epos Documentation

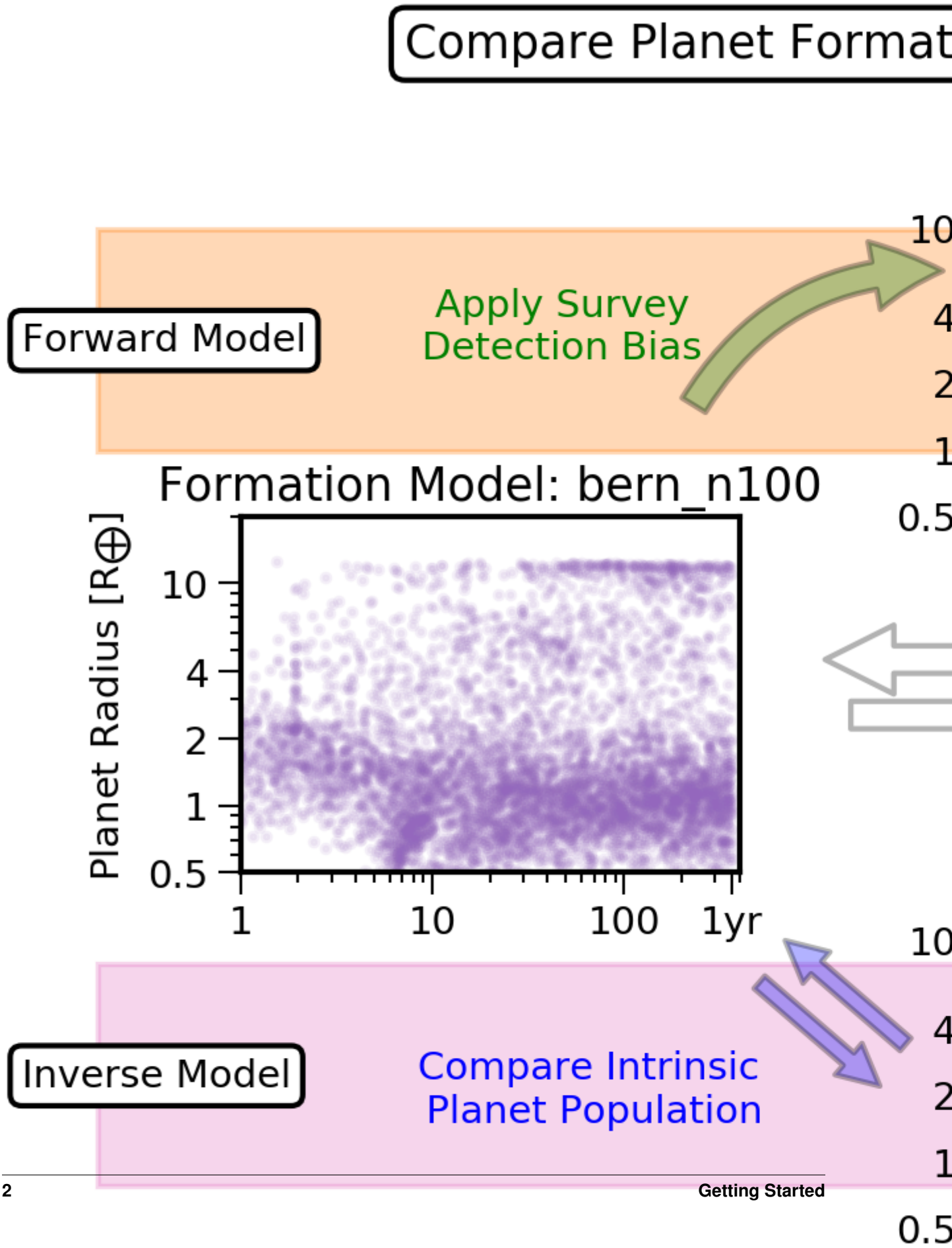
Release 3.0.1.dev4

Gijs Mulders

Jan 29, 2020

Getting Started

EPOS is a software package to simulate observations of exoplanet populations. It provides an interface between planet formation simulations and exoplanet surveys such as Kepler, accounting for detection biases in transit and radial velocity surveys.


Compare Planet Format
Forward Model
Apply Survey
Detection Bias
Formation Model: bern_n100
Planet Radius [R⊕]
10
4
2
1
0.5
1
10
100
1yr
Inverse Model
Compare Intrinsic
Planet Population

CHAPTER 1

Applications

Use EPOS to calculate:

- Planet occurrence rate distributions (*notebook*)
- Planetary system architectures (*notebook*)
- Synthetic observations of planet formation models
- Planet populations in radial velocity surveys
- Binned planet occurrence rates

CHAPTER 2

Install

EPOS is written in Python 3 and hosted on github.

The quickest way to install epos is via pip:

```
pip install epospy
```

CHAPTER 3

Getting Started

Familiarize yourself with the code is to take a look at the *Notebooks*

3.1 Installation

3.1.1 Pip

You can now install epos with `pip`:

```
pip install epospy
```

Depending on your OS, you might have to use `sudo` or:

```
sudo -H pip install epospy
```

You should now be able to import the EPOS module into python:

```
>>> import EPOS
```

3.1.2 Github

Download the source from github, preferably using git:

```
git clone https://github.com/GijsMulders/epos
```

In the future you can update to the latest version with:

```
git pull
```

You will need to add the EPOS directory to your path if you want to run from outside the main folder.

3.1.3 Dependencies

EPOS has the following dependencies:

- python 2.7
- numpy 1.13+
- scipy
- matplotlib 2.0+
- astropy

The following software is required if you want to use the MCMC part of EPOS

- emcee for the MCMC fitting
- corner.py for the corner plots

`pip` should take care of these dependencies automatically but if you installed via `github` you will have to install them manually.

3.1.4 Testing

To test the basic functionality of EPOS, you can run the test scripts. First, copy the test and example scripts to a local directory:

```
>>> import EPOS
>>> EPOS.scripts.install()
```

Navigate to the scripts directory (`epos-scripts/tests/`) and run each of:

```
./test_1_survey.py
./test_2_montecarlo.py
./test_3_mcmc.py
./test_4_multicore.py
./test_5_occurrence.py
```

If you don't have ipython installed, you can also run `python test_1_survey.py` or copy-paste the script into a terminal.

Each command is described in the comments, and provides an introduction to the basic functionality.

3.2 Examples

There are a number of example scripts included that demonstrate different functionality. The scripts contains comments describing each command. These scripts are copied to the `epos-scripts/examples/` directory by `EPOS.scripts.install()`.

3.2.1 Parametric mode

The first example shows how to fit a parametric function in planet radius and orbital period to Kepler data. The script can be run with:

```
./example_1_parametric_mode.py
```

Here, is a step-by-step description of each set of commands. First, load EPOS and set the output directory name

```
>>> import EPOS
>>> epos= EPOS.epos(name='example_1')
```

Second, load the observations (Kepler DR25 planet candidate list) and survey detection efficiency

```
>>> obs, survey= EPOS.kepler.dr25(Huber=True, Vetting=True, score=0.9)
>>> epos.set_observation(**obs)
>>> epos.set_survey(**survey)
```

Next, define the parametric distribution function that will be fitted to the data. (The example is a broken power-law in both the planet size and distance dimension from fitfunctions.py, but you can define your own)

```
>>> epos.set_parametric(EPOS.fitfunctions.brokenpowerlaw2D)
```

Now initialize the fit parameters with their initial values and limits (min, max). The first parameter is the normalization, defined as the number of planets per star over the simulated radius-period range:

```
>>> epos.fitpars.add('pps',              2.0,     min=0)
```

The next six parameters (one break point and two indices for each dimension) control the shape of the period-radius distribution

```
>>> epos.fitpars.add('P break', 10.,     min=2,   max=50, is2D=True)
>>> epos.fitpars.add('a_P',              1.5,     min=0,                    is2D=True)
>>> epos.fitpars.add('b_P',              0.0,     dx=0.1,                   is2D=True)
>>> epos.fitpars.add('R break', 3.0,     min=1.0,max=5,   is2D=True)
>>> epos.fitpars.add('a_R',              0.0,     dx=0.1,                   is2D=True)
>>> epos.fitpars.add('b_R',              -4.,     fixed=True,      is2D=True)
```

Note that the last parameter is fixed and thus not fitted for. `dx` is a parameter that controls the initial distribution of walkers when the inital value is zero.

Next, define the simulation range.

```
>>> epos.set_ranges(xtrim=[0,730],ytrim=[0.3,20.],xzoom=[2,400],yzoom=[1,6], Occ=True)
```

For transits, `x` refers to orbital period and `y` refers to planet size. The simulated range is that supplied by the detection efficiency grid and trimmed (`trim`) to the given values. For the observational comparison we `zoom` in a bit further.

Now we're ready to go! Run the code once with the initial values:

```
>>> EPOS.run.once(epos)
```

Then run the mcmc chain

```
>>> EPOS.run.mcmc(epos, nMC=1000, nwalkers=100, nburn=200, threads=20, Saved=True)
```

This runs multi-core with 20 threads. Saved indicates whether the mcmc will be skipped if a previously saved chain is present on disk. `Saved=False` always reruns the chain.

Define a set of bins where planet occurrence rates are calculated, both from the data and from integrating the fitted planet distributions, and calculate all the rates

```
>>> epos.set_bins(xbins=[[2,400],[0.9*365,2.2*365]], ybins=[[1,6],[0.7,1.5]]) # eta_
↪zoom, eta_earth
>>> EPOS.occurrence.all(epos)
```

Last, plot everything.

```
>>> EPOS.plot.survey.all(epos)
>>> EPOS.plot.input.all(epos)
>>> EPOS.plot.output.all(epos)
>>> EPOS.plot.mcmc.all(epos)
>>> EPOS.plot.occurrence.all(epos)
```

Plots will appear in the `png/example_1/` subfolder

3.2.2 Multi-planet Mode

```
./example_2_multiplanet_mode.py
```

In multi-planet mode, the first planet in the system is drawn from a parametric distribution same as above. However, we adjust the initial guess for the slope after the period break

```
>>> epos.fitpars.add('b_P',      -1,      max=1,   dx=0.1, is2D=True)
```

Next we tell epos to draw additional planets in the system assuming the spacing between adjacent planets is drawn from a dimensionless distribution:

```
>>> epos.set_multi(spacing='dimensionless')
```

We generate 10 planets per system.

```
>>> epos.fitpars.add('npl', 10, fixed=True)
```

The fit parameters for the dimensionless distribution are:

```
>>> epos.fitpars.add('log D', -0.3)
>>> epos.fitpars.add('sigma', 0.2, min=0)
```

Other properties of the planetary systems can also be fit for (or not):

```
>>> epos.fitpars.add('dR', 0.01, fixed=True) # Dispersion in planet radii
>>> epos.fitpars.add('inc', 2.0) # mode of mutual inclinations
>>> epos.fitpars.add('f_iso', 0.4) # Fraction of isotropic systems
>>> epos.fitpars.add('f_cor', 0.5, fixed=True) # Correlated noise
```

Then proceed as in single-planet mode.

3.2.3 Planet Formation Mode

Example 3 is a template for using EPOS with a planet formation / population synthesis model.

:: ./example_3_population_synthesis.py

Generate some random data in the same format as the outcome of a planet formation model. (These are 77 systems with 8 planets each)

```
>>> n= 616
>>> sma= 10.**np.random.uniform(-1.3,1,n)
>>> mass= 10.** (3.*np.random.power(0.5, n))
>>> radius= 10.** np.random.power(0.5, n)
```

(continues on next page)

(continued from previous page)

```
>>> inc= np.random.rayleigh(2, n)
>>> starID= np.repeat(np.arange(n/8), 8)
```

Load the planet formaton model into EPOS as a dictionary:

```
>>> pfm= {'sma':sma, 'mass':mass,'radius':radius, 'inc':inc, 'starID':starID}
>>> epos.set_population('Planet Formation Model', **pfm)
```

Set 1 in 5 stars to have planetary systems

```
>>> epos.fitpars.add('eta', 0.2, isnorm=True)
```

Optionally, use a mass-radius relation if the model does not simulate planetary radii:

```
>>> epos.set_massradius(EPOS.massradius.CK17, 'Chen & Kipping 2017', masslimits=[0.1,
↪100])
```

Define a polygonic bin for mini-Neptunes

```
>>> xmin, xmax= 3, 200
>>> ymin, ymax= 1.2, 4
>>> xb, yb= 100, 2.2
>>> xyMN= [[xmin,ymax],[xmax,ymax],[xmax,ymin], [xb,ymin], [xmin,yb]]
>>> epos.set_bins_poly([xyMN],
  labels=['Mini-\nNeptunes')
```

Then run epos and save/plot as usual

```
>>> EPOS.run.once(epos)
>>> EPOS.occurrence.all(epos)
>>> EPOS.plot.survey.all(epos)
>>> EPOS.plot.input.all(epos, imin=1e-4, color='C8')
>>> EPOS.plot.occurrence.all(epos, color='C8', alpha_fac=50.)
>>> EPOS.plot.output.all(epos, color='C8')
```

3.2.4 Radial Velocity Surveys

Example 5 shows how to estimate the distribution of planets from a radial velocity survey:

```
./example_5_radial_velocity.py
```

Two commands are different from fitting a transit survey: First, tell epos that we are doing a radial velocity survey (RV=True), that we are not doing the Monte Carlo simulation (MC=False) and that we are fitting for the planet mass distribution (Msini=True)

```
>>> epos= EPOS.epos(name='example_5', RV=True, MC=False, Msini=True)
```

Second, we define the radial velocity survey data, here from Mayor+ 2011

```
>>> obs, survey= EPOS.rv.Mayor2011()
>>> epos.set_observation(**obs)
>>> epos.set_survey(**survey)
```

3.2.5 Planet Ocurrence Rates

Example 9 shows how to estimate occurrence rates using the inverse detection efficiency method. You can run the entire script with:

```
./example_9_occurrence_rate_inverse.py
```

Here, is a step-by-step description of each set of commands. First, load EPOS and set the output directory name

```
>>> import EPOS
>>> epos= EPOS.epos(name='example_9')
```

Second, load the observations (Kepler DR25 planet candidate list) and survey dectetion efficiency

```
>>> obs, survey= EPOS.kepler.dr25(Huber=True, Vetting=True, score=0.9)
>>> epos.set_observation(**obs)
>>> epos.set_survey(**survey)
```

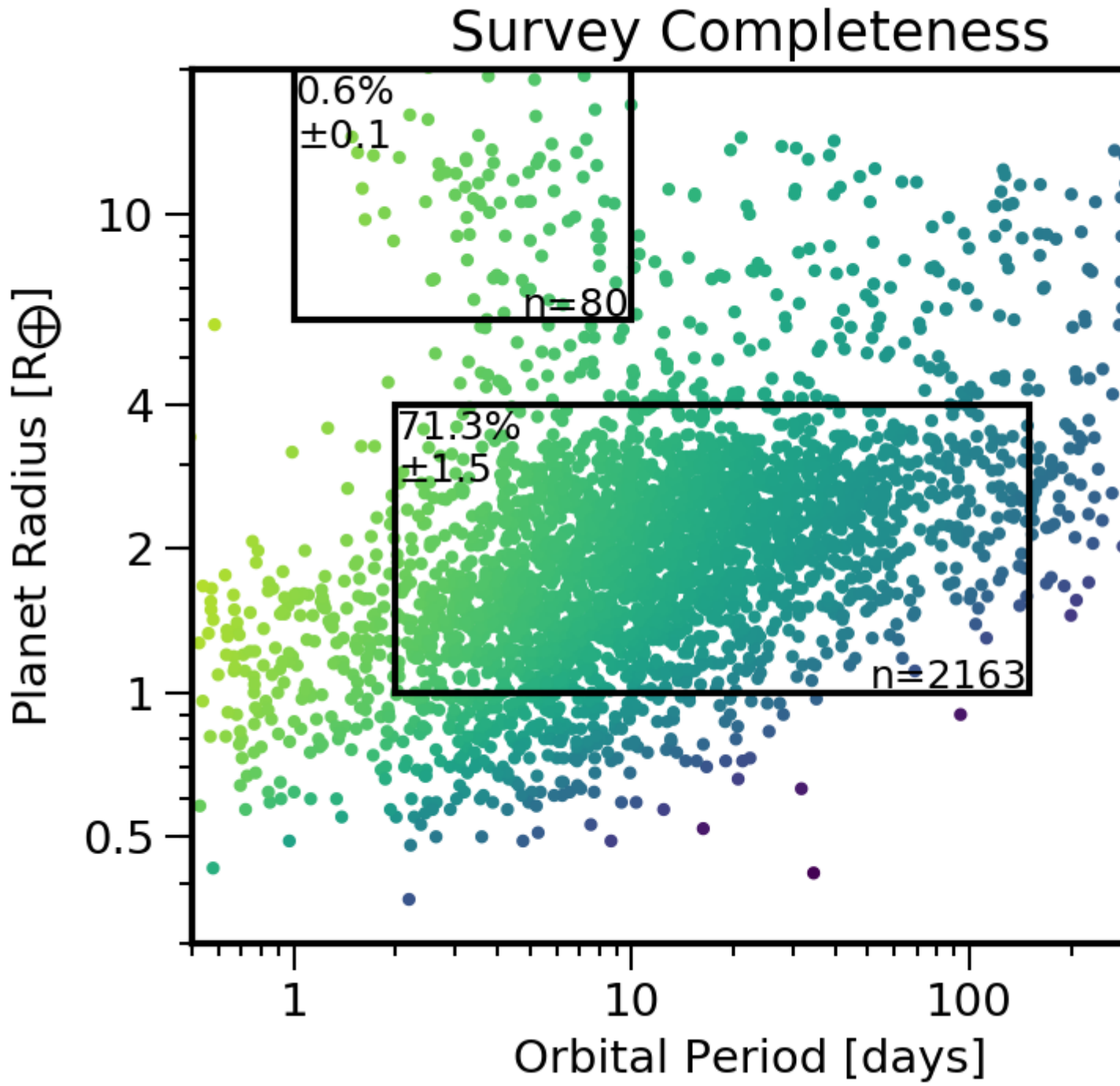
Next, define the occurrence rate bins for hot Jupiters and super-earths/mini-Neptunes:

```
>>> x_HJ= [1,10] # Orbital period range in days
>>> y_HJ= [7,20] # Planet size range in earth radii
>>> x_SEMN, y_SEMN= [2,150],[1.0,4.0] # super-Earths/mini-Neptunes
>>> epos.set_bins(xbins=[x_HJ, x_SEMN], ybins=[y_HJ, y_SEMN])
```

The rates are then calculated, plotted, and saved

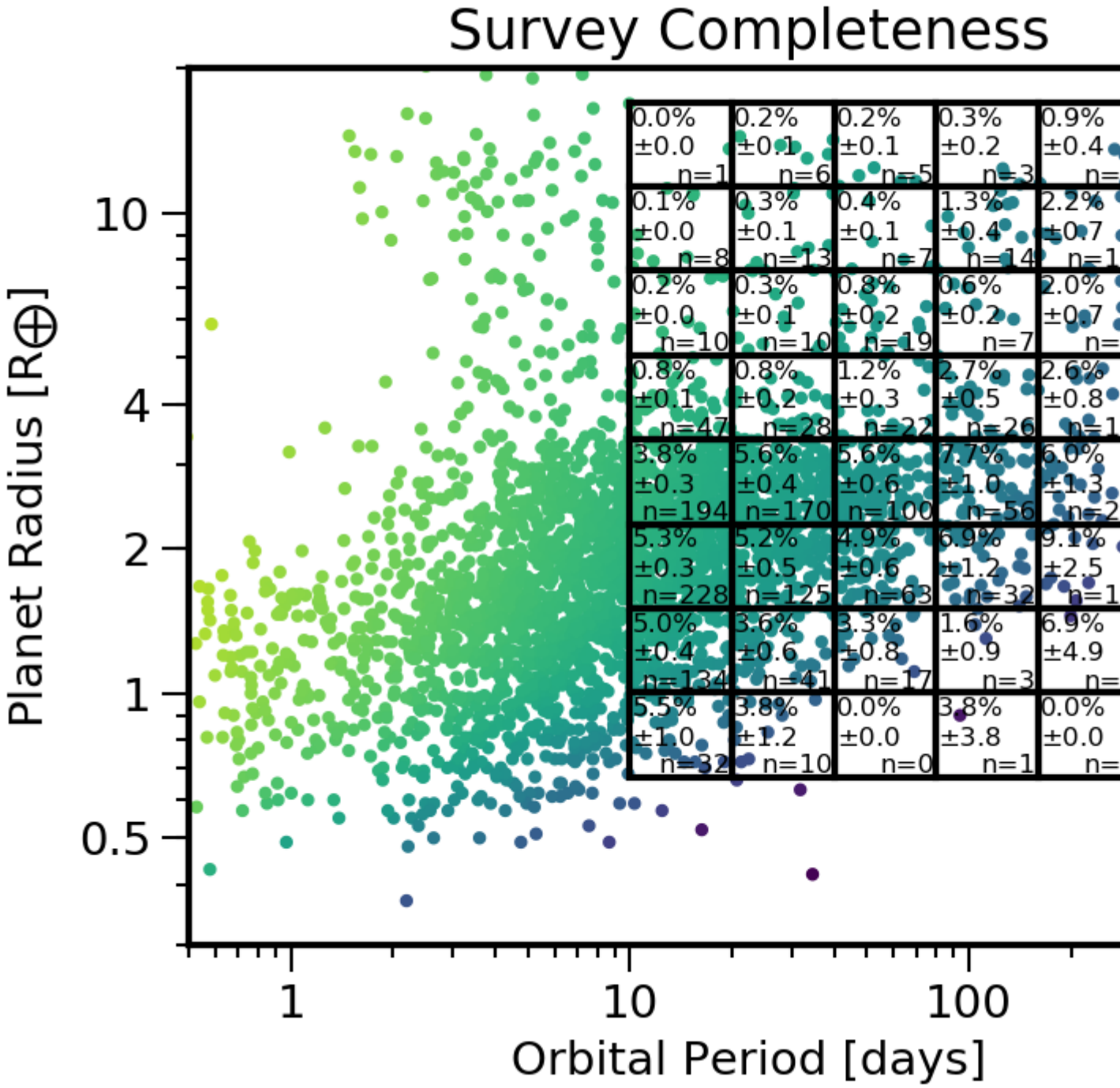
```
>>> EPOS.occurrence.all(epos)
>>> EPOS.save.occurrence(epos)
>>> EPOS.plot.occurrence.all(epos)
```

The output appears in `png/occurrence/bins.png` and should look like this:



Alternatively, you can generate a 1D or 2D grid of bins, for example the SAG13 grid:

```
>>> import numpy as np
>>> epos.set_bins(xgrid=np.geomspace(10,640,7),
   ybins=np.geomspace(0.67,17,9), Grid=True)
```



3.3 FAQ

3.3.1 Frequently asked questions

If you have any difficulties or questions running EPOS that are not addressed in the documentation or FAQ please contact gdmulders@gmail.com

I'm getting an AttributeError: 'module' object has no attribute 'geomspace'

Please upgrade to numpy 1.13 or a more recent version

3.4 Notebooks

3.4.1 Examples

- *Parametric Mode* (`save to disk`)
- *Multi-planet Mode* (`save to disk`)
- *Custom exoplanet survey* (`save to disk`)

The notebooks are also viewable on github

3.4.2 Papers

- Mulders+ 2018 (see tutorials *Parametric Mode* and *Multi-planet Mode*)
- Fernandes+ 2019
- Mulders+ 2019 (in prep)

These script are are also viewable on this github folder and you can copy these scripts to a local directory with:

```
>>> import EPOS
>>> EPOS.scripts.install()
```

3.5 Parametric Mode

Estimate the intrinsic distribution of planets by fitting a double broken power-law in period and radius to the Kepler data

[1]:
```
import EPOS
import numpy as np
import matplotlib.pyplot as plt
```

initialize the EPOS class

[2]:
```
epos= EPOS.epos(name='example_1')
```

```
 |~| epos 3.0.0.dev2 |~|

Using random seed 1099836816
```

Read in the kepler dr25 exoplanets and survey efficiency packaged with EPOS

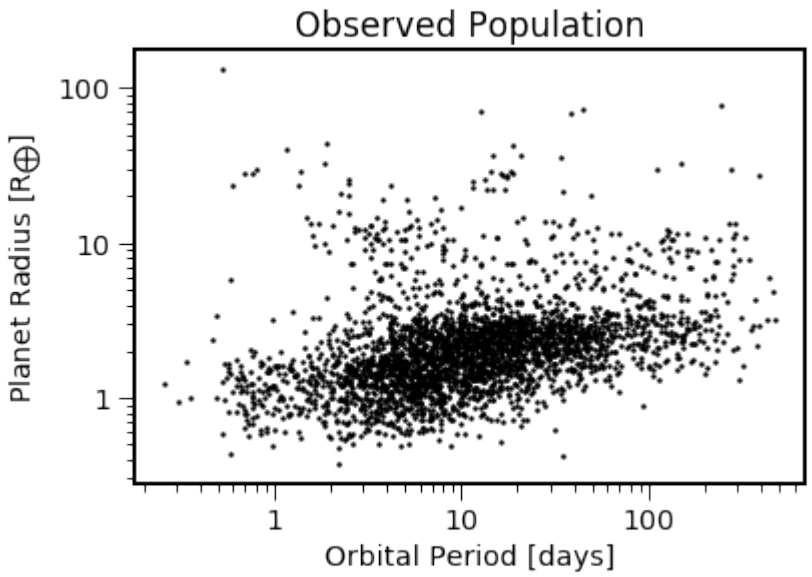
[3]:
```
obs, survey= EPOS.kepler.dr25(Huber=True, Vetting=True, score=0.9)
```

```
Loading planets from temp/q1_q17_dr25_koi.npz
  6853/7995 dwarfs
  3525 candidates, 3328 false positives
  3040+1 with score > 0.90
```

Load and display the observed planet candidates

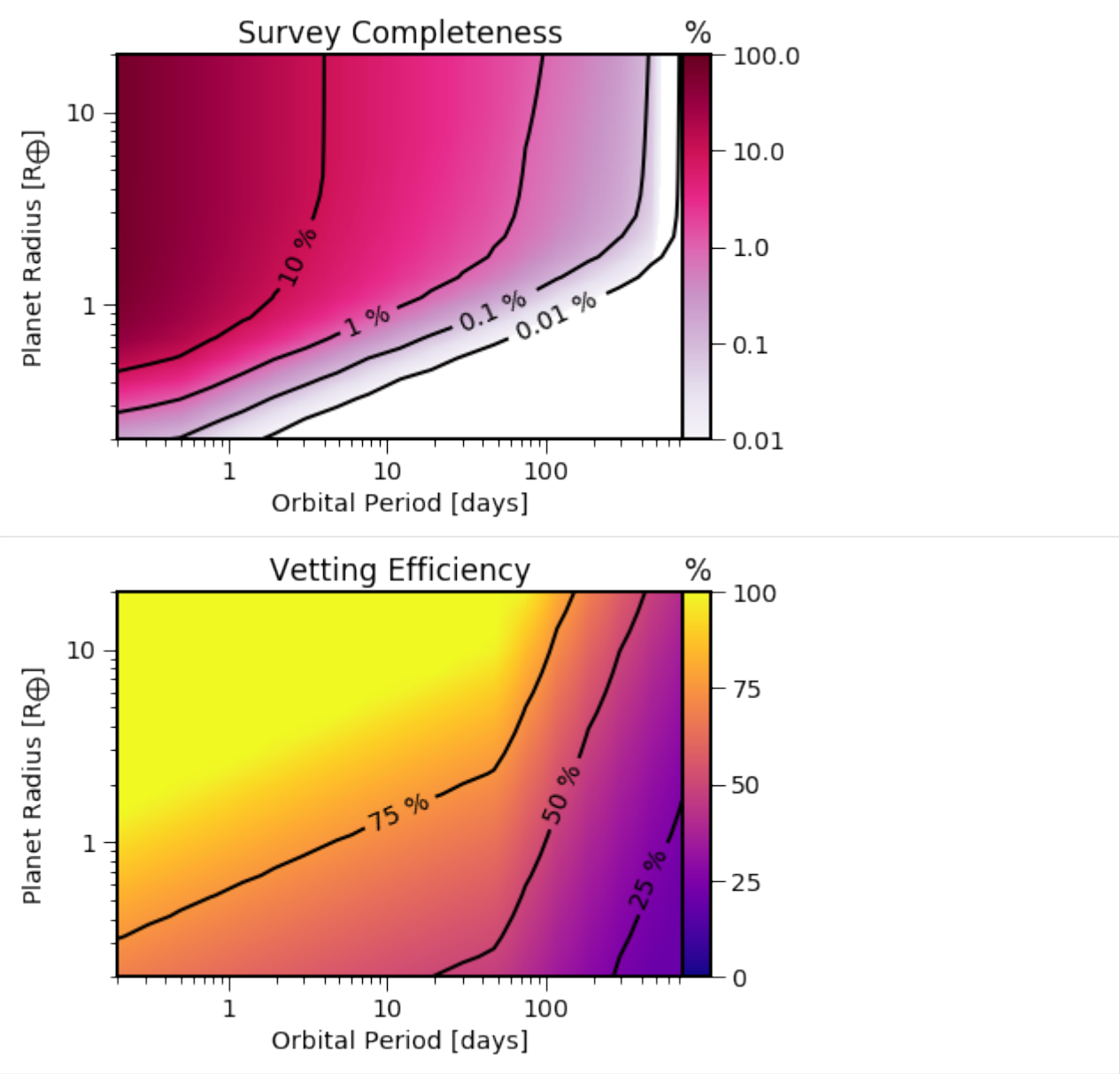
[4]:
```
epos.set_observation(**obs)
EPOS.plot.survey.observed(epos, NB=True, PlotBox=False)
```

```
Observations:
  159238 stars
  3041 planets

  1840 singles, 487 multis
  - single: 1840
  - double: 324
  - triple: 113
  - quad: 38
  - quint: 10
  - sext: 2
```



Load and display the survey detection efficiency

[5]:
```
epos.set_survey(**survey)
EPOS.plot.survey.completeness(epos, NB=True, PlotBox=False)
EPOS.plot.survey.vetting(epos, PlotBox=False, NB=True)
```



Define the function that describes the intrinsic planet population. Here we use a double broken power-law from EPOS.fitfunctions.

[6]:
```
epos.set_parametric(EPOS.fitfunctions.brokenpowerlaw2D)
```

brokenpowerlaw2D takes 8 parameters. The two dependent parameters are the period and radius. There are 6 free parameters (xp, p1, p2, yp, p3, p4) and a normalization parameter. Let's define them:

The normalization parameter, labeled pps, defines the integrated planet occurrence rate over the radius and period range (defined later on). Let's use two planets per star as a starting guess, and exclude negative numbers with the min keyword.

[7]:
```
epos.fitpars.add('pps',              2.0,    min=0,                    isnorm=True)
```

Initialize the 6 free parameters that define the shape of the 2D period-radius distribution, and their allowed ranges (min,max).

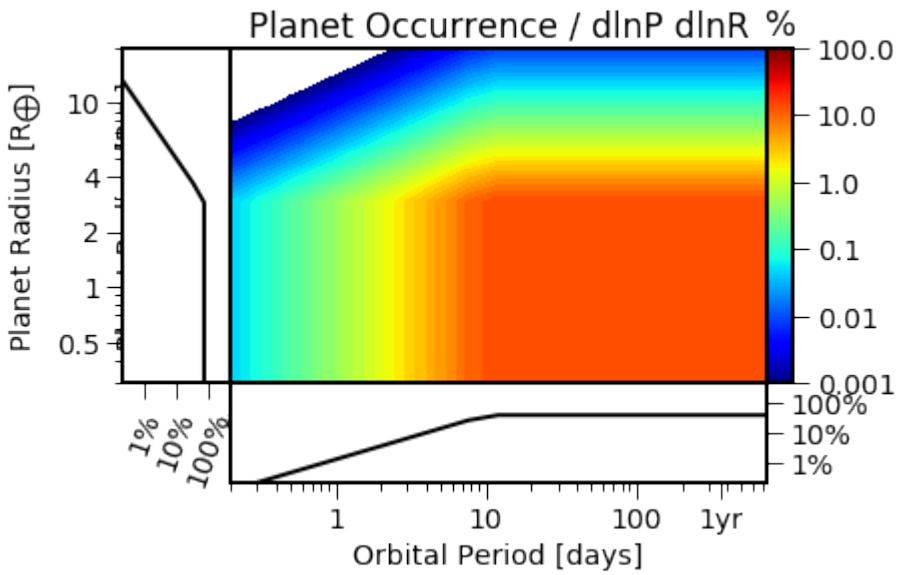
[8]:
```
epos.fitpars.add('P break', 10.,    min=2,  max=50, is2D=True)
epos.fitpars.add('a_P',             1.5,    min=0,                  is2D=True)
epos.fitpars.add('b_P',             0.0,    dx=0.1,                 is2D=True)
epos.fitpars.add('R break', 3.0,    min=1.0,max=5,  is2D=True)
epos.fitpars.add('a_R',             0.0,    dx=0.1,                 is2D=True)
epos.fitpars.add('b_R',             -4.,    fixed=True,     is2D=True)
```

define the simulated range (trim) and the range compared to observations (zoom)

[9]:
```
epos.set_ranges(xtrim=[0,730],ytrim=[0.3,20.],xzoom=[2,400],yzoom=[1,6], Occ=True)
```

Show the intial distribution

[10]:
```
EPOS.plot.parametric.panels(epos, NB=True)
```



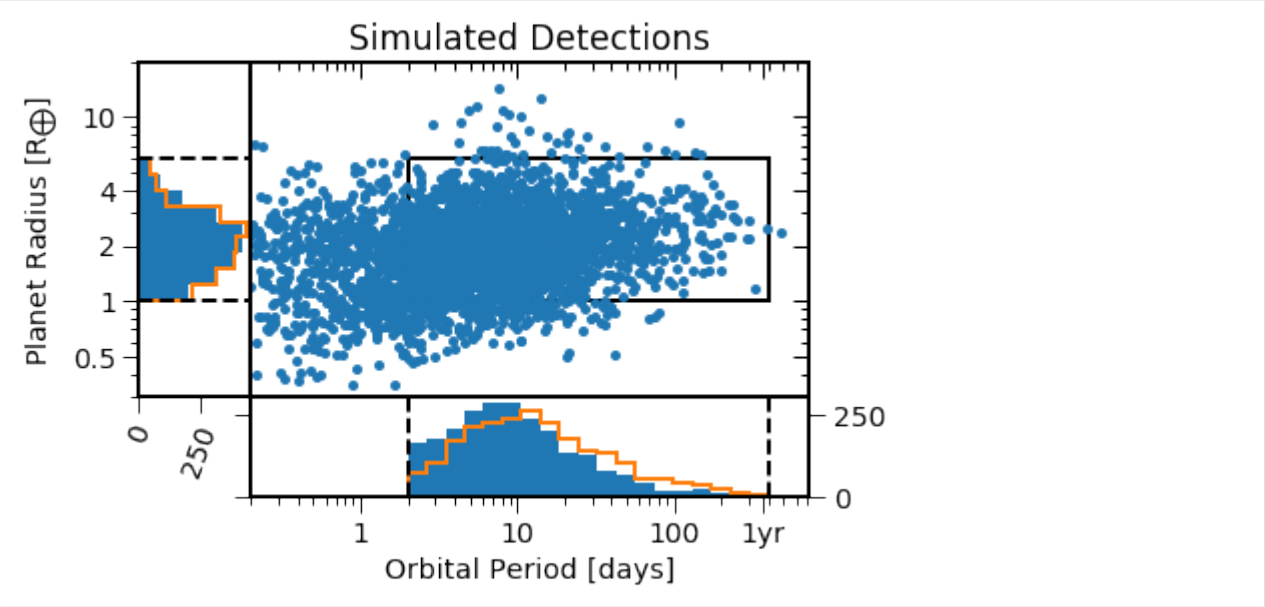
Generate an observable planet population with the inital guess and compare it to Kepler

[11]:
```
EPOS.run.once(epos)
```

```
Preparing EPOS run...
  6 fit parameters

Starting the first MC run

Goodness-of-fit
  logp= -54.1
  - p(n=2398)=0.44
  - p(x)=3.1e-23
  - p(y)=0.24
  observation comparison in 0.003 sec
Finished one MC in 0.063 sec
```

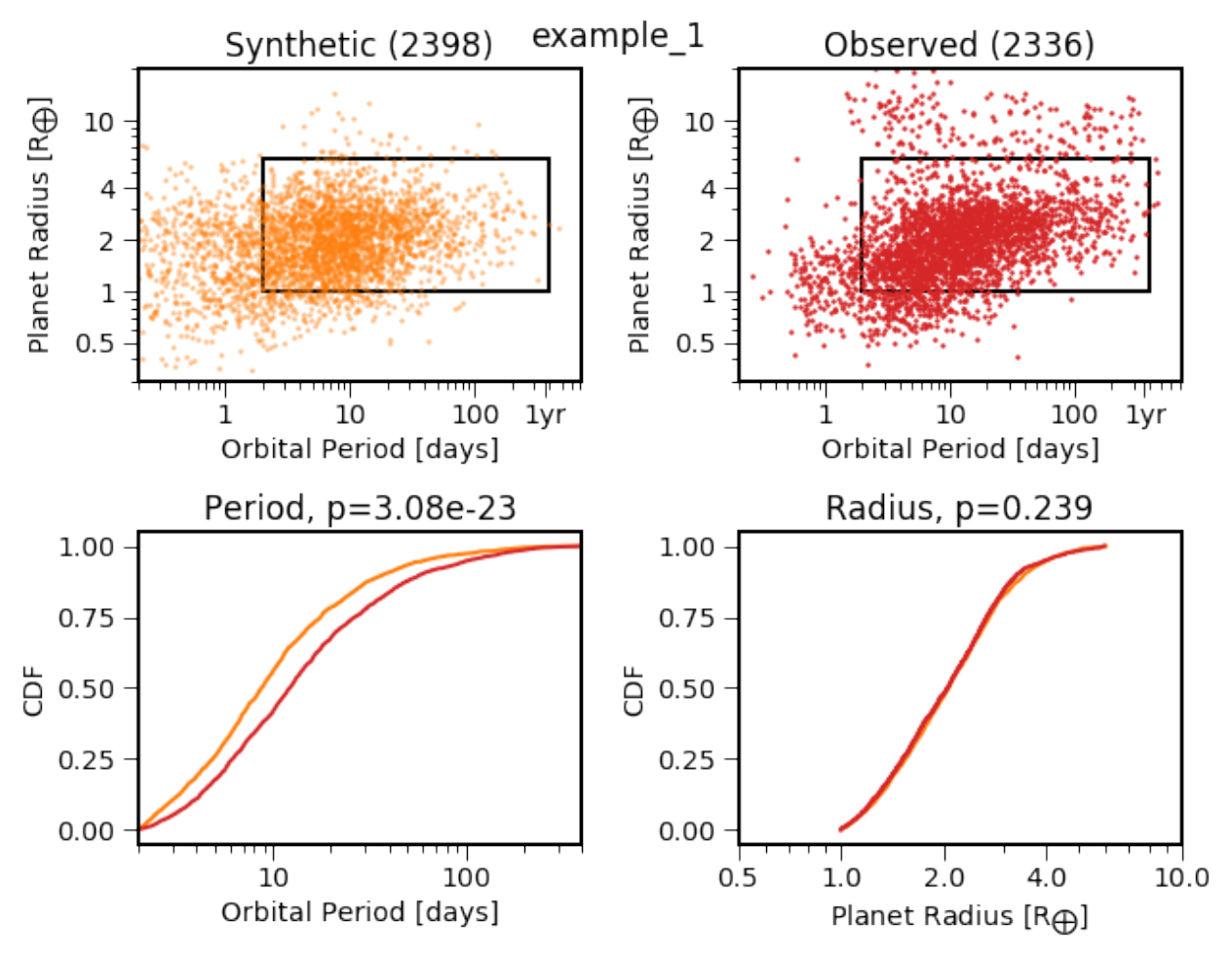
Show the simulated detections

[12]:
```
EPOS.plot.periodradius.panels(epos, NB=True)
```



Show the cumlative distributions used for the summary statistics

[13]:
```
EPOS.plot.periodradius.cdf(epos, NB=True)
```



Looks like the simulated period distribution is a bit different from what is observed. Let's minimize the distance between the distributions using emcee. (Note the counter doesn't work yet)

[14]:
```
EPOS.run.mcmc(epos, nMC=1000, nwalkers=100, nburn=200, threads=8, Saved=True)
```

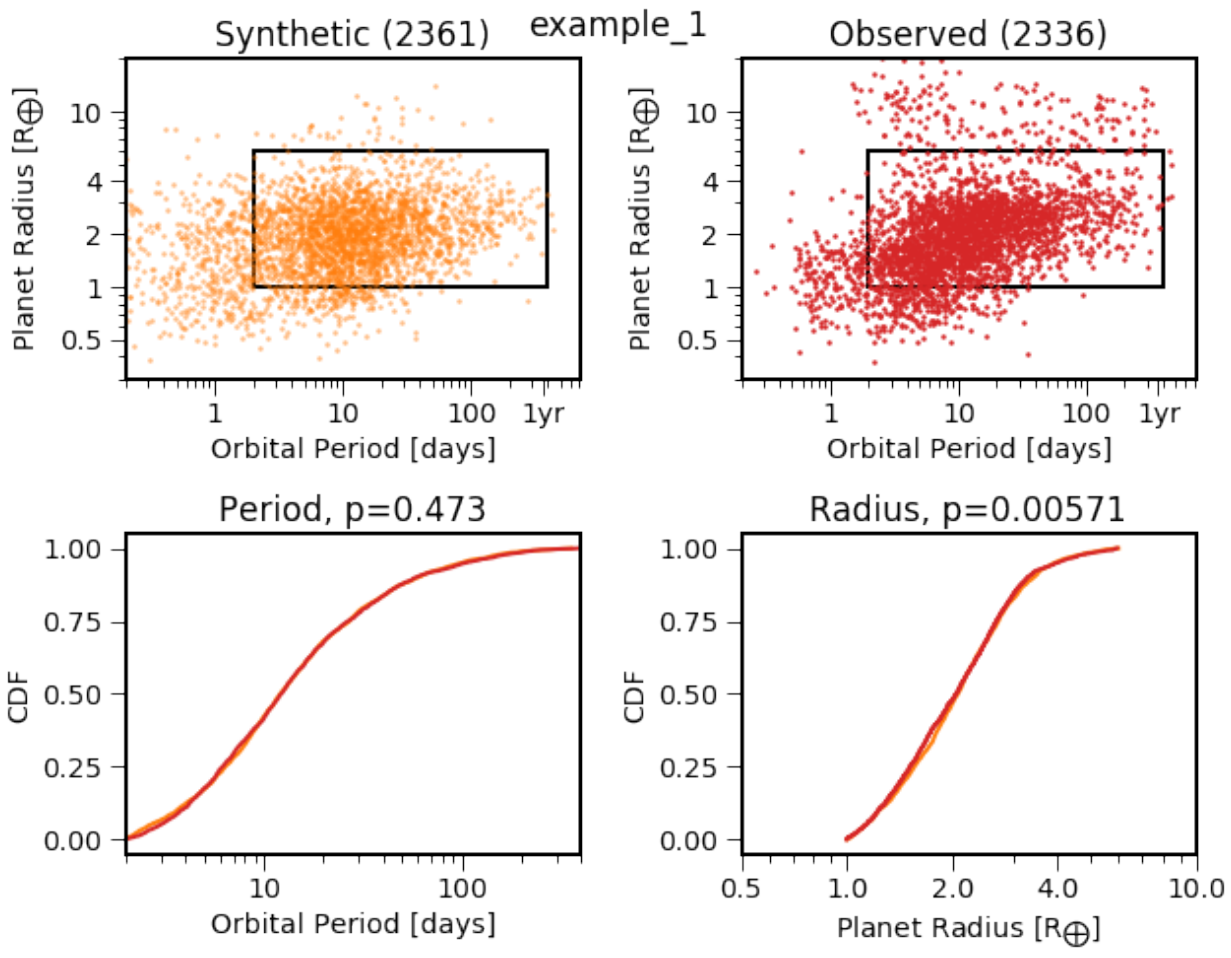
```
Loading saved status from chain/example_1/100x1000x6.npz

NOTE: Random seed changed: 2553310461 to 1099836816

MC-ing the 30 samples to plot

Best-fit values
  pps= 3.92 +1.18 -0.959
  P break= 13 +4.11 -3.24
  a_P= 1.57 +0.444 -0.213
  b_P= 0.184 +0.129 -0.144
  R break= 2.93 +0.159 -0.184
  a_R= -0.29 +0.312 -0.194

Starting the best-fit MC run

Goodness-of-fit
```

(continues on next page)

(continued from previous page)

```
 logp= -6.0
 - p(n=2361)=0.87
 - p(x)=0.47
 - p(y)=0.0057

 Akaike/Bayesian Information Criterion
 - k=6, n=2336
 - BIC= 58.6
 - AIC= 24.1, AICc= 0.9
 observation comparison in 0.004 sec
```

[15]:
```
EPOS.plot.periodradius.cdf(epos, NB=True)
```

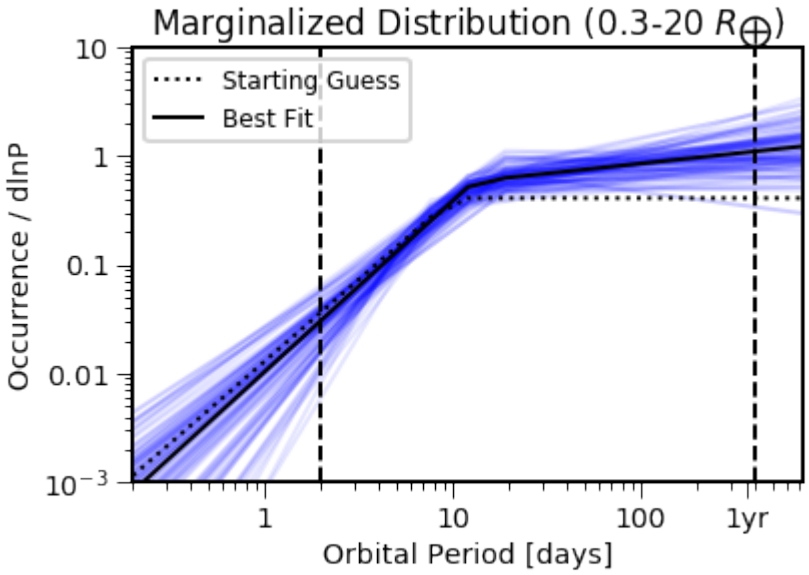


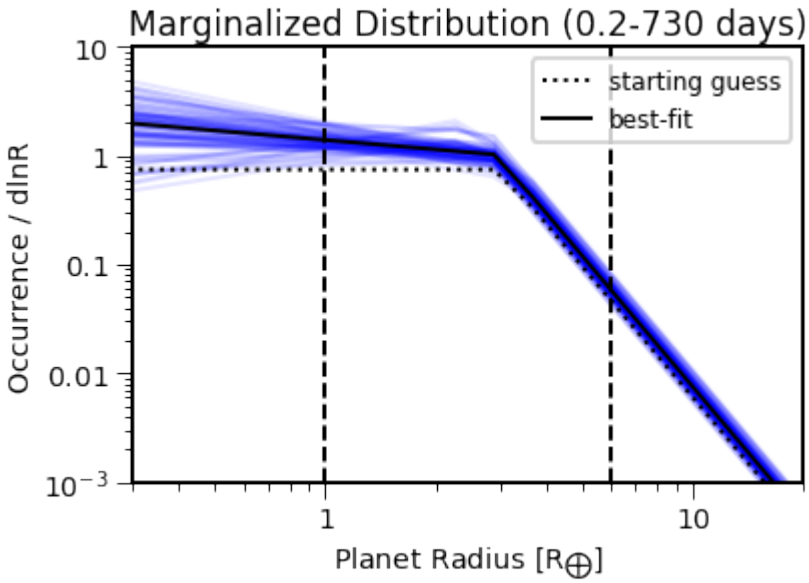
That looks better!

[16]:
```
#EPOS.plot.mcmc.corners(epos, NB=True)
#EPOS.plot.mcmc.chain(epos, NB=True)
```

Let's look at the posterior distributions

[17]:
```
EPOS.plot.parametric.oneD_x(epos, MCMC=True, NB=True)
EPOS.plot.parametric.oneD_y(epos, MCMC=True, NB=True)
```





And compare them to the planet occurrence rates

[18]:
```
EPOS.occurrence.all(epos)
EPOS.plot.parametric.oneD_x(epos, MCMC=True, NB=True, Occ=True)
EPOS.plot.parametric.oneD_y(epos, MCMC=True, NB=True, Occ=True)
```

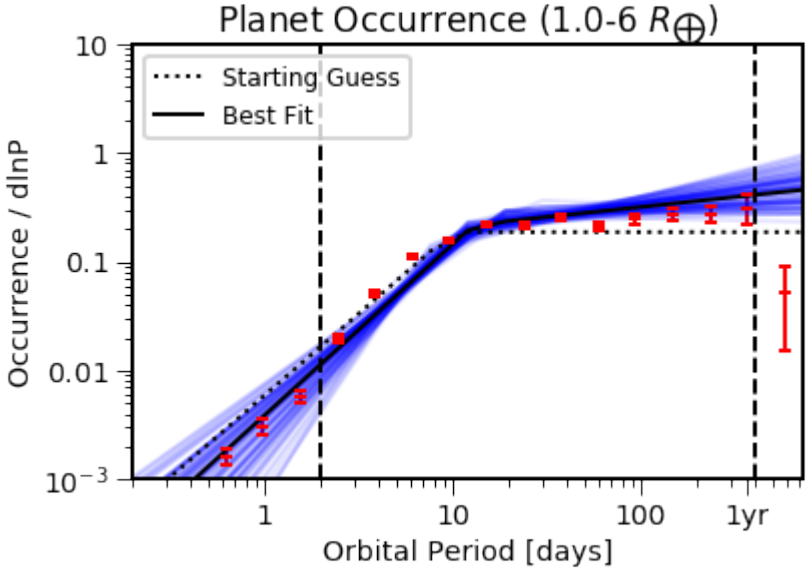
```
Interpolating planet occurrence

  x zoom bins
  x: [2,400], y: [0.25,0.32], n=0, comp=nan, occ=0
  x: [2,400], y: [0.32,0.41], n=1, comp=0.0011, occ=0.0055
  x: [2,400], y: [0.41,0.53], n=8, comp=0.002, occ=0.82
  x: [2,400], y: [0.53,0.67], n=42, comp=0.0067, occ=0.11
  x: [2,400], y: [0.67,0.86], n=106, comp=0.016, occ=0.11
```

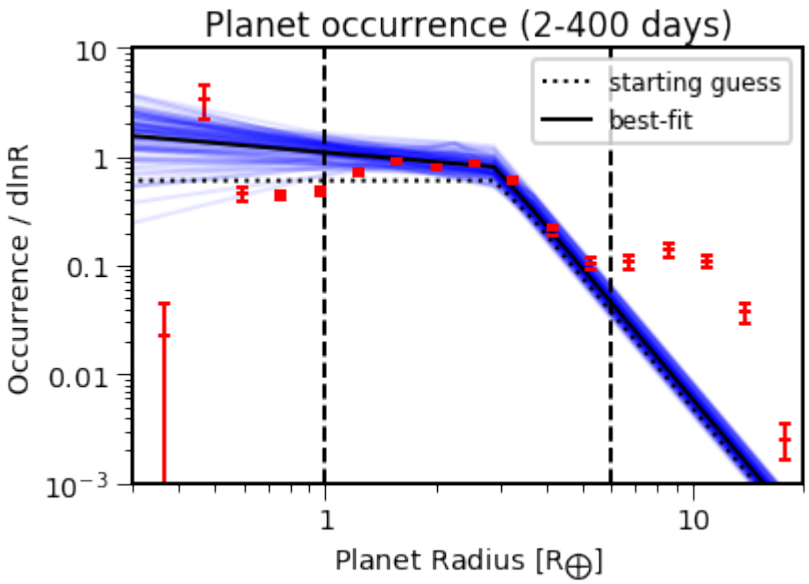
(continues on next page)

(continued from previous page)

```
  x: [2,400], y: [0.86,1.1], n=184, comp=0.029, occ=0.12
  x: [2,400], y: [1.1,1.4], n=346, comp=0.039, occ=0.18
  x: [2,400], y: [1.4,1.8], n=475, comp=0.042, occ=0.22
  x: [2,400], y: [1.8,2.3], n=466, comp=0.035, occ=0.19
  x: [2,400], y: [2.3,2.9], n=520, comp=0.035, occ=0.21
  x: [2,400], y: [2.9,3.7], n=290, comp=0.031, occ=0.15
  x: [2,400], y: [3.7,4.7], n=104, comp=0.037, occ=0.051
  x: [2,400], y: [4.7,6], n=54, comp=0.038, occ=0.025
  x: [2,400], y: [6,7.6], n=46, comp=0.041, occ=0.026
  x: [2,400], y: [7.6,9.7], n=48, comp=0.034, occ=0.034
  x: [2,400], y: [9.7,12], n=51, comp=0.054, occ=0.027
  x: [2,400], y: [12,16], n=25, comp=0.061, occ=0.009
  x: [2,400], y: [16,20], n=8, comp=0.091, occ=0.00062

  y zoom bins
  x: [0.2,0.315], y: [1,6], n=1, comp=0.52, occ=1.2e-05
  x: [0.315,0.498], y: [1,6], n=4, comp=0.41, occ=6.2e-05
  x: [0.498,0.785], y: [1,6], n=32, comp=0.27, occ=0.00075
  x: [0.785,1.24], y: [1,6], n=43, comp=0.2, occ=0.0014
  x: [1.24,1.95], y: [1,6], n=58, comp=0.14, occ=0.0026
  x: [1.95,3.08], y: [1,6], n=135, comp=0.099, occ=0.0088
  x: [3.08,4.86], y: [1,6], n=259, comp=0.072, occ=0.024
  x: [4.86,7.66], y: [1,6], n=381, comp=0.05, occ=0.051
  x: [7.66,12.1], y: [1,6], n=390, comp=0.037, occ=0.072
  x: [12.1,19.1], y: [1,6], n=389, comp=0.027, occ=0.1
  x: [19.1,30.1], y: [1,6], n=267, comp=0.019, occ=0.098
  x: [30.1,47.4], y: [1,6], n=217, comp=0.014, occ=0.12
  x: [47.4,74.8], y: [1,6], n=120, comp=0.0093, occ=0.097
  x: [74.8,118], y: [1,6], n=85, comp=0.0057, occ=0.11
  x: [118,186], y: [1,6], n=58, comp=0.0033, occ=0.12
  x: [186,293], y: [1,6], n=32, comp=0.0021, occ=0.12
  x: [293,463], y: [1,6], n=11, comp=0.0008, occ=0.14
  x: [463,730], y: [1,6], n=2, comp=0.00053, occ=0.024
```

```
/Users/mulders/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2920:␣
↪RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/Users/mulders/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:85:␣
↪RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```





Now let's extrapolate into the Habitable Zone

[19]:
```
epos.set_bins(xbins=[[0.9*365,2.2*365]], ybins=[[0.7,1.5]])
EPOS.occurrence.all(epos)
```

```
Interpolating planet occurrence

  Observed Planets
  x: [328,803], y: [0.7,1.5], n=0, comp=nan, occ=0

  x zoom bins
  x: [2,400], y: [0.25,0.32], n=0, comp=nan, occ=0
  x: [2,400], y: [0.32,0.41], n=1, comp=0.0011, occ=0.0055
  x: [2,400], y: [0.41,0.53], n=8, comp=0.002, occ=0.82
```

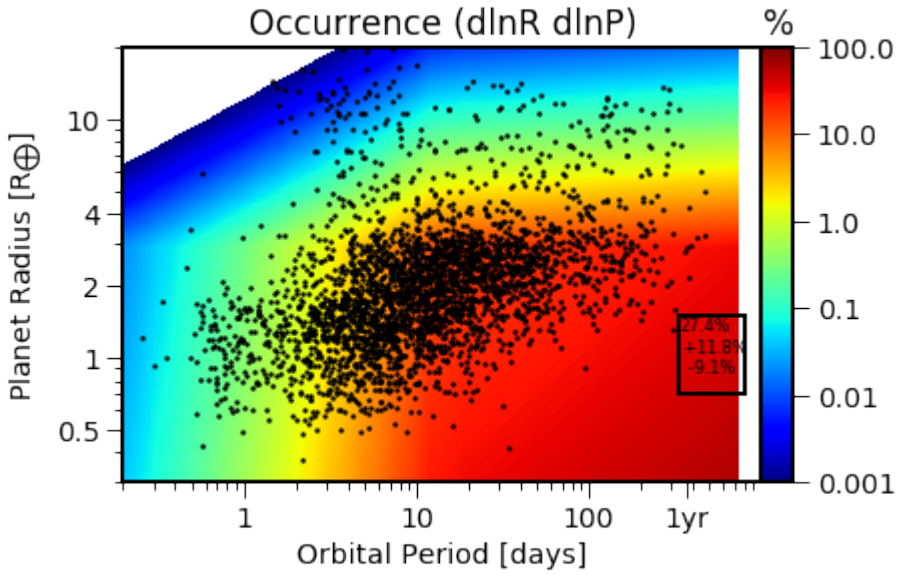
(continues on next page)

(continued from previous page)

```
  x: [2,400], y: [0.53,0.67], n=42, comp=0.0067, occ=0.11
  x: [2,400], y: [0.67,0.86], n=106, comp=0.016, occ=0.11
  x: [2,400], y: [0.86,1.1], n=184, comp=0.029, occ=0.12
  x: [2,400], y: [1.1,1.4], n=346, comp=0.039, occ=0.18
  x: [2,400], y: [1.4,1.8], n=475, comp=0.042, occ=0.22
  x: [2,400], y: [1.8,2.3], n=466, comp=0.035, occ=0.19
  x: [2,400], y: [2.3,2.9], n=520, comp=0.035, occ=0.21
  x: [2,400], y: [2.9,3.7], n=290, comp=0.031, occ=0.15
  x: [2,400], y: [3.7,4.7], n=104, comp=0.037, occ=0.051
  x: [2,400], y: [4.7,6], n=54, comp=0.038, occ=0.025
  x: [2,400], y: [6,7.6], n=46, comp=0.041, occ=0.026
  x: [2,400], y: [7.6,9.7], n=48, comp=0.034, occ=0.034
  x: [2,400], y: [9.7,12], n=51, comp=0.054, occ=0.027
  x: [2,400], y: [12,16], n=25, comp=0.061, occ=0.009
  x: [2,400], y: [16,20], n=8, comp=0.091, occ=0.00062

  y zoom bins
  x: [0.2,0.315], y: [1,6], n=1, comp=0.52, occ=1.2e-05
  x: [0.315,0.498], y: [1,6], n=4, comp=0.41, occ=6.2e-05
  x: [0.498,0.785], y: [1,6], n=32, comp=0.27, occ=0.00075
  x: [0.785,1.24], y: [1,6], n=43, comp=0.2, occ=0.0014
  x: [1.24,1.95], y: [1,6], n=58, comp=0.14, occ=0.0026
  x: [1.95,3.08], y: [1,6], n=135, comp=0.099, occ=0.0088
  x: [3.08,4.86], y: [1,6], n=259, comp=0.072, occ=0.024
  x: [4.86,7.66], y: [1,6], n=381, comp=0.05, occ=0.051
  x: [7.66,12.1], y: [1,6], n=390, comp=0.037, occ=0.072
  x: [12.1,19.1], y: [1,6], n=389, comp=0.027, occ=0.1
  x: [19.1,30.1], y: [1,6], n=267, comp=0.019, occ=0.098
  x: [30.1,47.4], y: [1,6], n=217, comp=0.014, occ=0.12
  x: [47.4,74.8], y: [1,6], n=120, comp=0.0093, occ=0.097
  x: [74.8,118], y: [1,6], n=85, comp=0.0057, occ=0.11
  x: [118,186], y: [1,6], n=58, comp=0.0033, occ=0.12
  x: [186,293], y: [1,6], n=32, comp=0.0021, occ=0.12
  x: [293,463], y: [1,6], n=11, comp=0.0008, occ=0.14
  x: [463,730], y: [1,6], n=2, comp=0.00053, occ=0.024

  posterior per bin
  x: [328,803], y: [0.7,1.5], area=0.68, eta_0=0.1
  gamma= 40.2% +17.4% -13.4%
  eta= 27.4% +11.8% -9.1%

  Binned occurrence rate metrics
  x: (n=12, k=6)
    chi^2= 147.3, reduced= 24.6
    bic= 45.0
    aic= 71.9, AICc= 1208.1
  y: (n=7, k=6)
    chi^2= 79.3, reduced= 79.3
    bic= 28.7
    aic= 42.6, AICc= inf
/Users/mulders/EPOS/EPOS/analytics.py:67: RuntimeWarning: divide by zero encountered␣
↪in double_scalars
  cfactor= (2.* k_free**2. + 2.*k_free) / (n_data- k_free - 1.)
```

And visualize the distribution

[20]:
```
epos.plotpars['textsize']= 8 # Shrink text to fit in the box
epos.xtrim[1]= 1000 # Adjust plot axes
EPOS.plot.occurrence.integrated(epos, MCMC=True,Planets=True, NB=True)
```



[ ]:

3.6 Multi-planet Mode

Estimate the intrinsic distribution of planetary systems by fitting a population of multi-planet systems to Kepler data

[1]:
```
import EPOS
import numpy as np
import matplotlib.pyplot as plt
```

initialize the EPOS class

[2]:
```
epos= EPOS.epos(name='example_2')
```

```
 |~| epos 3.0.0.dev2 |~|


Using random seed 493439428
```

Read in the kepler dr25 exoplanets and survey efficiency packaged with EPOS

[3]:
```
obs, survey= EPOS.kepler.dr25(Huber=True, Vetting=True, score=0.9)
epos.set_observation(**obs)
epos.set_survey(**survey)
```

```
Loading planets from temp/q1_q17_dr25_koi.npz
  6853/7995 dwarfs
```

(continues on next page)

(continued from previous page)

```
  3525 candidates, 3328 false positives
  3040+1 with score > 0.90

Observations:
  159238 stars
  3041 planets

  1840 singles, 487 multis
  - single: 1840
  - double: 324
  - triple: 113
  - quad: 38
  - quint: 10
  - sext: 2
```

Define the function that describes the intrinsic planetary system population. Here we use a double broken power-law from EPOS.fitfunctions to deinfe the location and size of the innermost planet in each system.

[4]:
```
epos.set_parametric(EPOS.fitfunctions.brokenpowerlaw2D)
```

brokenpowerlaw2D takes 8 parameters. The two dependent parameters are the period and radius. There are 6 free parameters (xp, p1, p2, yp, p3, p4) and a normalization parameter. Let's define them:

The normalization parameter, labeled pps, defines the fraction of stars with planetary systems. Let's assign a planetary system to 40% of stars, and exclude negative numbers with the min keyword.

[5]:
```
epos.fitpars.add('pps', 0.4, min=0, isnorm=True)
```

Initialize the 6 parameters that define the distribution of the innermost planets, fixing the radius distribution.

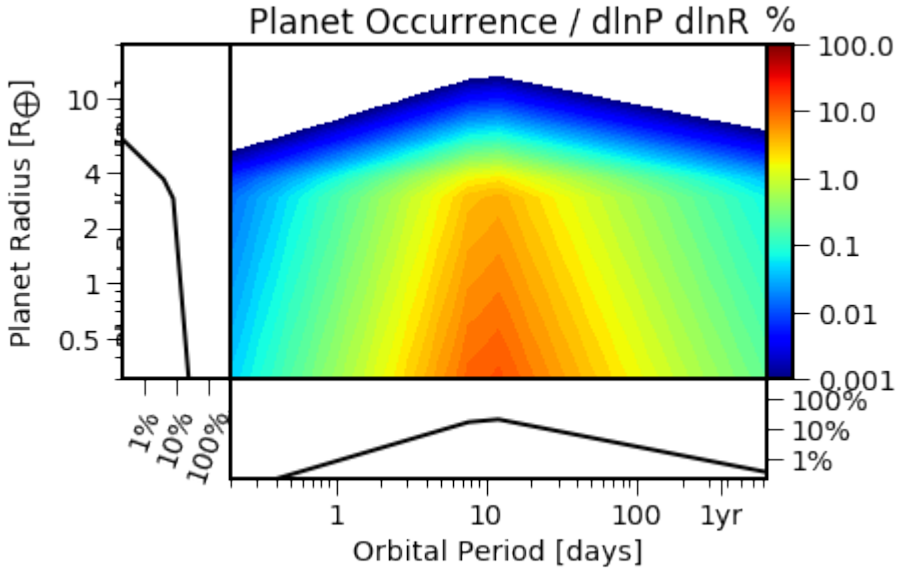
[6]:
```
epos.fitpars.add('P break', 10., min=2, max=50, is2D=True)
epos.fitpars.add('a_P', 1.5, min=0, is2D=True)
epos.fitpars.add('b_P', -1, max=1, dx=0.1, is2D=True)
epos.fitpars.add('R break', 3.3, fixed=True, is2D=True)
epos.fitpars.add('a_R', -0.5, fixed=True, is2D=True)
epos.fitpars.add('b_R', -6., fixed=True, is2D=True)
```

define the simulated range (trim) and the range compared to observations (zoom)

[7]:
```
epos.set_ranges(xtrim=[0,730],ytrim=[0.3,20.],xzoom=[2,400],yzoom=[1,6], Occ=True)
```

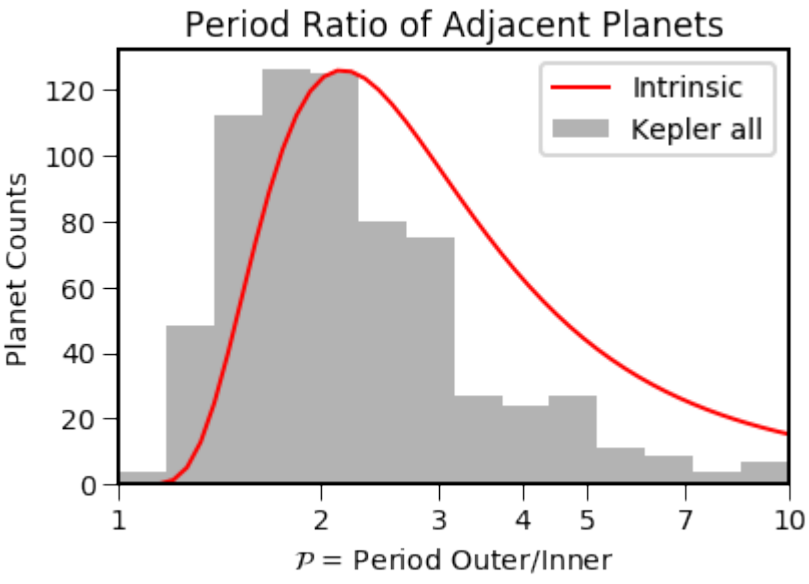
Show the inner planet distribution

[8]:
```
EPOS.plot.parametric.panels(epos, NB=True)
```



Define the locations of additional planets in the system. Here, let's use 10 planets per system, with the spacing drawn from a dimensionless distribution

[9]:
```
epos.set_multi(spacing='dimensionless')
epos.fitpars.add('npl', 10, fixed=True) # planets per system
epos.fitpars.add('log D', -0.3)
epos.fitpars.add('sigma', 0.2, min=0)
EPOS.plot.multi.periodratio(epos, Input=True, MC=False, NB=True)
```



Define the mutual inclinations, drawn from Rayleigh distribution with mode 2 degrees

[10]:
```
epos.fitpars.add('inc', 2.0)                                    # mode of mutual␣
↪inclinations
```

Fraction of system with high mutual inclunations to fit the Kepler dichotomy

[11]:
```
epos.fitpars.add('f_iso', 0.4)                                    # Fraction of isotropic␣
↪systems
```

Add a dispersion to the radii of planets in each system

[12]:
```
epos.fitpars.add('dR', 0.01, fixed=True)
```

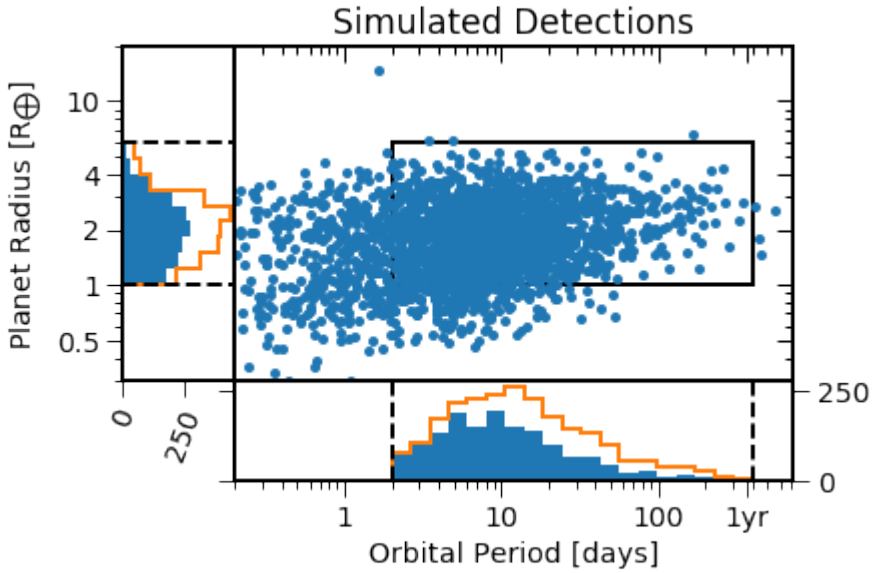
Generate an observable planet population with the inital guess and compare it to Kepler

[13]:
```
EPOS.run.once(epos)
```

```
Preparing EPOS run...
  6 fit parameters
  Set f_cor to default 0.5

Starting the first MC run
  63695/356370 systems
  Average mutual inc=2.4 degrees

  356370 planets, 9647 transit their star
  - single: 4632
  - double: 1082
  - triple: 456
  - quad: 217
  - quint: 81
  - sext: 24
  - sept: 6
  - oct: 3
  9647 transiting planets, 2145 detectable
  - single: 1263
  - double: 226
  - triple: 87
  - quad: 29
  - quint: 7
  - sext: 3

Goodness-of-fit
  logp= -160.4
  - p(n=1548)=1.9e-58
  - p(x)=3.5e-08
  - p(N_k)=0.84
  - p(P ratio)=0.027
  - p(P inner)=0.0014
  observation comparison in 0.044 sec
Finished one MC in 0.475 sec
```

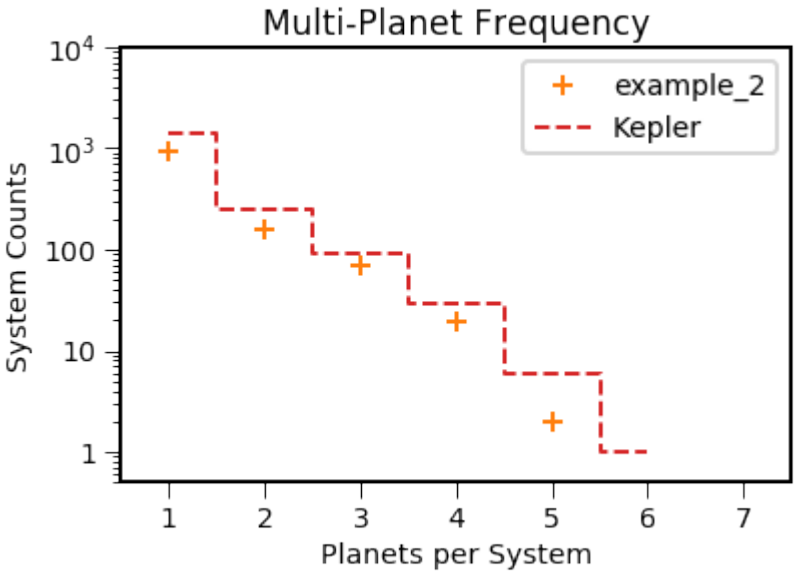
Show the simulated detections

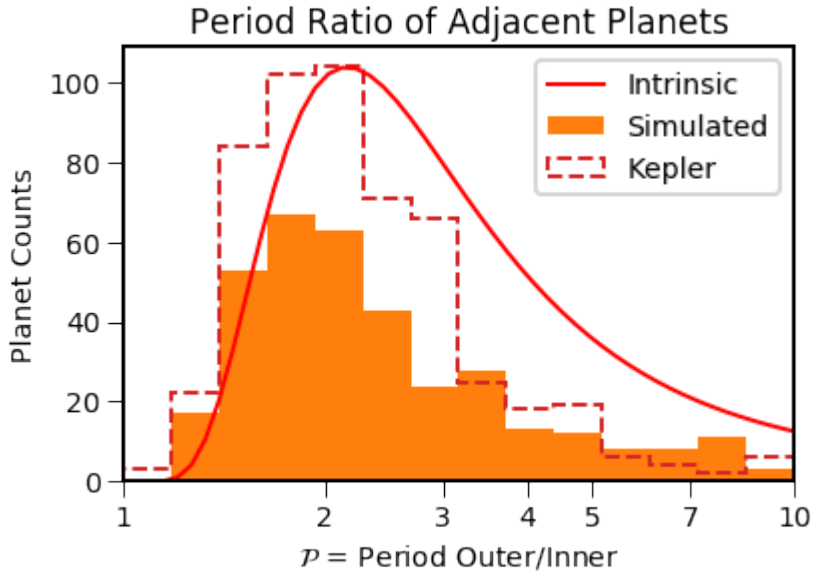
[14]:
```
EPOS.plot.periodradius.panels(epos, NB=True)
```

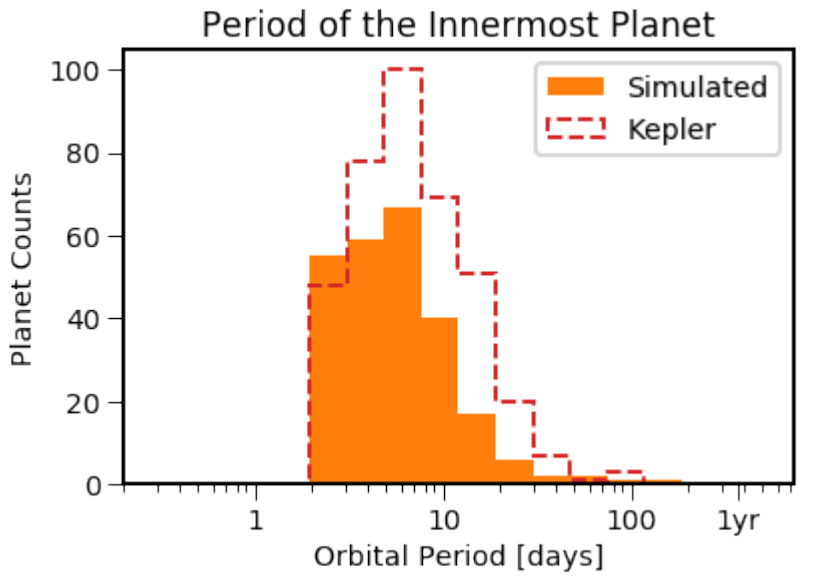


And the detectable planet architectures, compared to the initial distributions

[15]:
```
EPOS.plot.multi.multiplicity(epos, MC=True, NB=True)
EPOS.plot.multi.periodratio(epos, Input=True, MC=True, NB=True)
EPOS.plot.multi.periodinner(epos, MC=True, NB=True)
```







The simulated distributions are a bit different from what is observed. Let's minimize the distance between the distributions using emcee. (Note the counter doesn't work yet)

[16]:
```
EPOS.run.mcmc(epos, nMC=100, nwalkers=20, nburn=20, threads= 8, Saved=True) # ~20 mins
#EPOS.run.mcmc(epos, nMC=500, nwalkers=50, nburn=200, threads= 8, Saved=True) # ~20
↪mins
#EPOS.run.mcmc(epos, nMC=1000, nwalkers=100, nburn=200, threads=20, Saved=True) # ~5
↪hrs
```

```
Loading saved status from chain/example_2/20x100x8.npz

NOTE: Random seed changed: 3119675631 to 493439428

MC-ing the 30 samples to plot
Mercury analogues < 3.3% +1.2% -1.1%
```

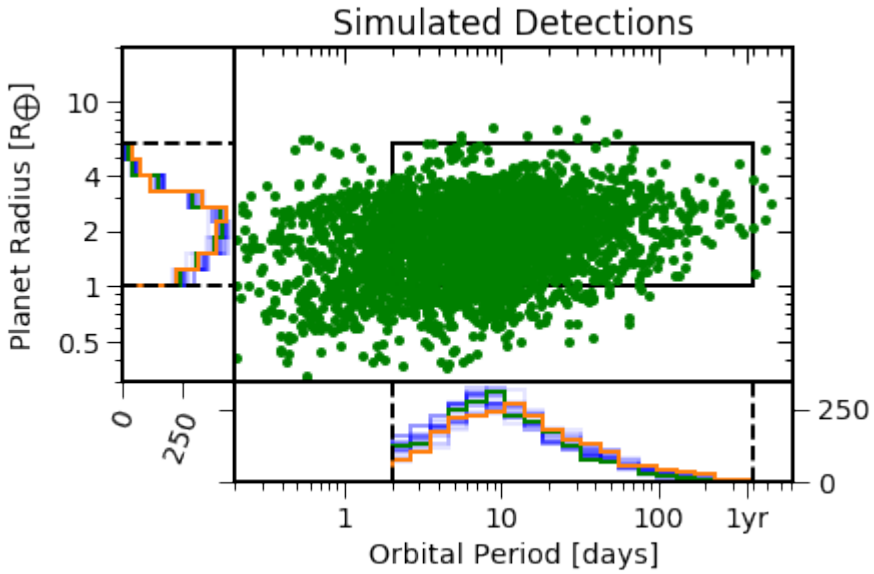
(continues on next page)

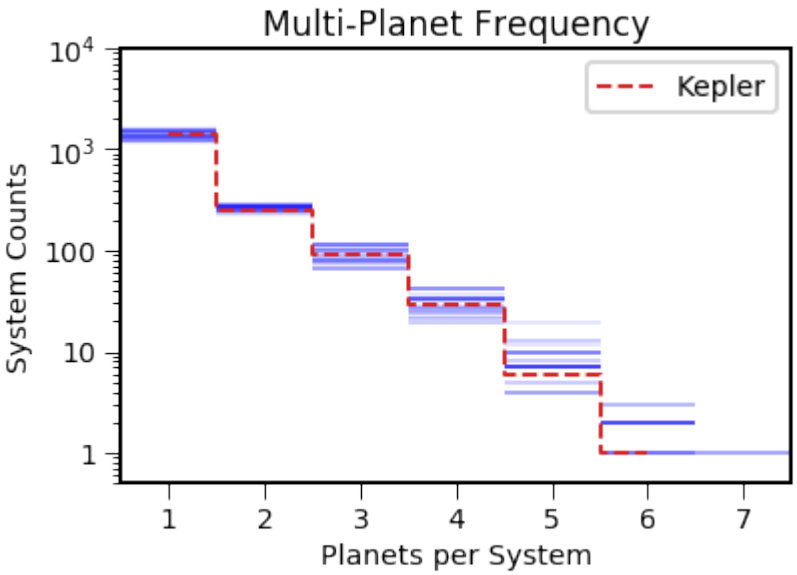
(continued from previous page)

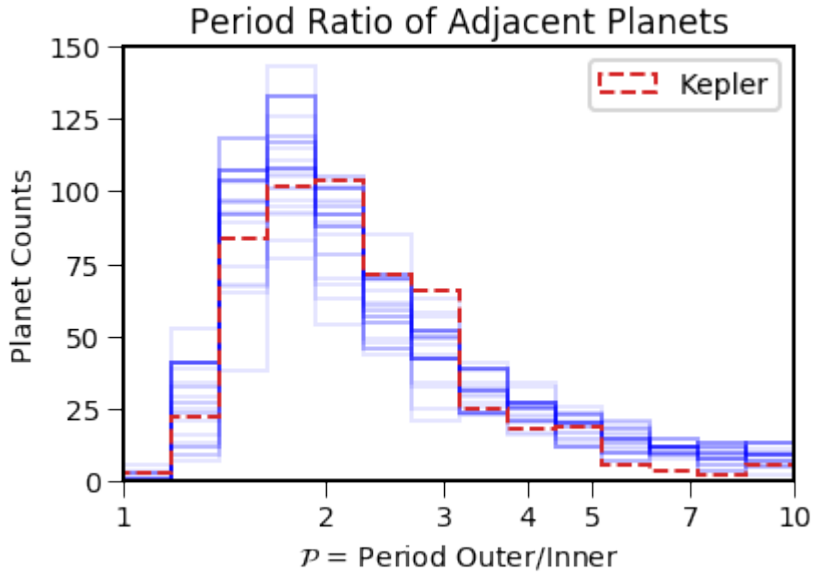
```
  1 sigma UL 3.9%
  2 sigma UL 5.1%
  3 sigma UL 12.2%
Venus analogues < 0.9% +0.4% -0.3%
  1 sigma UL 1.2%
  2 sigma UL 1.6%
  3 sigma UL 4.9%

Best-fit values
  pps= 0.57 +0.0583 -0.0365
  P break= 10.8 +1.87 -1.39
  a_P= 1.61 +0.246 -0.191
  b_P= -1.3 +0.171 -0.107
  log D= -0.345 +0.0311 -0.0343
  sigma= 0.187 +0.0225 -0.019
  inc= 1.92 +0.684 -0.67
  f_iso= 0.418 +0.0635 -0.0459

Starting the best-fit MC run
  90776/565953 systems
  Average mutual inc=2.3 degrees

  565953 planets, 14997 transit their star
  - single: 6926
  - double: 1572
  - triple: 783
  - quad: 353
  - quint: 133
  - sext: 52
  - sept: 20
  - oct: 5
  - nint: 1
  14997 transiting planets, 3087 detectable
  - single: 1743
  - double: 354
  - triple: 135
  - quad: 34
  - quint: 14
  - sext: 3
  - sept: 1

Goodness-of-fit
  logp= -13.7
  - p(n=2277)=0.47
  - p(x)=0.0014
  - p(N_k)=0.94
  - p(P ratio)=0.11
  - p(P inner)=0.016

  Akaike/Bayesian Information Criterion
  - k=8, n=2336
  - BIC= 89.4
  - AIC= 43.4, AICc= 2.7
  observation comparison in 0.083 sec
```

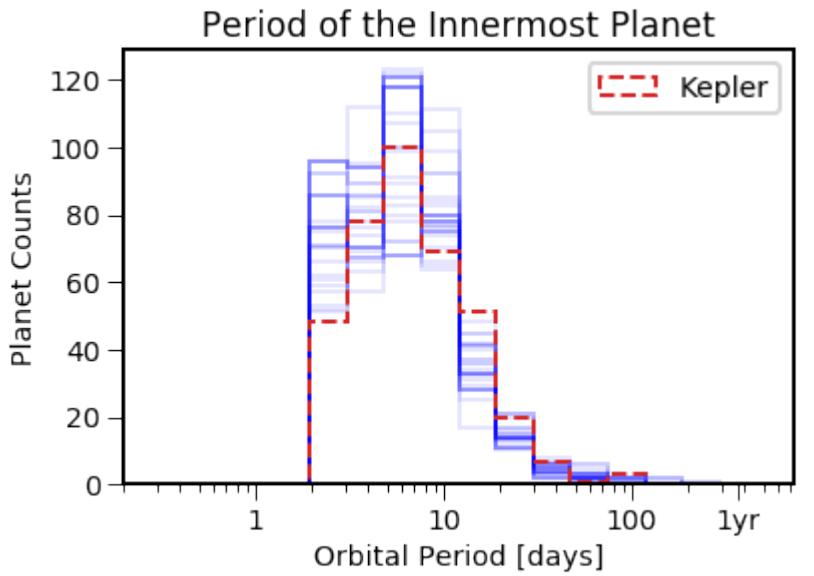
Let's look at the posterior distributions

[17]:
```
EPOS.plot.periodradius.panels(epos, MCMC=True, NB=True)
EPOS.plot.multi.multiplicity(epos, MCMC=True, MC=True, NB=True)
EPOS.plot.multi.periodratio(epos, MCMC=True, MC=True, NB=True)
EPOS.plot.multi.periodinner(epos, MCMC=True, MC=True, NB=True)
```









Save the planet populations to a csv file

[18]:
```
popdict= dict(all_systems=epos.population,
              transiting_systems= epos.population['system'],
             transiting_planets= epos.synthetic_survey)

for key, pop in popdict.items():
    fname= 'csv/'+epos.name+'_'+key
    EPOS.save.to_csv(fname, starID= pop['ID'], Period_days= pop['P'], Radius_earth=␣
↪pop['Y'])
```

[ ]:
```
```

3.7 Radial Velocity

Example script fitting a power-law in period and M sin i to radial velocity data

First, let's import EPOS

[1]:
```
import EPOS
```

Then intialize the EPOS class for use with a radial velocity survey using the RV keyword. We don't want to use the Monte Carlo simulation because of the small sample size (MC=False). And we want to fit the instrinsic mass distribution, so we need to convert to M sin i before fitting the observed planets (Msini=False).

[2]:
```
epos= EPOS.epos(name='radial_velocity', RV=True, MC=False, Msini=True)
```

```
 |~| epos 3.0.0.dev4 |~|


Using random seed 322504493
Survey: None selected
```

Now we load the detected exoplanets and the survey completeness from the Mayor+ 2011 paper

[3]:
```
obs, survey= EPOS.rv.Mayor2011()
epos.set_observation(**obs, Verbose=False)
epos.set_survey(**survey)
```

We are going to use a double broken power-law in mass and period to parametrize the intrinsic planet population. This function is pre-defined in the EPOS.fitfunctions module

[4]:
```
epos.set_parametric(EPOS.fitfunctions.brokenpowerlaw2D)
```

Let's make an initial guess for each parameter. For the MCMC to run smoothly, we also define an upper and lower bound to some parameters with the min and max keywords.

The first parameter is a normalization parameter that defines the number of planets per star in the simulated range:

[5]:
```
epos.fitpars.add('pps', 1.0, min=1e-3)
```

brokenpowerlaw2D uses 6 parameters, each indicated with the is2D keyword.

The first three parameters are the location of the break and the power-law indices before and after the break

[6]:
```
epos.fitpars.add('P break', 1e3, min=100, max=7e3, is2D=True)
epos.fitpars.add('a_P', 1.0, min=0, max=3, is2D=True)
epos.fitpars.add('b_P', -0.5, min=-3, max=0, is2D=True)
```

The next three parameter define the planet mass distribution. We only use a single power-law in this example, so we fix the other two parameters.

[7]:
```
epos.fitpars.add('M break', 10.0, fixed=True, is2D=True)
epos.fitpars.add('a_M', 0.0, fixed=True, is2D=True)
epos.fitpars.add('b_M', -0.5, dx=0.1, is2D=True)
```

Next, let's define the range of planet parameters that is simulated (trim) and the range compared to observations (zoom)

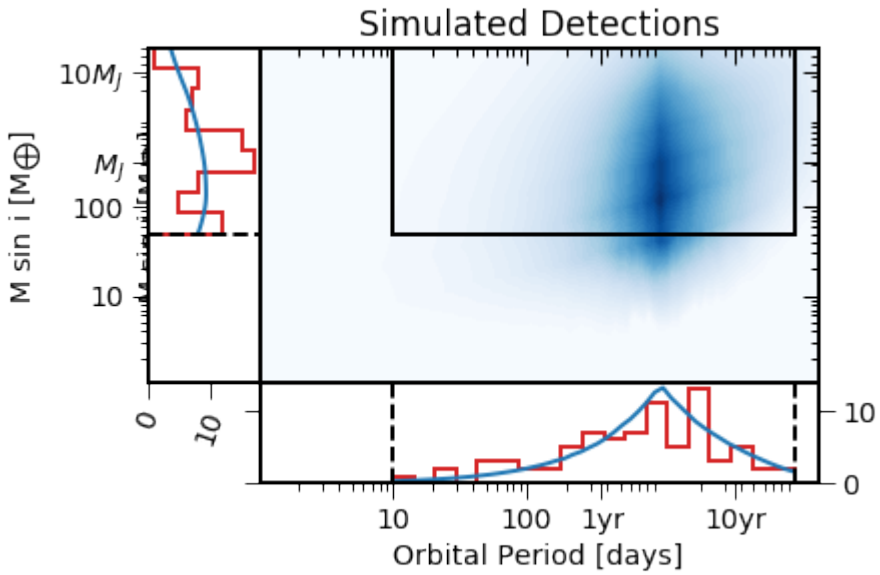
Units are period (days) and planet mass (earth mass)

[8]:
```
epos.set_ranges(xtrim=[1,1e5],ytrim=[1,1e5],xzoom=[10,1e4],yzoom=[50,1e4], Occ=True)
```

Now, let's run epos once to see what out initial distribution looks like and how it compares to observations

[9]:
```
EPOS.run.once(epos)
EPOS.plot.periodradius.panels(epos, NB=True)
```

```
Preparing EPOS run...
  6 fit parameters

Starting the first noMC run
Finished one noMC in 0.007 sec
```



Blue is the simulated distribution, red the detected planets.

Now let's calculate the occurrence rates for comparison

[10]:
```
EPOS.occurrence.all(epos)
epos.plotpars['occrange']= [2e-4,2.]
EPOS.plot.parametric.oneD_x(epos, Occ=True, NB=True)
EPOS.plot.parametric.oneD_y(epos, Occ=True, NB=True)
```

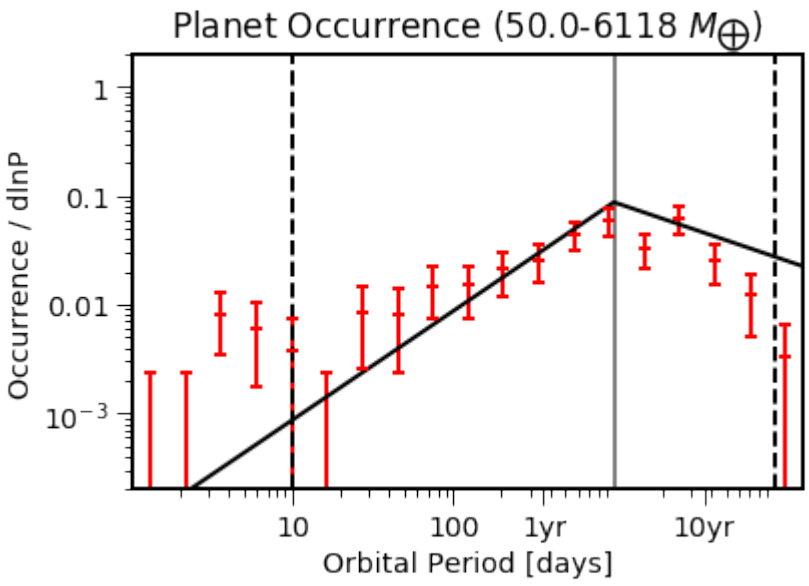
```
Interpolating planet occurrence

  x zoom bins
  x: [10,1e+04], y: [1,1.7], n=0, comp=nan, occ=0
  x: [10,1e+04], y: [1.7,2.6], n=1, comp=0.0027, occ=0.45
  x: [10,1e+04], y: [2.6,4.1], n=3, comp=0.024, occ=0.17
  x: [10,1e+04], y: [4.1,6.5], n=6, comp=0.079, occ=0.13
  x: [10,1e+04], y: [6.5,10], n=7, comp=0.16, occ=0.061
  x: [10,1e+04], y: [10,16], n=17, comp=0.19, occ=0.11
  x: [10,1e+04], y: [16,26], n=14, comp=0.2, occ=0.097
  x: [10,1e+04], y: [26,40], n=4, comp=0.23, occ=0.021
  x: [10,1e+04], y: [40,64], n=7, comp=0.34, occ=0.028
  x: [10,1e+04], y: [64,1e+02], n=9, comp=0.36, occ=0.034
  x: [10,1e+04], y: [1e+02,1.6e+02], n=4, comp=0.55, occ=0.009
```

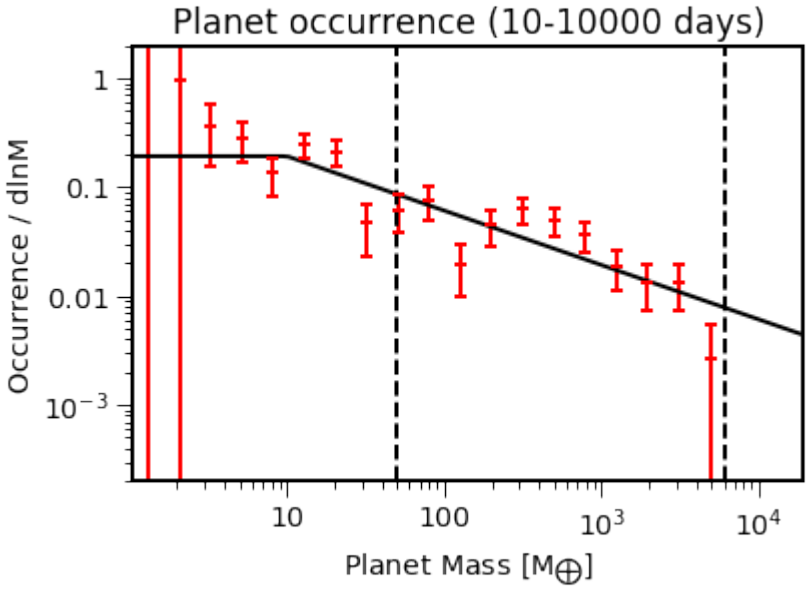
(continues on next page)

(continued from previous page)

```
  x: [10,1e+04], y: [1.6e+02,2.5e+02], n=8, comp=0.5, occ=0.02
  x: [10,1e+04], y: [2.5e+02,4e+02], n=14, comp=0.61, occ=0.029
  x: [10,1e+04], y: [4e+02,6.2e+02], n=13, comp=0.71, occ=0.022
  x: [10,1e+04], y: [6.2e+02,9.9e+02], n=10, comp=0.76, occ=0.017
  x: [10,1e+04], y: [9.9e+02,1.6e+03], n=6, comp=0.86, occ=0.0086
  x: [10,1e+04], y: [1.6e+03,2.5e+03], n=5, comp=0.99, occ=0.0062
  x: [10,1e+04], y: [2.5e+03,3.9e+03], n=5, comp=0.99, occ=0.0062
  x: [10,1e+04], y: [3.9e+03,6.1e+03], n=1, comp=0.99, occ=0.0012

  y zoom bins
  x: [1.02,1.69], y: [50,6.1e+03], n=0, comp=nan, occ=0
  x: [1.69,2.8], y: [50,6.1e+03], n=0, comp=nan, occ=0
  x: [2.8,4.63], y: [50,6.1e+03], n=3, comp=0.89, occ=0.0041
  x: [4.63,7.68], y: [50,6.1e+03], n=2, comp=0.83, occ=0.0031
  x: [7.68,12.7], y: [50,6.1e+03], n=1, comp=0.64, occ=0.0019
  x: [12.7,21.1], y: [50,6.1e+03], n=0, comp=nan, occ=0
  x: [21.1,35], y: [50,6.1e+03], n=2, comp=0.58, occ=0.0043
  x: [35,58.1], y: [50,6.1e+03], n=2, comp=0.7, occ=0.0041
  x: [58.1,96.3], y: [50,6.1e+03], n=4, comp=0.72, occ=0.0074
  x: [96.3,160], y: [50,6.1e+03], n=4, comp=0.65, occ=0.0076
  x: [160,265], y: [50,6.1e+03], n=5, comp=0.68, occ=0.011
  x: [265,439], y: [50,6.1e+03], n=7, comp=0.74, occ=0.013
  x: [439,727], y: [50,6.1e+03], n=12, comp=0.71, occ=0.022
  x: [727,1.21e+03], y: [50,6.1e+03], n=13, comp=0.64, occ=0.03
  x: [1.21e+03,2e+03], y: [50,6.1e+03], n=8, comp=0.68, occ=0.016
  x: [2e+03,3.31e+03], y: [50,6.1e+03], n=12, comp=0.59, occ=0.031
  x: [3.31e+03,5.5e+03], y: [50,6.1e+03], n=6, comp=0.66, occ=0.013
  x: [5.5e+03,9.11e+03], y: [50,6.1e+03], n=3, comp=0.6, occ=0.0061
  x: [9.11e+03,1.51e+04], y: [50,6.1e+03], n=1, comp=0.72, occ=0.0017
```

```
/Users/mulders/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:3118:␣
↪RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/Users/mulders/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:85:␣
↪RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```





The model population (black) looks reasonably close to the estimated occurrence rates (red).

Now let's run the MCMC chain to generate a planet population that better matches the data.

(This can take a long time, so the Saved keyword allows you to read in a previously saved run if it exists)

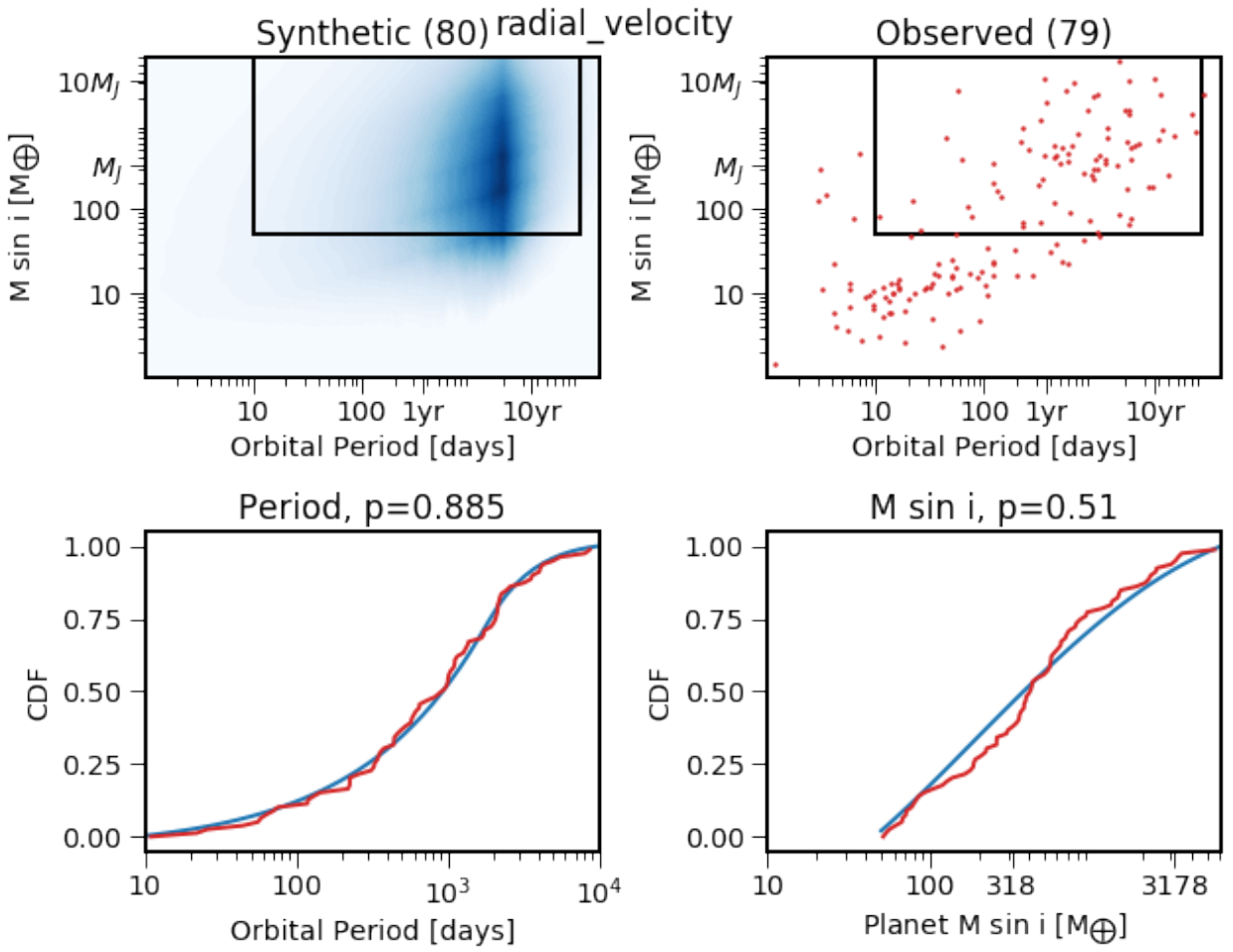
[11]:
```
EPOS.run.mcmc(epos, nMC=1000, nwalkers=100, nburn=200, threads=20, Saved=True)
EPOS.plot.periodradius.cdf(epos, NB=True)
```

```
Loading saved status from chain/radial_velocity/100x1000x5.npz

NOTE: Random seed changed: 1885962102 to 322504493

MC-ing the 30 samples to plot

Best-fit values
  pps= 0.94 +0.237 -0.184
  P break= 1.99e+03 +1.12e+03 -1.11e+03
  a_P= 0.706 +0.324 -0.155
  b_P= -1.06 +0.786 -1.29
  b_M= -0.462 +0.0569 -0.0635

Starting the best-fit MC run
nobs=80 (x:88,y:73)

Goodness-of-fit
  logp= -0.8
  - p(n=80)=0.99
  - p(x)=0.88
  - p(y)=0.51

  Akaike/Bayesian Information Criterion
  - k=5, n=79
  - BIC= 23.5
  - AIC= 11.6, AICc= 9.5
```

(continues on next page)

(continued from previous page)

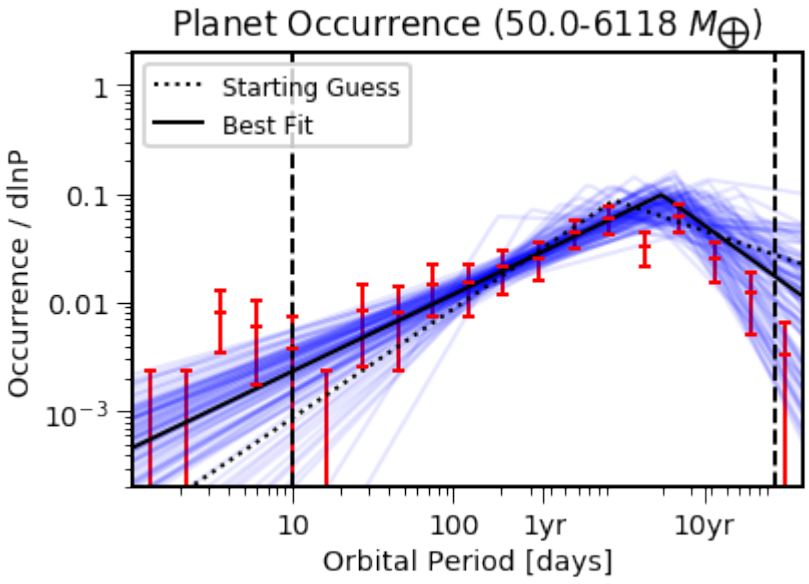
```
Observation comparison in 0.001 sec
```

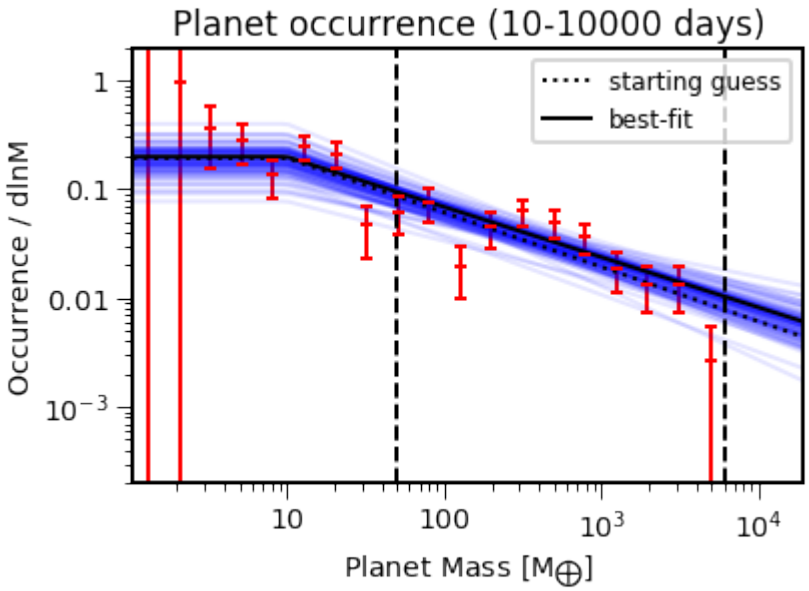


The MCMC has minimized the distance between the simulated orbital period and Msini distribution (blue) and the observed planets (red).

Now let's take a look at the posterior planet occurrence rate distributions, and compare them with the occurrence rates.

[12]:
```
EPOS.plot.parametric.oneD_x(epos, MCMC=True, NB=True, Occ=True)
EPOS.plot.parametric.oneD_y(epos, MCMC=True, NB=True, Occ=True)
```





Now we have distribution that describes the ocurrence in different regiosn of parameter space. Let's see what the prediciton for directly imaged giant planets would be (1-10 au, 1-20 Mjup)

[13]:
```
Pbin= [365.24 * au**1.5 for au in [10,100.]]
Mbin= [Mj * EPOS.cgs.Mjup/EPOS.cgs.Mearth for Mj in [1,20]]
epos.set_bins(xbins=[Pbin], ybins=[Mbin])
EPOS.occurrence.parametric(epos)
```

```
posterior per bin
x: [1.15e+04,3.65e+05], y: [3.2e+02,6.4e+03], area=10.35, eta_0=0.014
gamma= 0.1% +0.2% -0.0%
eta= 0.6% +2.2% -0.5%
```

That's about 1%, consistent with the upper limits for FGK stars

3.8 Custom exoplanet survey

EPOS was designed to work with different exoplanet surveys.

This notebook shows how to load a (mock) exoplanet catolog and (mock) survey detection efficiency.

[1]:
```
import EPOS
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
```

initialize the EPOS class

[2]:
```
epos= EPOS.epos(name='custom_survey')
```

```
 |~| epos 3.0.1.dev3 |~|

Initializing 'custom_survey'

Using random seed 4144380222
Survey: None selected
```

3.8.1 Exoplanet catalog

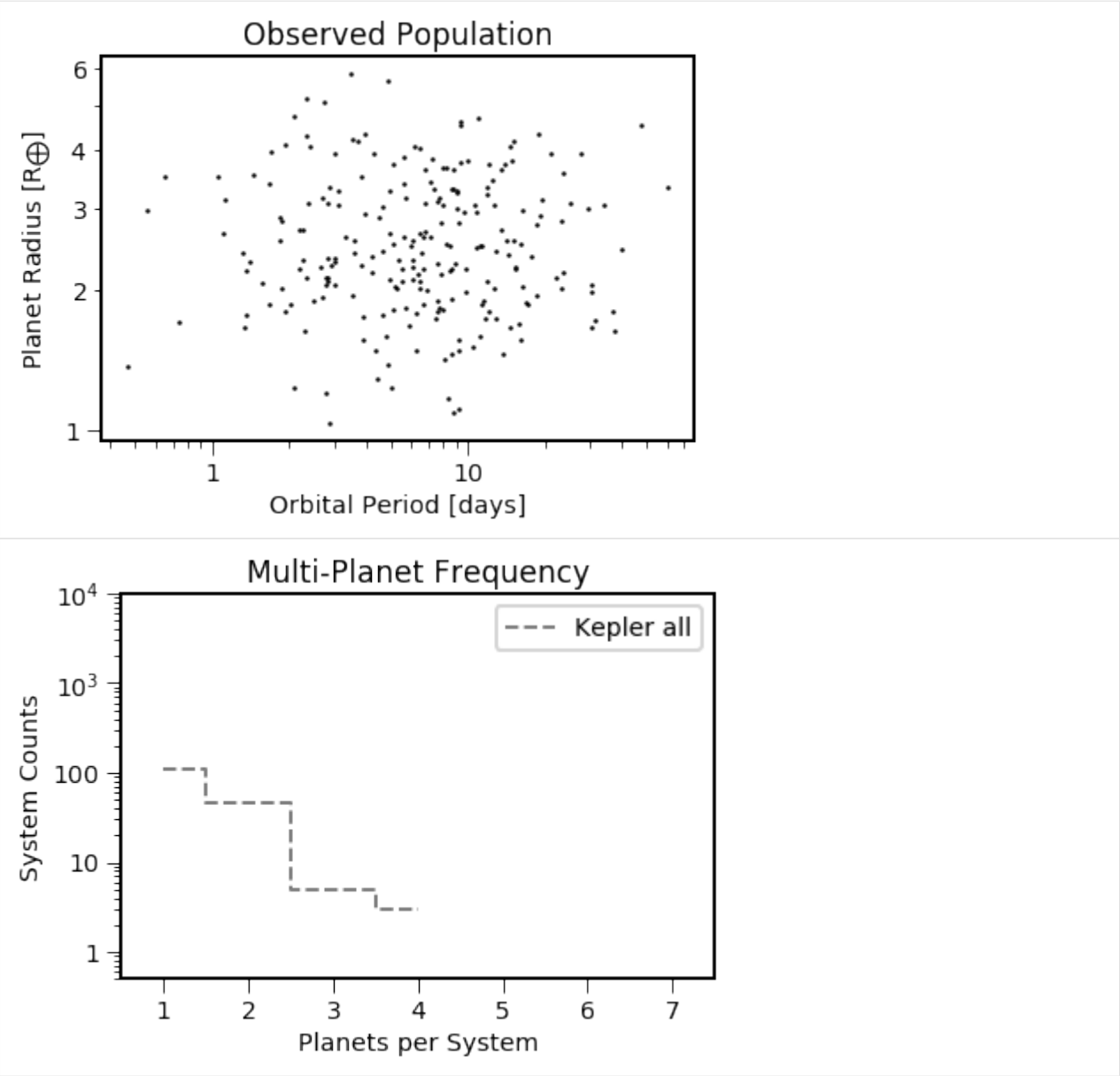
Generate a mock exoplanet catalog with 131 planets detected in a survey of 18k stars

[3]:
```
np.random.seed(76543)
nstars= 18000
npl= 231
period= 10.**np.random.normal(loc=0.8, scale=0.4, size=npl) # days
radius= 10.**np.random.normal(loc=0.4, scale=0.15, size=npl) # earth radii
starID= np.random.choice(np.arange(nstars/50), npl, replace=True) # star identifier␣
↪for each planet to identify multis (can be any format)
```

Load the mock catalog into EPOS

[4]:
```
epos.set_observation(xvar=period, yvar=radius, starID=starID, nstars=nstars)
EPOS.plot.survey.observed(epos, NB=True, PlotBox=False)
EPOS.plot.multi.multiplicity(epos, NB=True)
```

```
Observations:
  18000 stars
  231 planets

  110 singles, 55 multis
  - single: 110
  - double: 47
  - triple: 5
  - quad: 3
```



3.8.2 Detection efficiency

Generate a mock detection efficiency on a grid in period and radius

Note: the geometric detection efficiency factor is calculated by EPOS from the star mass and radius, so it does not need to be provided.

[5]:
```
period_grid= np.geomspace(0.5, 300, 17)
radius_grid= np.geomspace(0.3, 20, 19)
Rstar= 1 # solar radius
Mstar= 1 # solar mass

P, R= np.meshgrid(period_grid, radius_grid, indexing='ij')
detection_efficiency= scipy.stats.norm.cdf(np.log10(R),
```

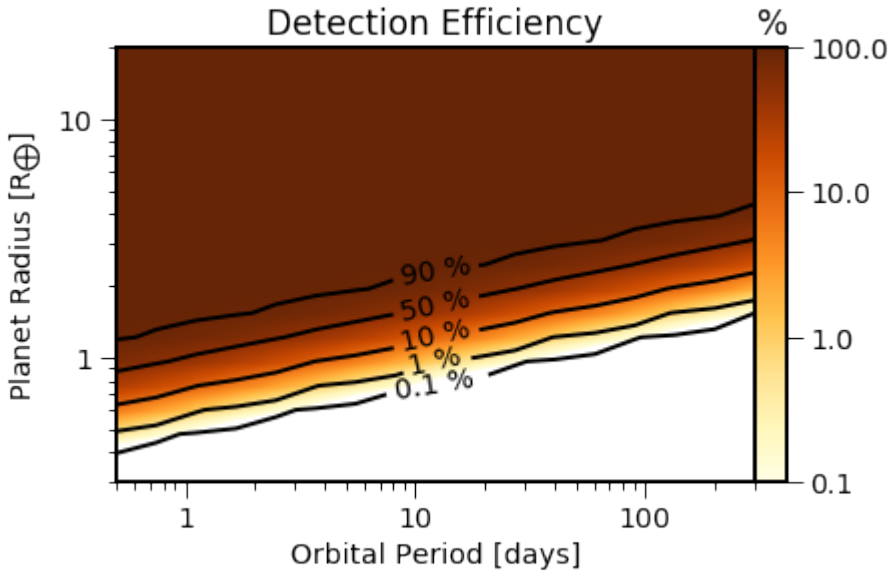
(continues on next page)

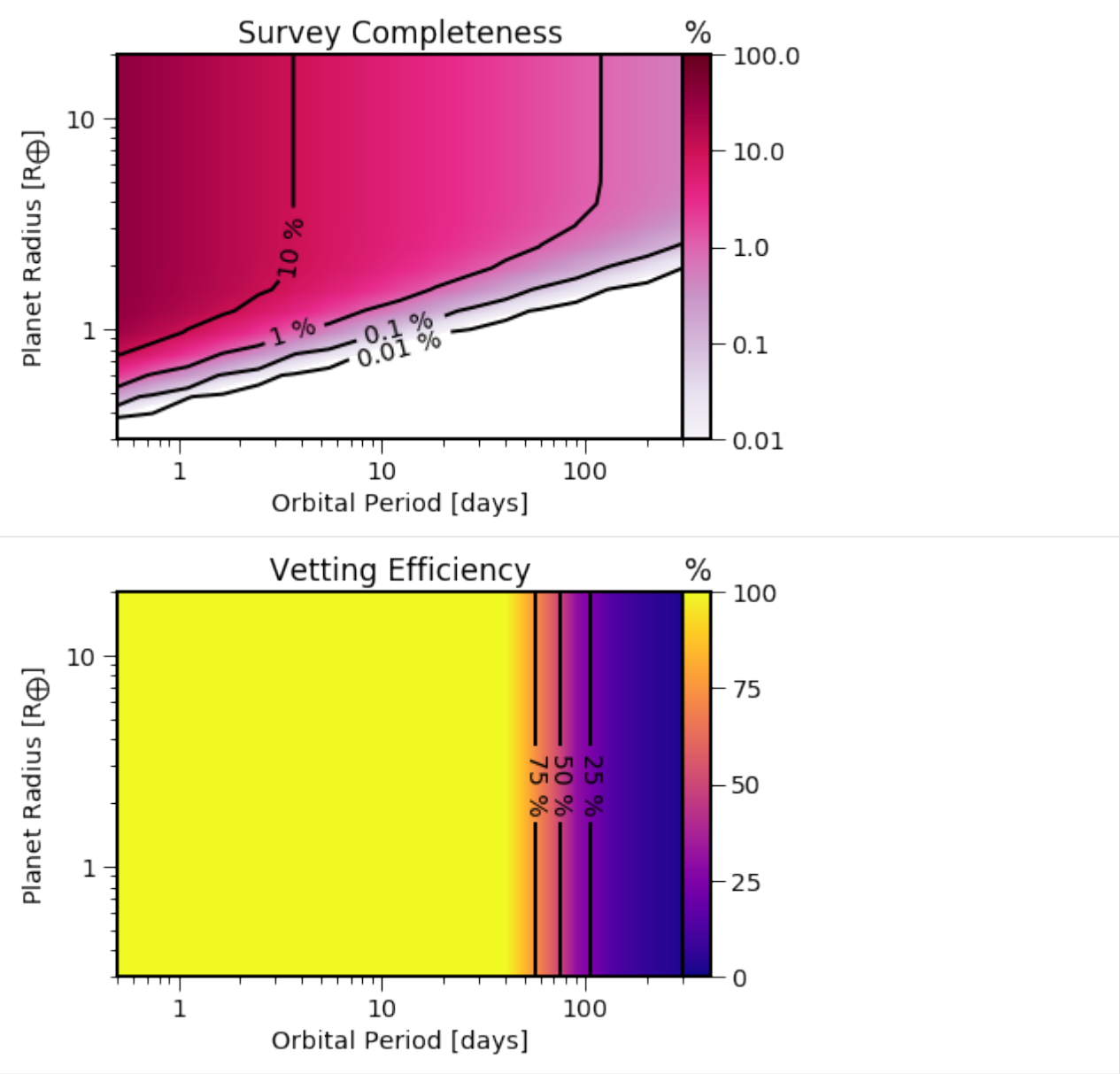
(continued from previous page)

```
                loc=0.0+0.2*np.log10(P), scale=0.1)

#vetting_efficiency=None # optional
vetting_efficiency= np.minimum(1, (P/50)**-2)
```

Load and display the survey detection efficiency

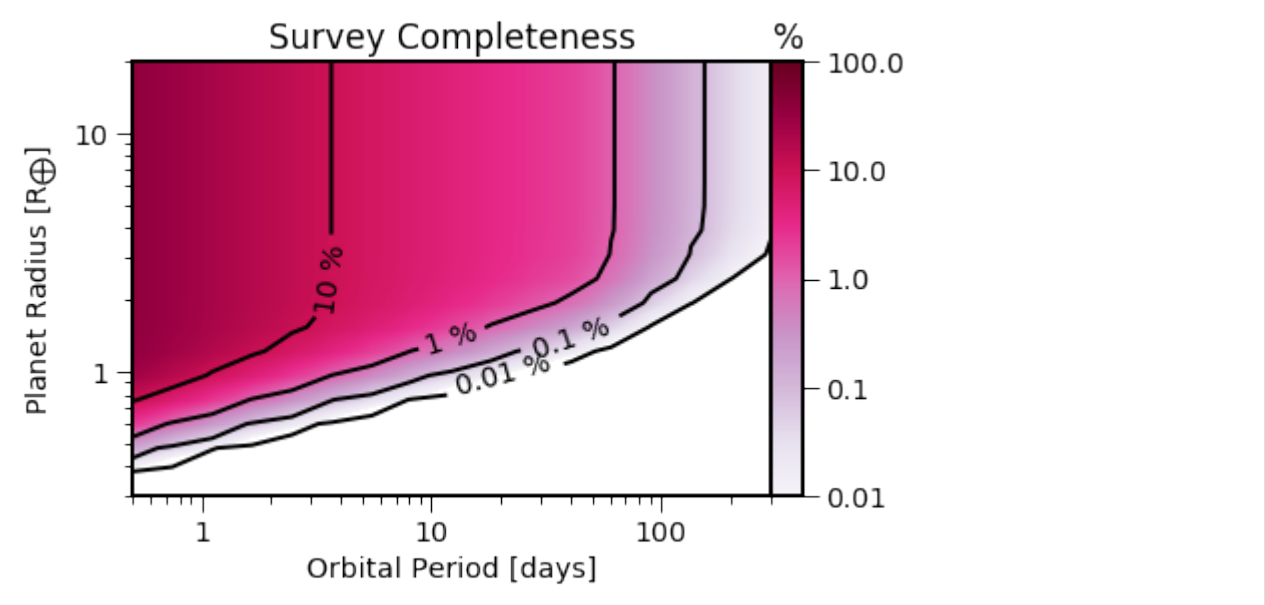
[6]:
```
epos.set_survey(xvar= period_grid,
                yvar= radius_grid,
                eff_2D= detection_efficiency,
                vet_2D= vetting_efficiency,
                Rstar=Rstar, Mstar=Mstar
                )
EPOS.plot.survey.completeness(epos, NB=True, Transit=True, Vetting=False,␣
↪PlotBox=False)
EPOS.plot.survey.completeness(epos, NB=True, Transit=False, Vetting=False,␣
↪PlotBox=False)
EPOS.plot.survey.vetting(epos, PlotBox=False, NB=True)
```





Show the combined completeness

[7]:
```
EPOS.plot.survey.completeness(epos, NB=True, Vetting=True, PlotBox=False)
```



3.8.3 Occurrence Rates

Calculate Occurrence rates for the mock survey

First define bins where to calculate occurrence rates

[8]:
```
epos.set_bins(xbins=[[5,20]], ybins=[[1.4,6]])
```

Define regions where observational comparisons are made (also used for some occurrence rate calculations)

[9]:
```
epos.set_ranges(
    xtrim=[0,730],ytrim=[0.3,20.], # trim the detection efficiency grid
    xzoom=[1,100],yzoom=[1,4], # zoomed region for occurrence rates
    Occ=True) # calculate occurrence along period and radius grid
```

Calculate occurrence rates

[10]:
```
EPOS.occurrence.all(epos)
```

```
Interpolating planet occurrence

  Observed Planets
  x: [5,20], y: [1.4,6], n=123, comp=0.05, occ=0.16

  x zoom bins
  x: [1,100], y: [0.3,0.38], n=0, comp=nan, occ=0
  x: [1,100], y: [0.38,0.48], n=0, comp=nan, occ=0
  x: [1,100], y: [0.48,0.6], n=0, comp=nan, occ=0
  x: [1,100], y: [0.6,0.76], n=0, comp=nan, occ=0
  x: [1,100], y: [0.76,0.96], n=0, comp=nan, occ=0
  x: [1,100], y: [0.96,1.2], n=5, comp=0.02, occ=0.039
  x: [1,100], y: [1.2,1.5], n=11, comp=0.036, occ=0.023
  x: [1,100], y: [1.5,1.9], n=39, comp=0.062, occ=0.091
  x: [1,100], y: [1.9,2.4], n=61, comp=0.08, occ=0.066
```

(continues on next page)

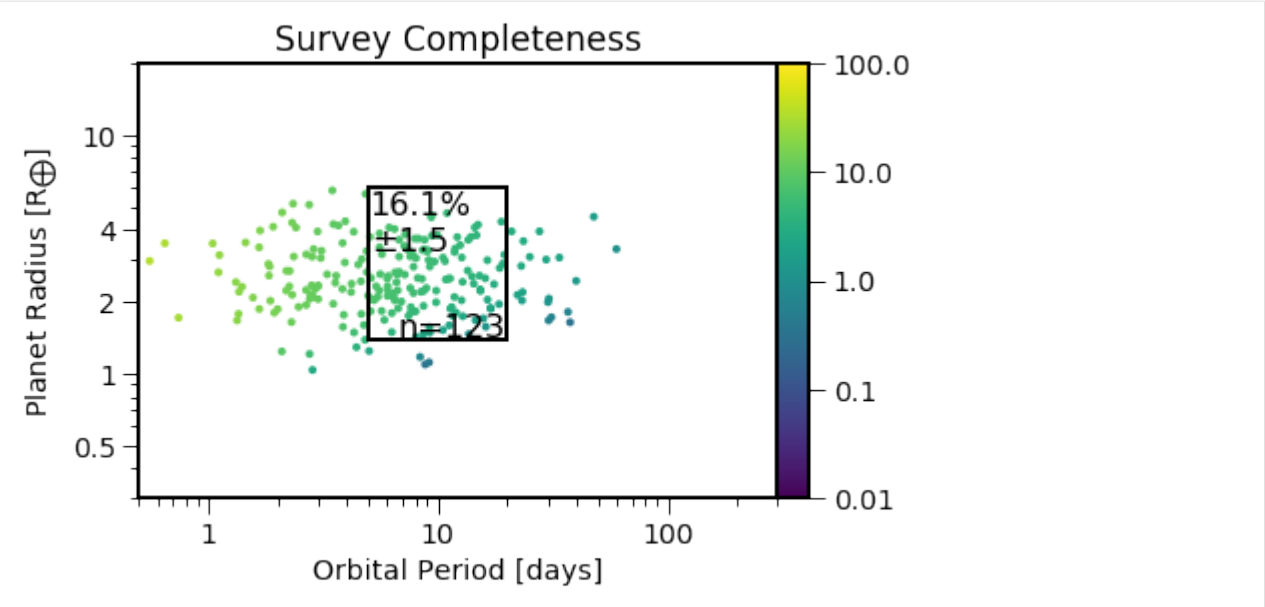
(continued from previous page)

```
  x: [1,100], y: [2.4,3.1], n=49, comp=0.074, occ=0.049
  x: [1,100], y: [3.1,3.9], n=37, comp=0.077, occ=0.038
  x: [1,100], y: [3.9,4.9], n=21, comp=0.082, occ=0.021
  x: [1,100], y: [4.9,6.2], n=4, comp=0.11, occ=0.0021
  x: [1,100], y: [6.2,7.9], n=0, comp=nan, occ=0
  x: [1,100], y: [7.9,9.9], n=0, comp=nan, occ=0
  x: [1,100], y: [9.9,13], n=0, comp=nan, occ=0
  x: [1,100], y: [13,16], n=0, comp=nan, occ=0
  x: [1,100], y: [16,20], n=0, comp=nan, occ=0

  y zoom bins
  x: [0.5,0.746], y: [1,4], n=3, comp=0.32, occ=0.00052
  x: [0.746,1.11], y: [1,4], n=2, comp=0.23, occ=0.00049
  x: [1.11,1.66], y: [1,4], n=8, comp=0.19, occ=0.0023
  x: [1.66,2.47], y: [1,4], n=17, comp=0.15, occ=0.0066
  x: [2.47,3.69], y: [1,4], n=24, comp=0.11, occ=0.014
  x: [3.69,5.51], y: [1,4], n=27, comp=0.076, occ=0.022
  x: [5.51,8.21], y: [1,4], n=44, comp=0.062, occ=0.041
  x: [8.21,12.2], y: [1,4], n=39, comp=0.042, occ=0.085
  x: [12.2,18.3], y: [1,4], n=24, comp=0.032, occ=0.049
  x: [18.3,27.2], y: [1,4], n=11, comp=0.027, occ=0.023
  x: [27.2,40.6], y: [1,4], n=10, comp=0.014, occ=0.063
  x: [40.6,60.6], y: [1,4], n=1, comp=0.01, occ=0.0054
  x: [60.6,90.4], y: [1,4], n=0, comp=nan, occ=0
  x: [90.4,135], y: [1,4], n=0, comp=nan, occ=0
  x: [135,201], y: [1,4], n=0, comp=nan, occ=0
  x: [201,300], y: [1,4], n=0, comp=nan, occ=0
```

```
/Users/mulders/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:3257:␣
↪RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/Users/mulders/anaconda3/lib/python3.7/site-packages/numpy/core/_methods.py:161:␣
↪RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

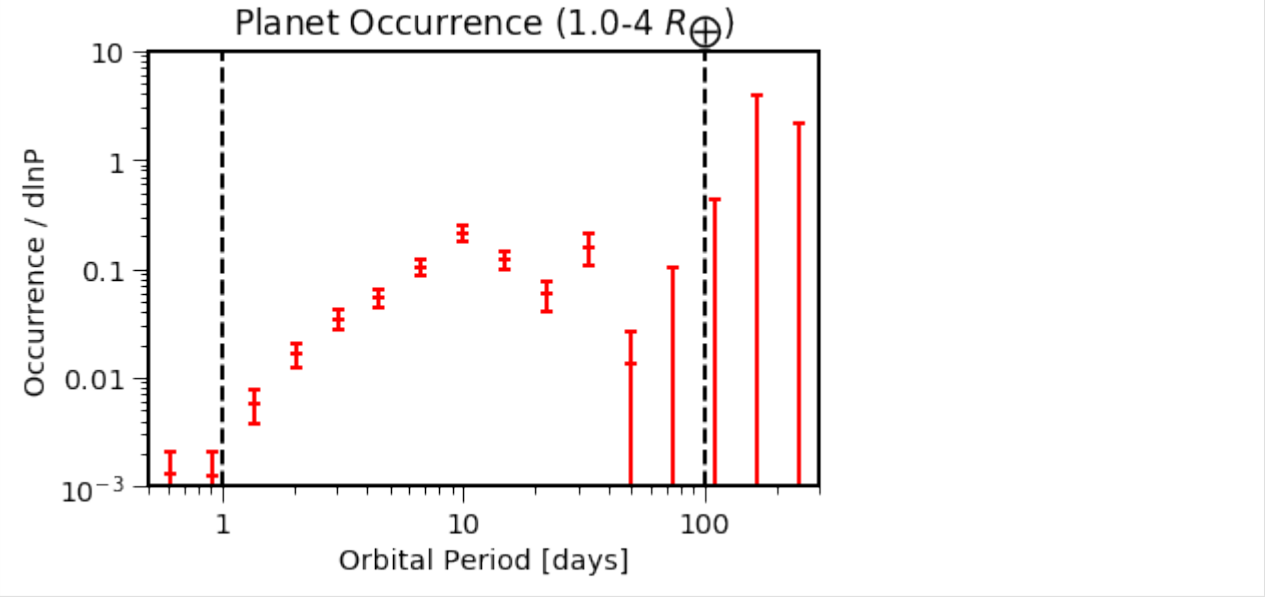
Plot the planet catalog with color-coded completeness

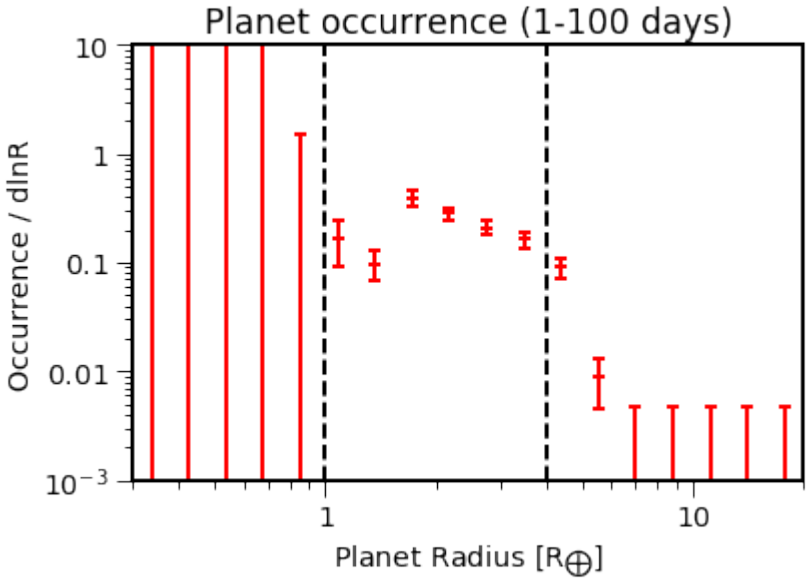
[11]:
```
EPOS.plot.occurrence.colored(epos, Bins=True, NB=True)
```



And plot the occurrence rates as function of period and radius

[12]:
```
EPOS.plot.parametric.oneD_x(epos, NB=True, Occ=True, Init=False)
EPOS.plot.parametric.oneD_y(epos, NB=True, Occ=True, Init=False)
```





3.9 API

This generates the API, showing some documented functions

(this is a link to *EPOS.epos*)

3.9.1 The epos class

class EPOS.**epos**(*name*, *Debug=False*, *seed=True*, *title=None*, *RV=False*, *Norm=False*, *MC=True*, *Msini=False*, *survey=None*)

The epos class

Description: Initialize

Parameters

- **name** (*str*) – name to use for directories
- **survey** (*str*) – Survey data to load. [‘Kepler’, ‘RV’]. Default None
- **RV** (*bool*) – Compare to radial velocity instead of transits
- **MC** (*bool*) – Generate planet population by random draws
- **Msini** (*bool*) – Convert the planet mass distribution into an Msini distribution
- **Debug** (*bool*) – Log more output for debugging
- **seed** (*int*) – Same random number for each simulation? True, None, or int
- **Norm** (*bool*) – normalize pdf (deprecated?)

name

name

Type str

plotdir
plot directory

Type str

RV
Compare to Radial Velocity instead of transit data

Type bool

Multi
Do multi-planet statistics

Type bool

RandomPairing
multis are randomly paired

Type bool

Parametric
parametric planet population?

Type bool

Debug
Verbose logging

Type bool

seed
Random seed, can be any of int, True, or None

jsondir = None
EPOS mode

plotpars = None
Load Default Settings for each Survey

set_bins(*xbins=[[1, 10]], ybins=[[1, 10]], xgrid=None, ygrid=None, Grid=False, MassRadius=False*)
Initialize period-radius (or mass) bins for occurrence rate calculations

Description: Bins can be generated from a grid, f.e. xgrid=[1,10,100], or from a list of bin edges, f.e. xbins= [[1,10],[10,100]]

Parameters

- **xbins** (*list*) – (list of) period bin edges
- **ybins** (*list*) – (list of) radius/mass bin edges
- **xgrid** (*list*) – period bin in interfaces
- **ygrid** (*list*) – radius/mas bin interfaces
- **Grid** (*bool*) – If true, create a 2D grid from bins: nbins = nx $\star$ ny. If false, pair x and y bins: nbins == nx == ny

set_bins_poly(*polys*, *labels=None*)
Initialize polygonic bins for occurrence rate calculations

Description: each polygone is a Nx2 numpy array of (x,y) coordinates

Parameters **polys** (*list*) – (list of) polygones

set_observation (*xvar*, *yvar*, *starID*, *nstars=168620.0*, *radiusError=0.1*, *score=None*, *tdur=None*, *Verbose=True*)
Observed planet population

Parameters

- **xvar** – planet orbital period [list]
- **yvar** – planet radius or M sin i [list]
- **ID** – planet ID [list]
- **nstars** – number of stars surveyed

Note: Some pre-defined planet populations from Kepler can be generated from `EPOS.kepler.dr25()`

set_parametric (*func*)
Define a parametric function to generate the planet size-period distribution

Description: Function should be callable as func(X, Y, *fitpars2d) with X(np.array): period Y(np.array): size (radius or mass) The list of fit parameters fitpars2d will be constructed from parameters added using *EPOS.fitparameters.add()* with is2D=True

Note: Some pre-defined functions can be found in `EPOS.fitfunctions`

Parameters **func** (*function*) – callable function

set_survey (*xvar*, *yvar*, *eff_2D*, *Rstar=1.0*, *Mstar=1.0*, *vet_2D=None*)
Survey detection efficiency (completeness) :param xvar: planet orbital period grid [list]' :param yvar: planet radius or M sin i grid [list] :param eff_2D: 2D matrix of detection efficiency :param Rstar: stellar radius, for calculating transit probability :param Mstar: stellar mass, for period-semimajor axis conversion

Note: Some pre-defined detection efficiencies from Kepler can be generated from `EPOS.kepler`

class EPOS.**fitparameters**
Holds the fit parameters. Usually initialized in epos.fitpars

add (*key*, *value*, *fixed=False*, *min=-inf*, *max=inf*, *dx=None*, *text=None*, *is2D=False*, *isnorm=False*)
Add a fit parameter

Parameters

- **key** (*str*) – fit parameter dictionary key
- **value** (*float*) – starting guess
- **fixed** (*bool*) – keep this parameter fixed
- **min** (*float*) – lower bound
- **max** (*float*) – upper bound
- **dx** (*float*) – initial dispersion for MCMC
- **text** (*str*) – plot safe name?
- **is2D** (*bool*) – use this parameter in the 2D parametric `EPOS.fitfunctions()`

- **isnorm** (*bool*) – this parameter is the normalization factor for the number of planet per star EPOS.fitfunctions()

3.9.2 Fit parameters

class EPOS.**fitparameters**

Holds the fit parameters. Usually initialized in epos.fitpars

add(*key*, *value*, *fixed=False*, *min=-inf*, *max=inf*, *dx=None*, *text=None*, *is2D=False*, *isnorm=False*)

Add a fit parameter

Parameters

- **key** (*str*) – fit parameter dictionary key
- **value** (*float*) – starting guess
- **fixed** (*bool*) – keep this parameter fixed
- **min** (*float*) – lower bound
- **max** (*float*) – upper bound
- **dx** (*float*) – initial dispersion for MCMC
- **text** (*str*) – plot safe name?
- **is2D** (*bool*) – use this parameter in the 2D parametric EPOS.fitfunctions()
- **isnorm** (*bool*) – this parameter is the normalization factor for the number of planet per star EPOS.fitfunctions()

CHAPTER 4

License / Attribution

EPOS was developed as part of the of the Earths in Other Solar Systems project, a NASA/NExSS program

Please cite the papers the two EPOS papers (Mulders et al. 2018; 2019) if you use epos. You can also cite the github repository directly

CHAPTER 5

List of Publications

Mulders et al. 2018 The Exoplanet Population Observation Simulator. I - The Inner Edges of Planetary Systems

Pascucci et al. 2018 A Universal Break in the Planet-to-star Mass-ratio Function of Kepler MKG Stars

Kopparapu et al. 2018 Exoplanet Classification and Yield Estimates for Direct Imaging Missions

Fernandes et al. 2019 Hints for a Turnover at the Snow Line in the Giant Planet Occurrence Rate

Mulders et al. 2019 The Exoplanet Population Observation Simulator. II - Population Synthesis in the Era of Kepler

Pascucci et al. 2019 The impact of stripped cores on the frequency of Earth-size planets in the habitable zone

CHAPTER 6

Version Notes:

1.0.1 first public release

1.0.2 pip installable version

1.1.0 radial velocity without Monte Carlo

2.0.2 planet formation models

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index