]*) – The orbital eccentricities.
- **im** (*float or array[floats]*) – The inclinations relative to the system invariable plane (radians).

Returns
normed_AMD – The NAMD of the system (unitless).

Return type
float

photoevap_boundary_Carrera2018(*R*, *P*)

Evaluate whether a planet is above or below the 'photo-evaporation' valley defined by Eq. 5 in Carrera et al. (2018).

Parameters

- **R** (*float*) – The planet radius (Earth radii).
- **P** (*float*) – The orbital period (days).

Returns
above_boundary – Whether the planet is above (1) or below (0) the boundary.

Return type
1 or 0

incl_mult_power_law_Zhu2018(*k*, *sigma_5=0.8*, *alpha=-3.5*)

Compute the Rayleigh scale of the mutual inclination distribution for a given planet multiplicity using the power-law relation.

Note: Default values for the power-law parameters `sigma_5` and `alpha` are set to best-fit values from Zhu et al. (2018).

Parameters

- **k** (*float or array[floats]*) – The planet multiplicity.
- **sigma_5** (*float or array[floats], default=0.8*) – The normalization (Rayleigh scale of the mutual inclination distribution at `k=5`) (degrees).
- **alpha** (*float or array[floats], default=-3.5*) – The power-law index.

Returns
sigma_k – The Rayleigh scale (degrees) of the mutual inclination distribution for multiplicity `k`.

Return type
float or array[floats]

cdf_normal(*x*, *mu=0.0*, *std=1.0*)

Compute the cumulative distribution function (CDF; i.e. the integral between `-inf` and `x` of) the normal distribution at `x` given a mean and standard deviation.

Note: Can deal with array inputs for `x`, `mu`, and `std` as long as they are the same shape.

Parameters

- **x** (*float or array[floats]*) – The position to evaluate the CDF (between `-inf` and `inf`).
- **mu** (*float or array[floats], default=0.*) – The mean of the normal distribution.

- **std** (*float or array[floats], default=1.*) – The standard deviation of the normal distribution.

Returns
cdf_x – The CDF at `x`.

Return type
float or array[floats]

`cdf_empirical`(*xdata*, *xeval*)

Compute the empirical cumulative distribution function (CDF) at `xeval` given a sample.

Note: Is designed to deal with either scalar or array inputs of `xeval`.

Parameters

- **xdata** (*array*) – The sample of data points.
- **xeval** (*float or array[floats]*) – The position to evaluate the CDF.

Returns
cdf_at_xeval – The CDF at `xeval`.

Return type
float or array[floats]

`calc_f_near_pratios`(*sssp_per_sys*, *pratios=[2.0, 1.5, 1.3333333333333333, 1.25]*, *pratio_width=0.05*)

Compute the intrinsic fraction of planets 'near' a period ratio with another planet for any period ratio in the given list of period ratios.

'Near' is defined as a period ratio between `pr` and `pratio*(1+pratio_width)` for `pratio` in `pratios`.

Note: Defaults to calculating the fraction of all planets near a mean-motion resonance (MMR), defined by `res_ratios`.

Parameters

- **sssp_per_sys** (*dict*) – The dictionary of summary statistics for a physical catalog of planets.
- **pratios** (*list, default=res_ratios*) – The list of period ratios to consider.
- **pratio_width** (*float, default=res_width*) – The fractional width to be considered 'near' a period ratio.

Returns
f_mmr – The fraction of planets being 'near' at least one of the period ratios.

Return type
float

`compute_ratios_adjacent`(*x*, *inverse=False*, *avoid_div_zeros=False*)

Compute an array of the adjacent ratios (e.g. `x[j+1]/x[j]`) of the terms in the input array `x`.

Parameters

- **x** (*array[float]*) – The input values.

- **inverse** (*bool, default=False*) – Whether to take the inverse ratios (i.e. `x[j]/x[j+1]` instead of `x[j+1]/x[j]`).
- **avoid_div_zeros** (*bool, default=False*) – Whether to avoid dividing by zeros. If True, will avoid computing ratios where the denominator is zero, and insert either 'inf' (when the numerator is non-zero) or 'nan' (when the numerator is also zero). If False, the computed ratios will still have the same results, but will produce `RuntimeWarning` messages.

Returns
ratios_adj – The ratios of adjacent pairs in `x`.

Return type
array[float]

Note: Will return an empty array if there are fewer than two elements in the input array.

`compute_ratios_all`(*x*, *inverse=False*, *avoid_div_zeros=False*)

Compute an array of all the unique ratios (`x[j]/x[i]` for all `j > i`) of the terms in the input array `x`.

Parameters

- **x** (*array[float]*) – The input values.
- **inverse** (*bool, default=False*) – Whether to take the inverse ratios (i.e. `x[j]/x[i]` for all `j < i` instead of for all `j > i`).
- **avoid_div_zeros** (*bool, default=False*) – Whether to avoid dividing by zeros (WARNING: not implemented yet!).

Returns
ratios_all – The ratios of all pairs in `x`.

Return type
array[float]

Note: Will return an empty array if there are fewer than two elements in the input array.

`zeta1`(*pratios*)

Compute the zeta statistic for each period ratio in `pratios` as defined in Fabrycky et al. (2014).

`split_colors_per_cdpp_bin`(*stars_cleaned*, *nbins=10*)

Return the indices of the bluer and redder stars that split the stellar sample into two samples based on the histogram of combined differential photometric precision (CDPP) values.

Note: Uses log-uniform bins for the histograms.

Parameters

- **stars_cleaned** (*structured array*) – A table of stars, including columns for 4.5hr CDPP (`rrmscdpp04p5`) and Gaia DR2 bp-rp color (`bp_rp`).
- **nbins** (*int, default=10*) – The number of bins to use.

Returns

- **bins** (*array[floats]*) – The bin edges.

- **i_blue_per_bin** (*array[floats]*) – The indices corresponding to the rows of bluer stars in each bin.
- **i_red_per_bin** (*array[floats]*) – The indices corresponding to the rows of redder stars in each bin.

`linear_fswp_bprp`(*bprp*, *bprp_med*, *fswp_med=0.5*, *slope=0.0*)

Evaluate the fraction of stars with planets (fswp) at a number of bp-rp colors using a linear relation.

Note: The fswp cannot be negative or greater than one.

Parameters

- **bprp** (*array[floats]*) – The bp-rp colors at which to evaluate the fswp.
- **bprp_med** (*float*) – The median bp-rp color (or some normalization point).
- **fswp_med** (*float, default=0.5*) – The fswp at bprp_med (normalization).
- **slope** (*float, default=0.*) – The slope of the linear relation.

Returns

fswp_bprp – The fswp at each bp-rp color.

Return type

array[floats]

`linear_alphaP_bprp`(*bprp*, *bprp_med*, *alphaP_med=0.5*, *slope=0.0*)

Evaluate the period power-law index at a number of bp-rp colors using a linear relation.

Parameters

- **bprp** (*array[floats]*) – The bp-rp colors at which to evaluate the fswp.
- **bprp_med** (*float*) – The median bp-rp color (or some normalization point).
- **alphaP_med** (*float, default=0.5*) – The period power-law index at bprp_med (normalization).
- **slope** (*float, default=0.*) – The slope of the linear relation.

Returns

alphaP_bprp – The period power-law index at each bp-rp color.

Return type

array[floats]

`bin_Nmult`(*Nmult_obs*, *m_geq=5*)

Bins the higher orders of the input multiplicity distribution.

Parameters

- **Nmult_obs** (*array[ints]*) – The multiplicity distribution for consecutive multiplicity orders (i.e. number of systems with 1,2,3,… planets).
- **m_geq** (*int, default=5*) – The multiplicity order at and above which to bin together.

Returns

Nmult_obs – The multiplicity distribution with multiplicity orders greater than or equal to m_geq binned together. For the default m_geq=5, this is the number of systems with 1,2,3,4,5+ planets.

Return type
array[ints]

Shannon_entropy(*p*)

Compute the Shannon entropy.

Note: Assumes base-natural log, although any choice is valid.

Parameters
p (*array[floats]*) – An array of probabilities (values between 0 and 1).

Returns
H – The Shannon entropy.

Return type
float

disequilibrium(*p*)

Compute the disequilibrium of a system.

Parameters
p (*array[floats]*) – An array of probabilities (values between 0 and 1).

Returns
D – The disequilibrium.

Return type
float

LMC_complexity(*K*, *p*)

Compute the Lopez-Ruiz, Mancini, and Calbet (1995) complexity, which is the product of Shannon entropy and disequilibrium.

Parameters

- **K** (*float*) – The normalization constant.
- **p** (*array[floats]*) – An array of probabilities (values between 0 and 1).

Returns
C – The LMC complexity.

Return type
float

Pearson_correlation_coefficient(*x*, *y*)

Compute the Pearson correlation coefficient of two variables.

Spearman_correlation_coefficient(*x*, *y*)

Compute the Spearman correlation coefficient of two variables.

This is the Pearson correlation coefficient applied to the ranks of the variables.

radii_star_ratio(*r*, *Rstar*)

Compute the sum of planet/star radius ratios for a system.

Parameters

- **r** (*array[floats]*) – The planet radii.
- **Rstar** (*float*) – The stellar radius.

Returns
mu – The sum of the planet/star radius ratios.

Return type
float

`partitioning`(*x*)

Compute the 'partitioning' of a quantity `x` for the system.

For example, if `x` is an array of planet masses, this is the 'mass partitioning' quantity.

Parameters
x (*array[floats]*) – The partitions of the quantity.

Returns
Q – The partitioning metric for the quantity (between 0 and 1).

Return type
float

`monotonicity_GF2020`(*x*)

Compute the 'monotonicity' (a measure of the degree of ordering) of a quantity `x` for the system, defined in Gilbert and Fabrycky (2020).

For example, if `x` is an array of planet radii, this is the 'radius monotonicity' quantity.

Parameters
x (*array[floats]*) – The array of quantities.

Returns
M – The monotonicity of the quantity.

Return type
float

`gap_complexity_GF2020`(*P*)

Compute the 'gap complexity' metric of a planetary system, as defined in Gilbert and Fabrycky (2020).

Parameters
P (*array[floats]*) – The array of orbital periods (days). Does not have to be sorted.

Returns
C – The gap complexity of the system.

Return type
float

`Cmax_table_GF2020`(*n*)

Return the value of 'C_max', based on the number of gaps `n` (i.e. the number of planets minus 1) in the system.

This is a normalization used in the gap complexity metric, with the values computed by Gilbert and Fabrycky (2020).

Note: The table of 'C_max' only goes up to `n`=9; for greater numbers of gaps, use the approximation function *syssimpyplots.general.Cmax_approx_GF2020()*.

`Cmax_approx_GF2020`(*n*)

Compute an approximation for the value of 'C_max' based on the number of gaps, defined in Gilbert and Fabrycky (2020).

class MidPointLogNorm(*args: Any, **kwargs: Any)

Set the midpoint of a colormap on a log scale, taken from: https://stackoverflow.com/questions/48625475/python-shifted-logarithmic-colorbar-white-color-offset-to-center (in a post by ImportanceOfBeingErnest)

1.6.2 load_sims.py

read_targets_period_radius_bounds(*file_name*)

Read the number of simulated targets and bounds for the planet periods and radii from a file.

Parameters
file_name (*str*) – The path/name of the file containing a header with simulation parameters.

Returns

- **N_sim** (*int*) – The number of simulated systems.
- **cos_factor** (*float*) – The cosine of the maximum inclination angle (relative to the sky plane) drawn for the reference planes of the simulated systems (between 0 and 1).
- **P_min** (*float*) – The minimum orbital period (days).
- **P_max** (*float*) – The maximum orbital period (days).
- **radii_min** (*float*) – The minimum planet radius (Earth radii).
- **radii_max** (*float*) – The maximum planet radius (Earth radii).

read_sim_params(*file_name*)

Read the simulation parameters from a file and output them in a dictionary.

Parameters
file_name (*str*) – The path/name of the file containing a header with simulation parameters.

Returns
param_vals – A dictionary containing the simulation parameters.

Return type
dict

The full list of possible parameters is defined in `param_symbols` (also exported by this module).

load_cat_phys(*file_name*)

Load a table with all the planets in a simulated physical catalog.

Parameters
file_name (*str*) – The path/name of the file for the physical catalog (should end with 'physical_catalog.csv').

Returns
cat_phys – A table with the physical properties of all the planets.

Return type
structured array

The table has the following columns:

- *target_id*: The index of the star in the simulation (e.g. 1 for the first star) which the planet orbits.
- *star_id*: The index of the star based on where it is in the input stellar catalog.
- *planet_mass*: The planet mass (solar masses).
- *planet_radius*: The planet radius (solar radii).

- *clusterid*: A cluster identifier.
- *period*: The orbital period (days).
- *ecc*: The orbital eccentricity.
- *incl*: The orbital inclination (radians) relative to the sky plane.
- *omega*: The argument of periapsis (radians) relative to the sky plane.
- *asc_node*: The argument of ascending node (radians) relative to the sky plane.
- *mean_anom*: The mean anomaly (radians) relative to the sky plane.
- *incl_invariable*: The orbital inclination (radians) relative to the system invariable plane.
- *asc_node_invariable*: The argument of ascending node (radians) relative to the system invariable plane.
- *star_mass*: The stellar mass (solar masses).
- *star_radius*: The stellar radius (solar radii).

`load_star_phys`(*file_name*)

Load a table of only the stars with planets in a simulated physical catalog.

Parameters
file_name (*str*) – The path/name of the file for the stellar physical catalog (should end with 'physical_catalog_stars.csv').

Returns
star_phys – A table with basic properties of the planet-hosting stars.

Return type
structured array

The table has the following columns:

- *target_id*: The index of the star in the simulation (e.g. 1 for the first star) which the planet orbits.
- *star_id*: The index of the star based on where it is in the input stellar catalog.
- *star_mass*: The stellar mass (solar masses).
- *star_radius*: The stellar radius (solar radii).
- *num_planets*: The number of planets in the system.

`load_planets_stars_phys_separate`(*file_name_path*, *run_number*)

Load individual files with the properties of all the planets and stars in a simulated physical catalog.

Note: Faster than *syssimpyplots.load_sims.load_cat_phys()* for large catalogs, but returns individual lists instead of a single table. Each list is ordered in the same way so the planet properties can be matched to each other.

Parameters

- **file_name_path** (*str*) – The path to the physical catalog.
- **run_number** (*str*) – The run number appended to the file names for the physical catalog.

Returns

- **clusterids_per_sys** (*list[list]*) – The cluster id's of each system.
- **P_per_sys** (*list[list]*) – The orbital periods (days) of each system.

- **radii_per_sys** (*list[list]*) – The planet radii (solar radii) of each system.
- **mass_per_sys** (*list[list]*) – The planet masses (solar masses) of each system.
- **e_per_sys** (*list[list]*) – The orbital eccentricities of each system.
- **inclmut_per_sys** (*list[list]*) – The orbital inclinations (radians) relative to the system invariable plane of each system.
- **incl_per_sys** (*list[list]*) – The orbital inclinations (radians) relative to the sky plane of each system.
- **Mstar_all** (*array[float]*) – The stellar mass (solar masses) of each system.
- **Rstar_all** (*array[float]*) – The stellar radius (solar radii) of each system.

`compute_basic_summary_stats_per_sys_cat_phys`(*clusterids_per_sys*, *P_per_sys*, *radii_per_sys*, *mass_per_sys*, *e_per_sys*, *inclmut_per_sys*, *incl_per_sys*, *Mstar_all*, *Rstar_all*)

Compute the basic summary statistics per system in a physical catalog.

Note: The input parameters should be returned by the function *syssimpyplots.load_sims.load_planets_stars_phys_separate()* and requires the individual lists to be ordered in the same way.

Parameters

- **clusterids_per_sys** (*list[list]*) – A list of lists with the cluster id's for each system.
- **P_per_sys** (*list[list]*) – A list of lists with the orbital periods (days) for each system.
- **radii_per_sys** (*list[list]*) – A list of lists with the planet radii (solar radii) for each system.
- **mass_per_sys** (*list[list]*) – A list of lists with the planet masses (solar masses) for each system.
- **e_per_sys** (*list[list]*) – A list of lists with the orbital eccentricities for each system.
- **inclmut_per_sys** (*list[list]*) – A list of lists with the orbital inclinations (radians) relative to the system invariable plane for each system.
- **incl_per_sys** (*list[list]*) – A list of lists with the orbital inclinations (radians) relative to the sky plane for each system.
- **Mstar_all** (*array[float]*) – The stellar mass (solar masses) for each system.
- **Rstar_all** (*array[float]*) – The stellar radius (solar radii) for each system.

Returns

sssp_per_sys_basic – A dictionary containing planetary and stellar properties for each system.

Return type

dict

The output is a dictionary containing the following fields:

- *Mmax*: The maximum planet multiplicity of any system.
- *Mtot_all*: The planet multiplicity of each system (1-d array).
- *clustertot_all*: The number of planet clusters in each system (1-d array).
- *pl_per_cluster_all*: The number of planets in each cluster (1-d array).

- *P_all*: The orbital periods (days) of each system (2-d array).
- *clusterids_all*: The cluster id's of each system (2-d array).
- *e_all*: The orbital eccentricities of each system (2-d array).
- *inclmut_all*: The orbital inclinations (radians) relative to system invariable plane of each system (2-d array).
- *incl_all*: The orbital inclinations (radians) relative to the sky plane of each system (2-d array).
- *radii_all*: The planet radii (Earth radii) of each system (2-d array).
- *mass_all*: The planet masses (Earth masses) of each system (2-d array).
- *Mstar_all*: The stellar mass (solar masses) of each system (1-d array).
- *Rstar_all*: The stellar radius (solar radii) of each system (1-d array).
- *mu_all*: The planet/star mass ratios of each system (2-d array).
- *a_all*: The semi-major axes (AU) of each system (2-d array).
- *AMD_all*: The AMDs (units of G*Mstar=1) of each system (2-d array).
- *AMD_tot_all*: The total AMD (units of G*Mstar=1) of each system (1-d array).

Warning: For the 2-d arrays, each row is padded with zeros (or negative ones), since different systems have different numbers of planets.

load_cat_phys_separate_and_compute_basic_summary_stats_per_sys(*file_name_path*, *run_number*)

Load a physical catalog and compute the basic summary statistics per system.

Wrapper for the functions *syssimpyplots.load_sims.load_planets_stars_phys_separate()* and *syssimpyplots.load_sims.compute_basic_summary_stats_per_sys_cat_phys()*.

Parameters

- **file_name_path** (*str*) – The path to the physical catalog.
- **run_number** (*str*) – The run number appended to the file names for the physical catalog.

Returns

sssp_per_sys_basic – A dictionary containing planetary and stellar properties for each system. See the documentation for *syssimpyplots.load_sims.compute_basic_summary_stats_per_sys_cat_phys()* for a description of the dictionary fields.

Return type

dict

compute_summary_stats_from_cat_phys(*cat_phys=None*, *star_phys=None*, *file_name_path=None*, *run_number=''*, *load_full_tables=False*, *compute_ratios=<function compute_ratios_adjacent>*, *match_observed=True*)

Compute detailed summary statistics per system in a simulated physical catalog.

Note: This function can be used by either passing a `cat_phys` and `star_phys` for the physical catalog, or by passing a `file_name_path` and `run_number` from which to load the physical catalog. If the latter, will load the individual files with the planet and star properties using *syssimpyplots.load_sims.load_planets_stars_phys_separate()*.

Parameters

- **cat_phys** (*structured array, default=None*) – A table with the physical properties of all the planets.
- **star_phys** (*structured array, default=None*) – A table with the basic properties of the planet-hosting stars.
- **file_name_path** (*str, default=None*) – The path to the physical catalog.
- **run_number** (*str, default=''*) – The run number appended to the file names for the physical catalog.
- **load_full_tables** (*bool, default=False*) – Whether to load full tables of the physical catalogs. Required to be True if also want to match the physical planets to the observed planets.
- **compute_ratios** (*func, default=compute_ratios_adjacent*) – The function to use for computing ratios; can be either *syssimpyplots.general.compute_ratios_adjacent()* or *syssimpyplots.general.compute_ratios_all()*.
- **match_observed** (*bool, default=True*) – Whether to match the physical planets to the observed planets. If True, the output will also contain a field *det_all*.

Returns

- **sssp_per_sys** (*dict*) – A dictionary containing the planetary and stellar properties for each system (2-d and 1-d arrays).
- **sssp** (*dict*) – A dictionary containing the planetary and stellar properties of all planets (1-d arrays).

The outputs are two dictionaries. `sssp_per_sys` contains the following fields:

- *det_all*: The detection flags (1=detected, 0=undetected) of the planets in each system (2-d array). Only returned if *match_observed=True*.
- *Mtot_all*: The number of planets in each system (1-d array).
- *clusterids_all*: The cluster id's of each system (2-d array).
- *P_all*: The orbital periods (days) of each system (2-d array).
- *a_all*: The semi-major axes (AU) of each system (2-d array).
- *radii_all*: The planet radii (Earth radii) of each system (2-d array).
- *mass_all*: The planet masses (Earth masses) of each system (2-d array).
- *mu_all*: The planet/star mass ratios of each system (2-d array).
- *e_all*: The orbital eccentricities of each system (2-d array).
- *inclmut_all*: The orbital inclinations (radians) relative to the system invariable plane of each system (2-d array).
- *incl_all*: The orbital inclinations (radians) relative to the sky plane of each system (2-d array).
- *AMD_all*: The AMDs (units of G*Mstar=1) of each system (2-d array).
- *Rm_all*: The period ratios of each system (2-d array).
- *radii_ratio_all*: The planet radius ratios of each system (2-d array).
- *N_mH_all*: The planet spacings in mutual Hill radii of each system (2-d array).

- *dynamical_mass*: The 'dynamical mass' of each system (1-d array).
- *radii_partitioning*: The 'radius partitioning' of each multi-planet system (1-d array).
- *radii_monotonicity*: The 'radius monotonicity' of each multi-planet system (1-d array).
- *gap_complexity*: The 'gap complexity' of each system with 3+ planets (1-d array).

`sssp` contains the following fields:

- *Mstar_all*: The stellar mass (solar masses) of each system (1-d array).
- *Rstar_all*: The stellar radius (solar radii) of each system (1-d array).
- *clustertot_all*: The number of planet clusters in each system (1-d array).
- *AMD_tot_all*: The total AMD (units of G*Mstar=1) of each system (1-d array).
- *pl_per_cluster_all*: The number of planets in each cluster (1-d array).
- *P_all*: The orbital periods (days) of all planets (1-d array).
- *radii_all*: The radii (Earth radii) of all planets (1-d array).
- *mass_all*: The masses (Earth masses) of all planets (1-d array).
- *e_all*: The orbital eccentricities of all planets (1-d array).
- *inclmut_all*: The orbital inclinations (radians) of all planets relative to the system invariable planes (1-d array).
- *incl_all*: The orbital inclinations (radians) of all planets relative to the sky plane (1-d array).
- *radii_above_all*: The radii (Earth radii) of all planets above the photo-evaporation boundary* (1-d array).
- *radii_below_all*: The radii (Earth radii) of all planets below the photo-evaporation boundary* (1-d array).
- *Rm_all*: The orbital period ratios (1-d array).
- *radii_ratio_all*: The planet radii ratios (1-d array).
- *N_mH_all*: The planet spacings in mutual Hill radii (1-d array).
- *radii_ratio_above_all*: The planet radii ratios for all planets above the photo-evaporation boundary* (1-d array).
- *radii_ratio_below_all*: The planet radii ratios for all planets below the photo-evaporation boundary* (1-d array).
- *radii_ratio_across_all*: The planet radii ratios for all planets across the photo-evaporation boundary* (1-d array).

Note: *The photo-evaporation boundary is defined by the function *syssimpyplots.general.photoevap_boundary_Carrera2018()*.

Warning: For the 2-d arrays, each row is padded with zeros (or negative ones), since different systems have different numbers of planets.

load_cat_obs(*file_name*)

Load a table with all the planets in a simulated observed catalog.

Parameters
file_name (*str*) – The path/name of the file for the observed catalog (should end with 'observed_catalog.csv').

Returns
cat_obs – A table with the observed properties of all the planets.

Return type
structured array

The table has the following columns:

- *target_id*: The index of the star in the simulation (e.g. 1 for the first star) which the planet orbits.
- *star_id*: The index of the star based on where it is in the input stellar catalog.
- *period*: The observed orbital period (days).
- *period_err*: The uncertainty in observed orbital period (days).
- *depth*: The observed transit depth.
- *depth_err*: The uncertainty in observed transit depth.
- *duration*: The observed transit duration (days).
- *duration_err*: The uncertainty in observed transit duration (days).
- *star_mass*: The stellar mass (solar masses).
- *star_radius*: The stellar radius (solar radii).

load_star_obs(*file_name*)

Load a table of only the stars with observed planets in a simulated observed catalog.

Parameters
file_name (*str*) – The path/name of the file for the stellar physical catalog (should end with 'observed_catalog_stars.csv')

Returns
star_obs – A table with the basic properties of the observed planet-hosting stars.

Return type
structured array

The table has the same columns as those returned by the function *syssimpyplots.load_sims.load_star_phys()*.

load_planets_stars_obs_separate(*file_name_path*, *run_number*)

Load individual files with the properties of all the planets and stars in a simulated observed catalog.

Note: Faster than *syssimpyplots.load_sims.load_cat_obs()* for large catalogs, but returns individual lists instead of a single table. Each list is ordered in the same way (low to high observed multiplicity) so the planet properties can be matched to each other.

Parameters

- **file_name_path** (*str*) – The path to the observed catalog.
- **run_number** (*str*) – The run number appended to the file names for the observed catalog.

Returns

- **P_per_sys** (*list[list]*) – The observed orbital periods (days) of each system.
- **D_per_sys** (*list[list]*) – The observed transit depths of each system.
- **tdur_per_sys** (*list[list]*) – The observed transit durations (days) of each system.
- **Mstar_per_sys** (*array[float]*) – The stellar mass (solar masses) of each system.
- **Rstar_per_sys** (*array[float]*) – The stellar radius (solar radii) of each system.

`count_planets_from_loading_cat_obs_stars_only`(*file_name_path=None*, *run_number=''*, *Rstar_min=0.0*, *Rstar_max=10.0*, *Mstar_min=0.0*, *Mstar_max=10.0*, *teff_min=0.0*, *teff_max=10000.0*, *bp_rp_min=-5.0*, *bp_rp_max=5.0*)

Count the number of observed planets in each system (and the resulting observed multiplicity distribution), given a set of stellar cuts.

Note: Loads an 'observed_catalog_stars.csv' file (using *syssimpyplots.load_sims.load_star_obs()*) and a table of cleaned Kepler target stars (using *syssimpyplots.compare_kepler.load_Kepler_stars_cleaned()*).

Parameters

- **file_name_path** (*str, default=None*) – The path to the observed catalog.
- **run_number** (*str, default=''*) – The run number appended to the file names for the observed catalog.
- **Rstar_min** (*float, default=0.*) – The minimum stellar radius (solar radii) to include in the sample.
- **Rstar_max=** (*float, default=10.*) – The maximum stellar radius (solar radii) to include in the sample.
- **Mstar_min** (*float, default=0.*) – The minimum stellar mass (solar masses) to include in the sample.
- **Mstar_max** (*float, default=10.*) – The maximum stellar mass (solar masses) to include in the sample.
- **teff_min** (*float, default=0.*) – The minimum stellar effective temperature (K) to include in the sample.
- **teff_max** (*float, default=1e4*) – The maximum stellar effective temperature (K) to include in the sample.
- **bp_rp_min** (*float, default=-5.*) – The minimum Gaia DR2 bp-rp color to include in the sample.
- **bp_rp_max** (*float, default=5.*) – The maximum Gaia DR2 bp-rp color to include in the sample.

Returns

- **Mtot_obs** (*array[int]*) – The number of observed planets in each system.
- **Nmult_obs** (*array[int]*) – The observed multiplicity distribution (number of observed systems at each multiplicity order).

compute_summary_stats_from_cat_obs(*cat_obs=None*, *star_obs=None*, *file_name_path=None*, *run_number=''*, *P_min=0.0*, *P_max=300.0*, *Rstar_min=0.0*, *Rstar_max=10.0*, *Mstar_min=0.0*, *Mstar_max=10.0*, *teff_min=0.0*, *teff_max=10000.0*, *bp_rp_min=-5.0*, *bp_rp_max=5.0*, *i_stars_custom=None*, *compute_ratios=<function compute_ratios_adjacent>*)

Compute detailed summary statistics per system in a simulated observed catalog.

Note: This function can be used by either passing a `cat_obs` and `star_obs` for the observed catalog, or by passing a `file_name_path` and `run_number` from which to load the observed catalog. If the latter, will load the individual files with the planet and star properties using *syssimpyplots.load_sims.load_planets_stars_obs_separate()*.

Parameters

- **cat_obs** (*structured array, default=None*) – A table with the observed properties of the planets.
- **star_obs** (*structured array, default=None*) – A table with the basic properties of the observed planet-hosting stars.
- **file_name_path** (*str, default=None*) – The path to the observed catalog.
- **run_number** (*str, default=''*) – The run number appended to the file names for the observed catalog.
- **P_min** (*float, default=0.*) – The minimum orbital period to include in the sample.
- **P_max** (*float, default=300.*) – The maximum orbital period to include in the sample.
- **Rstar_min** (*float, default=0.*) – The minimum stellar radius (solar radii) to include in the sample.
- **Rstar_max** (*float, default=10.*) – The maximum stellar radius (solar radii) to include in the sample.
- **Mstar_min** (*float, default=0.*) – The minimum stellar mass (solar masses) to include in the sample.
- **Mstar_max** (*float, default=10.*) – The maximum stellar mass (solar masses) to include in the sample.
- **teff_min** (*float, default=0.*) – The minimum stellar effective temperature (K) to include in the sample.
- **teff_max** (*float, default=10.*) – The maximum stellar effective temperature (K) to include in the sample.
- **bp_rp_min** (*float, default=-5.*) – The minimum Gaia DR2 bp-rp color to include in the sample.
- **bp_rp_max** (*float, default=5.*) – The maximum Gaia DR2 bp-rp color to include in the sample.
- **i_stars_custom** (*array[int], default=None*) – An array of indices for the stars in the Kepler stellar catalog to be included in the sample.
- **compute_ratios** (*func, default=compute_ratios_adjacent*) – The function to use for computing ratios; can be either *syssimpyplots.*

general.compute_ratios_adjacent() or *syssimpyplots.general.compute_ratios_all()*.

Returns

- **sss_per_sys** (*dict*) – A dictionary containing the planetary and stellar properties for each observed system (2-d and 1-d arrays).
- **sss** (*dict*) – A dictionary containing the planetary and stellar properties of all observed planets (1-d arrays).

The outputs are two dictionaries. `sss_per_sys` contains the following fields:

- *Mstar_obs*: The stellar mass (solar masses) of each system (1-d array).
- *Rstar_obs*: The stellar radius (solar radii) of each system (1-d array).
- *teff_obs*: The stellar effective temperature (K) of each system (1-d array).
- *bp_rp_obs*: The Gaia DR2 bp-rp color (mag) of each system (1-d array).
- *e_bp_rp_obs*: The extinction in bp-rp color interpolated from a model, of each system (1-d array).
- *cdpp4p5_obs*: The 4.5 hr duration combined differential photometric precision of each system (1-d array).
- *Mtot_obs*: The observed number of planets in each system (1-d array).
- *P_obs*: The observed orbital periods (days) of each system (2-d array).
- *D_obs*: The observed transit depths of each system (2-d array).
- *tdur_obs*: The observed transit durations of each system (2-d array).
- *tdur_tcirc_obs*: The observed transit durations normalized by the durations of the circular orbits of each system (2-d array).
- *radii_obs*: The observed planet radii (Earth radii) of each system (2-d array).
- *Rm_obs*: The observed period ratios of each system (2-d array).
- *D_ratio_obs*: The observed transit depth ratios of each system (2-d array).
- *xi_obs*: The observed period-normalized transit duration ratios ('xi' values) of each system (2-d array).
- *xi_res_obs*: The observed 'xi' values of planet-pairs near a resonance* (2-d array).
- *xi_res32_obs*: The observed 'xi' values of planet-pairs near the 3:2 resonance (2-d array).
- *xi_res21_obs*: The observed 'xi' values of planet-pairs near the 2:1 resonance (2-d array).
- *xi_nonres_obs*: The observed 'xi' values of planet-pairs not near any resonances* (2-d array).
- *radii_star_ratio*: The observed sum of planet/star radius ratios of each system (1-d array).
- *radii_partitioning*: The observed 'radius partitioning' of each multi-planet system (1-d array).
- *radii_monotonicity*: The observed 'radius monotonicity' of each multi-planet system (1-d array).
- *gap_complexity*: The observed 'gap complexity' of each system with 3+ planets (1-d array).

`sss` contains the following fields:

- *Mstar_obs*: The stellar mass (solar masses) of each system, repeated for each planet in the system (1-d array).
- *Rstar_obs*: The stellar radius (solar radii) of each system, repeated for each planet in the system (1-d array).
- *teff_obs*: The stellar effective temperature (K) of each system, repeated for each planet in the system (1-d array).

- *bp_rp_obs*: The Gaia DR2 bp-rp color (mag) of each system, repeated for each planet in the system (1-d array).
- *e_bp_rp_obs*: The extinction in bp-rp color interpolated from a model, of each system, repeated for each planet in the system (1-d array).
- *cdpp4p5_obs*: The 4.5 hr duration combined differential photometric precision of each system, repeated for each planet in the system (1-d array).
- *Nmult_obs*: The observed multiplicity distribution (1-d array).
- *P_obs*: The observed periods (days) of all observed planets (1-d array).
- *D_obs*: The observed transit depths of all observed planets (1-d array).
- *tdur_obs*: The observed transit duration (hrs) of all observed planets (1-d array).
- *tdur_tcirc_obs*: The observed transit durations normalized by the durations of the circular orbits, of all observed planets (1-d array).
- *tdur_tcirc_1_obs*: The observed transit durations normalized by the durations of the circular orbits, of all observed single planets (1-d array).
- *tdur_tcirc_2p_obs*: The observed transit durations normalized by the durations of the circular orbits, of all observed multi-planets (1-d array).
- *radii_obs*: The observed radii (Earth radii) of all observed planets (1-d array).
- *D_above_obs*: The observed transit depths of all planets above the photo-evaporation boundary** (1-d array).
- *D_below_obs*: The observed transit depths of all planets below the photo-evaporation boundary** (1-d array).
- *Rm_obs*: The observed period ratios (1-d array).
- *D_ratio_obs*: The observed transit depth ratios (1-d array).
- *xi_obs*: The observed 'xi' values (1-d array).
- *xi_res_obs*: The observed 'xi' values of all planet-pairs near a resonance* (1-d array).
- *xi_res32_obs*: The observed 'xi' values of all planet-pairs near the 3:2 resonance (1-d array).
- *xi_res21_obs*: The observed 'xi' values of all planet-pairs near the 2:1 resonance (1-d array).
- *xi_nonres_obs*: The observed 'xi' values of all planet-pairs not near any resonances* (1-d array).
- *D_ratio_above_obs*: The observed transit depth ratios of all planets above the photo-evaporation boundary** (1-d array).
- *D_ratio_below_obs*: The observed transit depth ratios of all planets below the photo-evaporation boundary** (1-d array).
- *D_ratio_across_obs*: The observed transit depth ratios of all planets across the photo-evaporation boundary** (1-d array).

Note: *As defined by `res_ratios` and `res_width` in *general.py*.

**The photo-evaporation boundary defined by the function *syssimpyplots.general.photoevap_boundary_Carrera2018()*.

Warning:

1. For the 2-d arrays, each row is padded with zeros (or negative ones), since different systems have different numbers of observed planets.
2. The observed transit durations (*tdur_obs*, and thus also fields involving *tdur_tcirc_obs* and *xi_obs*) can be zero!

combine_sss_or_sssp_per_sys(*s1*, *s2*)

Combine two dictionaries of summary statistics (e.g., two simulated catalogs).

Note: Requires both dictionaries to have the exact same fields.

Parameters

- **s1** (*dict*) – A dictionary of summary statistics.
- **s2** (*dict*) – A dictionary of summary statistics.

Returns

scombined – The combined dictionary of summary statistics.

Return type

dict

load_cat_phys_multiple_and_compute_combine_summary_stats(*file_name_path*, *run_numbers=range(1, 11)*, *load_full_tables=False*, *compute_ratios=<function compute_ratios_adjacent>*, *match_observed=True*)

Load multiple simulated physical catalogs and compute detailed summary statistic for all the catalogs combined.

Parameters

- **file_name_path** (*str*) – The path to the physical catalogs.
- **run_number** (*range, default=range(1,11)*) – The range of catalog run numbers over which we want to load and combine.
- **load_full_tables** (*bool, default=False*) – Whether to load full tables of the physical catalogs. Required to be True if also want to match the physical planets to the observed planets.
- **compute_ratios** (*func, default=compute_ratios_adjacent*) – The function to use for computing ratios; can be either *syssimpyplots.general.compute_ratios_adjacent()* or *syssimpyplots.general.compute_ratios_all()*.
- **match_observed** (*bool, default=True*) – Whether to match the physical planets to the observed planets. If True, the output will also contain a field *det_all*.

Returns

- **sssp_per_sys** (*dict*) – A dictionary containing the planetary and stellar properties for each system (2-d and 1-d arrays).
- **sssp** (*dict*) – A dictionary containing the planetary and stellar properties of all planets (1-d arrays).

The fields of `sssp_per_sys` and `sssp` are the same as those returned by *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.

1.6.3 compare_kepler.py

`load_Kepler_planets_cleaned()`

Load a table of the Kepler objects of interest (KOIs) from a CSV file.

Returns
planets_cleaned – A table with the properties of the KOIs.

Return type
structured array

The table has the following columns:

- *kepid*: The Kepler ID.
- *KOI*: The KOI number.
- *koi_disposition*: The disposition of the KOI.
- *koi_pdisposition*: (TODO: TBD).
- *koi_score*: The disposition score (between 0 and 1).
- *P*: The orbital period (days).
- *t_D*: The transit duration (hrs).
- *depth*: The transit depth (ppm).
- *Rp*: The planet radius (Earth radii).
- *teff*: The stellar effective temperature (K).
- *logg*: The log surface gravity of the star.
- *Rstar*: The stellar radius (solar radii).
- *Mstar*: The stellar mass (solar masses).

`load_Kepler_stars_cleaned()`

Load a table of Kepler target stars from a CSV file.

Returns
stars_cleaned – A table with the properties of the Kepler target stars.

Return type
structured array

The table has the following columns:

- *kepid*: The Kepler ID.
- *mass*: The stellar mass (solar masses).
- *radius*: The stellar radius (solar radii).
- *teff*: The stellar effective temperature (K).
- *bp_rp*: The Gaia DR2 bp-rp color (mag).
- *lum_val*: The luminosity (solar luminosities).
- *e_bp_rp_interp*: The extinction in bp-rp color (mag) interpolated from a model/binning.

- *e_bp_rp_true*: The extinction in bp-rp color (mag) as given in the Gaia DR2 catalog.
- *rrmscdpp04p5*: The root-mean-square combined differential photometric precision (CDPP) for 4.5 hr durations (ppm).

compute_summary_stats_from_Kepler_catalog(*P_min*, *P_max*, *radii_min*, *radii_max*, *Rstar_min=0.0*, *Rstar_max=10.0*, *Mstar_min=0.0*, *Mstar_max=10.0*, *teff_min=0.0*, *teff_max=10000.0*, *bp_rp_min=-5.0*, *bp_rp_max=5.0*, *i_stars_custom=None*, *compute_ratios=<function compute_ratios_adjacent>*)

Compute detailed summary statistics per system in the Kepler catalog.

Parameters

- **P_min** (*float*) – The minimum orbital period (days) to be included.
- **P_max** (*float*) – The maximum orbital period (days) to be included.
- **radii_min** (*float*) – The minimum planet radius (Earth radii) to be included.
- **radii_max** (*float*) – The maximum planet radius (Earth radii) to be included.
- **Rstar_min** (*float, default=0.*) – The minimum stellar radius (solar radii) to be included.
- **Rstar_max** (*float, default=10.*) – The maximum stellar radius (solar radii) to be included.
- **Mstar_min** (*float, default=0.*) – The minimum stellar mass (solar masses) to be included.
- **Mstar_max** (*float, default=10.*) – The maximum stellar mass (solar masses) to be included.
- **teff_min** (*float, default=0.*) – The minimum stellar effective temperature (K) to be included.
- **teff_max** (*float, default=0.*) – The maximum stellar effective temperature (K) to be included.
- **bp_rp_min** (*float, default=-5.*) – The minimum Gaia DR2 bp-rp color (mag) to be included.
- **bp_rp_max** (*float, default=5.*) – The maximum Gaia DR2 bp-rp color (mag) to be included.
- **i_stars_custom** (*array[int], default=None*) – An array of indices for the stars in the Kepler stellar catalog to be included.
- **compute_ratios** (*func, default=compute_ratios_adjacent*) – The function to use for computing ratios; can be either *syssimpyplots.general.compute_ratios_adjacent()* or *syssimpyplots.general.compute_ratios_all()*.

Returns

- **ssk_per_sys** (*dict*) – A dictionary containing the planetary and stellar properties for each observed system (2-d and 1-d arrays).
- **ssk** (*dict*) – A dictionary containing the planetary and stellar properties of all observed planets (1-d arrays).

The fields of `ssk_per_sys` and `ssk` are the same as those returned by *syssimpyplots.load_sims.compute_summary_stats_from_cat_obs()*.

CRPD_dist(*En*, *On*)

Compute the Cressie-Read Power Divergence (CRPD) statistic for observed planet multiplicity distributions.

Warning: Can potentially return negative values for extreme/edge cases!

Parameters

- **En** (*array[int]*) – The 'expected' (i.e. simulated) numbers of total systems with 1,2,3,... observed planets.
- **On** (*array[int]*) – The 'observed' (i.e. actual Kepler) numbers of total systems with 1,2,3,... observed planets.

Returns

rho – The CRPD statistic.

Return type

float

KS_dist_mult(*x1*, *x2*)

Compute the two-sample Kolmogorov-Smirnov (KS) distance between two discrete distributions taking on integer values.

Parameters

- **x1** (*array[int]*) – A sample of integers.
- **x2** (*array[int]*) – A sample of integers.

Returns

- **KS_dist** (*float*) – The KS distance between the two distributions (i.e. the greatest distance between the cumulative distributions).
- **KS_x** (*float*) – The x-value that corresponds to the greatest distance between the two cumulative distributions.

KS_dist(*x1*, *x2*)

Compute the two-sample Kolmogorov-Smirnov (KS) distance between two continuous distributions (i.e. no repeat values).

Parameters

- **x1** (*array[float]*) – A sample of real values.
- **x2** (*array[float]*) – A sample of real values.

Returns

- **KS_dist** (*float*) – The KS distance between the two distributions (i.e. the greatest distance between the cumulative distributions).
- **KS_x** (*float*) – The x-value that corresponds to the greatest distance between the two cumulative distributions.

AD_dist(*x1*, *x2*)

Compute the two-sample Anderson-Darling (AD) distance between two continuous distributions.

Implements Equation 1.2 of A. N. Pettitt (1976).

Note: Returns `np.inf` if there are not enough points (less than two) in either `x1` or `x2` for computing the AD distance.

Parameters

- **x1** (*array[float]*) – A sample of real values.
- **x2** (*array[float]*) – A sample of real values.

Returns
AD_dist – The AD distance between the two distributions.

Return type
float

`AD_dist2`(*x1*, *x2*)

Compute the two-sample Anderson-Darling (AD) distance between two continuous distributions.

Implements Equation 3 of Scholz & Stephens (1987). Tested to be equivalent to *syssimpyplots.compare_kepler.AD_dist()*.

Note: Returns `np.inf` if there are not enough points (less than two) in either `x1` or `x2` for computing the AD distance.

Parameters

- **x1** (*array[float]*) – A sample of real values.
- **x2** (*array[float]*) – A sample of real values.

Returns
AD_dist – The AD distance between the two distributions.

Return type
float

`AD_mod_dist`(*x1*, *x2*)

Compute a modified version of the two-sample Anderson-Darling (AD) distance between two continuous distributions.

Equivalent to the AD distance (implemented by *syssimpyplots.compare_kepler.AD_dist()* and *syssimpyplots.compare_kepler.AD_dist2()*) without the factor of 'n*m/N' in front of the integral, where 'n' and 'm' are the sample sizes (and 'N=n+m' is the combined sample size).

Note: Returns `np.inf` if there are not enough points (less than two) in either `x1` or `x2` for computing the AD distance.

Parameters

- **x1** (*array[float]*) – A sample of real values.
- **x2** (*array[float]*) – A sample of real values.

Returns
AD_dist – The (modified) AD distance between the two distributions.

Return type
float

`load_split_stars_model_evaluations_and_weights`(*file_name*)

Load a file containing the distances from many evaluations of the same model, and compute the weights for each distance term.

Parameters
file_name (*str*) – The path/name of the file containing the distances of many model evaluations.

Returns

- **Nmult_evals** (*dict*) – A dictionary containing an array of observed planet multiplicity distributions for each model evaluation, for each stellar sample (*all*, *bluer*, and *redder* fields).
- **d_all_keys_evals** (*dict*) – A dictionary containing an array of distance term names (strings) for each model evaluation, for each stellar sample.
- **d_all_vals_evals** (*dict*) – A dictionary containing an array of distances (corresponding to the distance term names) for each model evaluation, for each stellar sample.
- **weights_all** (*dict*) – A dictionary containing a dictionary for the weights corresponding to each distance term, for each stellar sample.

Note: The *bluer* and *redder* samples split the stellar sample into two equal sized samples of stars below and above the median Gaia DR2 bp-rp color, respectively.

Warning: Currently returns empty arrays in the *Nmult_evals* dictionary.

`load_split_stars_weights_only`()

Compute the weights for each distance term.

Wrapper to return just the weights from the function *syssimpyplots.compare_kepler.load_split_stars_model_evaluations_and_weights()*.

`compute_total_weighted_dist`(*weights*, *dists*, *dists_w*, *dists_include=[]*)

Compute the total weighted distance including a number of distance terms.

Also prints out the individual distance terms, their weights, and their unweighted and weighted distances.

Parameters

- **weights** (*dict*) – The dictionary containing the weights for to each distance term.
- **dists** (*dict*) – The dictionary containing the individual distance terms.
- **dists_w** (*dict*) – The dictionary containing the individual weighted distance terms.
- **dists_include** (*list[str], default=[]*) – The list of distance terms (strings) to include in the sum.

Returns
tot_dist_w – The total weighted distance of the included distance terms.

Return type
float

compute_distances_sim_Kepler(*sss_per_sys*, *sss*, *ssk_per_sys*, *ssk*, *weights*, *dists_include*, *N_sim*, *cos_factor=1.0*, *AD_mod=True*, *print_dists=True*)

Compute weighted and unweighted distances for a large collection of distance terms.

Parameters

- **sss_per_sys** (*dict*) – A dictionary of summary statistics per observed system in a simulated observed catalog.
- **sss** (*dict*) – A dictionary of summary statistics for all observed planets in a simulated observed catalog.
- **ssk_per_sys** (*dict*) – A dictionary of summary statistics per observed system in the Kepler catalog.
- **ssk** (*dict*) – A dictionary of summary statistics for observed all planets in the Kepler catalog.
- **weights** (*dict*) – A dictionary of the weights corresponding to each distance term.
- **dists_include** (*list[str]*) – A list of distance terms (strings) to be printed.
- **N_sim** (*int*) – The number of target stars (i.e. simulated systems) in the simulated catalog.
- **cos_factor** (*float, default=1.*) – The cosine of the maximum inclination angle (relative to the sky plane) drawn for the reference planes of the simulated systems (between 0 and 1).
- **AD_mod** (*bool, default=True*) – Whether to compute the modified AD distance (*syssimpyplots.compare_kepler.AD_mod_dist()*, if True) or the standard AD distance (*syssimpyplots.compare_kepler.AD_dist()*, if False).
- **print_dists** (*bool, default=True*) – Whether to print the distances corresponding to the terms in *dists_include*. If True, also prints the total numbers of observed planets and planet pairs in the simulated and Kepler catalogs.

Returns

- **dists** (*dict*) – A dictionary containing all the various (unweighted) distance terms.
- **dists_w** (*dict*) – A dictionary containing all the various (weighted) distance terms.

Note: The distance terms computed and included in `dists` and `dists_w` are not limited to the terms in `dists_include`.

1.6.4 plot_catalogs.py

Note: Currently under construction!

---

setup_fig_single(*fig_size*, *left*, *bottom*, *right*, *top*)

Set up a single-panel figure using matplotlib.gridspec.GridSpec.

Parameters

- **fig_size** (*tuple*) – The figure size, e.g. (16,8).
- **left** (*float*) – The left margin of the panel (between 0 and 1).
- **bottom** (*float*) – The bottom margin of the panel (between 0 and 1).

- **right** (*float*) – The right margin of the panel (between 0 and 1).
- **top** (*float*) – The top margin of the panel (between 0 and 1).

Returns
ax – The plotting axes for the figure.

Return type
matplotlib.axes._subplots.AxesSubplot

plot_panel_counts_hist_simple(*ax*, *x_sim*, *x_Kep*, *x_min=0*, *x_max=None*, *y_min=None*, *y_max=None*, *x_llim=None*, *x_ulim=None*, *normalize=False*, *N_sim_Kep_factor=1.0*, *log_y=False*, *c_sim=['k']*, *c_Kep=['k']*, *ls_sim=['-']*, *ms_Kep=['x']*, *lines_Kep=False*, *lw=1*, *labels_sim=['Simulated']*, *labels_Kep=['Kepler']*, *xticks_custom=None*, *xlabel_text='x'*, *ylabel_text='Number'*, *afs=20*, *tfs=20*, *lfs=16*, *legend=False*, *show_counts_sim=False*, *show_counts_Kep=False*)

Plot histogram(s) of discrete integer values on a given panel.

Parameters

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – The axes to plot on.
- **x_sim** (*list[array[int]]*) – A list with sample(s) of integers (e.g. simulated data).
- **x_Kep** (*list[array[int]]*) – A list with sample(s) of integers (e.g. Kepler data).
- **x_min** (*float, default=0*) – The minimum value to include.
- **x_max** (*float, optional*) – The maximum value to include.
- **y_min** (*float, optional*) – The lower y-axis limit.
- **y_max** (*float, optional*) – The upper y-axis limit.
- **x_llim** (*float, optional*) – The lower x-axis limit.
- **x_ulim** (*float, optional*) – The upper x-axis limit.
- **normalize** (*bool, default=False*) – Whether to normalize the histograms. If True, each histogram will sum to one.
- **N_sim_Kep_factor** (*float, default=1.*) – The number of simulated targets divided by the number of Kepler targets. If *normalize=False*, will divide the bin counts for the simulated data (*x_sim*) by this value to provide an equivalent comparison to the Kepler counts.
- **log_y** (*bool, default=False*) – Whether to plot the y-axis on a log-scale.
- **c_sim** (*list[str], default=['k']*) – A list of plotting colors for the histograms of simulated data.
- **c_Kep** (*list[str], default=['k']*) – A list of plotting colors for the histograms of Kepler data.
- **ls_sim** (*list[str], default=['-']*) – A list of line styles for the histograms of simulated data.
- **ms_Kep** (*list[str], default=['x']*) – A list of marker shapes for the histograms of Kepler data.
- **lines_Kep** (*bool, default=False*) – Whether to also draw the histogram lines for the Kepler data. Default value (False) will just draw marker points (given by *ms_Kep*) for the Kepler counts in each bin.
- **lw** (*float, default=1*) – The line width for the histograms.

- **labels_sim** (*list[str], default=['Simulated']*) – A list of legend labels for the histograms of simulated data.
- **labels_Kep** (*list[str], default=['Kepler']*) – A list of legend labels for the histograms of Kepler data.
- **xticks_custom** (*list or array[float], optional*) – The x-values at which to plot ticks.
- **xlabel_text** (*str, default='x'*) – The x-axis label.
- **ylabel_text** (*str, default='Number'*) – The y-axis label.
- **afs** (*int, default=20*) – The axes fontsize.
- **tfs** (*int, default=20*) – The text fontsize.
- **lfs** (*int, default=16*) – The legend fontsize.
- **legend** (*bool, default=False*) – Whether to show the legend.
- **show_counts_sim** (*bool, default=False*) – Whether to show the individual bin counts for the simulated data.
- **show_counts_Kep** (*bool, default=True*) – Whether to show the individual bin counts for the Kepler data.

plot_fig_counts_hist_simple(*fig_size*, *x_sim*, *x_Kep*, *x_min=0*, *x_max=None*, *y_min=None*, *y_max=None*, *x_llim=None*, *x_ulim=None*, *normalize=False*, *N_sim_Kep_factor=1.0*, *log_y=False*, *c_sim=['k']*, *c_Kep=['k']*, *ls_sim=['-']*, *ms_Kep=['x']*, *lines_Kep=False*, *lw=1*, *labels_sim=['Simulated']*, *labels_Kep=['Kepler']*, *xticks_custom=None*, *xlabel_text='x'*, *ylabel_text='Number'*, *afs=20*, *tfs=20*, *lfs=16*, *legend=False*, *show_counts_sim=False*, *show_counts_Kep=False*, *fig_lbrt=[0.15, 0.2, 0.95, 0.925]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot a figure with histogram(s) of discrete integer values.

Wrapper for the function *syssimpyplots.plot_catalogs.plot_panel_counts_hist_simple()*. Includes all of the parameters of that function, with the following additional parameters:

Parameters

- **fig_size** (*tuple*) – The figure size, e.g. '(16,8)'.
- **fig_lbrt** (*list[float], default=[0.15, 0.2, 0.95, 0.925]*) – The positions of the (left, bottom, right, and top) margins of the plotting panel (all values must be between 0 and 1).
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

Returns

ax – The plotting axes, if *save_fig=False*.

Return type

matplotlib.axes._subplots.AxesSubplot

plot_panel_pdf_simple(*ax*, *x_sim*, *x_Kep*, *x_min=None*, *x_max=None*, *y_min=None*, *y_max=None*, *n_bins=100*, *normalize=True*, *N_sim_Kep_factor=1.0*, *log_x=False*, *log_y=False*, *c_sim=['k']*, *c_Kep=['k']*, *ls_sim=['-']*, *ls_Kep=['-']*, *lw=1*, *alpha=0.2*, *labels_sim=['Simulated']*, *labels_Kep=['Kepler']*, *extra_text=None*, *xticks_custom=None*, *xlabel_text='x'*, *ylabel_text='Fraction'*, *afs=20*, *tfs=20*, *lfs=16*, *legend=False*)

Plot histogram(s) of continuous distributions on a given panel.

Parameters

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – The axes to plot on.
- **x_sim** (*list[array[float]]*) – A list with sample(s) of values (e.g. simulated data).
- **x_Kep** (*list[array[float]]*) – A list with sample(s) of values (e.g. Kepler data).
- **x_min** (*float, optional*) – The minimum value to include.
- **x_max** (*float, optional*) – The maximum value to include.
- **y_min** (*float, optional*) – The lower y-axis limit.
- **y_max** (*float, optional*) – The upper y-axis limit.
- **n_bins** (*int, default=100*) – The number of bins.
- **normalize** (*bool, default=True*) – Whether to normalize the histograms. If True, each histogram will sum to one.
- **N_sim_Kep_factor** (*float, default=1.*) – The number of simulated targets divided by the number of Kepler targets. If *normalize=False*, will divide the bin counts for the simulated data (*x_sim*) by this value to provide an equivalent comparison to the Kepler counts.
- **log_x** (*bool, default=False*) – Whether to plot the x-axis on a log-scale and use log-uniform bins.
- **log_y** (*bool, default=False*) – Whether to plot the y-axis on a log-scale.
- **c_sim** (*list[str], default=['k']*) – A list of plotting colors for the histograms of simulated data.
- **c_Kep** (*list[str], default=['k']*) – A list of plotting colors for the histograms of Kepler data.
- **ls_sim** (*list[str], default=['-']*) – A list of line styles for the histograms of simulated data.
- **ls_Kep** (*list[str], default=['-']*) – A list of line styles for the histograms of Kepler data.
- **lw** (*float, default=1*) – The line width for the histograms.
- **alpha** (*float, default=0.2*) – The transparency of the shading for the histograms of Kepler data (between 0 and 1).
- **labels_sim** (*list[str], default=['Simulated']*) – A list of legend labels for the histograms of simulated data.
- **labels_Kep** (*list[str], default=['Kepler']*) – A list of legend labels for the histograms of Kepler data.
- **extra_text** (*str, optional*) – Extra text to be displayed on the figure.
- **xticks_custom** (*list or array[float], optional*) – The x-values at which to plot ticks.
- **xlabel_text** (*str, default='x'*) – The x-axis label.
- **ylabel_text** (*str, default='Fraction'*) – The y-axis label.
- **afs** (*int, default=20*) – The axes fontsize.

- **tfs** (*int, default=20*) – The text fontsize.
- **lfs** (*int, default=16*) – The legend fontsize.
- **legend** (*bool, default=False*) – Whether to show the legend.

plot_fig_pdf_simple(*fig_size*, *x_sim*, *x_Kep*, *x_min=None*, *x_max=None*, *y_min=None*, *y_max=None*, *n_bins=100*, *normalize=True*, *N_sim_Kep_factor=1.0*, *log_x=False*, *log_y=False*, *c_sim=['k']*, *c_Kep=['k']*, *ls_sim=['-']*, *ls_Kep=['-']*, *lw=1*, *alpha=0.2*, *labels_sim=['Simulated']*, *labels_Kep=['Kepler']*, *extra_text=None*, *xticks_custom=None*, *xlabel_text='x'*, *ylabel_text='Fraction'*, *afs=20*, *tfs=20*, *lfs=16*, *legend=False*, *fig_lbrt=[0.15, 0.2, 0.95, 0.925]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot a figure with histogram(s) of continuous distributions.

Wrapper for the function *syssimpyplots.plot_catalogs.plot_panel_pdf_simple()*. Includes all of the parameters of that function, with the following additional parameters:

Parameters

- **fig_size** (*tuple*) – The figure size, e.g. '(16,8)'.
- **fig_lbrt** (*list[float], default=[0.15, 0.2, 0.95, 0.925]*) – The positions of the (left, bottom, right, and top) margins of the plotting panel (all values must be between 0 and 1).
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

Returns

ax – The plotting axes, if *save_fig=False*.

Return type

matplotlib.axes._subplots.AxesSubplot

plot_panel_cdf_simple(*ax*, *x_sim*, *x_Kep*, *x_min=None*, *x_max=None*, *y_min=0.0*, *y_max=1.0*, *log_x=False*, *c_sim=['k']*, *c_Kep=['k']*, *ls_sim=['-']*, *ls_Kep=['--']*, *lw=1*, *labels_sim=['Simulated']*, *labels_Kep=['Kepler']*, *extra_text=None*, *xticks_custom=None*, *xlabel_text='x'*, *ylabel_text='CDF'*, *one_minus=False*, *afs=20*, *tfs=20*, *lfs=16*, *legend=False*, *label_dist=False*)

Plot cumulative distribution functions (CDFs) for continuous distributions on a given panel.

Parameters

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – The axes to plot on.
- **x_sim** (*list[array[float]]*) – A list with sample(s) of values (e.g. simulated data).
- **x_Kep** (*list[array[float]]*) – A list with sample(s) of values (e.g. Kepler data).
- **x_min** (*float, optional*) – The minimum value to include.
- **x_max** (*float, optional*) – The maximum value to include.
- **y_min** (*float, default=0.*) – The lower y-axis limit.
- **y_max** (*float, default=1.*) – The upper y-axis limit.
- **log_x** (*bool, default=False*) – Whether to plot the x-axis on a log-scale.
- **c_sim** (*list[str], default=['k']*) – A list of plotting colors for the CDFs of simulated data.

- **c_Kep** (*list[str], default=['k']*) – A list of plotting colors for the CDFs of Kepler data.
- **ls_sim** (*list[str], default=['-']*) – A list of line styles for the CDFs of simulated data.
- **ls_Kep** (*list[str], default=['--']*) – A list of line styles for the CDFs of Kepler data.
- **lw** (*float, default=1*) – The line width for the CDFs.
- **labels_sim** (*list[str], default=['Simulated']*) – A list of legend labels for the CDFs of simulated data.
- **labels_Kep** (*list[str], default=['Kepler']*) – A list of legend labels for the CDFs of Kepler data.
- **extra_text** (*str, optional*) – Extra text to be displayed on the figure.
- **xticks_custom** (*list or array[float], optional*) – The x-values at which to plot ticks.
- **xlabel_text** (*str, default='x'*) – The x-axis label.
- **ylabel_text** (*str, default='CDF'*) – The y-axis label.
- **one_minus** (*bool, default=False*) – Whether to plot one minus the CDF.
- **afs** (*int, default=20*) – The axes fontsize.
- **tfs** (*int, default=20*) – The text fontsize.
- **lfs** (*int, default=16*) – The legend fontsize.
- **legend** (*bool, default=False*) – Whether to show the legend.
- **label_dist** (*bool, default=False*) – Whether to compute and show the distances between the simulated and Kepler CDFs on the lower-right corner of the plot. If True, will compute the KS distance (using *syssimpyplots.compare_kepler.KS_dist()*) and the modified AD distance (using *syssimpyplots.compare_kepler.AD_mod_dist()*) between each pair of *x_sim* and *x_Kep* arrays.

plot_fig_cdf_simple(*fig_size*, *x_sim*, *x_Kep*, *x_min=None*, *x_max=None*, *y_min=0.0*, *y_max=1.0*, *log_x=False*, *c_sim=['k']*, *c_Kep=['k']*, *ls_sim=['-']*, *ls_Kep=['--']*, *lw=1*, *labels_sim=['Simulated']*, *labels_Kep=['Kepler']*, *extra_text=None*, *xticks_custom=None*, *xlabel_text='x'*, *ylabel_text='CDF'*, *one_minus=False*, *afs=20*, *tfs=20*, *lfs=16*, *legend=False*, *label_dist=False*, *fig_lbrt=[0.15, 0.2, 0.95, 0.925]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot a figure with CDF(s) of continuous distributions.

Wrapper for the function *syssimpyplots.plot_catalogs.plot_panel_cdf_simple()*. Includes all of the parameters of that function, with the following additional parameters:

Parameters

- **fig_size** (*tuple*) – The figure size, e.g. '(16,8)'.
- **fig_lbrt** (*list[float], default=[0.15, 0.2, 0.95, 0.925]*) – The positions of the (left, bottom, right, and top) margins of the plotting panel (all values must be between 0 and 1).
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

Returns
ax – The plotting axes, if *save_fig=False*.

Return type
matplotlib.axes._subplots.AxesSubplot

plot_fig_mult_cdf_simple(*fig_size*, *x_sim*, *x_Kep*, *x_min=1*, *x_max=None*, *y_min=0.0*, *y_max=1.0*, *c_sim=['k']*, *c_Kep=['k']*, *ls_sim=['-']*, *ls_Kep=['--']*, *lw=1*, *labels_sim=['Simulated']*, *labels_Kep=['Kepler']*, *xticks_custom=None*, *xlabel_text='x'*, *ylabel_text='CDF'*, *afs=20*, *tfs=20*, *lfs=16*, *legend=False*, *fig_lbrt=[0.15, 0.2, 0.95, 0.925]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot a figure with CDF(s) of discrete integer values.

Analogous to the function `syssimpyplots.plot_catalogs.plot_fig_cdf_simple()` but for integer values; uses almost all of the same parameters.

plot_figs_systems_gallery(*x_per_sys*, *s_per_sys*, *x_min=2.0*, *x_max=300.0*, *log_x=True*, *s_norm=2.0*, *color_by='k'*, *colors_per_sys=None*, *det_per_sys=None*, *llabel_per_sys=None*, *llabel_text=None*, *llabel_fmt='{:.2f}'*, *xticks_custom=None*, *xlabel_text='Period P (days)'*, *title=None*, *legend=False*, *s_examples=[1.0, 3.0, 10.0]*, *s_units='$R_\\oplus$'*, *legend_fmt='${:.0f}$'*, *afs=16*, *tfs=16*, *lfs=8*, *sys_per_fig=100*, *line_every=1*, *max_figs=5*, *fig_size=(4, 8)*, *fig_lbrt=[0.1, 0.1, 0.9, 0.95]*, *save_name_base='gallery'*, *save_fmt='png'*, *save_fig=False*)

Plot a gallery of systems to visualize their architectures.

Parameters

- **x_per_sys** (*array[float]*) – The properties of the planets in each system to plot on the x-axis. This is a 2-d array (where each row is a separate system), typically of the orbital period (days).
- **s_per_sys** (*array[float]*) – The properties of the planets in each system to plot as the sizes of the points (which will be proportional to the squared values of *s_per_sys*). This is a 2-d array (where each row is a separate system), typically of the planet radii (Earth radii).
- **x_min** (*float, default=2.*) – The lower limit for the x-axis.
- **x_max** (*float, default=300.*) – The upper limit for the x-axis.
- **log_x** (*bool, default=True*) – Whether to plot the x-axis on a log-scale.
- **s_norm** (*float, default=2.*) – The normalization for the sizes of the points.
- **color_by** (*{'k', 'size_order', 'custom'}*) – A string indicating the way the points are colored. The options are: 'k' to color all points in black; 'size_order' to color the points by their size ordering; 'custom' to color the points by custom values given by *colors_per_sys*.
- **colors_per_sys** (*array[float], optional*) – A 2-d array for coloring the planets in each system, used if *color_by='custom'*.
- **det_per_sys** (*array[float], optional*) – A 2-d array indicating which planets in each system are detected. If provided, will mark undetected planets with red outlines.
- **llabel_per_sys** (*array[float], optional*) – A 1-d array for labeling each system.
- **llabel_text** (*str, optional*) – The label for the labelings of each system.
- **llabel_fmt** (*str, optional*) – The string formatting of the labels in *llabel_per_sys*.
- **xticks_custom** (*list or array[float], optional*) – The x-values at which to plot ticks.
- **xlabel_text** (*str, default=r'Period P (days)'*) – The x-axis label.

- **title** (*str, optional*) – The title for the figures.
- **legend** (*bool, default=False*) – Whether to plot a legend. Note that this will not make an actual legend object, but will simply plot an additional "system" at the top with several example planets of sizes given by *s_examples*.
- **s_examples** (*list or array[float], default=[1.,3.,10.]*) – The example planet sizes to plot in the legend. Must have the same units as the values in *s_per_sys*.
- **s_units** (*str, default=r'R_oplus'*) – The text for displaying the units in the legend.
- **legend_fmt** (*str, default=r'${:.0f}$'*) – The string formatter for the example values in the legend.
- **afs** (*int, default=16*) – The axes fontsize.
- **tfs** (*int, default=16*) – The text fontsize.
- **lfs** (*int, default=8*) – The labels fontsize.
- **sys_per_fig** (*int, default=100*) – The number of systems to plot on each figure.
- **line_every** (*int, default=1*) – The number of systems before a horizontal line is plotted.
- **max_figs** (*int, default=5*) – The maximum number of figures to generate.
- **fig_size** (*tuple, default=(4,8)*) – The figure size.
- **fig_lbrt** (*list[float], default=[0.1, 0.1, 0.9, 0.95]*) – The positions of the (left, bottom, right, and top) margins of the plotting panel (all values must be between 0 and 1).
- **save_name_base** (*str, default='gallery'*) – The start of the file names for saving each figure.
- **save_fmt** (*str, default='png'*) – The file format for saving each figure (e.g., 'png', 'pdf', 'eps', 'jpg', etc.).
- **save_fig** (*bool, default=False*) – Whether to save the figures. If True, will save each figure in the working directory with the file names given by *save_name_base* with an index appended.

plot_figs_observed_systems_gallery_from_cat_obs(*ss_per_sys*, *sort_by='inner'*, *n_min=3*, *n_max=20*, *x_min=2.0*, *x_max=300.0*, *log_x=True*, *s_norm=2.0*, *color_by='k'*, *llabel=None*, *llabel_text=None*, *llabel_fmt='{:.2f}'*, *xticks_custom=None*, *xlabel_text='Period P (days)'*, *title=None*, *legend=False*, *s_examples=[1.0, 3.0, 10.0]*, *s_units='$R_\\oplus$'*, *legend_fmt='${:.0f}$'*, *afs=16*, *tfs=16*, *lfs=8*, *max_sys=100*, *sys_per_fig=100*, *line_every=1*, *max_figs=5*, *fig_size=(4, 8)*, *fig_lbrt=[0.1, 0.1, 0.9, 0.95]*, *save_name_base='gallery'*, *save_fmt='png'*, *save_fig=False*)

Plot a gallery of systems from an observed catalog.

Filters, samples, and sorts the systems before plotting them using the function *syssimpyplots.plot_catalogs.plot_figs_systems_gallery()*.

Parameters

- **ss_per_sys** (*dict*) – The dictionary containing the planetary and stellar properties for each system in an observed catalog (2-d and 1-d arrays). Can be a simulated catalog (i.e. returned by *syssimpyplots.load_sims.compute_summary_stats_from_cat_obs()*) or a Kepler catalog (i.e. returned by *syssimpyplots.compare_kepler.compute_summary_stats_from_Kepler_catalog()*).
- **sort_by** (*{'inner', 'multiplicity', 'stellar_mass'}, optional*) – The property to sort the systems by. Default is None (no sorting).
- **n_min** (*int, default=3*) – The minimum number of observed planets a system must have to be included.
- **n_max** (*int, default=20*) – The maximum number of observed planets a system must have to be included.
- **llabel** (*{'multiplicity', 'stellar_mass'}, optional*) – The property to left-label each system with. Default is None (no left-labels).

The remaining parameters are described in the documentation for the function *syssimpyplots.plot_catalogs.plot_figs_systems_gallery()*.

plot_figs_physical_systems_gallery_from_cat_phys(*sssp_per_sys*, *sssp*, *sort_by='inner'*, *n_min=1*, *n_max=20*, *n_det_min=1*, *n_det_max=10*, *x_min=2.0*, *x_max=300.0*, *log_x=True*, *s_norm=2.0*, *color_by='k'*, *mark_det=True*, *llabel=None*, *llabel_text=None*, *llabel_fmt='{:.2f}'*, *xticks_custom=None*, *xlabel_text='Period P (days)'*, *title=None*, *legend=False*, *s_examples=[1.0, 3.0, 10.0]*, *s_units='$R_\\oplus$'*, *legend_fmt='${:.0f}$'*, *afs=16*, *tfs=16*, *lfs=8*, *max_sys=100*, *sys_per_fig=100*, *line_every=1*, *max_figs=5*, *fig_size=(4, 8)*, *fig_lbrt=[0.1, 0.1, 0.9, 0.95]*, *save_name_base='gallery'*, *save_fmt='png'*, *save_fig=False*)

Plot a gallery of systems from a physical catalog.

Filters, samples, and sorts the systems before plotting them using the function *syssimpyplots.plot_catalogs.plot_figs_systems_gallery()*.

Parameters

- **sssp_per_sys** (*dict*) – The dictionary containing the planetary and stellar properties for each system in an physical catalog (2-d and 1-d arrays), such as one returned by *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*).
- **sort_by** (*{'inner', 'multiplicity', 'stellar_mass'}, optional*) – The property to sort the systems by. Default is None (no sorting).
- **n_min** (*int, default=3*) – The minimum number of physical planets a system must have to be included.
- **n_max** (*int, default=20*) – The maximum number of physical planets a system must have to be included.
- **n_det_min** (*int, default=1*) – The minimum number of detected planets a system must have to be included.
- **n_det_max** (*int, default=10*) – The maximum number of detected planets a system must have to be included.

- **color_by** (*{'k', 'size_order', 'cluster', 'mass', 'eccentricity', 'AMD'}*) – The way in which the planets are colored.
- **llabel** (*{'multiplicity', 'stellar_mass'}, optional*) – The property to left-label each system with.
- **mark_det** (*bool, default=True*) – Whether to mark the detected and undetected planets separately.

The remaining parameters are described in the documentation for the function *syssimpyplots.plot_catalogs.plot_figs_systems_gallery()*.

Note: The *mark_det=True* option requires the `sssp_per_sys` object to have been generated with *load_full_tables=True* and *match_observed=True*!

1.6.5 plot_params.py

Note: Currently under construction!

---

load_training_points(*dims*, *file_name_path=''*, *file_name=''*)

Load the table of model parameters and total weighted distances of the model iterations compared to the Kepler data, that was used for training the Gaussian process emulator.

Parameters

- **dims** (*int*) – The number of (free) model parameters.
- **file_name_path** (*str, default=''*) – The path to the file containing the results of the model iterations.
- **file_name** (*str, default=''*) – The name of the file containing the results of the model iterations (e.g., 'Active_params_recomputed_distances_table_best100000_every10.txt').

Returns

data_train – A dictionary of the model training points.

Return type

dict

The dictionary contains the following fields:

- *active_params_names*: The names of the (free) model parameters.
- *xtrain*: The (free) model parameter values for each iteration (2-d array).
- *ytrain*: The total weighted distance of the model compared to the Kepler data for each iteration (1-d array).

load_GP_table_prior_draws(*file_name*, *file_name_path=''*)

Load a table of model parameter draws from the prior.

transform_sum_diff_params(*xpoints*, *i*, *j*)

Transform two parameters into their sum and difference.

Parameters

- **xpoints** (*array[float]*) – An array of parameters for each iteration (2-d array).
- **i** (*int*) – The index of a parameter to transform.

- **j** (*int*) – The index of a parameter to transform.

Returns
xpoints_transformed – The array of parameters with the x_i and x_j parameters replaced by their sum (x_i+x_j) and difference (x_j-x_i). 2-d array with the same shape as *xpoints*.

Return type
array[float]

See also:

transform_sum_diff_params_inverse
Perform the inverse transformation (return the original parameters from their sum and difference).

transform_sum_diff_params_inverse(*xpoints*, *i*, *j*)

Transform the sum and difference of two parameters back into the original parameters.

Parameters

- **xpoints** (*array[float]*) – An array of parameters for each iteration (2-d array), where the x_i and x_j parameters are the sum and difference of two original parameters.
- **i** (*int*) – The index of a transformed parameter.
- **j** (*int*) – The index of a transformed parameter.

Returns
xpoints_transformed – The array of original parameters (2-d array). Equivalent to the transformation of x_i and x_j to (x_i-x_j)/2 and (x_i+x_j)/2.

Return type
array[float]

See also:

transform_sum_diff_params
Perform the sum and difference of two parameters (the inverse of this transformation).

make_cuts_GP_mean_std_post(*x_names*, *xprior_table*, *max_mean=numpy.inf*, *max_std=numpy.inf*, *max_post=numpy.inf*)

Return a restricted sample of parameters given cuts on the Gaussian process (GP) emulator predictions at those parameters.

Parameters

- **x_names** (*array[str]*) – The names of the parameters.
- **xprior_table** (*structured array*) – The table of parameter values and GP predictions at each iteration, with parameter/column names corresponding to *x_names*.
- **max_mean** (*float, default=inf*) – The maximum mean prediction to include.
- **max_std** (*float, default=inf*) – The maximum standard deviation to include.
- **max_post** (*float, default=inf*) – The maximum posterior draw value to include.

Returns
xpoints_cut – The table of parameter values at the iterations surviving all of the cuts.

Return type
structured array

plot_fig_hists_GP_draws(*fig_size*, *xprior_table*, *bins=100*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot figure for visualizing the Gaussian process emulator statistics.

Contains four panels for the histograms of the Gaussian process mean predictions, prediction draws, and standard deviations, along with a scatter plot of mean prediction vs. standard deviation.

Parameters

- **fig_size** (*tuple*) – The figure size, e.g. (16,8).
- **xprior_table** (*structured array*) – The table of parameter values and GP predictions at each iteration.
- **bins** (*int, default=100*) – The number of bins to use for the histograms.
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

plot_cornerpy_wrapper(*x_symbols*, *xpoints*, *xpoints_extra=None*, *c_extra='r'*, *s_extra=1*, *quantiles=[0.16, 0.5, 0.84]*, *verbose=False*, *fig=None*, *show_titles=True*, *label_kwargs={'fontsize': 20}*, *title_kwargs={'fontsize': 20}*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot a parameter space as a corner plot.

Wrapper for the corner.corner function from the corner.py package, with the option of over-plotting an additional set of points for comparison.

Parameters

- **x_symbols** (*list or array[str]*) – The list of the parameter names/symbols.
- **xpoints** (*array[float]*) – The sample of parameter values (2-d array).
- **xpoints_extra** (*array[float], optional*) – An additional sample of parameter values to plot (2-d array).
- **c_extra** (*str, default='r'*) – The color for plotting the additional sample of parameters.
- **s_extra** (*float, default=1*) – The marker size for plotting the additional sample of parameters.
- **quantiles** (*list, default=[0.16, 0.5, 0.84]*) – The quantiles to show on the histograms of the parameters as vertical lines.
- **verbose** (*bool, default=False*) – Whether to print the quantiles for the distribution of each parameter.
- **fig** (*matplotlib.figure.Figure, optional*) – An existing figure object to plot on.
- **show_titles** (*bool, default=True*) – Whether to show the quantiles for the distribution of each parameter above each histogram.
- **label_kwargs** (*dict, default={'fontsize': 20}*) – Extra parameters for setting the labels.
- **title_kwargs** (*dict, default={'fontsize': 20}*) – Extra parameters for setting the title.
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

Returns
fig – The figure, if *save_fig=False*.

Return type
matplotlib.figure.Figure

plot_contours_and_points_corner(*x_symbols*, *xlower*, *xupper*, *contour_2d_grids*, *xpoints=None*, *points_size=1.0*, *points_alpha=1.0*, *afs=10*, *tfs=12*, *lfs=10*, *fig_size=(16, 16)*, *fig_lbrtwh=[0.05, 0.05, 0.98, 0.98, 0.05, 0.05]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot contours for the evaluations of a function over 2-d grids of an n-d parameter space.

Parameters

- **x_symbols** (*list or array[str]*) – The list of the parameter names/symbols.
- **xlower** (*list or array[float]*) – The lower bounds for each parameter.
- **xupper** (*list or array[float]*) – The upper bounds for each parameter.
- **contour_2d_grids** (*array[float]*) – The array of 2-d grids for the evaluations of a function (3-d array of dimensions '(N,gdim,gdim)' where 'N' is the number of unique pairs of parameters, and 'gdim' is the number of grid points along each grid dimension).
- **xpoints** (*array[float], optional*) – An additional sample of parameter values to plot (2-d array).
- **points_size** (*float, default=1.*) – The point size for plotting the additional sample of parameters.
- **points_alpha** (*float, default=1.*) – The transparency of the points for plotting the additional sample of parameters.
- **afs** (*int, default=10*) – The axes fontsize.
- **tfs** (*int, default=12*) – The text fontsize.
- **lfs** (*int, default=10*) – The legend fontsize.
- **fig_size** (*tuple, default=(16,16)*) – The figure size.
- **fig_lbrtwh** (*list, default=[0.05, 0.05, 0.98, 0.98, 0.05, 0.05]*) – The positions of the (left, bottom, right, and top) margins of all the plotting panels (between 0 and 1), followed by the width and height space between the panels.
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

plot_2d_points_and_contours_with_histograms(*x*, *y*, *x_min=None*, *x_max=None*, *y_min=None*, *y_max=None*, *log_x=False*, *log_y=False*, *bins_hist=50*, *bins_cont=50*, *points_only=False*, *xlabel_text='x'*, *ylabel_text='y'*, *extra_text=None*, *plot_qtls=True*, *log_x_qtls=False*, *log_y_qtls=False*, *x_str_format='{:0.2f}'*, *y_str_format='{:0.2f}'*, *x_symbol='x'*, *y_symbol='y'*, *afs=20*, *tfs=20*, *lfs=16*, *fig_size=(8, 8)*, *fig_lbrtwh=[0.15, 0.15, 0.95, 0.95, 0.0, 0.0]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot a pair of parameters as a 2-d scatter plot with attached histograms of each parameter.

Parameters

- **x** (*array[float]*) – The values of the parameters (1-d array).
- **y** (*array[float]*) – The values of the parameters (1-d array).
- **x_min** (*float, optional*) – The minimum value of *x* to include.
- **x_max** (*float, optional*) – The maximum value of *x* to include.
- **y_min** (*float, optional*) – The minimum value of *y* to include.
- **y_max** (*float, optional*) – The maximum value of *y* to include.
- **log_x** (*bool, default=False*) – Whether to plot the x-axis on a log-scale (and take the log of the values in *x* for the histograms and quantiles)*.
- **log_y** (*bool, default=False*) – Whether to plot the y-axis on a log-scale (and take the log of the values in *y* for the histograms and quantiles)*.
- **bins_hist** (*int, default=50*) – The number of bins to use for the histograms.
- **bins_cont** (*int, default=50*) – The number of bins to use for the contour maps (along each parameter).
- **points_only** (*bool, default=False*) – Whether to plot only the parameter points instead of their contour maps.
- **xlabel_text** (*str, default='x'*) – The x-axis label.
- **ylabel_text** (*str, default='y'*) – The y-axis label.
- **extra_text** (*str, optional*) – Extra text to be displayed on the figure.
- **plot_qtls** (*bool, default=True*) – Whether to compute, print, and plot the quantiles of each parameter on the histograms.
- **log_x_qtls** (*bool, default=False*) – Whether to use the log of the *x* values for computing their quantiles. Only used if *log_x=True*.
- **log_y_qtls** (*bool, default=False*) – Whether to use the log of the *y* values for computing their quantiles. Only used if *log_y=True*.
- **x_str_format** (*str, default='{:0.2f}'*) – The string formatter for the *x* quantiles (e.g. to control how many significant figures are shown).
- **y_str_format** (*str, default='{:0.2f}'*) – The string formatter for the *y* quantiles (e.g. to control how many significant figures are shown).
- **x_symbol** (*str, default=r'x'*) – The symbol for the *x* parameter for labeling the quantiles.
- **y_symbol** (*str, default=r'y'*) – The symbol for the *y* parameter for labeling the quantiles.
- **afs** (*int, default=20*) – The axes fontsize.
- **tfs** (*int, default=20*) – The text fontsize.
- **lfs** (*int, default=16*) – The legend fontsize.
- **fig_size** (*tuple, default=(8,8)*) – The figure size.
- **fig_lbrtwh** (*list, default=[0.15, 0.15, 0.95, 0.95, 0., 0.]*) – The positions of the (left, bottom, right, and top) margins of all the plotting panels (between 0 and 1), followed by the width and height space between the panels (main plot and top/side histograms).
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.

- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

Returns
ax_main – The plotting axes for the main panel (scatter plot/contour map).

Return type
matplotlib.axes._subplots.AxesSubplot

Note: *The *log_x* and *log_y* parameters are options to take the log of the *x* and *y* inputs, NOT to mean that the input values are already logged! The user should always pass unlogged values for *x* and *y* (and their limits).

---

plot_function_heatmap_contours_given_irregular_points_corner(*x_symbols*, *xpoints*, *fpoints*, *xlower=None*, *xupper=None*, *show_points=True*, *points_size=1.0*, *points_alpha=1.0*, *afs=10*, *tfs=12*, *lfs=10*, *fig_size=(16, 16)*, *fig_lbrtwh=[0.05, 0.05, 0.98, 0.98, 0.05, 0.05]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot 2-d heat-maps and contours of a function evaluated on a set of irregularly spaced points in the n-d parameter space.

Parameters

- **x_symbols** (*list or array[str]*) – The list of the parameter names/symbols.
- **xpoints** (*array[float]*) – The sample of parameter values at which the function was evaluated (2-d array).
- **fpoints** (*array[float]*) – The function evaluations at the points given by *xpoints*.
- **xlower** (*list or array[float]*) – The lower bounds for each parameter.
- **xupper** (*list or array[float]*) – The upper bounds for each parameter.
- **show_points** (*bool, default=True*) – Whether to plot the individual parameter points.
- **points_size** (*float, default=1.*) – The point size for plotting the sample of parameters.
- **points_alpha** (*float, default=1.*) – The transparency of the points for plotting the sample of parameters.
- **afs** (*int, default=10*) – The axes fontsize.
- **tfs** (*int, default=12*) – The text fontsize.
- **lfs** (*int, default=10*) – The legend fontsize.
- **fig_size** (*tuple, default=(16,16)*) – The figure size.
- **fig_lbrtwh** (*list, default=[0.05, 0.05, 0.98, 0.98, 0.05, 0.05]*) – The positions of the (left, bottom, right, and top) margins of all the plotting panels (between 0 and 1), followed by the width and height space between the panels.
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

plot_function_heatmap_averaged_grid_given_irregular_points_corner(*x_symbols*, *xpoints*, *fpoints*, *flabel='f'*, *xlower=None*, *xupper=None*, *x_bins=20*, *show_cbar=True*, *show_points=True*, *points_size=1.0*, *points_alpha=1.0*, *afs=10*, *tfs=12*, *lfs=10*, *fig_size=(16, 16)*, *fig_lbrtwh=[0.05, 0.05, 0.98, 0.98, 0.05, 0.05]*, *save_name='no_name_fig.pdf'*, *save_fig=False*)

Plot 2-d heat-maps (using matplotlib.pyplot.imshow) on grids based on the evaluations of a function on a set of irregularly spaced points in the n-d parameter space.

The value of each grid bin is computed as the mean of the function evaluated at all of the points inside the bin.

Parameters

- **x_symbols** (*list or array[str]*) – The list of the parameter names/symbols.
- **xpoints** (*array[float]*) – The sample of parameter values at which the function was evaluated (2-d array).
- **fpoints** (*array[float]*) – The function evaluations at the points given by *xpoints*.
- **flabel** (*str, default='f'*) – The text label for the function.
- **xlower** (*list or array[float]*) – The lower bounds for each parameter.
- **xupper** (*list or array[float]*) – The upper bounds for each parameter.
- **x_bins** (*int, default=20*) – The number of bins to use for each dimension of each grid.
- **show_cbar** (*bool, default=True*) – Whether to show the colorbar for the function.
- **show_points** (*bool, default=True*) – Whether to plot the individual parameter points.
- **points_size** (*float, default=1.*) – The point size for plotting the sample of parameters.
- **points_alpha** (*float, default=1.*) – The transparency of the points for plotting the sample of parameters.
- **afs** (*int, default=10*) – The axes fontsize.
- **tfs** (*int, default=12*) – The text fontsize.
- **lfs** (*int, default=10*) – The legend fontsize.
- **fig_size** (*tuple, default=(16,16)*) – The figure size.
- **fig_lbrtwh** (*list, default=[0.05, 0.05, 0.98, 0.98, 0.05, 0.05]*) – The positions of the (left, bottom, right, and top) margins of all the plotting panels (between 0 and 1), followed by the width and height space between the panels.
- **save_name** (*str, default='no_name_fig.pdf'*) – The file name for saving the figure.
- **save_fig** (*bool, default=False*) – Whether to save the figure. If True, will save the figure in the working directory with the file name given by *save_name*.

1.6.6 compute_RVs.py

Attention: This module may be moved into a separate, individual package in the future.

`M_anom`(*t*, *T0*, *P*)

Compute the mean anomaly of an orbit at a given time.

Note: The parameters must have the same units (of time), but the actual units can be freely chosen (e.g., yrs).

Parameters

- **t** (*float*) – The time at which the mean anomaly is evaluated.
- **T0** (*float*) – The reference epoch.
- **P** (*float*) – The orbital period.

Returns
M – The mean anomaly (radians) at time *t*.

Return type
float

`fzero_E_anom`(*E*, *t*, *T0*, *P*, *e*)

Evaluate Kepler's equation moved to one side (solve for the zeros of this function to get solutions for the eccentric anomaly, *E*).

Note: The parameters *t*, *T0*, and *P* must have the same units (of time), but the actual units can be freely chosen (e.g., yrs).

Parameters

- **E** (*float*) – The eccentric anomaly (radians).
- **t** (*float*) – The time at which the anomalies are evaluated.
- **T0** (*float*) – The reference epoch.
- **P** (*float*) – The orbital period.
- **e** (*float*) – The orbital eccentricity.

Return type
Kepler's equation moved to one side.

`E_anom`(*t*, *T0*, *P*, *e*)

Solve for the eccentric anomaly at a given time.

Note: The parameters *t*, *T0*, and *P* must have the same units (of time), but the actual units can be freely chosen (e.g., yrs).

Parameters

- **t** (*float*) – The time at which the eccentric anomaly is evaluated.
- **T0** (*float*) – The reference epoch.
- **P** (*float*) – The orbital period.
- **e** (*float*) – The orbital eccentricity.

Returns
E – The eccentric anomaly (radians).

Return type
float

nu_anom(*E*, *e*)

Compute the true anomaly from the eccentric anomaly and orbital eccentricity.

Parameters

- **E** (*float*) – The eccentric anomaly (radians).
- **e** (*float*) – The orbital eccentricity.

Returns
nu – The true anomaly (radians).

Return type
float

RV_true(*t*, *K*, *P*, *T0=0.0*, *e=0.0*, *w=0.0*, *gamma=0.0*)

Compute the true radial velocity of an orbit at a given time.

Parameters

- **t** (*float*) – The time (yrs) at which to compute the radial velocity.
- **K** (*float*) – The radial velocity semi-amplitude (m/s).
- **P** (*float*) – The orbital period (yrs).
- **T0** (*float, default=0.*) – The reference epoch (yrs).
- **e** (*float, default=0.*) – The orbital eccentricity.
- **w** (*float, default=0.*) – The argument of pericenter (radians).
- **gamma** (*float, default=0.*) – The radial velocity reference zero-point/offset (m/s).

Returns
V_r – The radial velocity (m/s) at time *t*.

Return type
float

RV_true_sys(*t*, *K_sys*, *P_sys*, *T0_sys*, *e_sys*, *w_sys*, *gamma=0.0*)

Compute the true radial velocity of a planetary system at a given time.

This is the sum of the radial velocities of the individual planets in the system, each computed using *syssimpyplots.compute_RVs.RV_true()*.

Parameters

- **t** (*float*) – The time (yrs) at which to compute the radial velocity.
- **K_sys** (*array[float]*) – The radial velocity semi-amplitudes (m/s) of the planets.
- **P_sys** (*array[float]*) – The orbital periods (yrs).

- **T0_sys** (*array[float]*) – The reference epochs (yrs).
- **e_sys** (*array[float]*) – The orbital eccentricities.
- **w_sys** (*array[float]*) – The arguments of pericenter (radians).
- **gamma** (*float, default=0.*) – The radial velocity reference zero-point/offset (m/s).

Returns
V_r – The radial velocity (m/s) at time *t*.

Return type
float

rv_K(*m*, *P*, *e=None*, *i=None*, *Mstar=1.0*)

Compute the radial velocity semi-amplitude of each planet in a system.

Parameters

- **m** (*array[float]*) – The planet masses (Earth masses).
- **P** (*array[float]*) – The orbital periods (days).
- **e** (*array[float], optional*) – The orbital eccentricities (assumed all zero if not provided).
- **i** (*array[float], optional*) – The orbital inclinations (radians) relative to the sky plane (assumed to be all 90 degrees, i.e. edge-on, if not provided).
- **Mstar** (*float, default=1.*) – The stellar mass (solar masses).

Returns
K – The radial velocity semi-amplitudes (m/s).

Return type
array[float]

fit_rv_K_single_planet_model_GLS(*t_obs*, *RV_obs*, *covarsc*, *P*, *T0=0.0*, *e=0.0*, *w=0.0*)

Fit the radial velocity (RV) semi-amplitude (K) of a single planet model given an RV time series.

Assumes fixed/known values for all other parameters of the planetary orbit, and uses generalized least squares (GLS) to solve for K.

Parameters

- **t_obs** (*array[float]*) – The observation times (days) corresponding to the RV observations.
- **RV_obs** (*array[float]*) – The RV observations (m/s).
- **covarsc** (*array[float]*) – The covariance matrix of measurement uncertainties (2-d array).
- **P** (*float*) – The orbital period (days) of the planet.
- **T0** (*float, default=0.*) – The reference epoch (days).
- **e** (*float, default=0.*) – The orbital eccentricity.
- **w** (*float, default=0.*) – The argument of pericenter (radians).

Returns

- **K_hat** (*float*) – The estimator for the RV semi-amplitude (m/s).
- **sigma_K** (*float*) – The estimated uncertainty in the RV semi-amplitude (m/s).

fit_rv_Ks_multi_planet_model_GLS(*t_obs*, *RV_obs*, *covarsc*, *P_all*, *T0_all*, *e_all*, *w_all*)

Fit the radial velocity (RV) semi-amplitudes (K's) of a multi-planet model given an RV time series.

Assumes fixed/known values for all other parameters of the planetary orbits, and uses generalized least squares (GLS) to solve for K.

Parameters

- **t_obs** (*array[float]*) – The observation times (days) corresponding to the RV observations.
- **RV_obs** (*array[float]*) – The RV observations (m/s).
- **covarsc** (*array[float]*) – The covariance matrix of measurement uncertainties (2-d array).
- **P_all** (*array[float]*) – The orbital periods (days).
- **T0_all** (*array[float]*) – The reference epochs (days).
- **e_all** (*array[float]*) – The orbital eccentricities.
- **w_all** (*array[float]*) – The arguments of pericenter (radians).

Returns

- **K_hat_all** (*array[float]*) – The estimators for the RV semi-amplitudes (m/s).
- **sigma_K_all** (*array[float]*) – The estimated uncertainties in the RV semi-amplitudes (m/s).

conditionals_dict(*P_cond_bounds=None*, *Rp_cond_bounds=None*, *Mp_cond_bounds=None*, *det=True*)

Create a dictionary with conditionals for planetary systems.

Allows for setting bounds on orbital period, planet radius, planet mass, and transit detectability.

Parameters

- **P_cond_bounds** (*list, optional*) – The [lower, upper] bounds on orbital period (days).
- **Rp_cond_bounds** (*list, optional*) – The [lower, upper] bounds on planet radius (Earth radii).
- **Mp_cond_bounds** (*list, optional*) – The [lower, upper] bounds on planet mass (Earth masses).
- **det** (*bool, default=True*) – Whether to require the planets to be detected in transits.

Returns
conds – A dictionary of conditionals.

Return type
dict

The dictionary contains the following fields:

- *P_lower*: The lower bound on orbital period (days).
- *P_upper*: The upper bound on orbital period (days).
- *Rp_lower*: The lower bound on planet radius (Earth radii).
- *Rp_upper*: The upper bound on planet radius (Earth radii).
- *Mp_lower*: The lower bound on planet mass (Earth masses).
- *Mp_upper*: The upper bound on planet mass (Earth masses).
- *det*: Whether to require the planets to be detected in transits (True/False).

condition_planets_bools_per_sys(*sssp_per_sys*, *conds*)

Compute a 2-d array of booleans indicating the planets passing the conditionals in a simulated physical catalog.

The 2-d array will match the shape of the 2-d arrays of planet properties in `sssp_per_sys`.

Parameters

- **sssp_per_sys** (*dict*) – The dictionary containing the planetary and stellar properties for each system in a physical catalog (2-d and 1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.
- **conds** (*dict*) – The dictionary of conditionals, e.g. returned by the function *syssimpyplots.compute_RVs.conditionals_dict()*.

Returns

bools_cond_per_sys – The 2-d array of booleans indicating which planets pass all the conditionals in *conds*.

Return type

array[bool]

Note: The output includes rows without any conditioned planets (i.e. rows with all False values), since it must have the same shape as the 2-d arrays of planet properties in `sssp_per_sys` in order to be able to index them.

---

condition_systems_indices(*sssp_per_sys*, *conds*)

Compute an array of indices indicating which systems in a simulated physical catalog contain at least one conditioned planet.

Parameters

- **sssp_per_sys** (*dict*) – The dictionary containing the planetary and stellar properties for each system in a physical catalog (2-d and 1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.
- **conds** (*dict*) – The dictionary of conditionals, e.g. returned by the function *syssimpyplots.compute_RVs.conditionals_dict()*.

Returns

i_cond – The array of indices indicating which systems contain at least one conditioned planet.

Return type

array[int]

plot_systems_gallery_with_RVseries_conditional(*sssp_per_sys*, *sssp*, *conds*, *outputs_RVs=None*, *mark_undet=True*, *fit_RVs=False*, *N_obs_all=None*, *repeat=1000*, *_1obs=1.0*, *t_obs_=0.2*, *fig_size=(9, 10)*, *seed=None*, *N_sample=20*, *N_per_plot=20*, *afs=12*, *tfs=12*, *save_name_base='no_name_fig'*, *save_fig=False*)

Plot a gallery of systems conditioned on a given planet, along with their radial velocity (RV) time series.

Parameters

- **sssp_per_sys** (*dict*) – The dictionary containing the planetary and stellar properties for each system in a physical catalog (2-d and 1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.
- **sssp** (*dict*) – A dictionary containing the planetary and stellar properties of all planets in a physical catalog (1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.

- **conds** (*dict*) – The dictionary of conditionals, e.g. returned by the function *syssimpyplots.compute_RVs.conditionals_dict()*.
- **outputs_RVs** (*structured array, optional*) – A table of RV simulation results (TODO: need more details).
- **mark_undet** (*bool, default=True*) – Whether to outline the undetected planets with red.
- **fit_RVs** (*bool, default=False*) – Whether to simulate how many RV observations are required to measure the RV semi-amplitude of the conditioned planet in each system.
- **N_obs_all** (*list[int], optional*) – The numbers of RV observations to test, if *fit_RVs=True*.
- **repeat** (*int, default=1000*) – The number of times to repeat the RV simulations for each system, if *fit_RVs=True*.
- **_1obs** (*float, default=1.*) – The single-measurement RV precision (m/s), if *fit_RVs=True*.
- **t_obs_** (*float, default=0.2*) – The standard deviation (days) in nightly RV observation times, if *fit_RVs=True*.
- **fig_size** (*tuple, default=(9,10)*) – The figure size.
- **seed** (*int, optional*) – A random seed, for reproducible results.
- **N_sample** (*int, default=20*) – The number of systems with a conditioned planet to sample and plot.
- **N_per_plot** (*int, default=20*) – The number of systems to plot per figure.
- **afs** (*int, default=12*) – The axes fontsize.
- **tfs** (*int, default=12*) – The text fontsize.
- **save_name_base** (*str, default='no_name_fig'*) – The start of the file names for saving the figures.
- **save_fig** (*bool, default=False*) – Whether to save the figures. If True, will save each figure in the working directory with the file name given by *save_name_base* with an index appended.

fit_RVobs_systems_conditional(*sssp_per_sys*, *sssp*, *conds*, *N_obs_all*, *cond_only=False*, *fit_sys_cond=True*, *fit_all_planets=False*, *N_sample=20*, *repeat=1000*, *_1obs=1.0*, *obs_mode='daily'*)

Simulate the fitting of radial velocity (RV) observations to a sample of systems with a conditioned planet from a simulated physical catalog, to measure the RV semi-amplitudes of the conditioned planets.

Parameters

- **sssp_per_sys** (*dict*) – The dictionary containing the planetary and stellar properties for each system in a physical catalog (2-d and 1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.
- **sssp** (*dict*) – A dictionary containing the planetary and stellar properties of all planets in a physical catalog (1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.
- **conds** (*dict*) – The dictionary of conditionals, e.g. returned by the function *syssimpyplots.compute_RVs.conditionals_dict()*.
- **N_obs_all** (*list[int]*) – The numbers of RV observations to test.

- **cond_only** (*bool, default=False*) – Whether to simulate an idealized case in which there are no planets other than the conditioned planet in each system.
- **fit_sys_cond** (*bool, default=True*) – Whether to simulate the realistic case in which we fit the RV semi-amplitude of the conditioned planet only.
- **fit_all_planets** (*bool, default=False*) – Whether to simulate an optimistic case in which we fit the RV semi-amplitudes of all the planets in each system.
- **N_sample** (*int, default=20*) – The number of systems with a conditioned planet to sample.
- **repeat** (*int, default=1000*) – The number of times to repeat the RV simulations for each system.
- **_1obs** (*float, default=1.*) – The single-measurement RV precision (m/s).
- **obs_mode** (*{'daily', 'random'}*) – How the RV observation times are drawn. *daily* will draw times spaced by a day with some added variation to avoid aliasing; *random* will draw completely random times over a set baseline (150 days).

Returns
outputs – A table with the averaged results of the RV simulations for each conditioned system.

Return type
structured array

The table has the following default columns:

- *id_sys*: The index of the system.
- *n_pl*: The total number of planets in the system.
- *P_cond*: The period (days) of the conditioned planet.
- *Rp_cond*: The radius (Earth radii) of the conditioned planet.
- *Mp_cond*: The mass (Earth masses) of the conditioned planet.
- *K_cond*: The RV semi-amplitude (m/s) of the conditioned planet.
- *K_max*: The maximum RV semi-amplitude (m/s) of the planets in the system.
- *K_sum*: The sum of the RV semi-amplitudes (m/s) of the planets in the system.

If *cond_only=True*, will also have the following columns:

- *N_obs_min_{XX}p_ideal*: The minimum number of RV observations required to measure *K_cond* to within {XX} percent error in the ideal case, where {XX} = 30, 20, 10, and 5.
- *rms_sigma_K_{XX}p_ideal*: The root mean square of the uncertainties in the measured *K_cond* when *K_cond* is measured to within {XX} percent error in the ideal case, where {XX} = 30, 20, 10, and 5.
- *rmsd_best_ideal*: The best (fractional) root mean square deviation of the measured *K_cond* from the true *K_cond* in the ideal case, from the number of RV observations tested.

If *fit_sys_cond=True*, will also have the following columns:

- *N_obs_min_{XX}p*: The minimum number of RV observations required to measure *K_cond* to within {XX} percent error in the realistic case, where {XX} = 30, 20, 10, and 5.
- *rms_sigma_K_{XX}p*: The root mean square of the uncertainties in the measured *K_cond* when *K_cond* is measured to within {XX} percent error in the realistic case, where {XX} = 30, 20, 10, and 5.
- *rmsd_best*: The best (fractional) root mean square deviation of the measured *K_cond* from the true *K_cond* in the realistic case, from the number of RV observations tested.

If *fit_all_planets=True*, will also have the following columns:

- *N_obs_min_{XX}p_fitall*: The minimum number of RV observations required to measure *K_cond* to within {XX} percent error in the optimistic case, where {XX} = 30, 20, 10, and 5.
- *rms_sigma_K_{XX}p_fitall*: The root mean square of the uncertainties in the measured *K_cond* when *K_cond* is measured to within {XX} percent error in the optimistic case, where {XX} = 30, 20, 10, and 5.
- *rmsd_best_fitall*: The best (fractional) root mean square deviation of the measured *K_cond* from the true *K_cond* in the optimistic case, from the number of RV observations tested.

fit_RVobs_single_planets_vs_K(*K_array*, *N_obs_all*, *P_bounds*, *alpha_P=0.0*, *sigma_ecc=0.25*, *repeat=1000*, *_1obs=1.0*, *t_obs_=0.2*)

Simulate the fitting of radial velocity (RV) observations of single planets as a function of their RV semi-amplitude (K).

Note: Draws single planets with periods from a power-law distribution and eccentricities from a Rayleigh distribution, while assuming a fixed array of K (thus, the mass of the planet will change).

Parameters

- **K_array** (*list or array[float]*) – The RV semi-amplitudes (m/s) to simulate.
- **N_obs_all** (*array[int]*) – The numbers of RV observations to test.
- **P_bounds** (*list or tuple*) – The (lower, upper) bounds on the orbital periods (days).
- **alpha_P** (*float, default=0.*) – The power-law index for the period distribution.
- **sigma_ecc** (*float, default=0.25*) – The Rayleigh scale for the eccentricity distribution.
- **repeat** (*int, default=1000*) – The number of times to repeat the RV simulations for each system.
- **_1obs** (*float, default=1.*) – The single-measurement RV precision (m/s).
- **t_obs_** (*float, default=0.2*) – The standard deviation (days) in nightly RV observation times.

Returns

outputs – A table with the averaged results of the RV simulations for each single planet.

Return type

structured array

The table has the following columns:

- *K*: The RV semi-amplitude (m/s) of the planet.
- *N_obs_min_20p*: The minimum number of RV observations required to measure *K* to within 20 percent error.
- *rmsd_best*: The best (fractional) root mean square deviation of the measured *K* from the true *K*, from the number of RV observations tested.

Warning: The results (each row) averages over the periods and eccentricities drawn for each planet, which are not independent from *K* and may have large effects on the minimum number of observations required!

plot_scatter_K_vs_P_conditional(*sssp_per_sys*, *sssp*, *conds*, *log_y=False*, *fig_size=(8, 5)*, *afs=20*, *tfs=20*, *lfs=16*, *save_name_base='no_name_fig'*, *save_fig=False*)

Plot the radial velocity semi-amplitude (*K*) versus orbital period (*P*) for all planets in systems with a conditioned planet.

Makes two figures: (1) a scatter plot of *K/K_max* versus *P*, where *K_max* is the maximum RV semi-amplitude of the planets in each system, and (2) a scatter plot of *K* versus *P*.

Parameters

- **sssp_per_sys** (*dict*) – The dictionary containing the planetary and stellar properties for each system in a physical catalog (2-d and 1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.
- **sssp** (*dict*) – A dictionary containing the planetary and stellar properties of all planets in a physical catalog (1-d arrays), e.g. returned by the function *syssimpyplots.load_sims.compute_summary_stats_from_cat_phys()*.
- **conds** (*dict*) – The dictionary of conditionals, e.g. returned by the function *syssimpyplots.compute_RVs.conditionals_dict()*.
- **log_y** (*bool, default=False*) – Whether to plot the y-axis (*K/K_max* or *K*) on a log-scale.
- **fig_size** (*tuple, default=(8,5)*) – The figure size.
- **afs** (*int, default=20*) – The axes fontsize.
- **tfs** (*int, default=20*) – The text fontsize.
- **lfs** (*int, default=16*) – The legend fontsize.
- **save_name_base** (*str, default='no_name_fig'*) – The start of the file names for saving the figures.
- **save_fig** (*bool, default=False*) – Whether to save the figures. If True, will save each figure in the working directory with the file name given by *save_name_base* with either 'v1' (for *K/K_max* vs. *P*) or 'v2' (for *K* vs. *P*) appended.

generate_latex_table_RVobs_systems_conditional(*outputs_RVs*, *nan_str='>1000'*, *N_sample=40*)

Make a LaTeX syntax-formatted table with the results of a simulation fitting radial velocity (RV) observations.

Parameters

- **outputs_RVs** (*structured array*) – A table of RV simulation results (TODO: need more details).
- **nan_str** (*str, default=r'>1000'*) – The string to replace NaN values.
- **N_sample** (*int, default=40*) – The number of rows/systems from *outputs_RVs* to include in the table.

Returns

table_array – An array of strings that will produce the rows of the LaTeX table.

Return type

array[str]

fit_line_loglog_Nobs_K_single_planets(*outputs_ideal*, *_1obs*, *p0*)

Fit a line to a number of points of log(N_obs) versus log(K), where 'N_obs' is the minimum number of radial velocity (RV) observations required to measure the RV semi-amplitude ('K') to within 20 percent error.

Equivalent to fitting N_obs as a power-law function of K.

Parameters

- **outputs_ideal** (*dict*) – The dictionary containing the results of the RV simulations of the ideal case, including the fields *K* for the RV semi-amplitudes (m/s) and *N_obs_min_20p* for the minimum number of RV observations required to measure *K* to within 20 percent error.
- **_1obs** (*float*) – The single measurement RV precision (m/s) to serve as the normalization point (also represented by 'K_norm').
- **p0** (*list*) – The initial guesses for the parameters of the line, [normalization, slope] where 'normalization' is the 'N_obs' at 'K_norm'.

Returns

- **logN** (*float*) – The log of the normalization of the fitted line, log10(N_obs) at 'K_norm'.
- **slope** (*float*) – The slope of the fitted line.

linear_logNobs_logK(*K*, *K_norm*, *Nobs_norm*, *slope*, *Nobs_min=5*, *round_to_ints=True*)

Return the required number of radial velocity (RV) observations ('N_obs') as a function of the RV semi-amplitude ('K') given a linear relation for log(N_obs) versus log(K).

Parameters

- **K** (*list or array[float]*) – The RV semi-amplitudes (m/s) at which to predict 'N_obs'.
- **K_norm** (*float*) – The normalization point (m/s) corresponding to *Nobs_norm*.
- **Nobs_norm** (*int*) – The required number of RV observations at the normalization point *K_norm*.
- **slope** (*float*) – The slope of the linear relation.
- **Nobs_min** (*int, default=5*) – The minimum number of RV observations possible. All predicted values less than this value will be set to this value.
- **round_to_ints** (*bool, default=True*) – Whether to round each resulting value of 'N_obs' to the nearest integer.

Returns

Nobs_K – The required number of RV observations at each RV semi-amplitude in *K*.

Return type

array[int] or array[float]

CHAPTER

TWO

PUBLICATIONS

The scripts in this package have been used to produce the figures and results in several papers:

- Architectures of Exoplanetary Systems. I: A Clustered Forward Model for Exoplanetary Systems around Kepler's FGK Stars

  Matthias Y. He, Eric B. Ford, Darin Ragozzine, 2019, MNRAS, 490, 4575-4605

- Architectures of Exoplanetary Systems. II: An Increase in Inner Planetary System Occurrence Towards Later Spectral Types for Kepler's FGK Dwarfs

  Matthias Y. He, Eric B. Ford, Darin Ragozzine, 2021a, AJ, 161, 16-40

- Architectures of Exoplanetary Systems. III: Eccentricity and Mutual Inclination Distributions of AMD-stable Planetary Systems

  Matthias Y. He, Eric B. Ford, Darin Ragozzine, Daniel Carrera, 2020, AJ, 160, 276-314

- Friends and Foes: Conditional Occurrence Rates of Exoplanet Companions and their Impact on Radial Velocity Follow-up Surveys

  Matthias Y. He, Eric B. Ford, Darin Ragozzine, 2021b, AJ, 162, 216-238

PYTHON MODULE INDEX

S

INDEX

L

M

N

P

R

S

T

Z