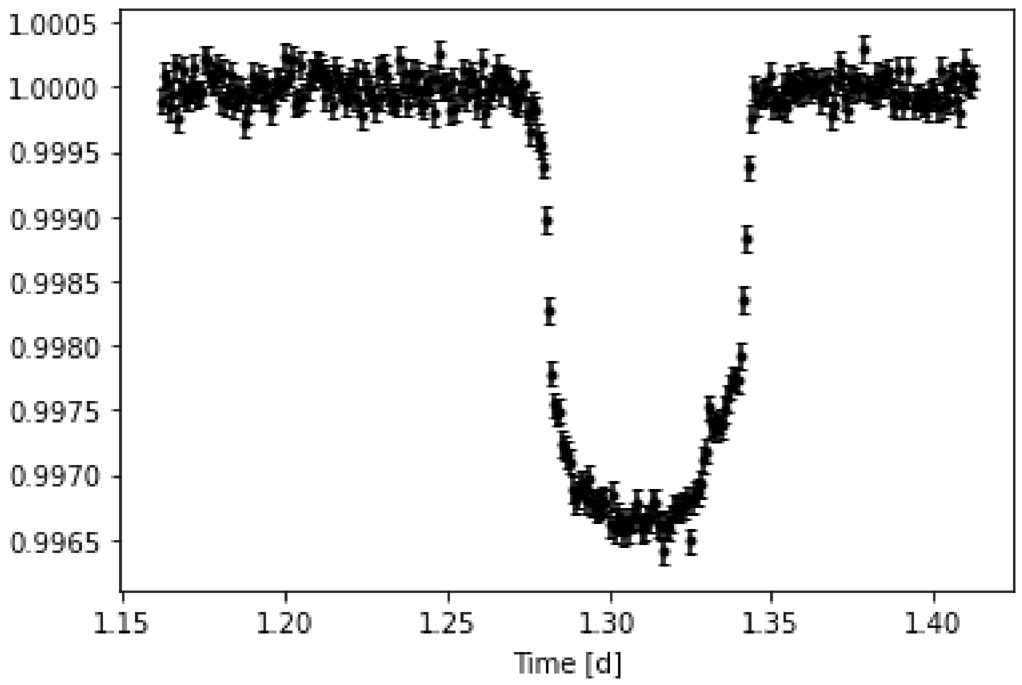
JWST Transit Fitting Tutorial

June 21, 2021

0.1 Step 0: Loading and Plotting the Light Curve Data

[1]:
```python
import batman
from pandas import read_csv
import numpy as np
import emcee

import model
import plots
import sampling
import postprocess as pp

data = read_csv("Devoir2_Data.csv")
t = data["time"].values
flux = data["flux"].values
eflux = np.ones_like(t) * 1e-4  # Given in problem

plots.transit_plot(t, flux, eflux)
```



0.2 Steps 1 and 2: model and probabilities

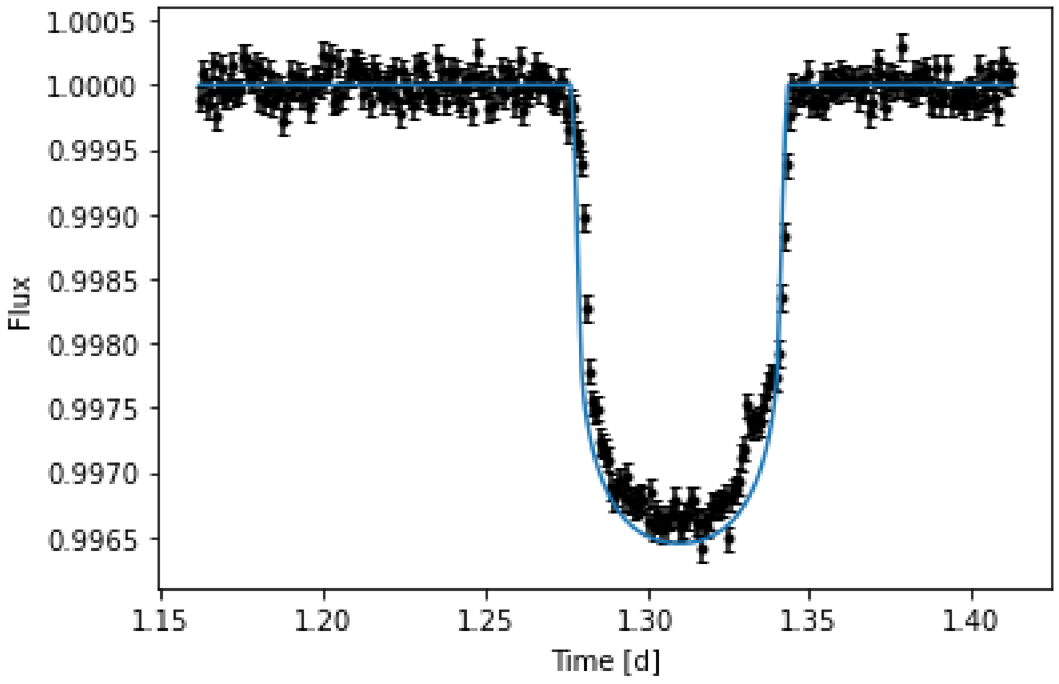
The model and probabilities are defined in `model.py`. This framework allows for multiple orbiting objects to be modelled.

0.3 Step 3: Solve numerically

First we set the model up with reasonable guesses on parameter values. The file `sampling.py` contains the sampling function, the file `postprocess.py` contains the analysis summary functions, and the `plots.py` file contains plotting utilities.

[2]:
```
pguess = [1.31, 0.055, 0.1]

# Fixed parameters
params_pl = batman.TransitParams()
params_pl.per = 3.0
params_pl.a = 15
params_pl.ecc = 0.0
params_pl.w = 90.0
params_pl.limb_dark = "quadratic"  # Basic quadratic limb darkening
params_pl.u = [0.3] * 2

# Free parameters
```

```
params_pl.t0 = pguess[0]
params_pl.rp = pguess[1]
b = pguess[2]
params_pl.inc = model.impact_to_inc(b, params_pl.a)

batmodel_pl = batman.TransitModel(params_pl, t)


plots.transit_plot(t, flux, eflux, model_flux=model.forward_model(pguess,␣
 ↪params_pl, batmodel_pl), show_res=False)
```



0.3.1 Part 1)

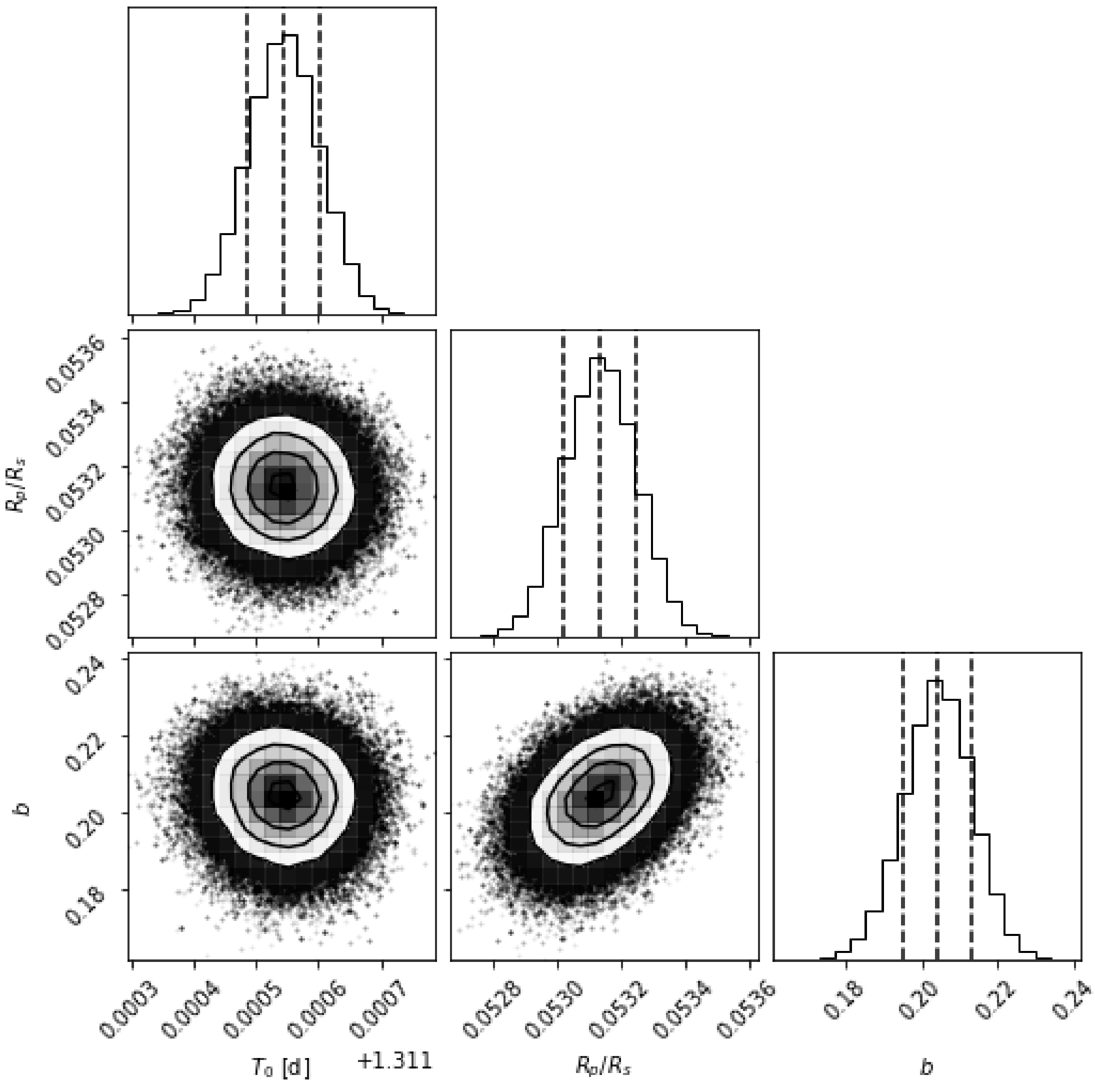
We first run the MCMC and show that the chain is more than 50 times the autocorrelation length, to check convergence.

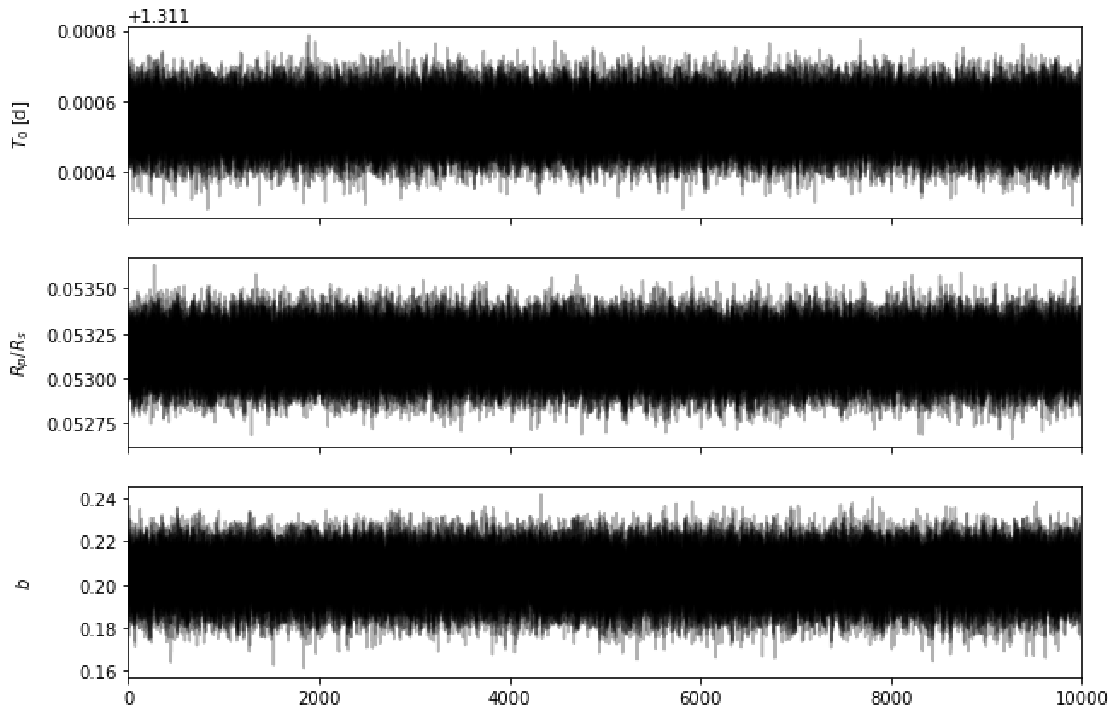
```
[3]: nwalk = 50
nburn = 1000
nstep = 10000
moves = emcee.moves.StretchMove(a=5.0)  # This helps having acc. rate between 0.
 ↪25-0.5
flatchain, chain = sampling.run_mcmc(nwalk, nburn, nstep, pguess, model.
 ↪log_prob, (flux, eflux, params_pl, batmodel_pl), moves=moves)
```

```
100%|          | 11000/11000 [00:31<00:00, 354.83it/s]

Nsteps / Autocorrelation length: [399.87431449 390.90095775 260.78745585]
Mean acceptance rate: 0.34924363636363637
```

```
[4]: labels = [f"$T_0$ [d]", "$R_p/R_s$", "$b$"]
     pmed_mcmc = pp.mcmc_processing(flatchain, chain, labels=labels)
```

```
$T_0$ [d]: 1.3115439946503258 + 5.6904512419686526e-05 - 5.6775299361655485e-05
$R_p/R_s$: 0.053134702711296736 + 0.00011215544965019475 -
0.00011251425682965582
$b$: 0.20426225037943044 + 0.008846991318255476 - 0.009051993519496787
```





We see from these plots that the chains are well-behaved. We also have the parameter values with the 1 sigma uncertainty given above. The mean acceptance rate is reasonable (between 25-50%) and the autocorrelation length is much smaller than the length of the chains (by a factor larger than 50, as required), which indicates convergence.

0.3.2 Part 2)

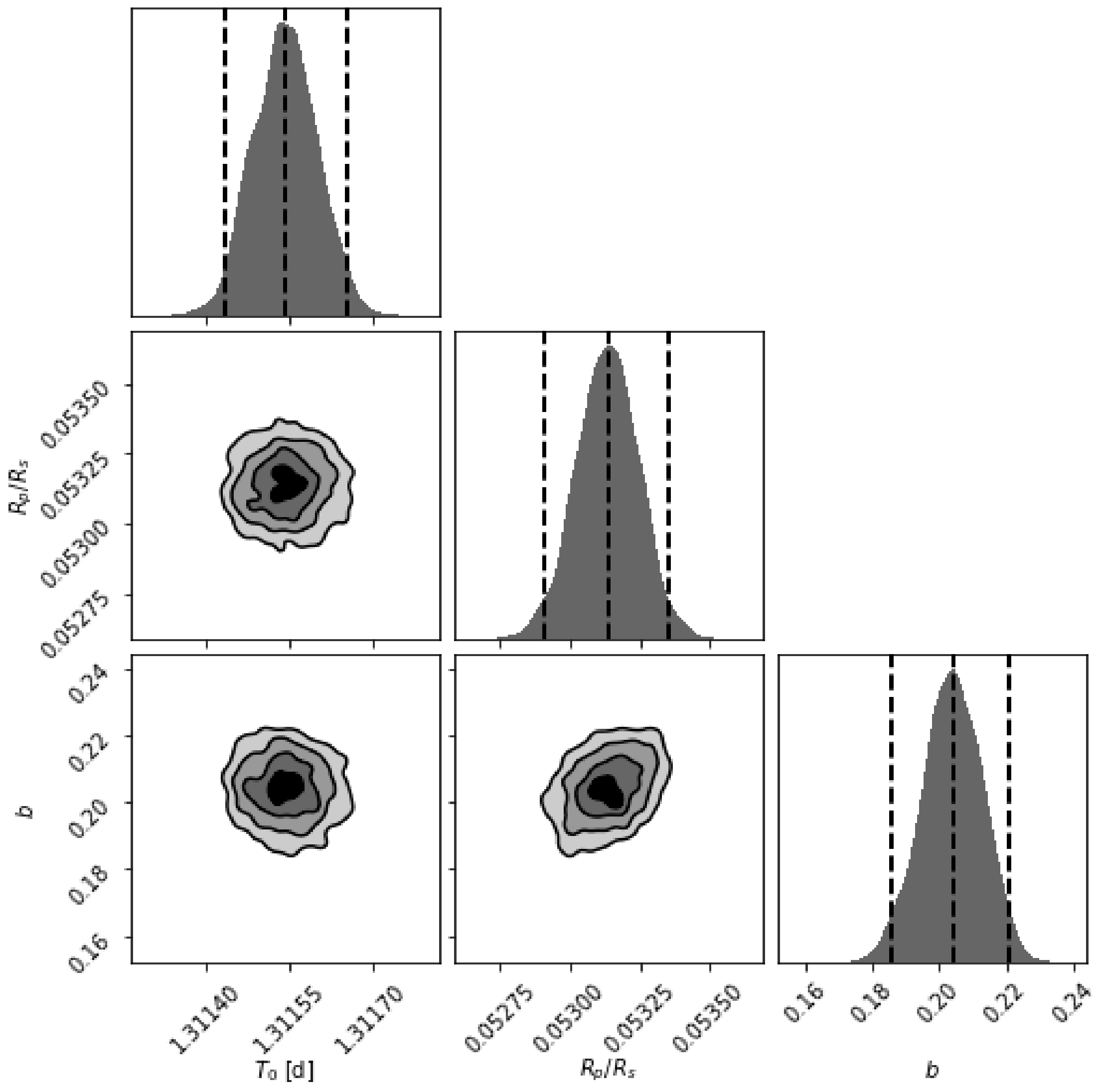
We can now perform nested sampling for the same problem

[5]:
```
from importlib import reload
reload(sampling)
nlive = 500
results = sampling.run_ns(model.log_like, model.prior_transform,␣
 ↪ndim=len(pguess), nlive=500, logl_args=(flux, eflux, params_pl, batmodel_pl))
```

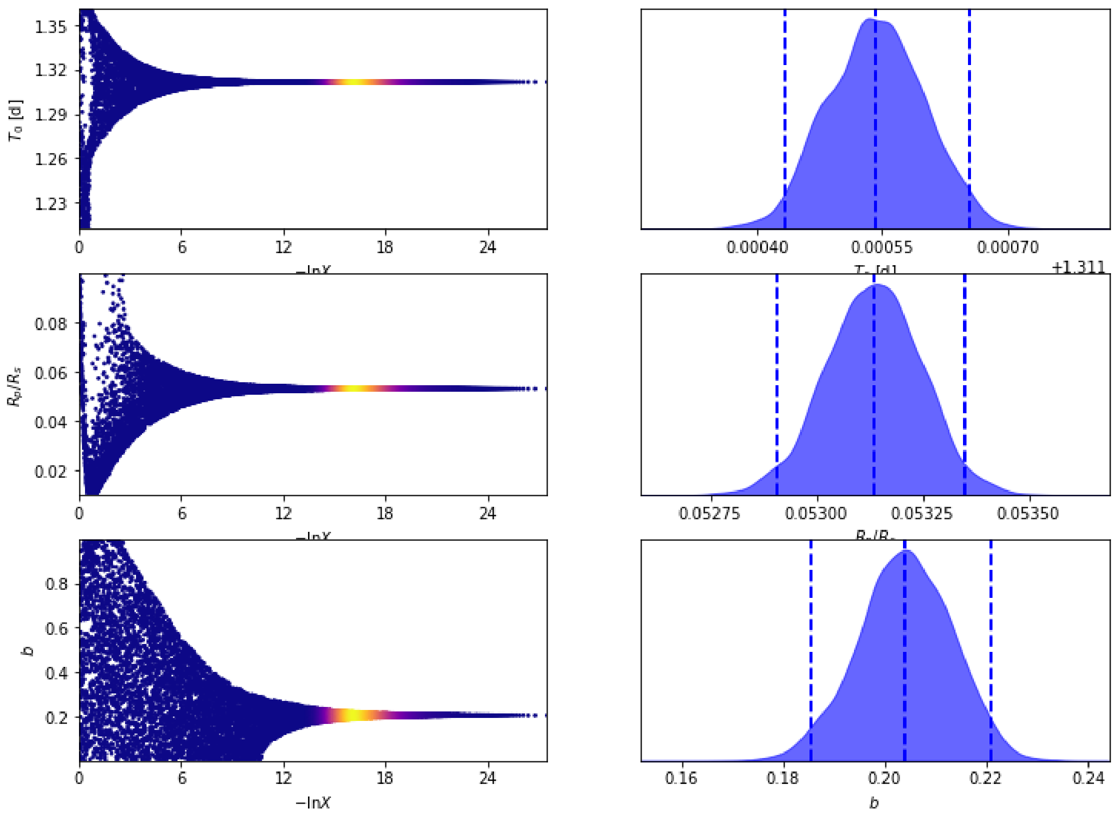
```
10634it [00:17, 603.35it/s, +500 | bound: 29 | nc: 1 | ncall: 39558 | eff(%):
28.146 | loglstar:   -inf < 2233.444 <    inf | logz: 2216.808 +/-  0.246 |
dlogz:  0.000 >  0.010]
```

[6]:
```
pmed_ns = pp.ns_processing(results, labels=labels)
```

```
$T_0$ [d]: 1.311542610621481 + 5.8126712614292586e-05 - 6.062058159228556e-05
$R_p/R_s$: 0.05313475660706046 + 0.00011215092440973462 - 0.00011404166241166941
$b$: 0.20409922359522703 + 0.009004467997890592 - 0.009275124461866152
```



R_p/R_s
0.05350
0.05325
0.05300
0.05275
b
0.24
0.22
0.20
0.18
0.16
1.31140
1.31155
1.31170
T_0 [d]
0.05275
0.05300
0.05325
0.05350
R_p/R_s
0.16
0.18
0.20
0.22
0.24
b



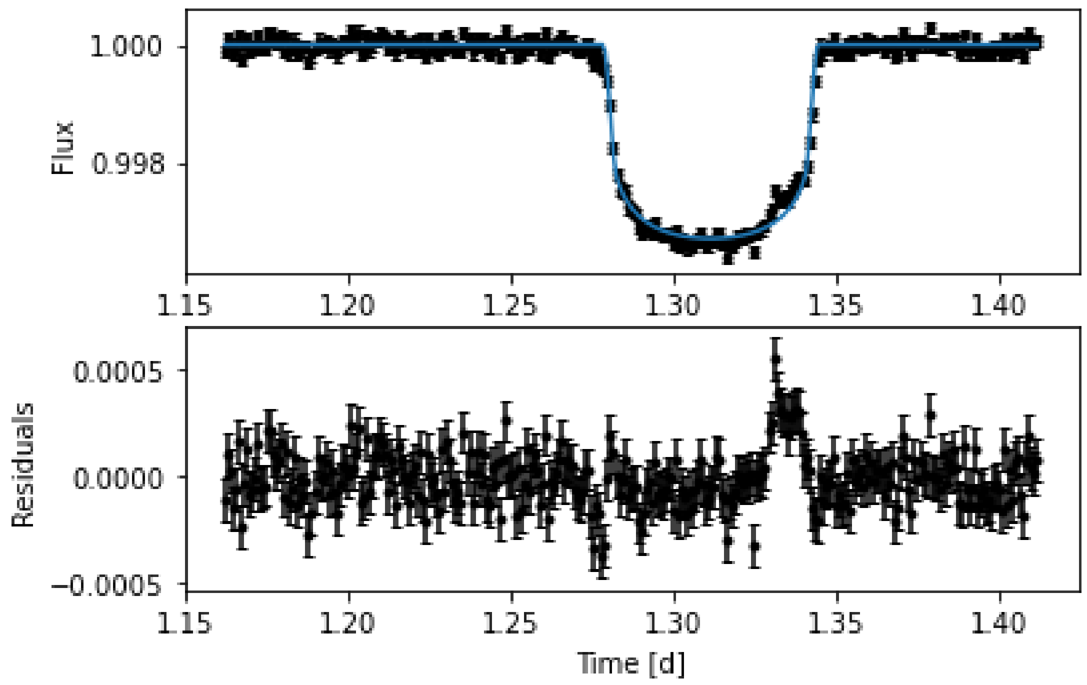
[7]:
```
logz_pl = results.logz[-1]
```

The nested sampling reached its stopping criterion defined by the error in logz, which indicates convergence. with well-behaved distributions.

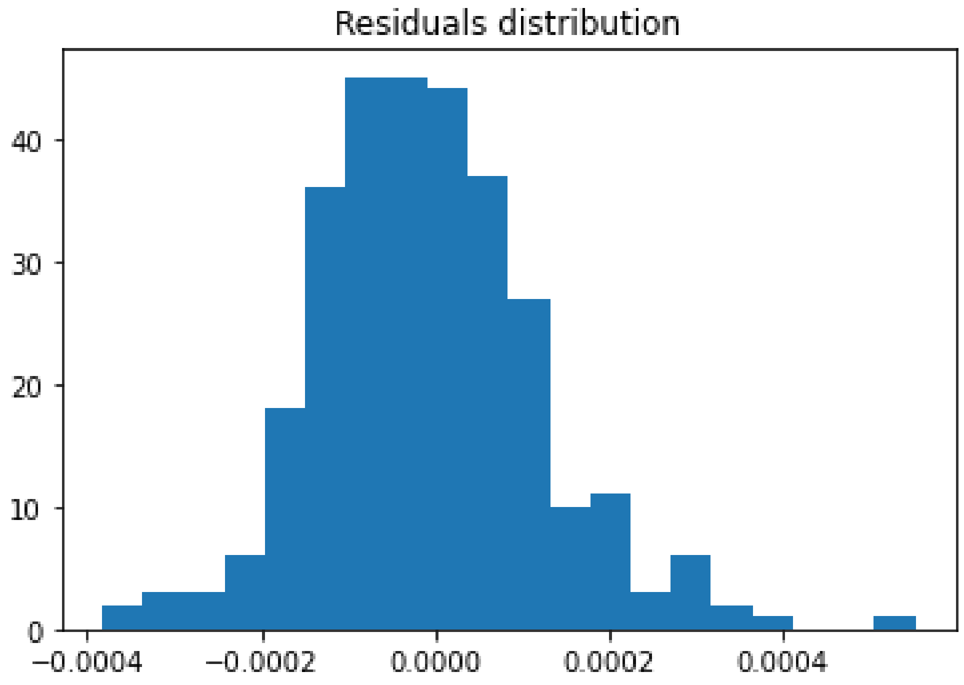
0.4 Step 4

We can check the residuals and we clearly see a pattern. This could be caused by a moon transiting with the planet.

[8]:
```
model_flux = model.forward_model(pmed_ns, params_pl, batmodel_pl)
plots.transit_plot(t, flux, eflux, model_flux)
pp.check_residuals(flux, eflux, model_flux)
```



```
Reduced chi2: 1.6931827842386833
RMS of residuals: 0.00012987777864106363
Mean Error bar: 0.00010000000000000002
Ratio: 1.298777786410636
```



0.5 Step 5

The physical model and probabilities already allow a second object, so we will create the batman object and pass it to the functions

[9]:
```
params_moon = batman.TransitParams()
params_moon.per = 3.0
params_moon.a = 15
params_moon.ecc = 0.0
params_moon.w = 90.0
params_moon.limb_dark = "quadratic"  # Basic quadratic limb darkening
params_moon.u = [0.3] * 2

pguess = [1.31, 0.055, 0.1]
pguess += [1.28, 0.011, 0.1]

# pguess += [0.01, 0.011, 0.1]
params_moon.t0 = pguess[0 + 3]
# params_moon.t0 = params_pl.t0 + pguess[0 + 3]
params_moon.rp = pguess[1 + 3]
b_moon = pguess[2 + 3]
params_moon.inc = model.impact_to_inc(b_moon, params_moon.a)

batmodel_moon = batman.TransitModel(params_moon, t)

params = [params_pl, params_moon]
batmodels = [batmodel_pl, batmodel_moon]
```

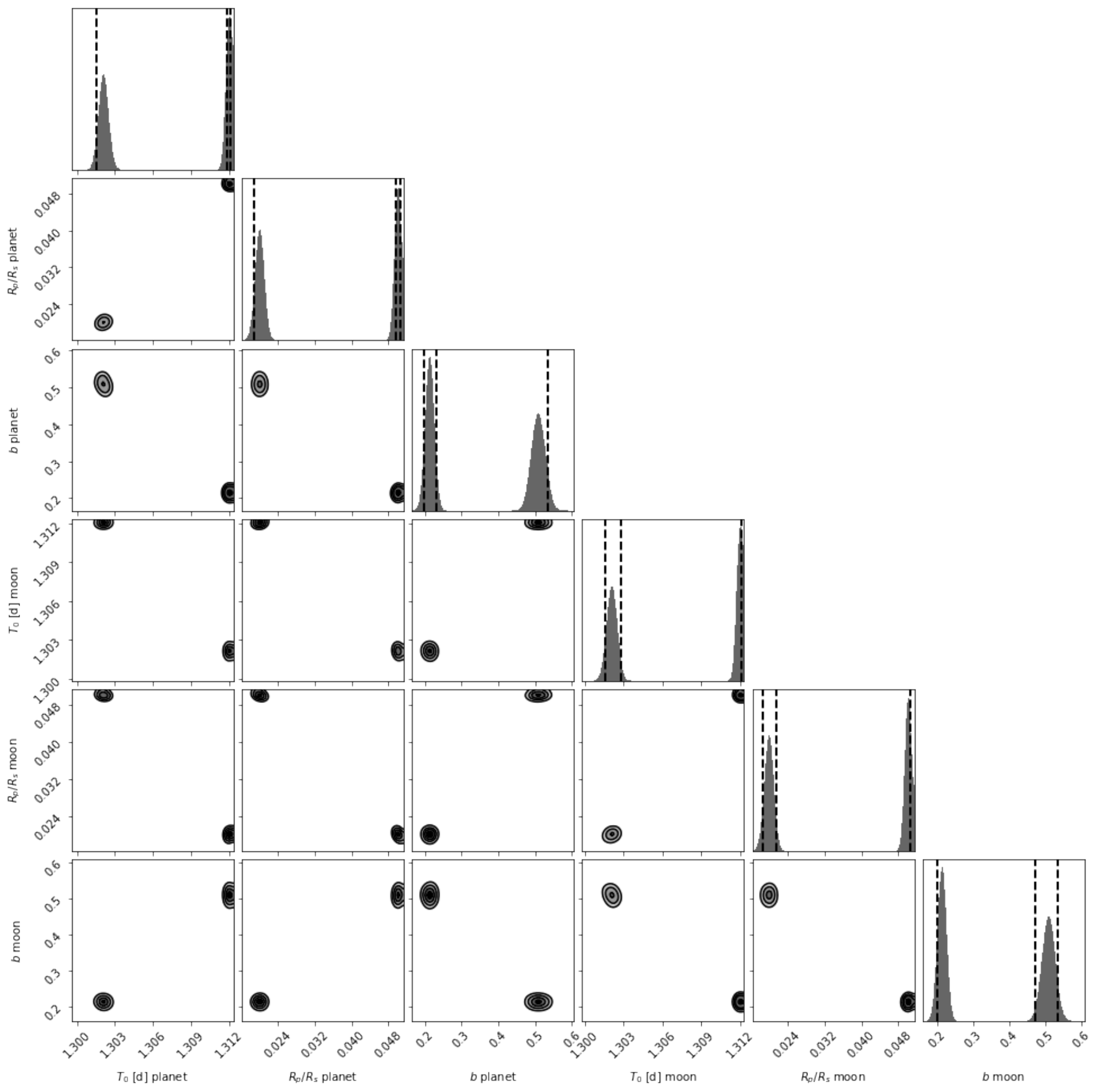
[10]:
```
nlive = 500
results = sampling.run_ns(model.log_like, model.prior_transform,␣
↪ndim=len(pguess), nlive=500, logl_args=(flux, eflux, params, batmodels),␣
↪ptform_kwargs=None)
```

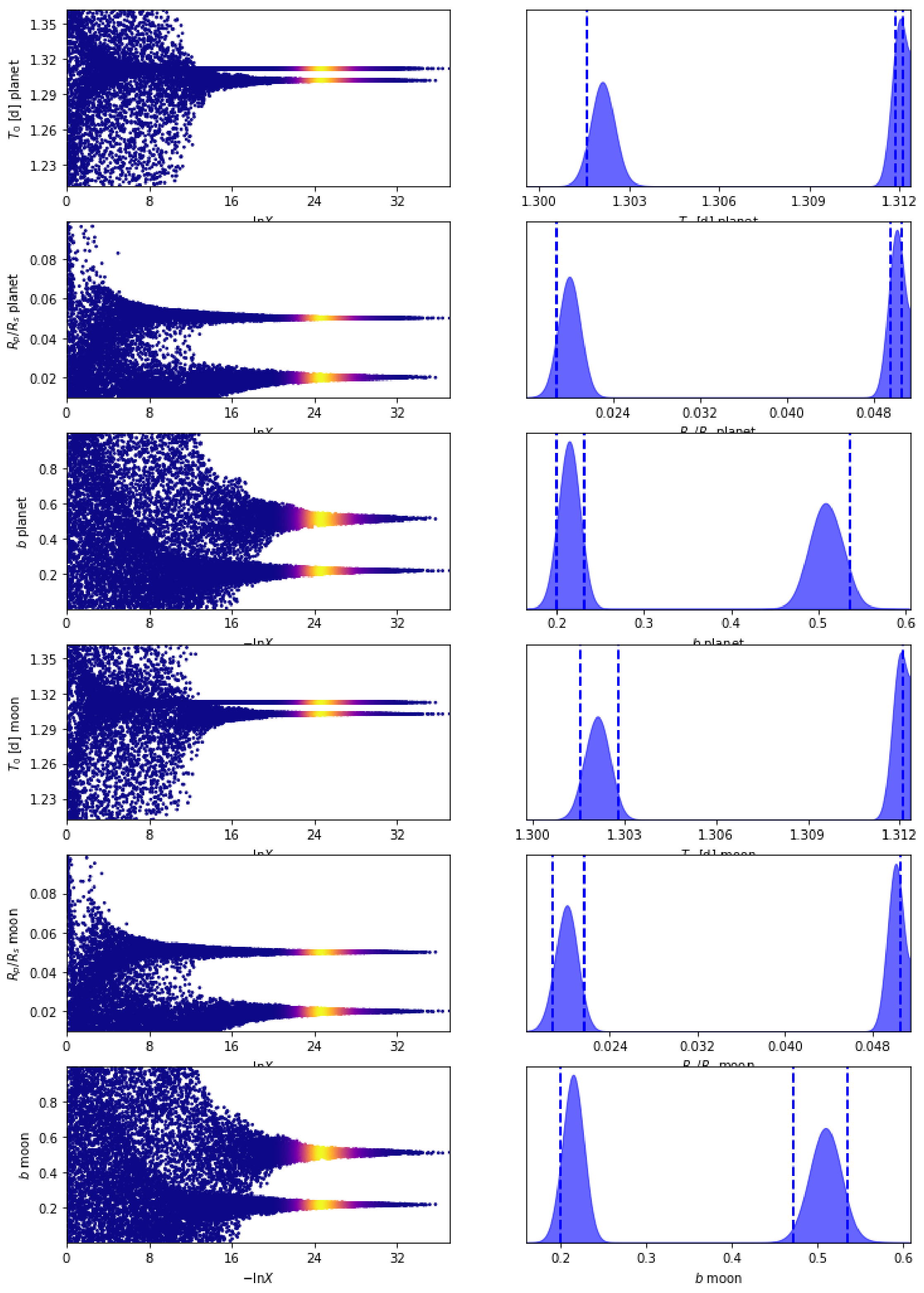
```
15462it [01:22, 187.11it/s, +500 | bound: 100 | nc: 1 | ncall: 93566 | eff(%):
17.060 | loglstar:   -inf < 2331.067 <    inf | logz: 2304.783 +/-  0.306 |
dlogz:  0.000 >  0.010]
```

[11]:
```
labels = [f"$T_0$ [d] planet", "$R_p/R_s$ planet", "$b$ planet"]
labels += [f"$T_0$ [d] moon", "$R_p/R_s$ moon", "$b$ moon"]
pmed_ns_moon = pp.ns_processing(results, labels=labels*2)
```

```
$T_0$ [d] planet: 1.3118556074578287 + 0.00018273534232737781 -
0.009883569780235524
$R_p/R_s$ planet: 0.04950696410585402 + 0.0007291500901802153 -
```

```
0.0298820234579093
$b$ planet: 0.23190952625152103 + 0.2845255675150713 - 0.02218762546306713
$T_0$ [d] moon: 1.3028183495474033 + 0.009218862606006573 - 0.000863114283037536
$R_p/R_s$ moon: 0.02161378095195406 + 0.028635767323516335 -
0.0019354837799265522
$b$ moon: 0.47169661361432585 + 0.045272246419804374 - 0.26121481089769666
```





The stopping criterion is reached and the plots seem to have stabilized. However there is a bimodality, which is explained below. The plots still appear to show convergence, with one of the modes clearly dominating for each parameter, but it surely is not ideal to have such a degeneracy.

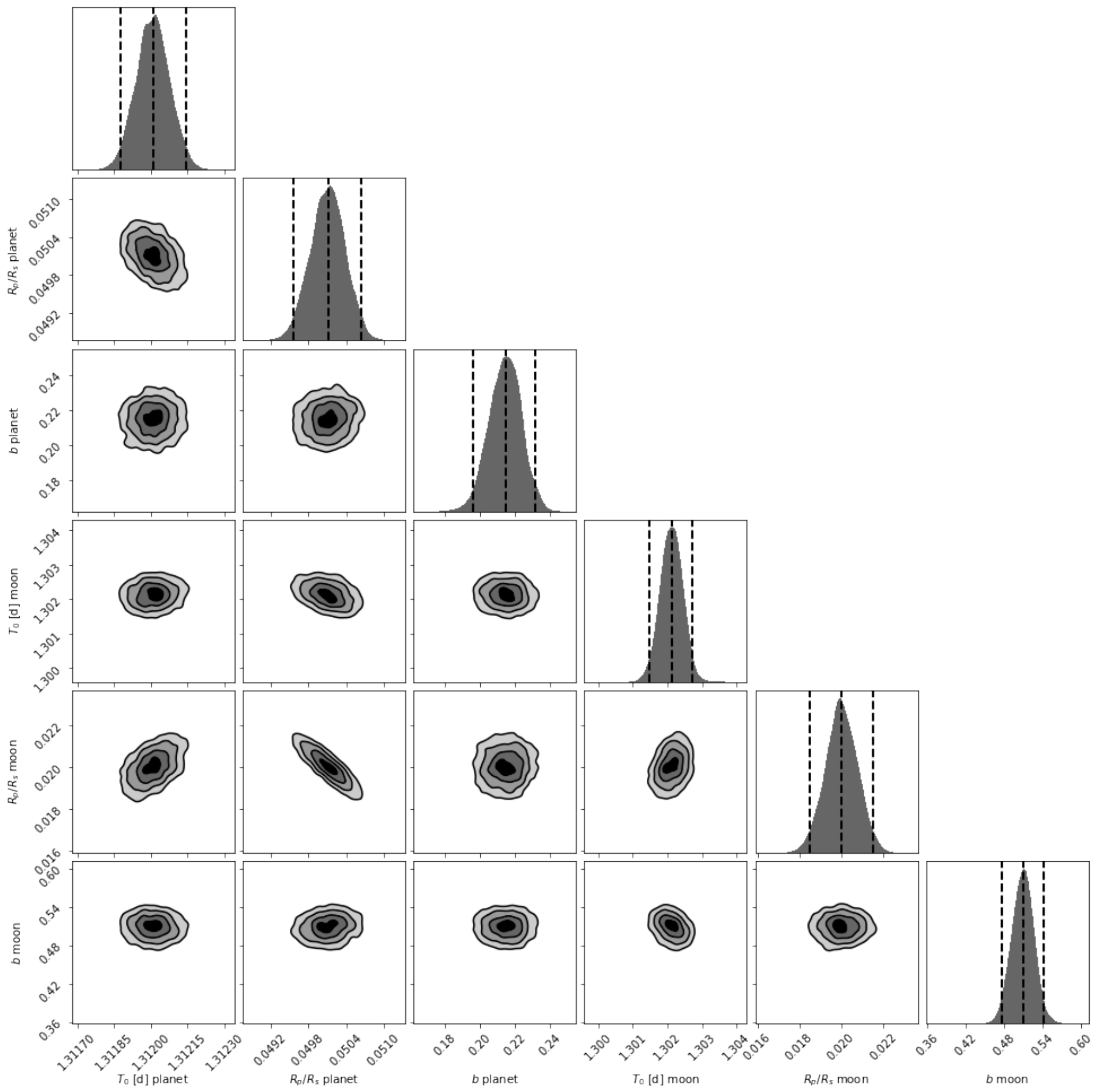
The distribution appears to be multi-modal because both objects are considered in each set of 3 parameters (each batman model). We can overcome this by ensuring that one of the objects (moon) is smaller than the other (planet), which would break the degeneracy. This appears to be a reasonable assumption here as one transit is much shallower, because moons are smaller than planets. We test this below by passing a keyword argument to the prior transform.

[12]:
```
nlive = 500
results = sampling.run_ns(model.log_like, model.prior_transform,␣
↪ndim=len(pguess), nlive=500, logl_args=(flux, eflux, params, batmodels),␣
↪ptform_kwargs={"force_smaller": True})
```

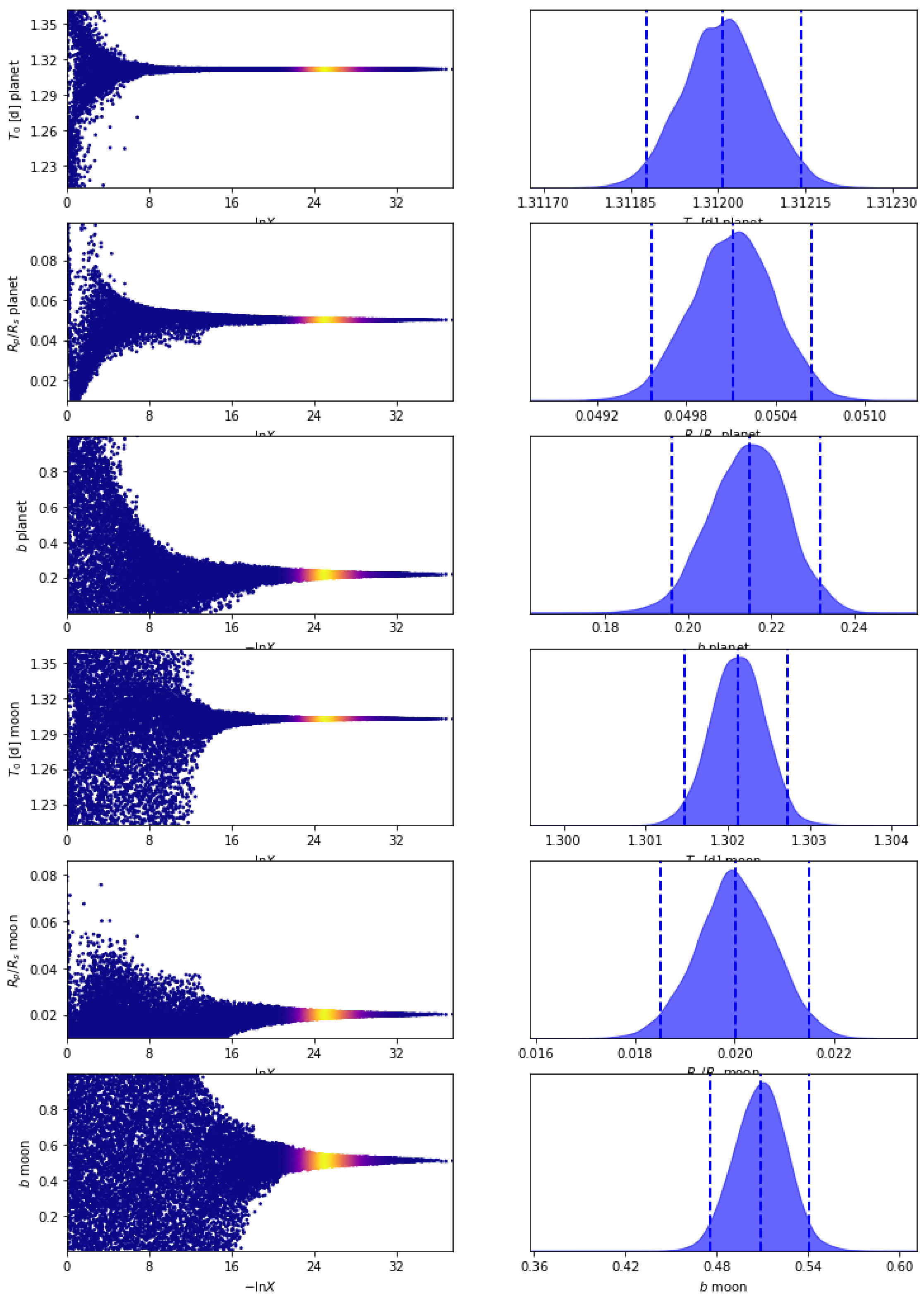
```
15666it [01:20, 193.94it/s, +500 | bound: 99 | nc: 1 | ncall: 91771 | eff(%):
17.616 | loglstar:   -inf < 2331.027 <    inf | logz: 2304.336 +/-  0.308 |
dlogz:  0.000 >  0.010]
```

[13]:
```
pmed_ns_moon = pp.ns_processing(results, labels=labels*2)
logz_moon = results.logz[-1]
```

```
$T_0$ [d] planet: 1.3120085674791435 + 6.728731725003101e-05 -
6.926193764034139e-05
$R_p/R_s$ planet: 0.050116082048003496 + 0.00026493869940261844 -
0.0002853844535839964
$b$ planet: 0.21458421536505248 + 0.00872778742341207 - 0.009444718807337549
$T_0$ [d] moon: 1.3021216612960504 + 0.00031372490841730105 -
0.0003227978036182133
$R_p/R_s$ moon: 0.0200037171142433 + 0.0007842071745880544 -
0.0007366072245794594
$b$ moon: 0.5090368642039036 + 0.016067711261730566 - 0.017285999547472397
```



R_p/R_s planet
b planet
T_0 [d] moon
R_p/R_s moon
b moon
T_0 [d] planet



This indeed looks better with a single mode everywhere. Because this run has better convergence, we will use its evidence for the “moon” model. By taking the ratio of this evidence with the one from the initial one-planet model, we can obtain the Bayes Factor.

```
[14]: print(f"ln(BF): {logz_moon - logz_pl}")
```

```
ln(BF): 87.5284368265211
```

By comparing the evidences, we obtained a bayes factor with a logarithm of almost 88. This shows very strong evidence in favor of the model that includes a moon in addition of the planet (based on the Jeffreys scale seen in class, more than 99% in favor of this model.

0.6 Acknowledgements

This tutorial was adapted and modified based on a class project solution submitted by Thomas Vandal for the course PHY6795 (Winter 2021, Prof. Björn Benneke)