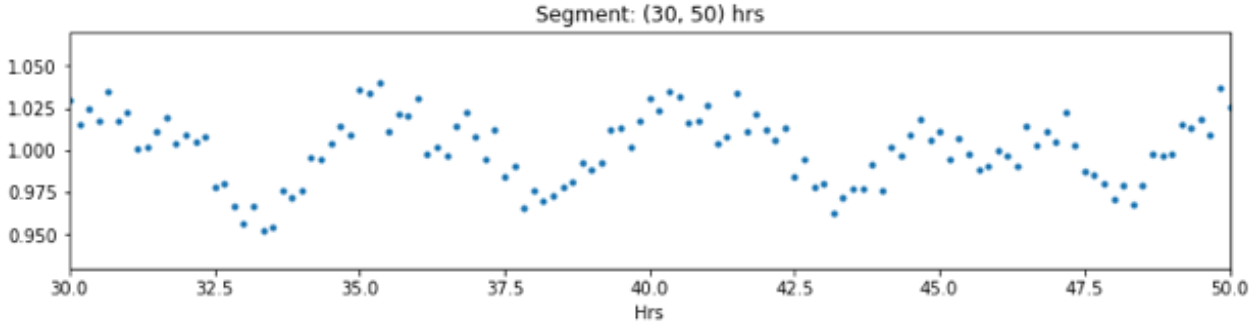
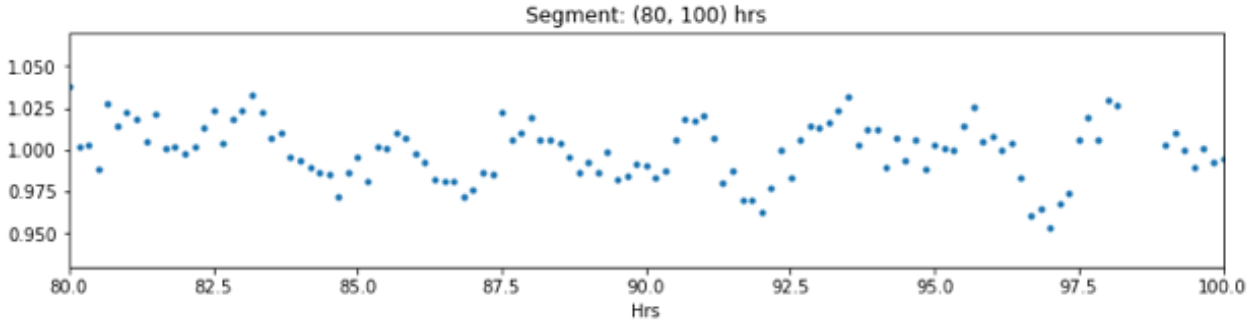
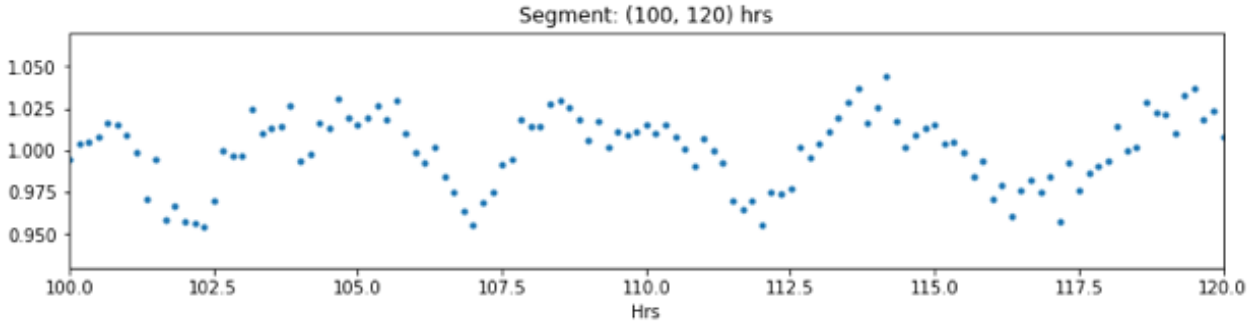
In [261…

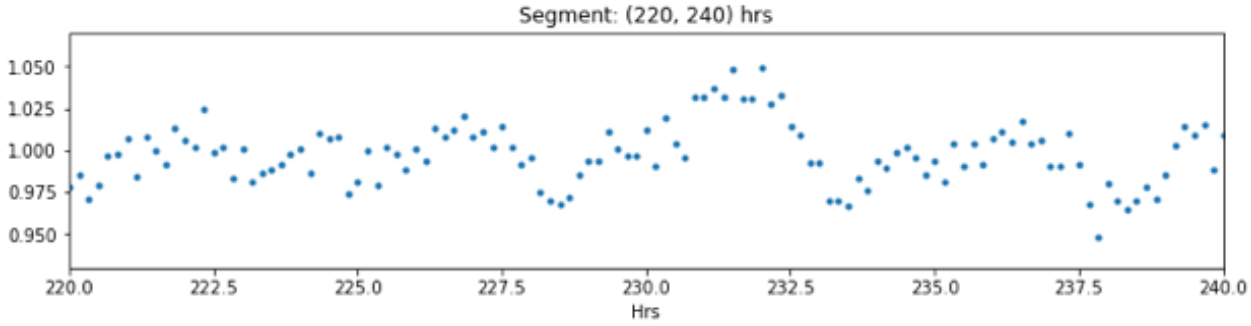
```
from AROMA import *
from AROMA.utils import *
from AROMA.config import *
import pandas as pd
from scipy.optimize import curve_fit

%matplotlib inline

pi = np.pi

def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return idx
print('def: find_nearest(array, value)')

import os
from os.path import join
array = os.path.abspath('').split('/')
homedir = '/'
for i in range(1,7):
   homedir = join(homedir, array[i])
homedir

data_path = '~/Documents/GitHub/aroma/AROMA_An_Exo_Rot_Mapping/data/proces
data = pd.read_csv(data_path, header=0, delimiter='\t')
plotPath = join(homedir, 'plots')

# data = data.query('TIME < 2293') # first half of sector 36
# data = data.query('TIME > 2293 & TIME < 2306') # second half of sector 3
# data = data.query('TIME < 2306') # full of sector 36

# data = data.query('TIME > 2306 & TIME < 2320') # first half of sector  3
# data = data.query('TIME > 2320') # second half of sector  37
# data = data.query('TIME > 2306') # full of sector 37

data = data # full light curve
fit_dir = join(homedir,'notebooks', 'periodSineFit_metadata')

time, lc = data.TIME, data.lc_corrected
# segments = [(30,80), (80,130), (150,200), (220,270), (330, 380)]
segments = [(30,50), (80,100), (100,120), (220,240), (330, 350)]
for seg in segments:
    plt.figure(figsize=(12, 2.5)), plt.plot(time, lc, ls='', marker='o', ms
```

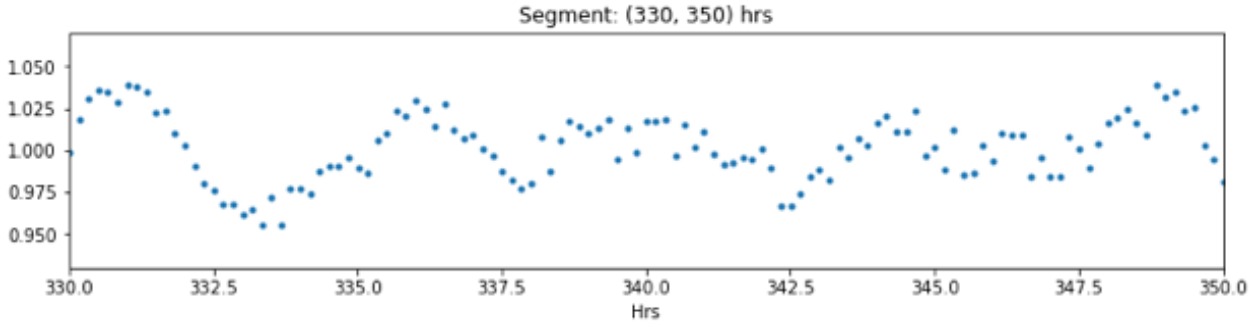
```
def: find_nearest(array, value)
```











Scipy Optimize Linear Regression Fit

Before doing our Markov Chain Monte Carlo (MCMC) Fit, we want to try a simple LR fit first to

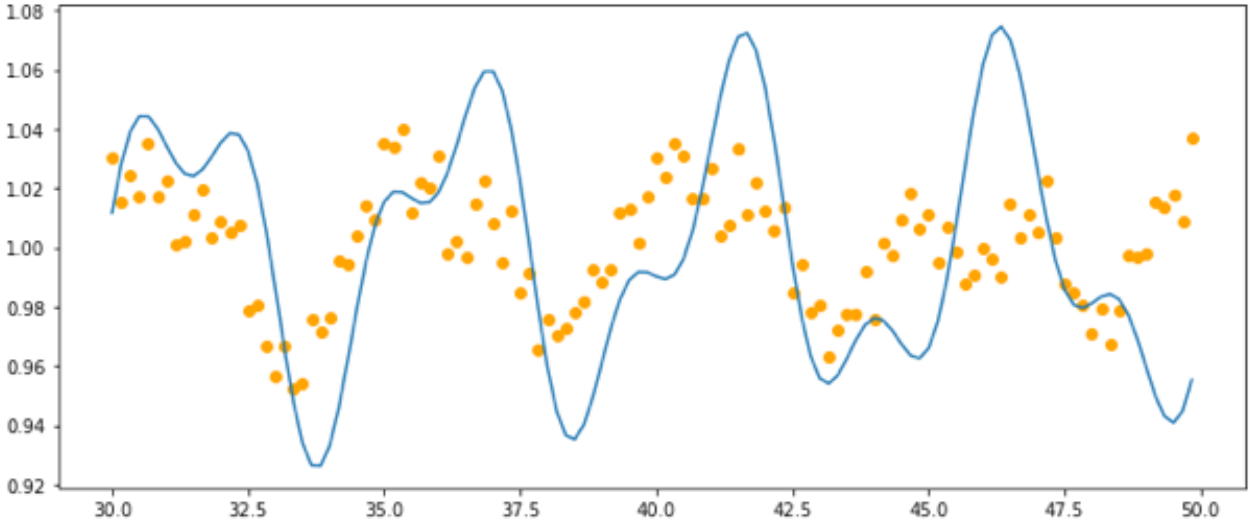
1. define good bounds for the priors
2. obtain good guesses for the initial guess

All these will goes into the MCMC fit!

In [282…

```
def altmodel(a1, f1, a2, f2, a3, f3, w1, w2, w3, t):
    model = 1+ a1*np.sin(w1*t + f1) + a2*np.sin(w2*t + f2) + a3*np.sin(w3*t
    return model

tlow, thigh = segments[0]
subset = data.query('TIME < %f & TIME >= %f'%(thigh, tlow))
time = np.array(subset.TIME)
lc = np.array(subset.lc_corrected)
lc_err = 0.05*np.array(subset.lc_corrected)

# guesses = [0.005, 0.5, 0.002, -1, 0.002, 3, 1.2, 1.5, 2.1]
lim = [(0, 0.05), (-10*pi, 10*pi),
       (0, 0.05), (-10*pi, 10*pi),
       (0, 0.05), (-10*pi, 10*pi),
       (1, 1.5), (2.5, 3), (1., 1.5)]

### GUESSES AND BOUNDS
# popt, pcov = curve_fit(altmodel, time, lc, p0=guesses, bounds=np.transpos
### BOUNDS ONLY
popt, pcov = curve_fit(altmodel, time, lc, bounds=np.transpose(lim))
### NONE
# popt, pcov = curve_fit(altmodel, time, lc)

a1, f1, a2, f2, a3, f3, w1, w2, w3 = popt
fit = altmodel(a1, f1, a2, f2, a3, f3, w1, w2, w3, t=time)

plt.figure(figsize=(12,5)), plt.plot(time, fit), plt.scatter(time, lc, c='
print(popt[0:2])
print(popt[2:4])
print(popt[4:6])
print(popt[6:9])
print(2*pi/popt[6:9], 'hours')
```

```
[0.02498714 0.0082991 ]
[ 0.02559212 -0.00761681]
[ 0.02504827 -0.02018933]
[1.25447043 2.74984946 1.24999876]
[5.00863566 2.28491973 5.02655322] hours
```



emcee Markov Chain Monte Carlo (MCMC)

Run the LR fit first to obtain `lim` bounds!

Declare four functions:

1. **model()**: The model function should take as an argument a list representing our θ vector, and return the model evaluated at that θ.
2. **lnlike()**: This function takes as an argument theta as well as the x, y, and y_err of your actual theta. It's job is to return a number corresponding to how good a fit your model is to your data for a given set of parameters, weighted by the error in your data points
3. **lnprior()**: This function is to check - before running the probability function on any set of parameters - that all variables are within their priors (in fact, this is where we set our priors). Reasonable bounds on the amplitudes can be drawn from the data (e.g. amplitudes can't be greate than overall signal, periods within expected bounds, no negative amplitudes, etc).
4. **lnprob()**: This function combines the steps above by running the lnprior function, and if the function returned -np.inf, passing that through as a return, and if not (if all priors are good), returning the lnlike for that model (by convention we say it's the lnprior output + lnlike output, since lnprior's output should be zero if the priors are good). lnprob needs to take as arguments theta, x, y, and y_err since these get passed through to lnlike.

In [279…

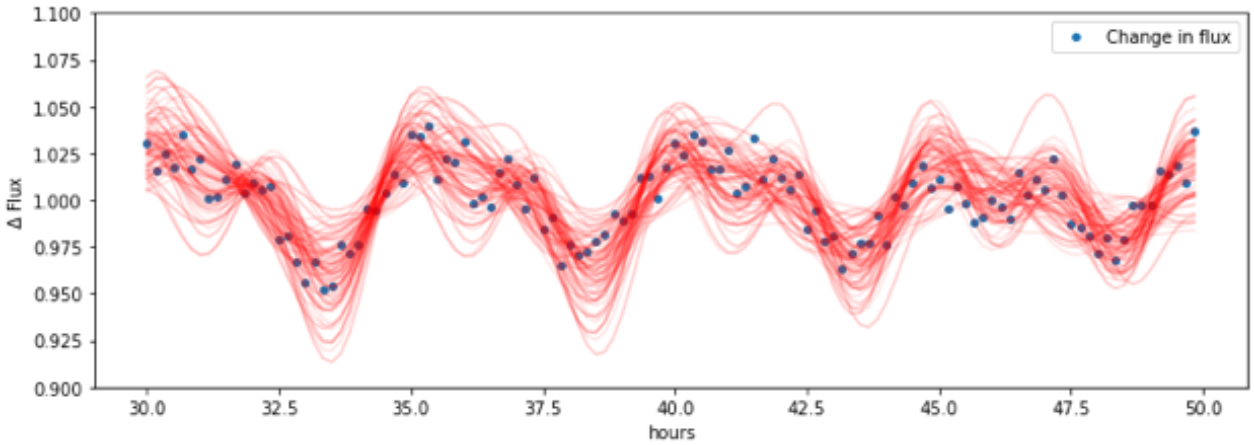
```
import emcee

def model(theta, t=time):
    a1, f1, a2, f2, a3, f3, w1, w2, w3 = theta
    model = 1 + a1*np.sin(w1*t + f1) + a2*np.sin(w2*t + f2) + a3*np.sin(w3
    return model

def lnlike(theta, x, y, yerr):
    LnLike = -0.5 * np.sum( ((y-model(theta,t=x))/yerr)**2 )
    return LnLike

def lnprior(theta, lim=lim):
    a1, f1, a2, f2, a3, f3, w1, w2, w3 = theta

    lim = [(1e-3, 1e-1), (-pi, pi),
           (1e-3, 1e-1), (-pi, pi),
           (1e-3, 1e-1), (-pi, pi),
           (1, 1.5),     (1, 1.5) , (2.5, 3)]

    mn, mx = np.array(lim).T
    x = theta[:, None]
    withinbounds = ((x >= mn) & (x <= mx)).all(1)

    if np.all(withinbounds):
        return 0.0
    else:
        return -np.inf

def lnprob(theta, x, y, yerr):
    lp = lnprior(theta)
    if np.isinf(lp):
        return -np.inf
    else:
        return lp + lnlike(theta, x, y, yerr) #recall if lp not -inf, its
```

1. We also need to set a value for nwalkers, which determines how many walkers are initialized in our MCMC. Let's use 500.
2. We need a variable called initial, which is an initial set of guesses (this will be the first theta, where the MCMC starts). **Foreman-Mackey & Hogg recommend that in many cases, running an optimizer first (e.g., from scipy) is the best way to select an initial starting value.**

In [298…

```
#set nwalkers
nwalkers = 200
niter = 10000
# initial = popt
# initial = [2e-2+5e-3*np.random.rand(), 5*np.random.rand(),
#            2e-2+5e-3*np.random.rand(), 5*np.random.rand(),
#            2e-2+5e-3*np.random.rand(), 5*np.random.rand(),
#            1.2+0.5*np.random.rand(), 3+0.5*np.random.rand(), 1.1+0.5*np.
### HYPEROPT guesses
initial = [1.08e-2, -2.48,
           1.66e-2, -0.69,
           1.14e-2, 1.49,
           1.156, 1.300, 2.528]
ndim = len(initial)

# p0=[]
# for i in range(nwalkers):
#     array = np.array([initial[0] + 1e-3*np.random.randn(1), initial[1] +
#                       initial[2] + 1e-3*np.random.randn(1), initial[3] +
#                       initial[4] + 1e-3*np.random.randn(1), initial[5] +
#                       initial[6] + 1e-1**np.random.randn(1), initial[7] +
#     p0.append(array.T)

weights = np.array((1e-2, 1e-3, 1e-2, 1e-3, 1e-2, 1e-3, 1e-3, 1e-3, 1e-3))
# weights = 1e-3
p0 = [np.array(initial) + weights*np.random.randn(ndim) for i in range(nwal

#create initial priors by scipy.optimize

def main(p0,nwalkers,niter,ndim,lnprob,data):
    sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args=df)

    print("Running burn-in...")
    p0, _, _ = sampler.run_mcmc(p0, 200, progress=True)
    sampler.reset()

    print("Running production...")
    pos, prob, state = sampler.run_mcmc(p0, niter, progress=True)

    return sampler, pos, prob, state

sampler, pos, prob, state = main(p0,nwalkers,niter,ndim,lnprob,df)

def plotter(sampler,t=time,flux=lc):
    plt.figure(figsize=(12,4))
    plt.plot(t,flux,label='Change in flux', ls='', marker='o', ms=4)
    samples = sampler.flatchain
    for theta in samples[np.random.randint(len(samples), size=100)]:
        plt.plot(t, model(theta, t), color="r", alpha=0.1)

    plt.xlabel('hours')
    plt.ylabel(r'$\Delta$ Flux')
    plt.legend(), plt.ylim((0.9, 1.1))
    plt.show()
plotter(sampler)
```

```
Running burn-in...
  0%|                                                                | 0/200 [00:00<?, ?
it/s]/Users/nguyendat/opt/anaconda3/lib/python3.9/site-packages/emcee/moves
/red_blue.py:99: RuntimeWarning: invalid value encountered in double_scalar
s
  lnpdiff = f + nlp - state.log_prob[j]
100%|████████████████████████████████████████████████| 200/200 [00:00<00:00, 242.90
it/s]
Running production...
100%|████████████████████████████████████████████████| 10000/10000 [00:39<00:00, 251.13
it/s]
```

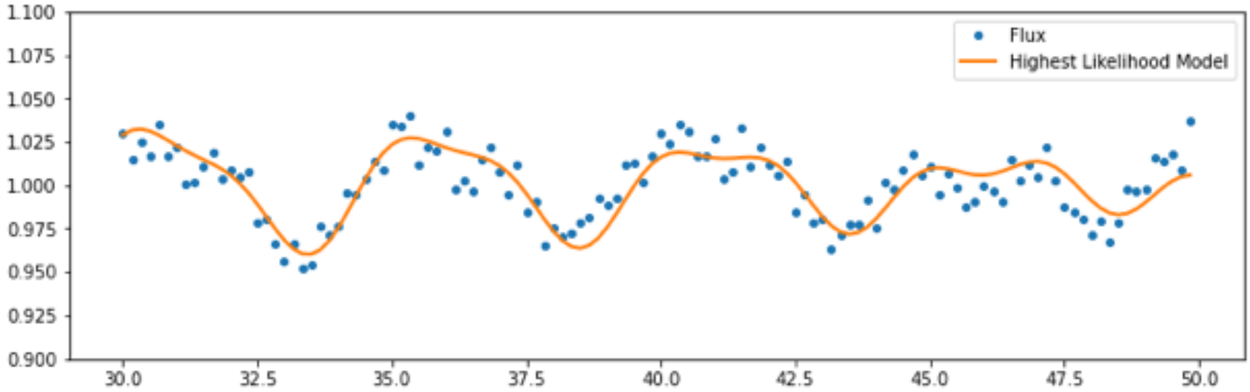


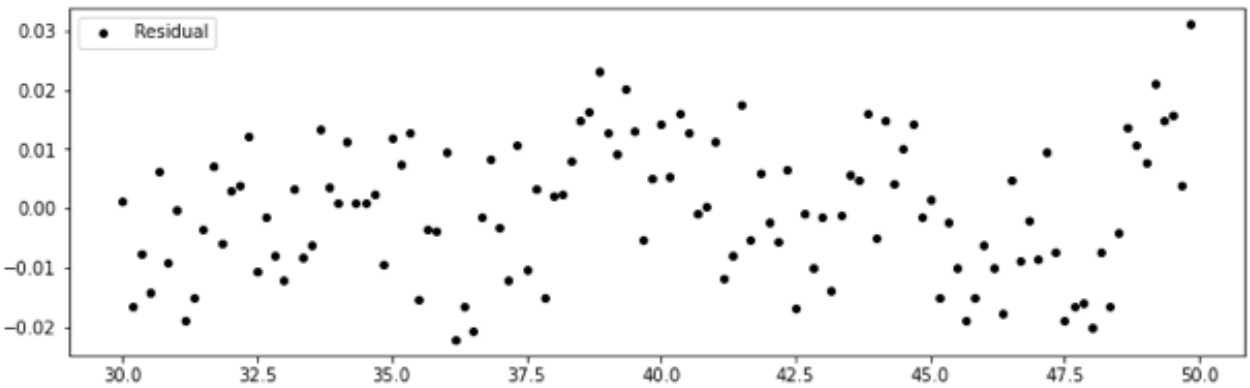
Now we plot the best fit solution and the prior probability distribution

In [299…

```
samples = sampler.flatchain

## BEST FIT SOLUTION
theta_max  = samples[np.argmax(sampler.flatlnprobability)]
print('Best theta fit')
print(theta_max[0:2])
print(theta_max[2:4])
print(theta_max[4:6])
print(theta_max[6:9])
print('Periods hours: ', 2*pi/theta_max[6:9])

best_fit_model = model(theta_max)
plt.figure(figsize=(12,8))

plt.subplot(211)
plt.plot(time,lc,label='Flux',ls='', marker='o', ms=4)
plt.plot(time,best_fit_model,label='Highest Likelihood Model', lw=2)
plt.ylim((0.9, 1.1)), plt.legend()
plt.subplot(212)
plt.plot(time,lc - best_fit_model,label='Residual', ls='', marker='o', ms=
plt.ylim((-0.06,0.06))
plt.show()
```

```
Best theta fit
[ 0.01748787 -2.4810307 ]
[ 0.01561891 -0.69121534]
[0.0085604  1.49026512]
[1.15457814 1.30077901 2.52869781]
Periods hours:  [5.44197493 4.83032495 2.48475136]
```



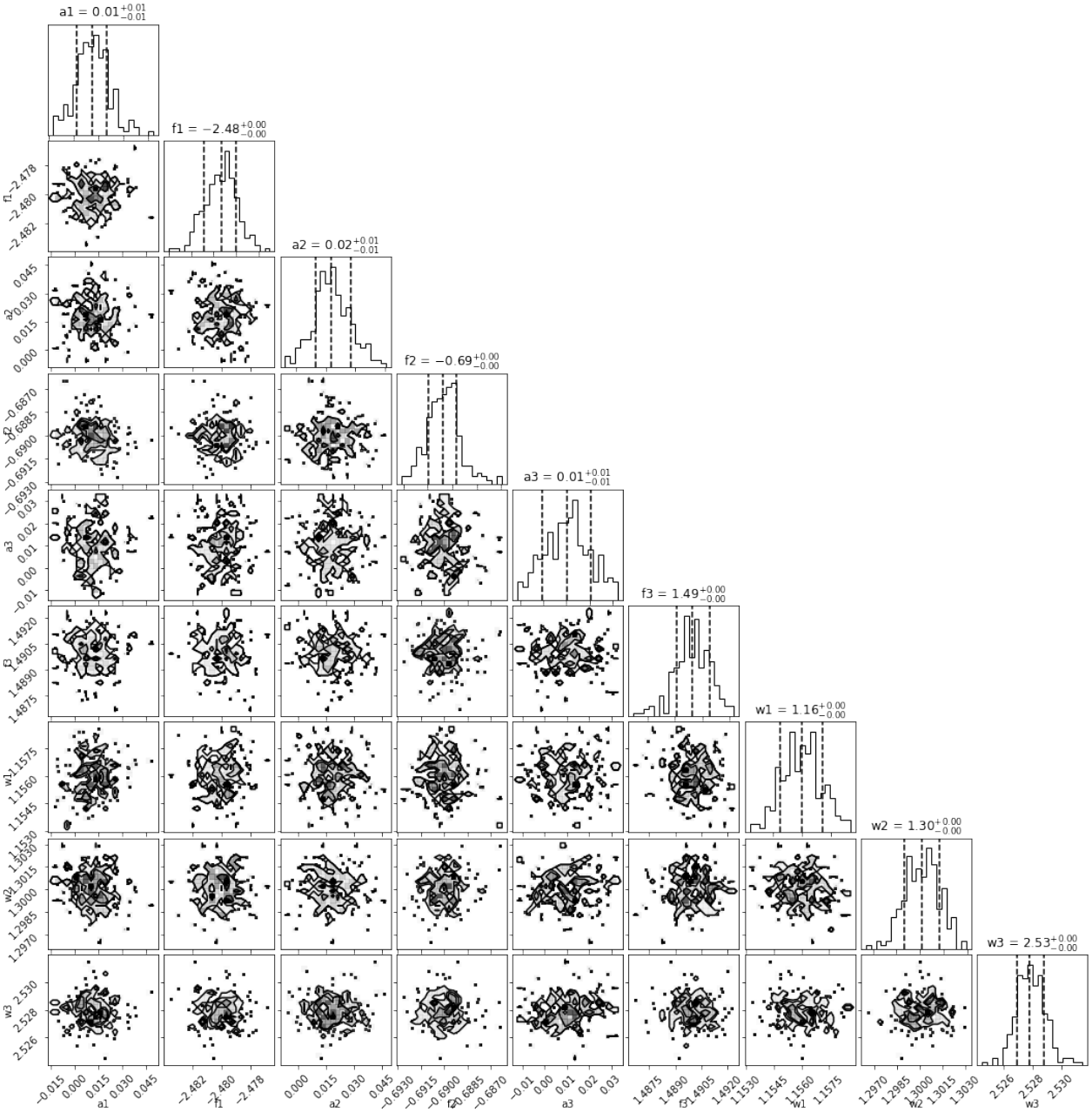


In [300…

```
## CORNER PLOT OF PRIOR PROBABILITY DISTRIBUTION
import corner
labels = ['a1', 'f1', 'a2', 'f2', 'a3', 'f3', 'w1', 'w2', 'w3']
p = plt.figure(figsize=(15,15))
fig = corner.corner(samples,show_titles=True,labels=labels,plot_datapoints=
```

```
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
```

```
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
WARNING:root:Too few points to create valid contours
```



In [ ]: