Lecture 14

MCMC convergence

and

pymc3 continued

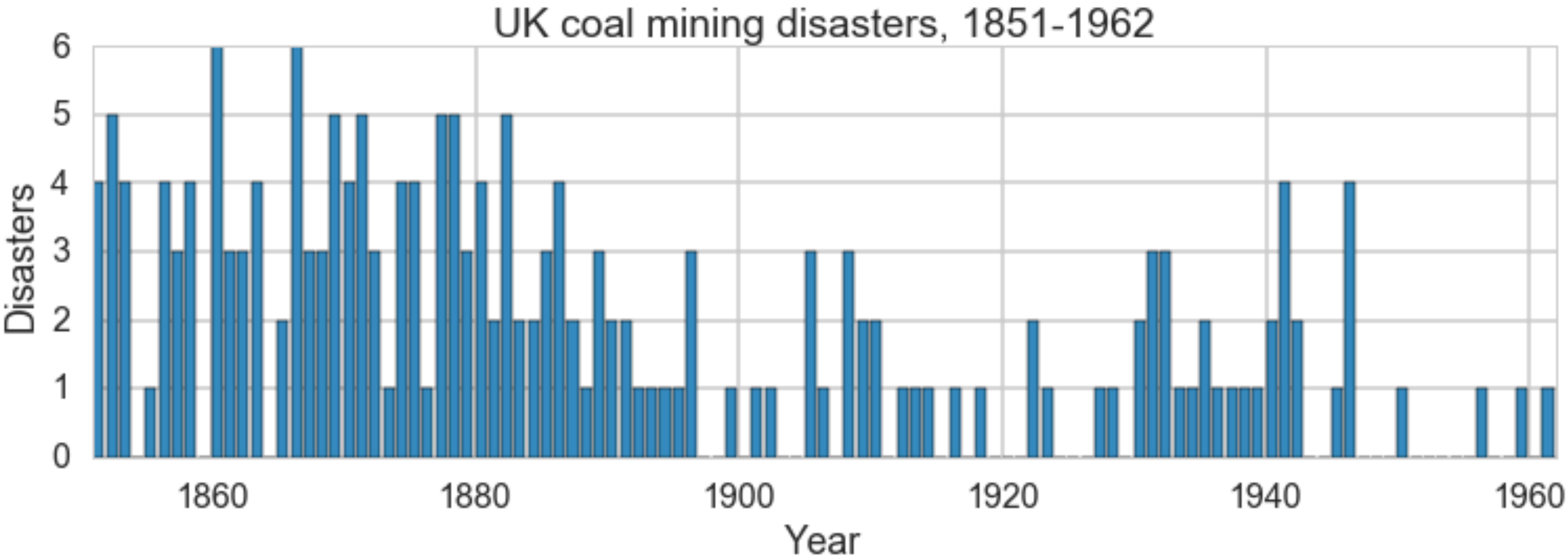
Last time

- the normal-normal model with MCMC
- then with pymc3
- bayesian regression and updating
- regularization and the ridge
- from the normal model to regression using pymc
- posterior vs predictive in regression problems

Today

- a switchpoint model
- more pymc3 practicalities
- data imputation using posterior predictives
- Gewecke, Gelman-Rubin, and ESS

- Regression with custom priors
- identifiability and sampling
- regression identifiability and lasso


UK coal mining disasters, 1851-1962
Disasters
Year
0
1
2
3
4
5
6
1860
1880
1900
1920
1940
1960

Model

$$y|\tau, \lambda_1, \lambda_2 \sim Poisson(r_t)$$

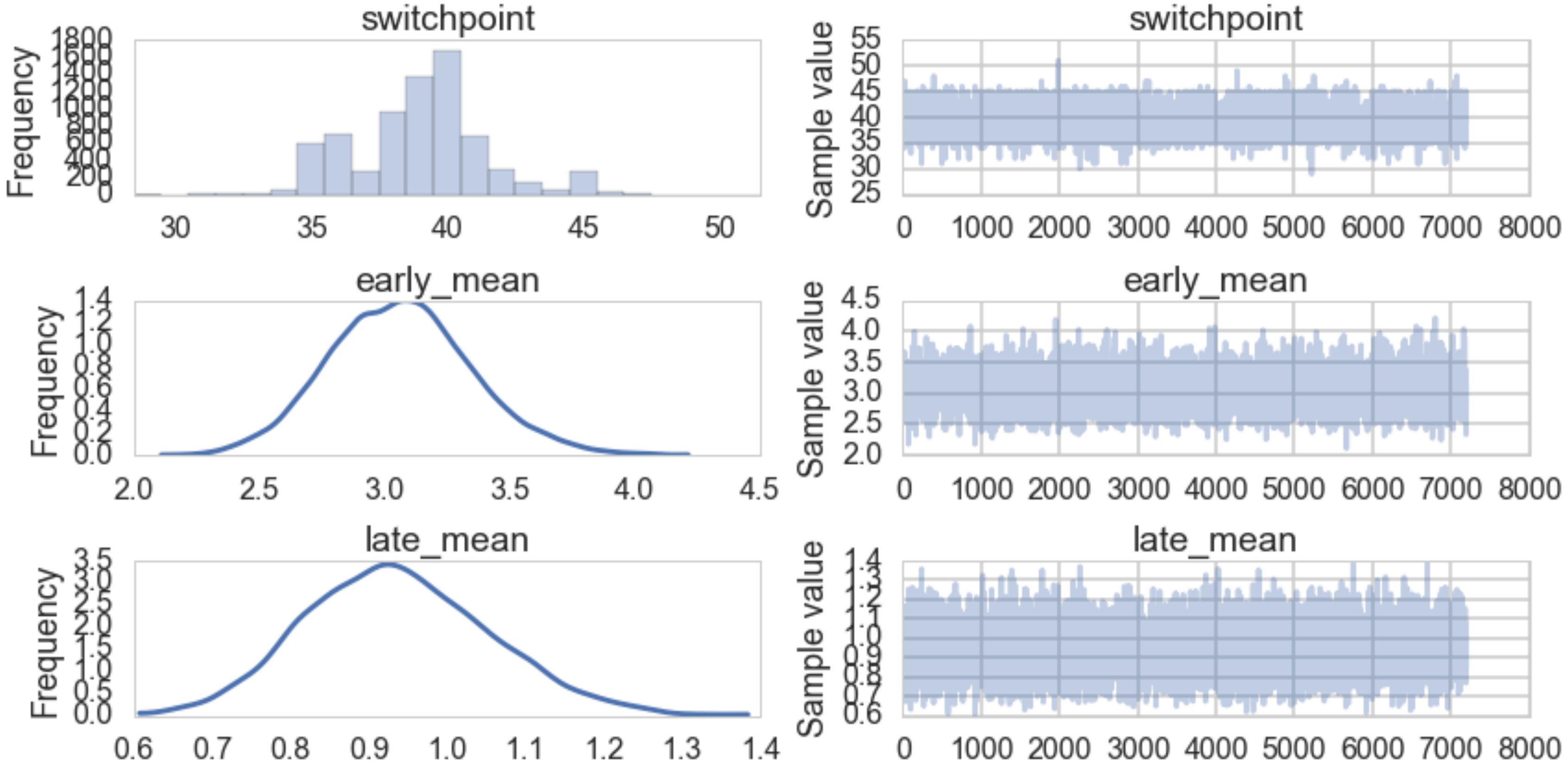
$$r_t = \lambda_1 \text{ if } t < \tau \text{ else } \lambda_2 \text{ for } t \in [t_l, t_h]$$
$$\tau \sim DiscreteUniform(t_l, t_h)$$

$$\lambda_1 \sim Exp(a)$$
$$\lambda_2 \sim Exp(b)$$

```python
from pymc3.math import switch
with pm.Model() as coaldis1:
    early_mean = pm.Exponential('early_mean', 1)
    late_mean = pm.Exponential('late_mean', 1)
    switchpoint = pm.DiscreteUniform('switchpoint', lower=0, upper=n_years)
    rate = switch(switchpoint >= np.arange(n_years), early_mean, late_mean)
    disasters = pm.Poisson('disasters', mu=rate, observed=disasters_data)
```

```python
with coaldis1:
    stepper=pm.Metropolis()
    trace = pm.sample(40000, step=stepper)
```

```
100%|██████████| 40000/40000 [00:12<00:00, 3326.53it/s] | 229/40000 [00:00<00:17, 2289.39it/s]
```



```
>>>coaldis1.vars #stochastics
[early_mean_log_, late_mean_log_, switchpoint]
>>>coaldis1.deterministics #deterministics
[early_mean, late_mean]
>>>coaldis1.observed_RVs
[disasters]
>>>ed=pm.Exponential.dist(1)
<class 'pymc3.distributions.continuous.Exponential'>
>>>ed.random(size=10)
array([ 1.18512233,  2.45533355,  0.04187961,  3.32967837,  0.0268889 ,
        0.29723148,  1.30670324,  0.23335826,  0.56203427,  0.15627659])
>>>type(switchpoint), type(early_mean)
(pymc3.model.FreeRV, pymc3.model.TransformedRV)
>>>switchpoint.logp({'switchpoint':55,
    'early_mean_log_':1, 'late_mean_log_':1})
array(-4.718498871295094)
```

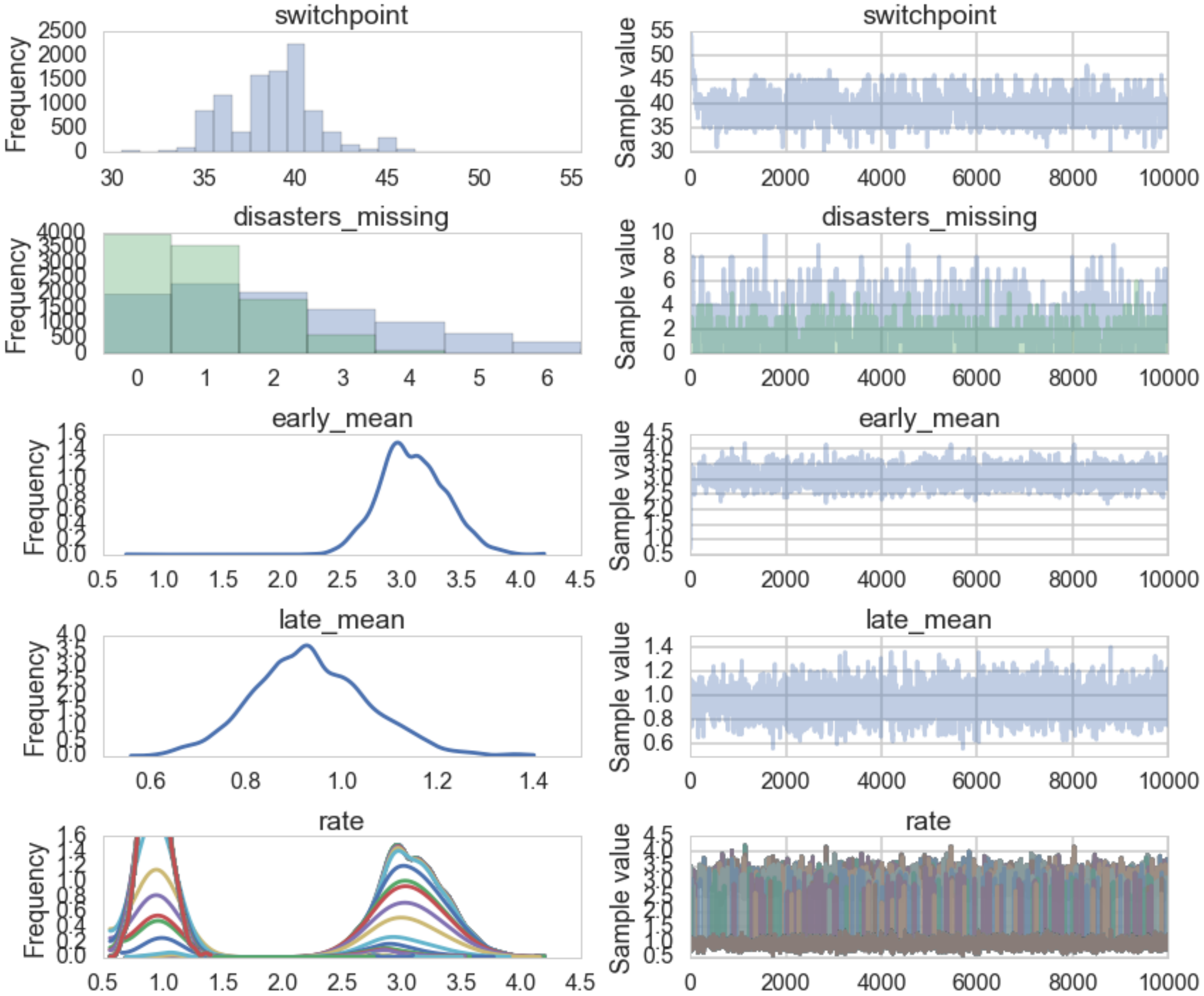
Imputation

```
>>>disasters_missing = np.array([ 4, 5, 4, 0, 1, 4, 3, 4, 0, 6, 3, 3, 4, 0, 2, 6,
3, 3, 5, 4, 5, 3, 1, 4, 4, 1, 5, 5, 3, 4, 2, 5,
2, 2, 3, 4, 2, 1, 3, -999, 2, 1, 1, 1, 1, 3, 0, 0,
1, 0, 1, 1, 0, 0, 3, 1, 0, 3, 2, 2, 0, 1, 1, 1,
0, 1, 0, 1, 0, 0, 0, 2, 1, 0, 0, 0, 1, 1, 0, 2,
3, 3, 1, -999, 2, 1, 1, 1, 1, 2, 4, 2, 0, 0, 1, 4,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1])
>>>disasters_masked = np.ma.masked_values(disasters_missing, value=-999)
```

An array with mask set to True where data is missing.

```
with pm.Model() as missing_data_model:
    switchpoint = pm.DiscreteUniform('switchpoint', lower=0, upper=len(disasters_masked))
    early_mean = pm.Exponential('early_mean', lam=1.)
    late_mean = pm.Exponential('late_mean', lam=1.)
    idx = np.arange(len(disasters_masked))
    rate = pm.Deterministic('rate', switch(switchpoint >= idx, early_mean, late_mean))
    disasters = pm.Poisson('disasters', rate, observed=disasters_masked)

with missing_data_model:
    stepper=pm.Metropolis()
    trace_missing = pm.sample(10000, step=stepper)

pm.summary(trace_missing, varnames=['disasters_missing'])
```

```
disasters_missing:

  Mean             SD               MC Error         95% HPD interval
  -------------------------------------------------------------------

  2.189            1.825            0.078            [0.000, 6.000]
  0.950            0.980            0.028            [0.000, 3.000]

  Posterior quantiles:
  2.5            25             50             75             97.5
  |--------------|==============|==============|--------------|

  0.000          1.000          2.000          3.000          6.000
  0.000          0.000          1.000          2.000          3.000
```

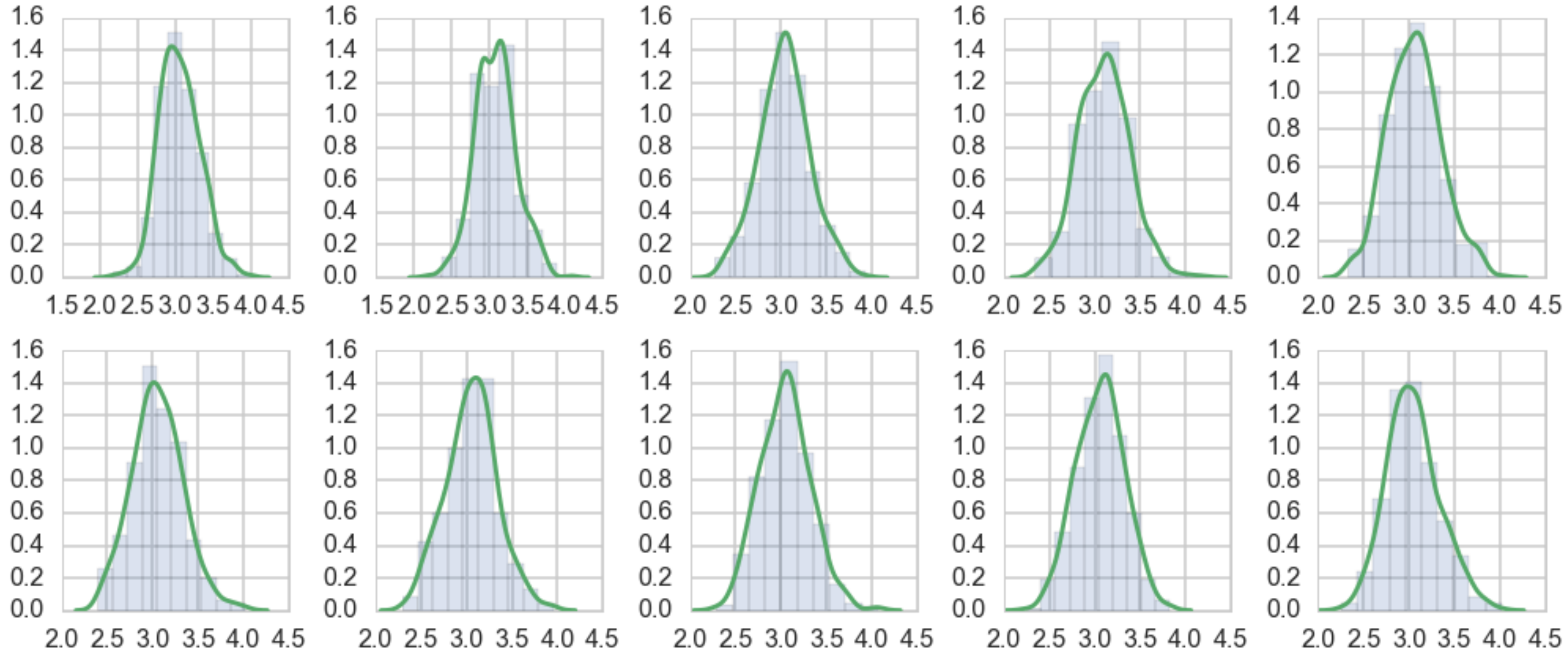


Model convergence

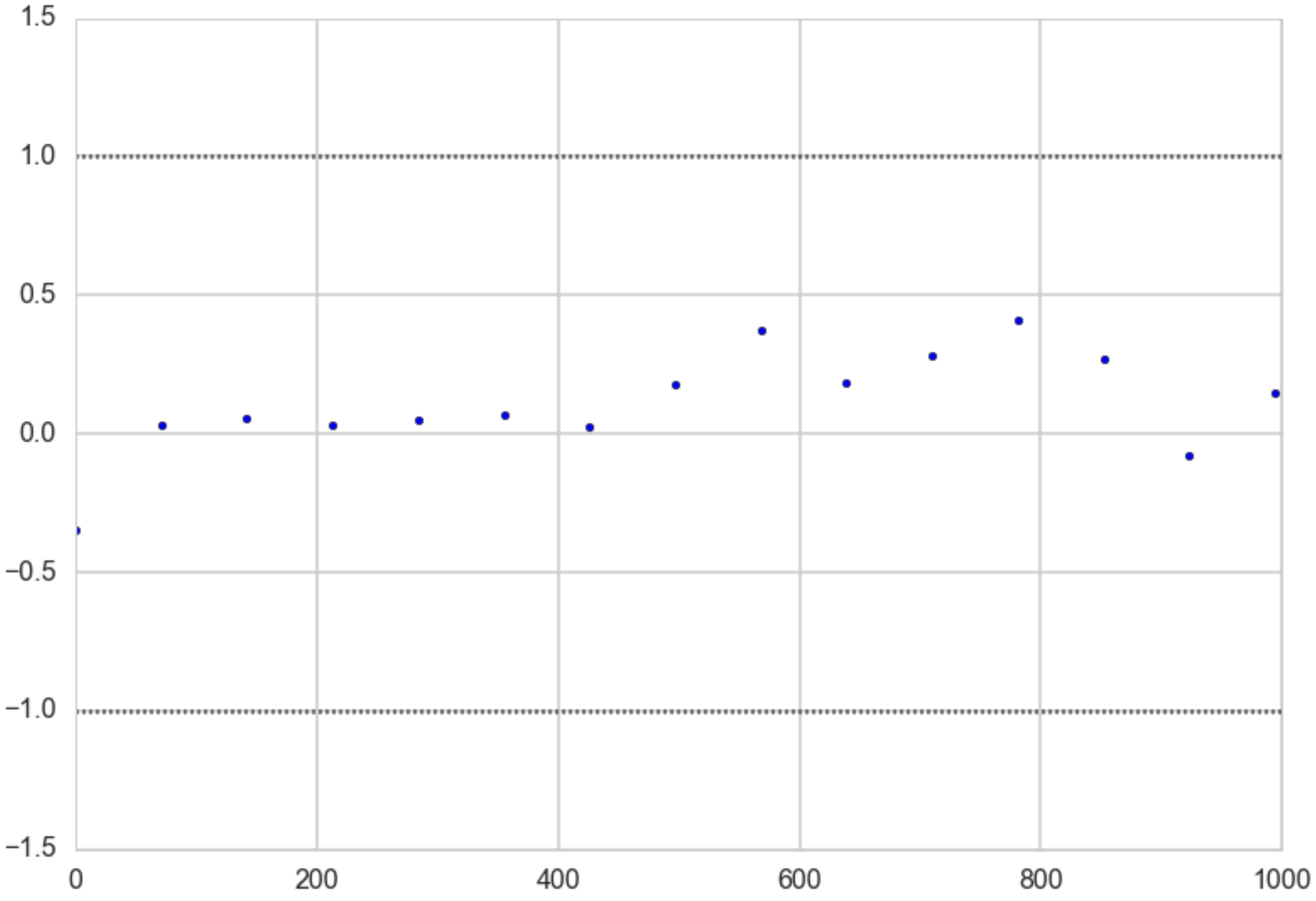
- traces white noisy
- diagnose autocorrelation, check parameter correlations

```
pm.trace_to_dataframe(trace).corr()
```

- visually inspect histogram every m samples
- traceplots from different starting points, different chains
- formal tests: Gewecke, Gelman-Rubin, Effective Sample Size



Gewecke: difference of means



$$H_0 : \mu_{\theta_1} - \mu_{\theta_2} = 0 \implies \mu_{\theta_1 - \theta_2} = 0$$

$$\sigma_{\theta_1 - \theta_2} = \sqrt{\frac{var(\theta_1)}{n_1} + \frac{var(\theta_2)}{n_2}}$$

$$|\mu_{\theta_1} - \mu_{\theta_2}| < 2\sigma_{\theta_1 - \theta_2}$$

```
with coaldis1:
    stepper=pm.Metropolis()
    tr = pm.sample(2000, step=stepper)

z = geweke(tr, intervals=15)

plt.scatter(*z['early_mean'].T)
plt.hlines([-1,1], 0, 1000, linestyles='dotted')
plt.xlim(0, 1000)
```

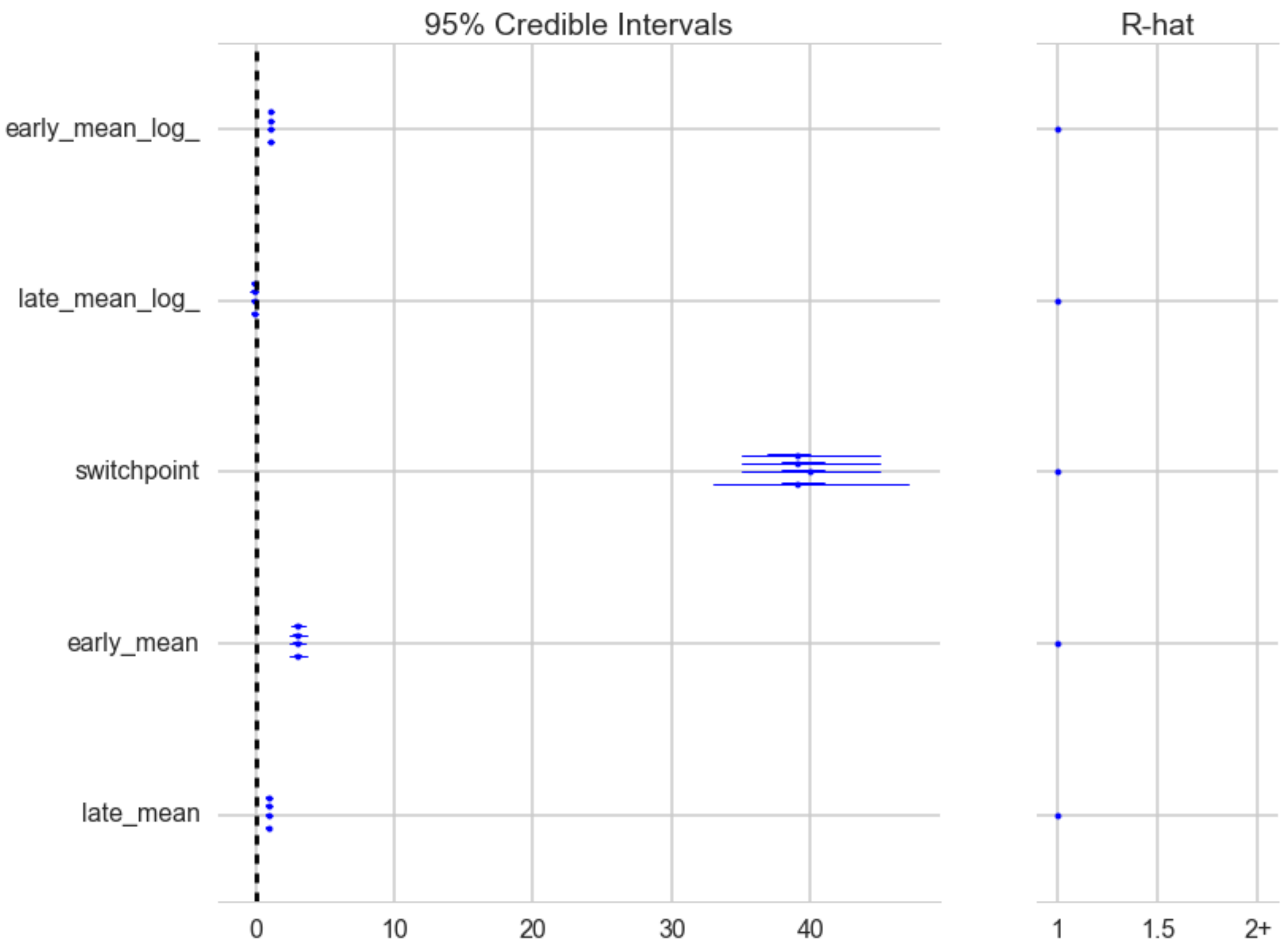
Gelman-Rubin

Multiple chains..compute within chain variance and compare to between chain variance

$$s_j^2 = \frac{1}{n-1}\sum_i (\theta_{ij} - \mu_{\theta_j})^2$$

$$w = \frac{1}{m}\sum_j s_j^2;\ \ \mu = \frac{1}{m}\sum_j \mu_{\theta_j}$$

$$B = \frac{n}{m-1}\sum_j (\mu_{\theta_j} - \mu)^2$$



Use weighted average of w and B to estimate variance of the stationary distribution `pm.gelman_rubin(trace)`:

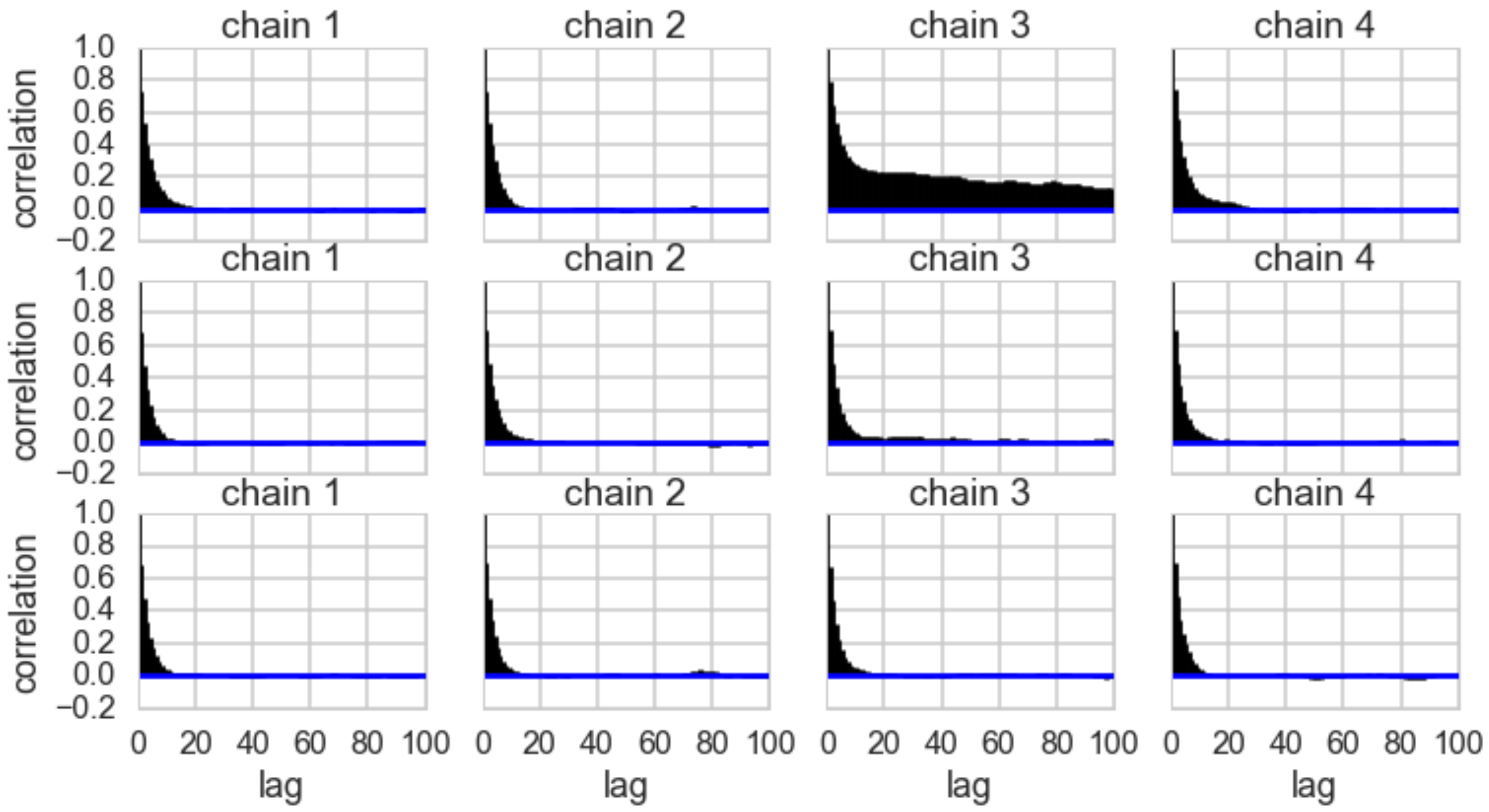
$$\hat{Var}(\theta) = (1 - \frac{1}{n})w + \frac{1}{n}B$$

Overestimates our variance, but unbiased under stationarity.

Ratio of the estimated distribution variance to asymptotic one:

$$\hat{R} = \sqrt{\frac{\hat{Var}(\theta)}{w}}$$

ESS: Effective Sample Size



IIDness of draws decreases

```
pm.effective_n(trace)
```

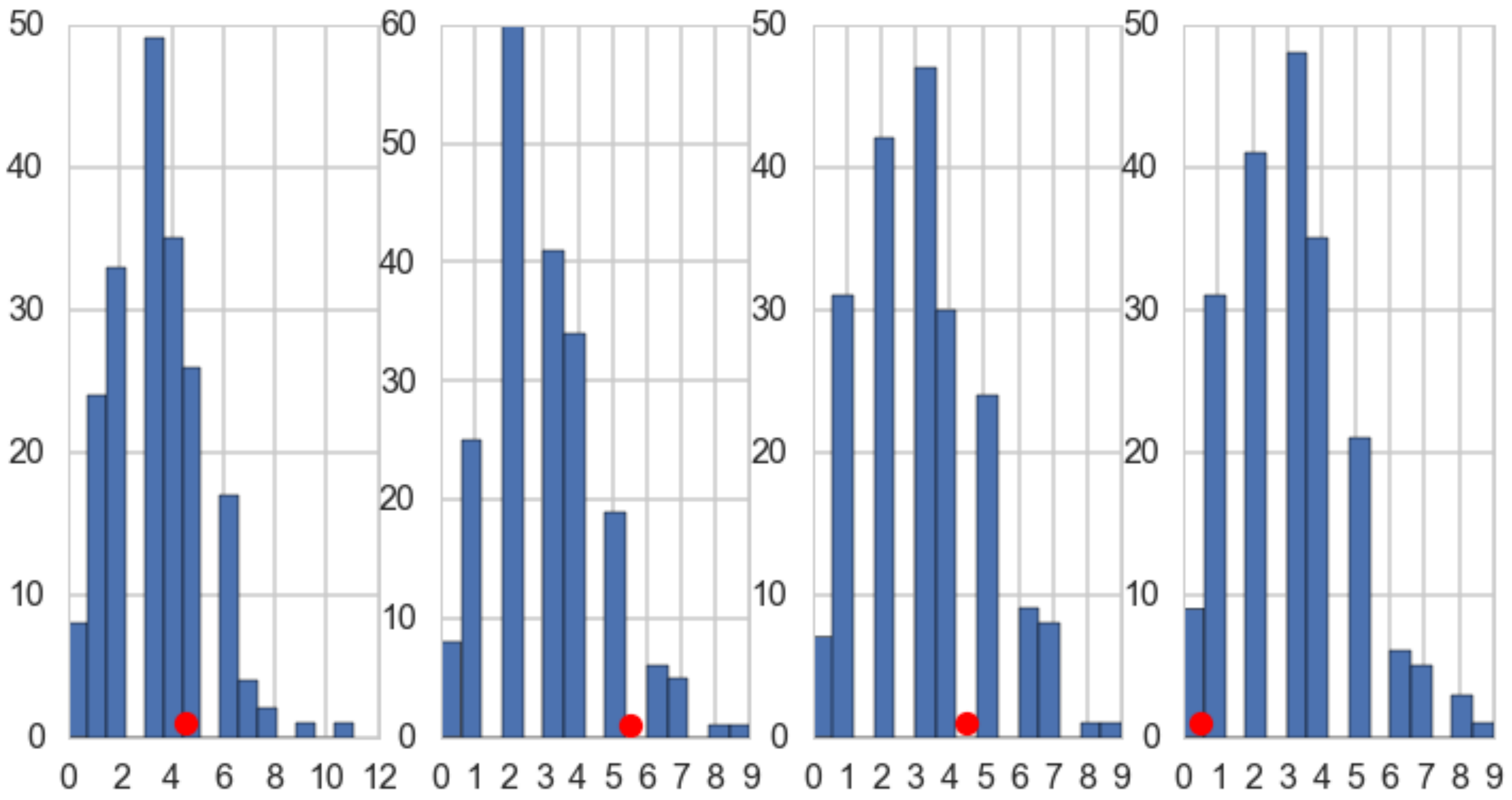
```
{'early_mean': 16857.0,
 'early_mean_log_': 12004.0,
 'late_mean': 27344.0,
 'late_mean_log_': 27195.0,
 'switchpoint': 195.0}
```

(40000 samples)

$$n_{eff} = \frac{mn}{1 + 2\sum_{\Delta t} \rho_{\Delta t}}$$

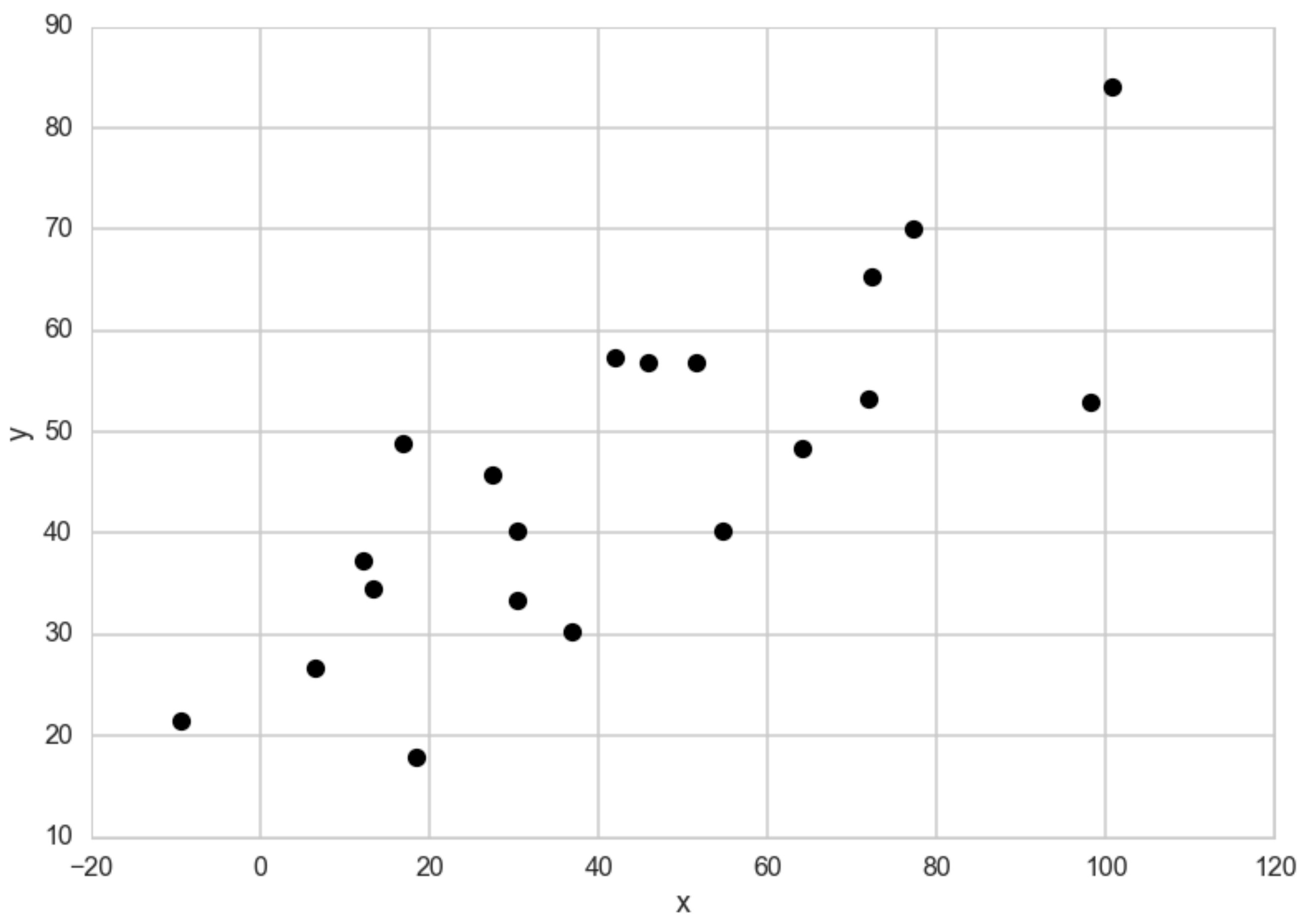
Posterior Predictive Checks

```
with coaldis1:
    sim = pm.sample_ppc(t2, samples=200)
```



Regression (again)

Data:



Model

$$\alpha \sim Uniform(-100, 100)$$
$$\beta \sim (1 + \beta^2)^{-3/2}$$

$$\sigma \sim 1/\sigma$$
$$\mu = \alpha + \beta x$$

$$y \sim N(\mu, \sigma)$$

Priors

For σ: $1/\sigma$ is Jeffrey's prior for $\sigma|\mu$.

For μ use symmetry: $y = \alpha + \beta x; x = \alpha^{'} + \beta^{'} y$

Thus $\alpha^{'} = \beta/\alpha$, and $\beta^{'} = 1/\beta$.

Jacobian is $\beta^3 \implies q(\alpha^{'}, \beta^{'}) = \beta^3 p(\alpha, \beta)$

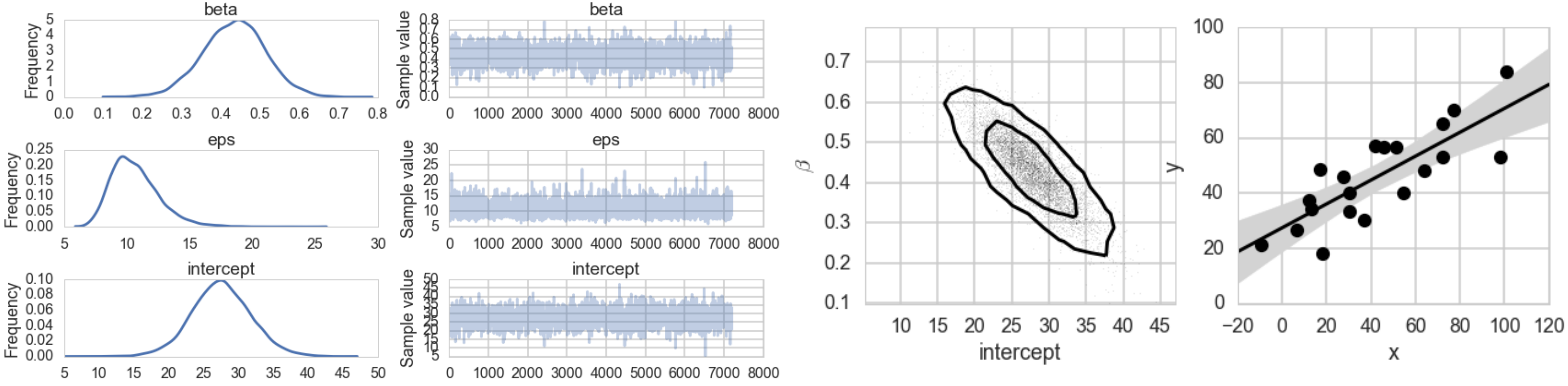
Thus $p(-\alpha/\beta, 1/\beta) = \beta^3 p(\alpha, \beta)$ gives $\beta \sim (1 + \beta^2)^{-3/2}$

Sampling

```python
import theano.tensor as T

with pm.Model() as model1:
    alpha = pm.Uniform('intercept', -100, 100)

    # Create custom densities, you must supply logp
    beta = pm.DensityDist('beta', lambda value: -1.5 * T.log(1 + value**2), testval=0)
    eps = pm.DensityDist('eps', lambda value: -T.log(T.abs_(value)), testval=1)

    # Create likelihood
    like = pm.Normal('y_est', mu=alpha + beta * xdata, sd=eps, observed=ydata)

with model1:
    stepper=pm.Metropolis()
    tracem1 = pm.sample(40000, step=stepper)
```

```
100%|██████████| 40000/40000 [00:10<00:00, 3952.57it/s] | 405/40000 [00:00<00:09, 4044.57it/s]
```

Results



Bug in pymc3 means `njobs` based tests fail. Run sequentially. Geweckes close to 0.

Non-Identifiability

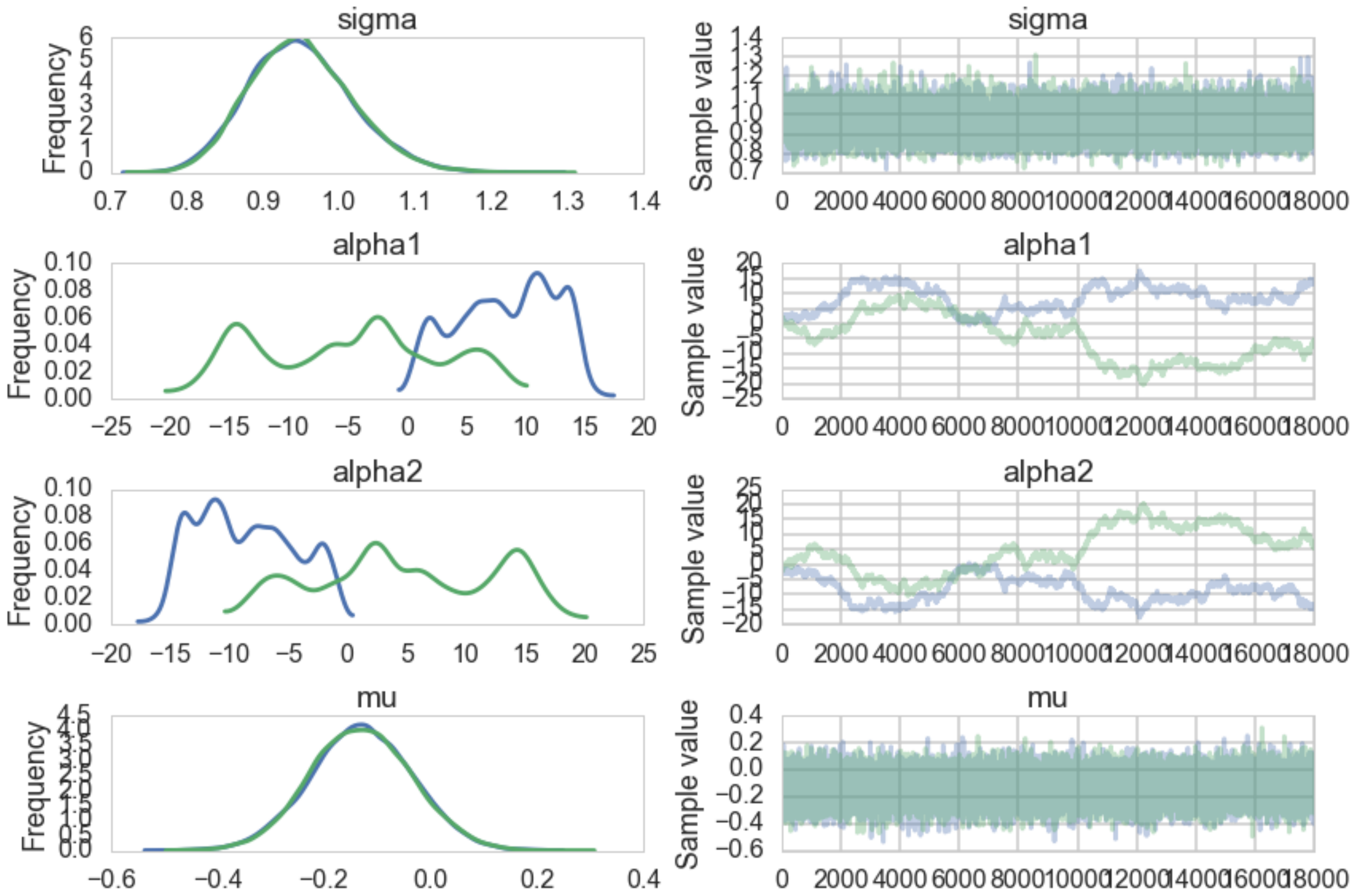
Generate data from N(0,1). Then fit:

$$y \sim N(\mu, \sigma)$$

$$\mu = \alpha_1 + \alpha_2$$

$$\alpha_1 \sim Unif(-\infty, \infty)$$
$$\alpha_2 \sim Unif(-\infty, \infty)$$
$$\sigma \sim HalfCauchy(0, 1)$$

Non-Identifiability




```
with pm.Model() as ni:
    sigma = pm.HalfCauchy("sigma", beta=1)
    alpha1=pm.Uniform('alpha1', lower=-10**6, upper=10**6)
    alpha2=pm.Uniform('alpha2', lower=-10**6, upper=10**6)
    mu = pm.Deterministic("mu", alpha1 + alpha2)
    y = pm.Normal("data", mu=mu, sd=sigma, observed=data)
    stepper=pm.Metropolis()
    traceni = pm.sample(100000, step=stepper, njobs=2)

100%|██████████| 100000/100000 [01:17<00:00, 1286.59it/s]| 1/100000 [00:00<3:20:31,  8.31it/s]
```

```
df=pm.trace_to_dataframe(traceni)
df.corr()
```

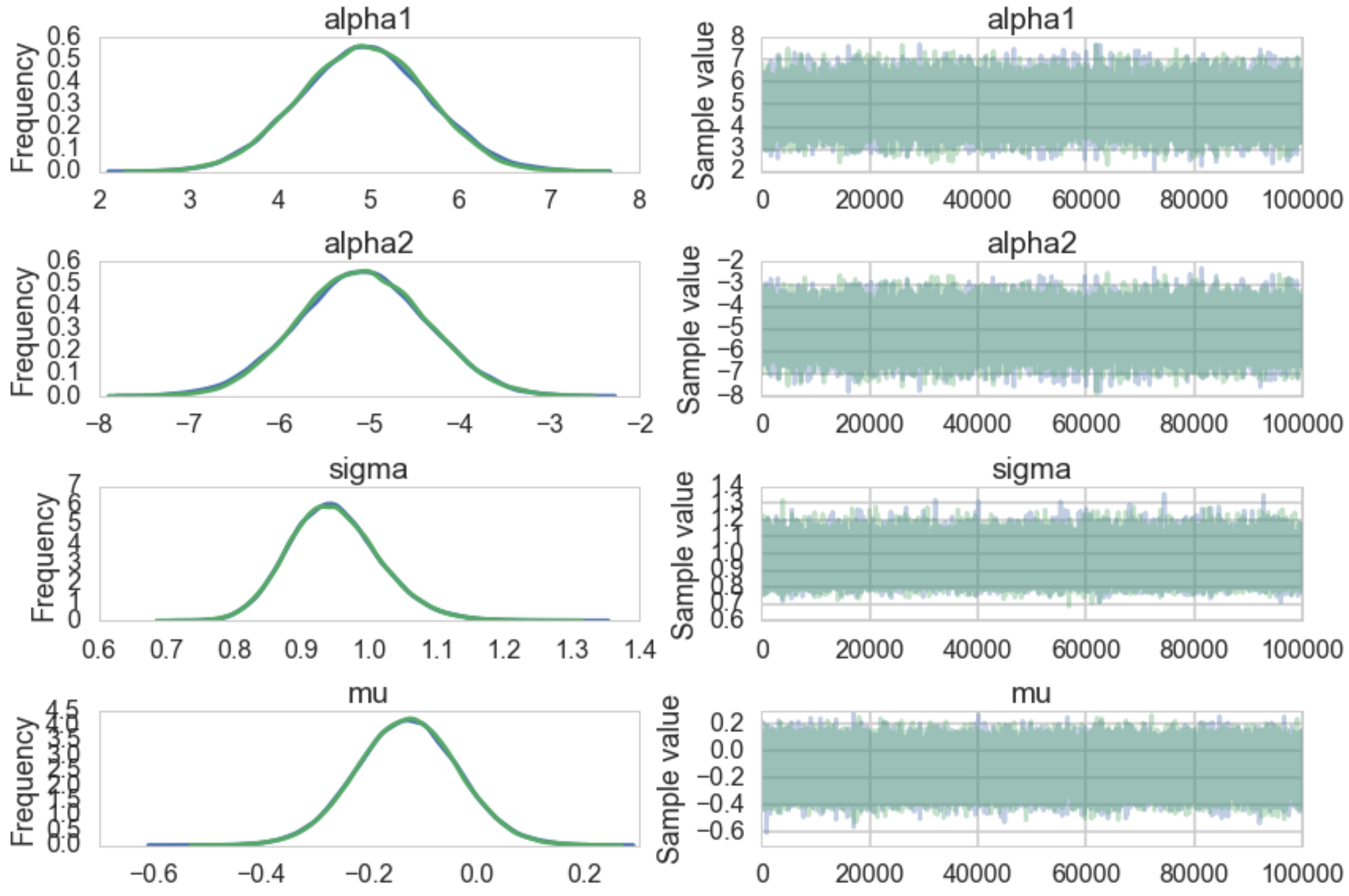
	sigma	mu	alpha1	alpha2
sigma	1.000000	-0.000115	-0.003153	0.003152
mu	-0.000115	1.000000	0.002844	0.008293
alpha1	-0.003153	0.002844	1.000000	-0.999938
alpha2	0.003152	0.008293	-0.999938	1.000000

```
>>>pm.effective_n(traceni)
{'alpha1': 1.0,
 'alpha1_interval_': 1.0,
 'alpha2': 1.0,
 'alpha2_interval_': 1.0,
 'mu': 26411.0,
 'sigma': 39215.0,
 'sigma_log_': 39301.0}
 >>>pm.gelman_rubin(traceni)
 {'alpha1': 1.7439881580327452,
  'alpha1_interval_': 1.7439881580160093,
  'alpha2': 1.743862659352983l,
  'alpha2_interval_': 1.7438626593368223,
  'mu': 0.99999710182062695,
  'sigma': 1.0000248056117549,
  'sigma_log_': 1.0000261752214563}
```

Attempt to fix

```
with pm.Model() as ni2:
    sigma = pm.HalfCauchy("sigma", beta=1)
    alpha1=pm.Normal('alpha1', mu=5, sd=1)
    alpha2=pm.Normal('alpha2', mu=-5, sd=1)
    mu = pm.Deterministic("mu", alpha1 + alpha2)
    y = pm.Normal("data", mu=mu, sd=sigma, observed=data)
    #stepper=pm.Metropolis()
    #traceni2 = pm.sample(100000, step=stepper, njobs=2)
    traceni2 = pm.sample(100000, njobs=2)

Average ELBO = -143.13: 100%|██████████| 200000/200000 [00:18<00:00, 10759.64it/s], 9912.87it/s]
100%|██████████| 100000/100000 [06:30<00:00, 255.83it/s]
```

NUTS sampler slower but covers better for this



We construct a model

$$y = 10x_1 + 10x_2 + 0.1x_3$$

where $x_1 \sim N(0,1)$, $x_2 = -x_1 + N(0, 10^{-3})$ and $x_3 \sim N(0,1)$

Thus our real model is

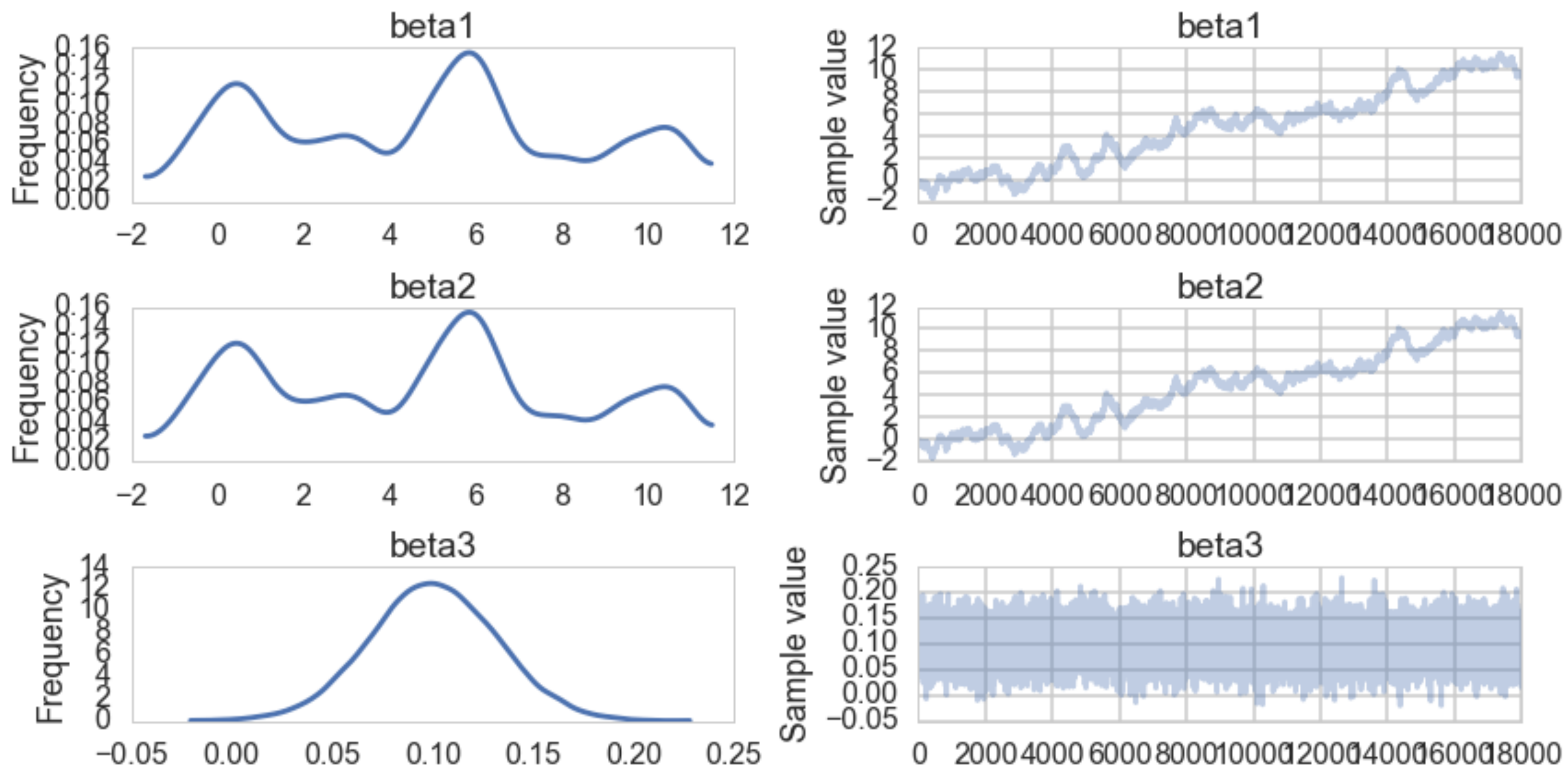
$$y = 10N(0, 10^{-3}) + 0.1N(0,1)$$

```
>>>np.dot(np.dot(np.linalg.inv(np.dot(X.T, X)), X.T), y)
array([ 10. ,  10. ,   0.1])
```

Model 1: uniform priors

```
beta_min = -10**6
beta_max = 10**6
with pm.Model() as uni:
    beta1 = pm.Uniform('beta1', lower=beta_min, upper=beta_max)
    beta2 = pm.Uniform('beta2', lower=beta_min, upper=beta_max)
    beta3 = pm.Uniform('beta3', lower=beta_min, upper=beta_max)
    mu = beta1*x1 + beta2*x2 + beta3*x3
    ys = pm.Normal('ys', mu=mu, tau=1.0, observed=y)
    stepper=pm.Metropolis()
    traceuni = pm.sample(100000, step=stepper)
```

```
100%|██████████| 100000/100000 [00:35<00:00, 2856.75it/s]| 1/100000 [00:00<4:16:19,  6.50it/s]
beta3:

  Mean             SD               MC Error         95% HPD interval
  -------------------------------------------------------------------

  0.100            0.032            0.000            [0.040, 0.165]

  Posterior quantiles:
  2.5            25             50             75             97.5
  |--------------|==============|==============|--------------|

  0.038          0.079          0.100          0.122          0.163
```

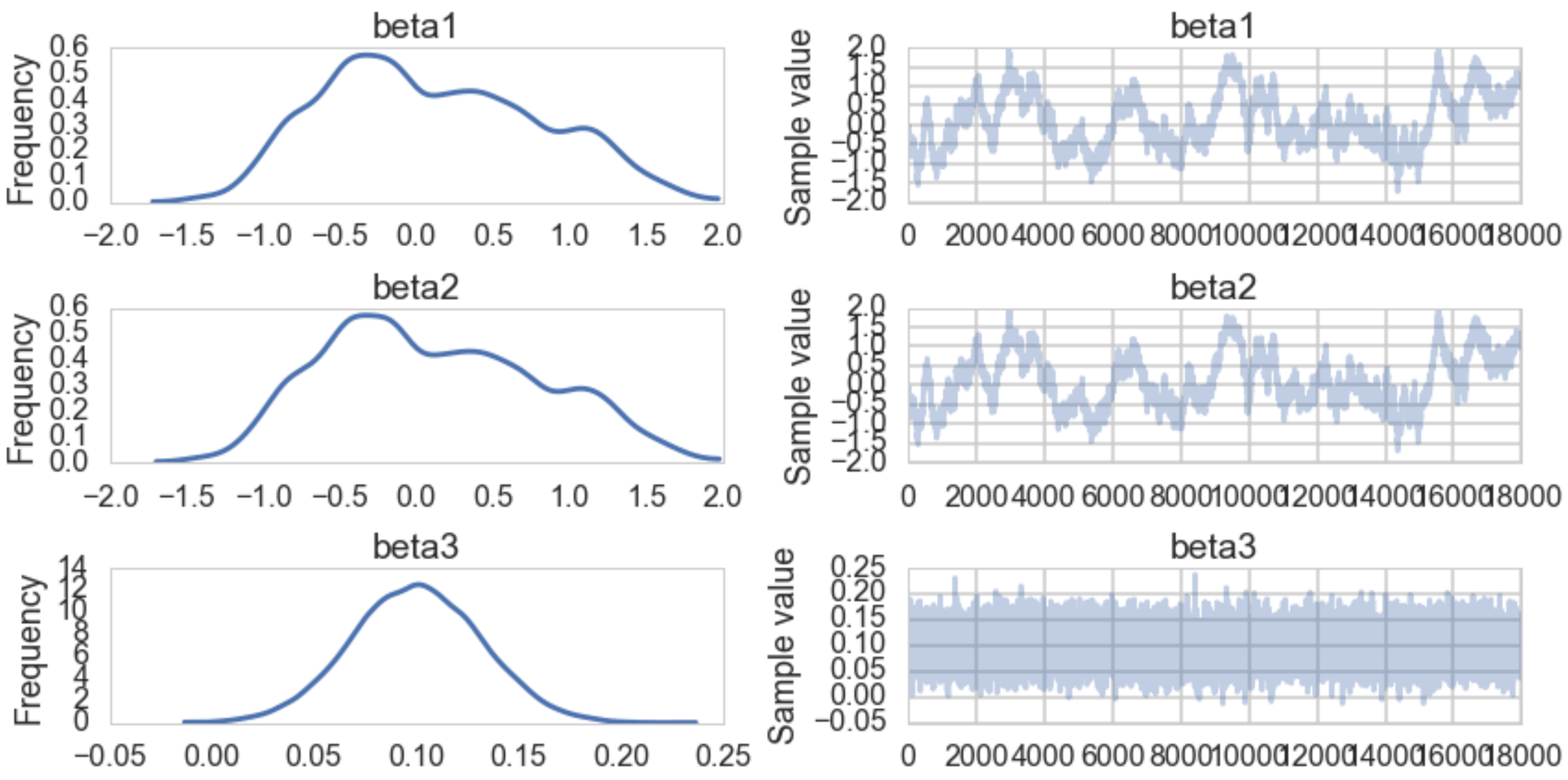


Model2: Ridge

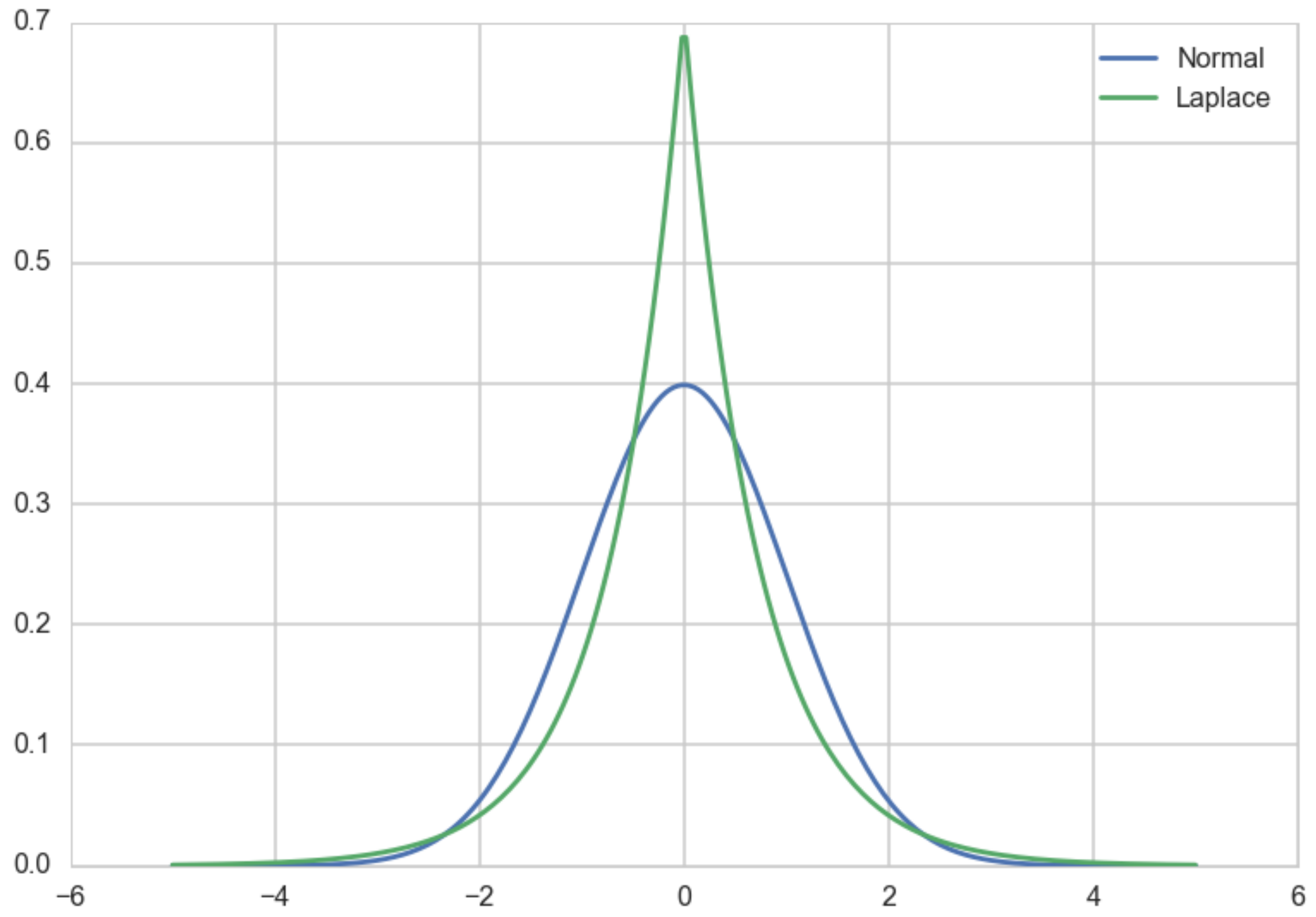
```python
with pm.Model() as ridge:
    beta1 = pm.Normal('beta1', mu=0, tau=1.0)
    beta2 = pm.Normal('beta2', mu=0, tau=1.0)
    beta3 = pm.Normal('beta3', mu=0, tau=1.0)
    mu = beta1*x1 + beta2*x2 + beta3*x3
    ys = pm.Normal('ys', mu=mu, tau=1.0, observed=y)
    stepper=pm.Metropolis()
    traceridge = pm.sample(100000, step=stepper)
```

```
100%|██████████| 100000/100000 [00:28<00:00, 3487.86it/s]| 68/100000 [00:00<02:27, 679.28it/s]
beta3:

  Mean             SD               MC Error         95% HPD interval
  -------------------------------------------------------------------

  0.100            0.032            0.000            [0.035, 0.159]

  Posterior quantiles:
  2.5            25             50             75             97.5
  |--------------|==============|==============|--------------|

  0.038          0.079          0.100          0.122          0.162
```

```python
with ridge:
    mapridge = pm.find_MAP()
{'beta1': array(0.004526796692482796),
 'beta2': array(0.005064112237104185),
 'beta3': array(0.10005872308519308)}
```



Laplace vs Gaussian Prior



Model 3: Lasso

```
b = 1.0 / np.sqrt(2.0 * sigma2)
with pm.Model() as lasso:
    beta1 = pm.Laplace('beta1', mu=0, b=b)
    beta2 = pm.Laplace('beta2', mu=0, b=b)
    beta3 = pm.Laplace('beta3', mu=0, b=b)
    mu = beta1*x1 + beta2*x2 + beta3*x3
    ys = pm.Normal('ys', mu=mu, tau=1.0, observed=y)
    stepper=pm.Metropolis()
    tracelasso = pm.sample(100000, step=stepper)
```

```
beta3:

  Mean             SD               MC Error         95% HPD interval
  -------------------------------------------------------------------

  0.099            0.032            0.000            [0.040, 0.164]

  Posterior quantiles:
  2.5            25             50             75             97.5
  |--------------|==============|==============|--------------|

  0.037          0.078          0.099          0.120          0.162
```

```
with lasso:
    maplasso = pm.find_MAP()
{'beta1': array(-7.255541060919206e-05),
 'beta2': array(8.485263161675386e-05),
 'beta3': array(0.10015818579834601)}
```