```
# One way to have some initial guess of model parameters (always needed):
# Run optimization in order to obtain a reference point to generate some
# initial guess for parameters in MCMC chains in later steps.
import scipy.optimize as opt
# any reasonable initial guess for 4 parameters in optimization procedure:
param_init=[1.0,1.0,1.0,1.0]
# nll is the negative log likelihood -- to be minizied:
nll= lambda *args: -lnlike(*args) # settings in opt.minimize...
result_opt = opt.minimize(nll, param_init, args=(x_true, y_true, z_obs))
print("optimazation:", result_opt['x'])
print("true value:",alpha_true, beta_x_true, beta_y_true, eps_true)
```

```
optimazation: [4.91230561 1.00815157 9.99252593 0.98342451]
true value: 5.0 1.0 10.0 1.0
```

```
# Now let's prepare many walkers/chains (to later sample the distribution)!
# It is unadvisable to use fewer walkers than twice the number of parameter-space dimensions
# Nparam = 4 here, suggested Nwalkers shall > 2 Nparam (= 8)
Nwalker,Ndim = 30,Nparam
# Set up (slightly different) initial parameter guess p0 for different walker/chain!!
rel_error_InitGuess=0.1   # relative (fractional) uncertainty for inital parameter guess around optimization results
#p0 = [param_true*(1+random.uniform(-rel_error_InitGuess,rel_error_InitGuess, size=Nparam)) for i in range(Nwalker)]
p0 = [result_opt['x']*(1+random.uniform(-rel_error_InitGuess,rel_error_InitGuess, size=Nparam)) for i in range(Nwalker)]
#p0 = [result_ls.x*(1+random.uniform(-rel_error_InitGuess,rel_error_InitGuess, size=Nparam)) for i in range(Nwalker)]
```

We know that MCMC is expected to fully sample the constructed (equilibrium) probability distribution $p(\vec{\theta})$, i.e., MCMC essentially samples the $\chi^2 = -\ln p(\vec{\theta})$ surface (for a Gaussian $p(\vec{\theta})$). Practically, we can first run an optimization procedure to get some initial guess for MCMC. However, there might be many local minima of χ^2 surface. Although a single chain might be able to jump between different local minima if it is long enough and with efficient sampling/update scheme, in order to maximize the chance to broadly sample the parameter space (to capture as many local minima as possible), we want to exploit multiple chains, each starting from a some different positions in the parameter space.

★ Fix mock dataset, do following experiment:

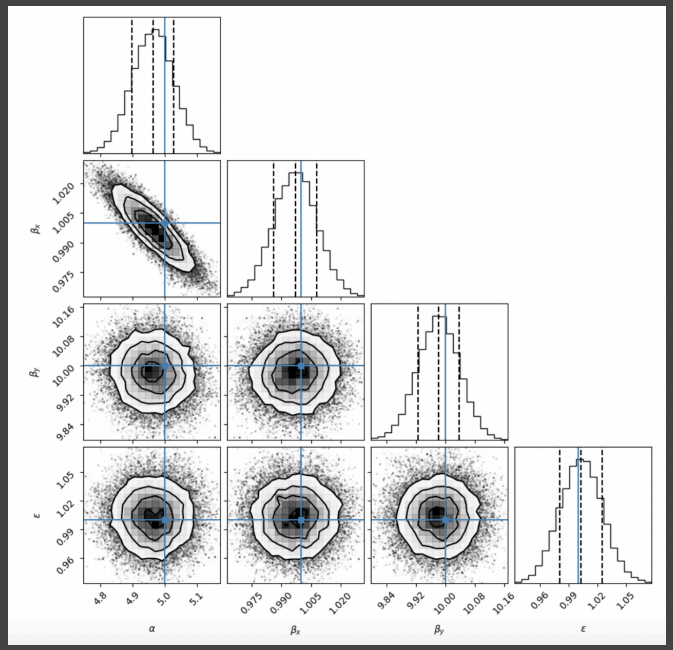
1. Change the prior function, observe how (1) the burn-in chain, (2) final marginalized posterior in the corner plot, change accordingly.

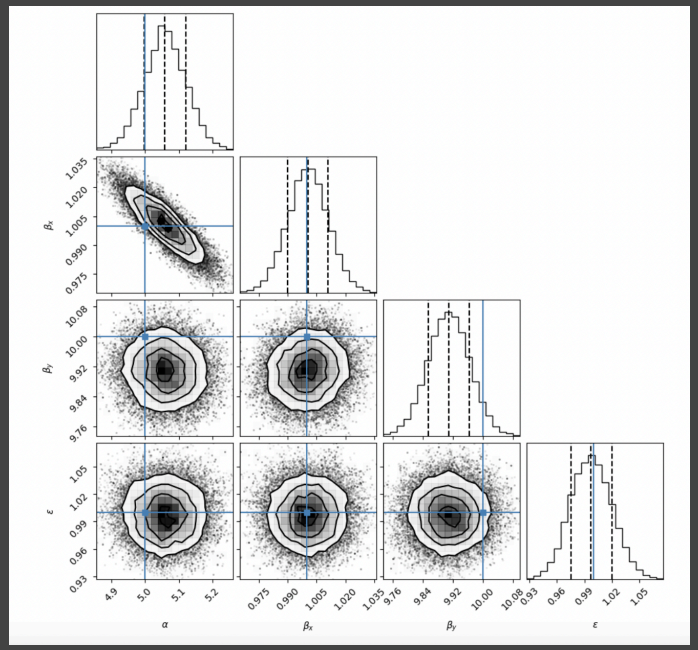
In [14]:
```
def lnprior(p):
    # The parameters are stored as a vector of values, so unpack them
    alpha,betax,betay,eps = p
    # We're using only uniform priors, and only eps has a lower bound
    if eps <= 0 or eps >=2 :
        return -inf
    return 0
```

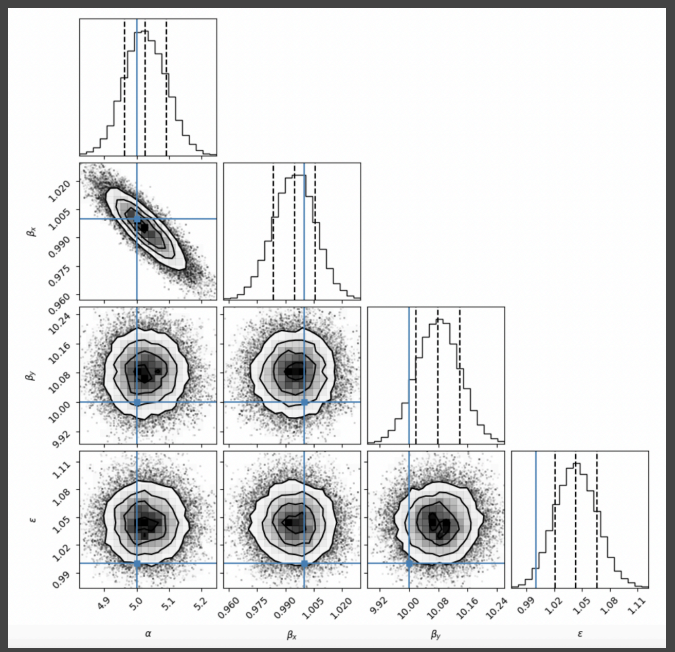
2. Change the prior function, observe how (1) the burn-in chain, (2) final marginalized posterior in the corner plot, change accordingly.

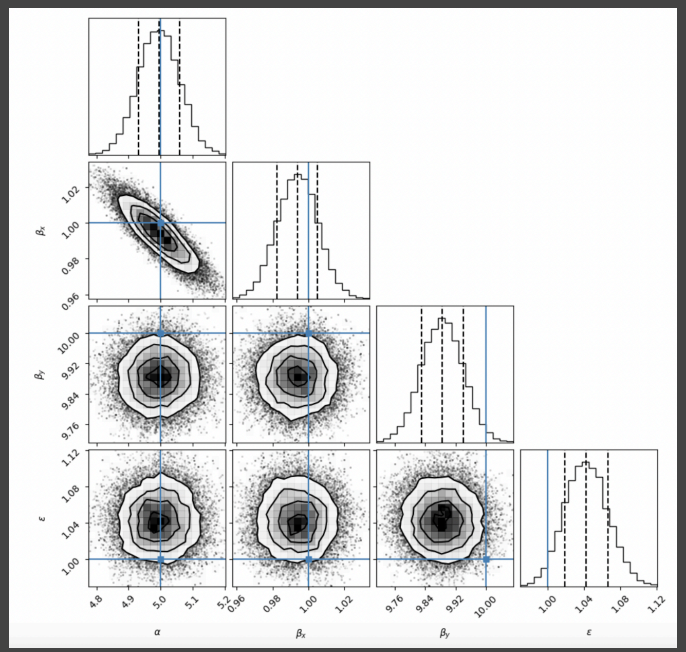
In [90]:
```
# Now let's prepare many walkers/chains (to later sample the distribution)!
# It is unadvisable to use fewer walkers than twice the number of parameter-space dimensions
# Nparam = 4 here, suggested Nwalkers shall > 2 Nparam (= 8)
Nwalker,Ndim = 30,Nparam

# Set up (slightly different) initial parameter guess p0 for different walker/chain!!
rel_error_InitGuess=1.0   # relative (fractional) uncertainty for inital parameter guess arou
p0 = [result_opt['x']*(1+random.uniform(-rel_error_InitGuess,rel_error_InitGuess, size=Nparar
```

1. Fix all settings and change the mock dataset by rerunning the program from the beginning, observe how the final results change.









2. Fix all settings and change the size of mock dataset, observe how the final results change.

★ In this simple exercise, fitting model is exactly the one that generates the mock data. See how the final inference results differ according to specific dataset.

★ Combining with experiments in the previous page, understand how the performance would depend on specific setup in inference procedure (e.g., choice of prior, initial guess etc) and how it is also limited given a specific dataset.