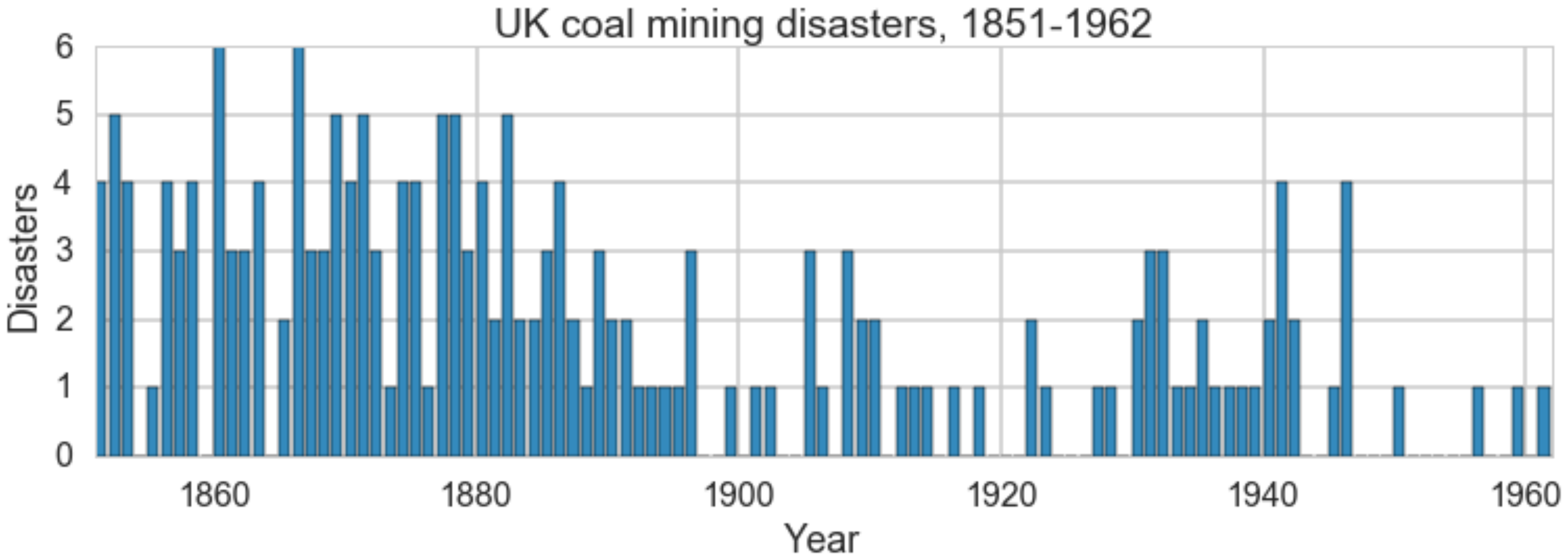
Lecture 18

Formal Tests, NUTS

Sampling with pymc3

Diagnostics

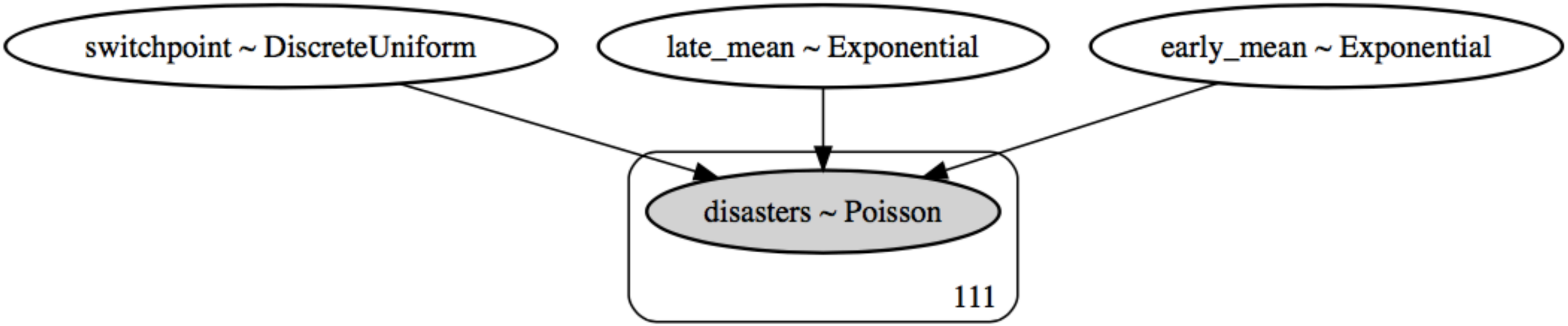

UK coal mining disasters, 1851-1962
Disasters
0
1
2
3
4
5
6
1860
1880
1900
1920
1940
1960
Year

Model

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

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

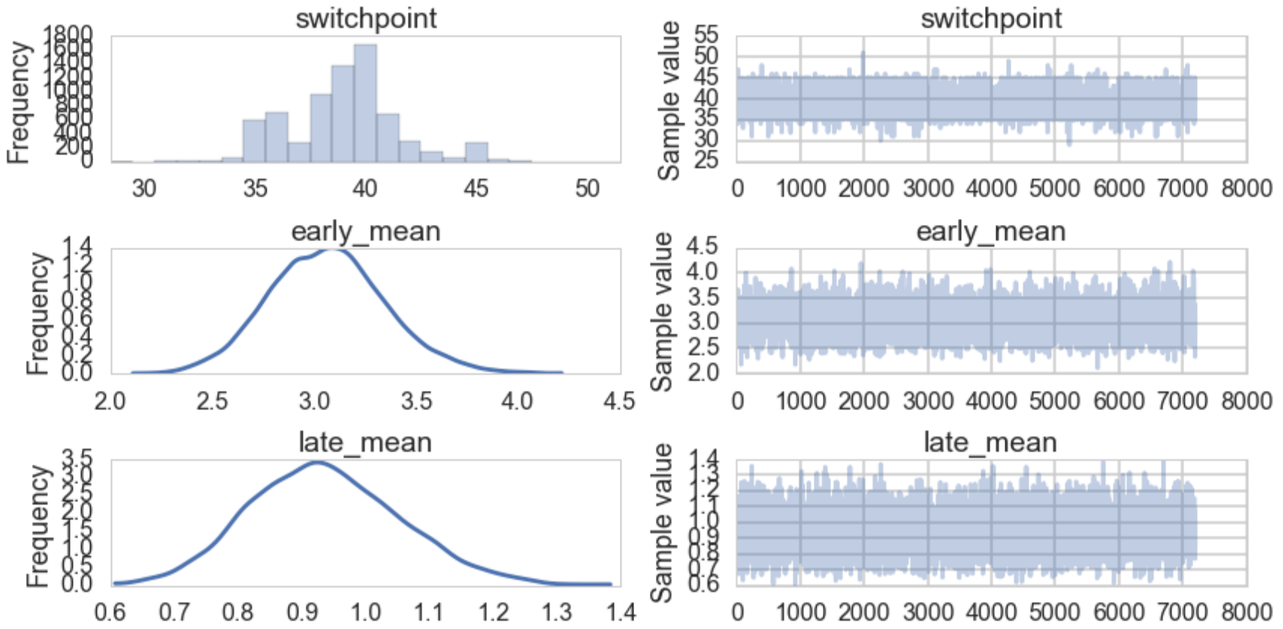
$$\lambda_1 \sim Exp(a)$$
$$\lambda_2 \sim Exp(b)$$


switchpoint ~ DiscreteUniform
late_mean ~ Exponential
early_mean ~ Exponential
disasters ~ Poisson
111

```
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)
```

```
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]
```

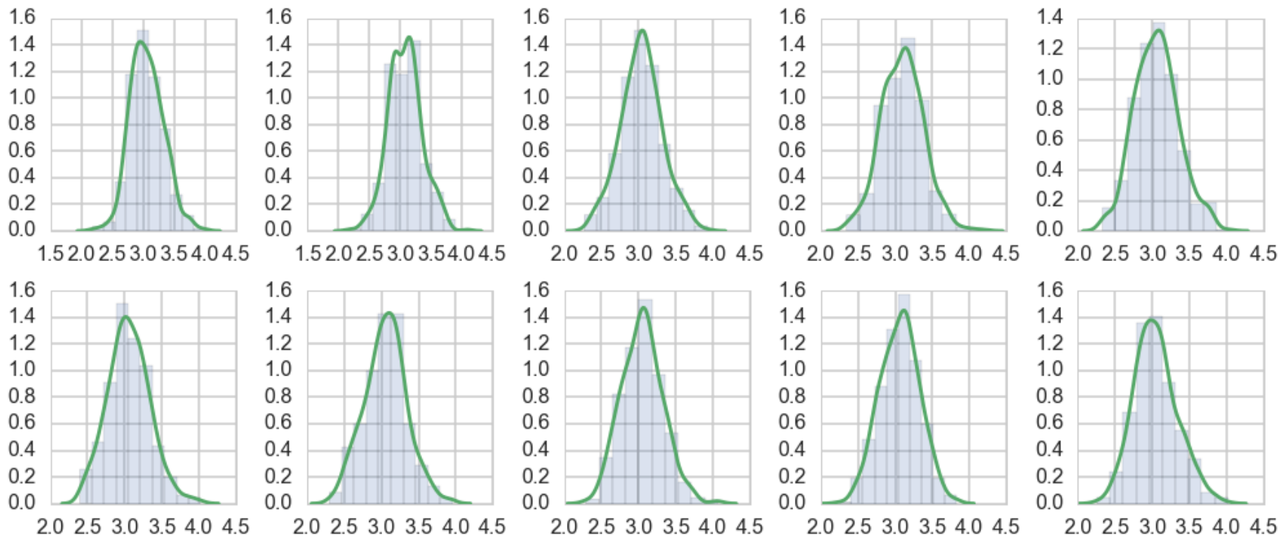


Model convergence

- 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



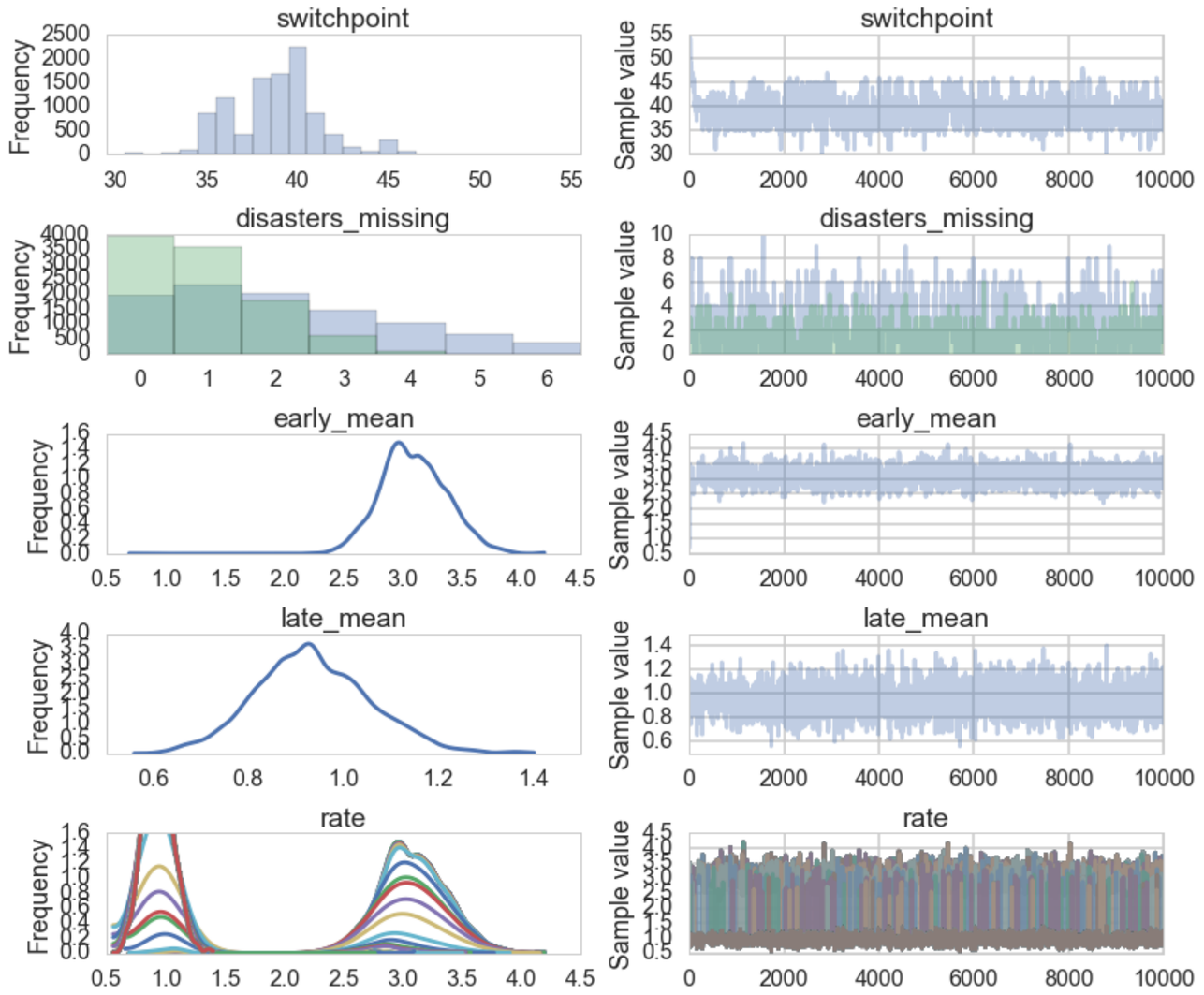
Imputation

```
>>>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.

```python
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
```

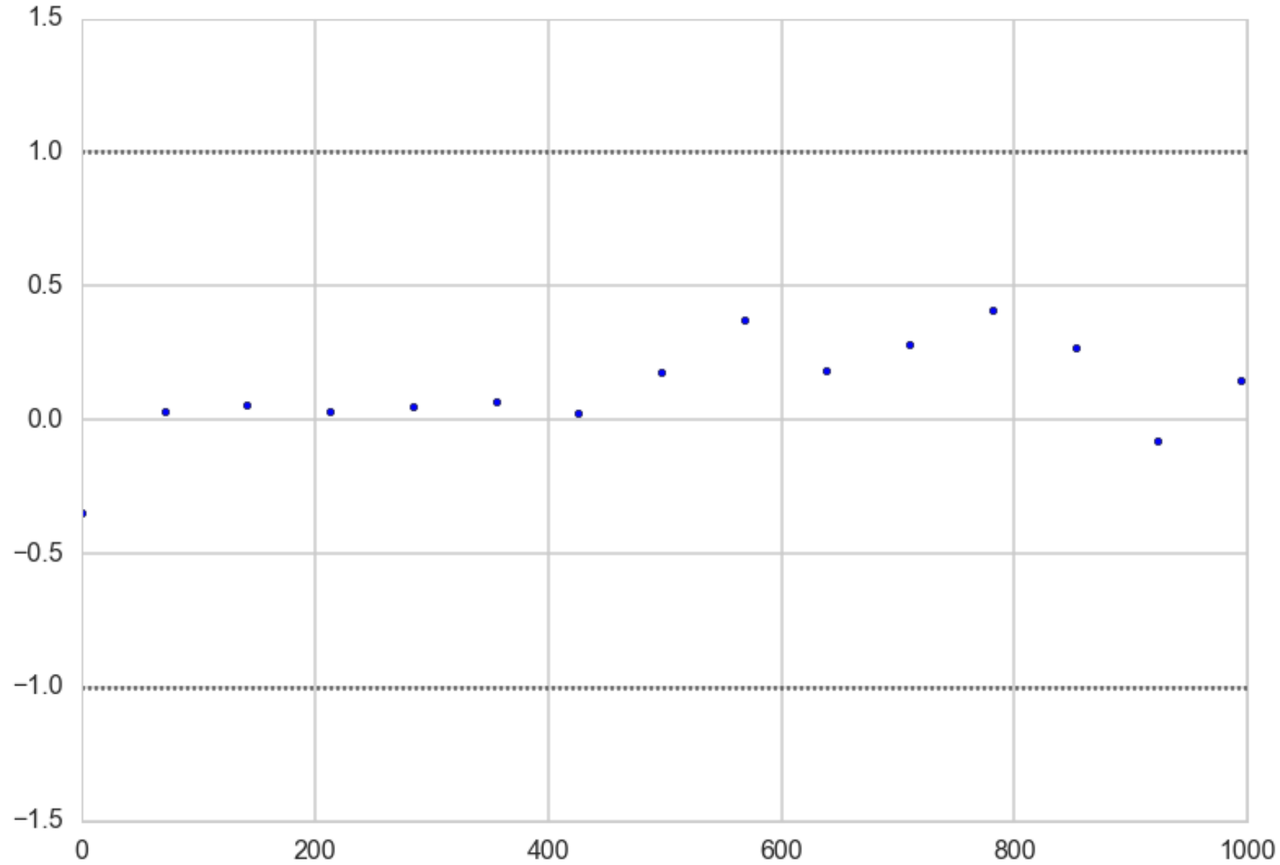


Gewecke: difference of means

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

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

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




```
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)
```

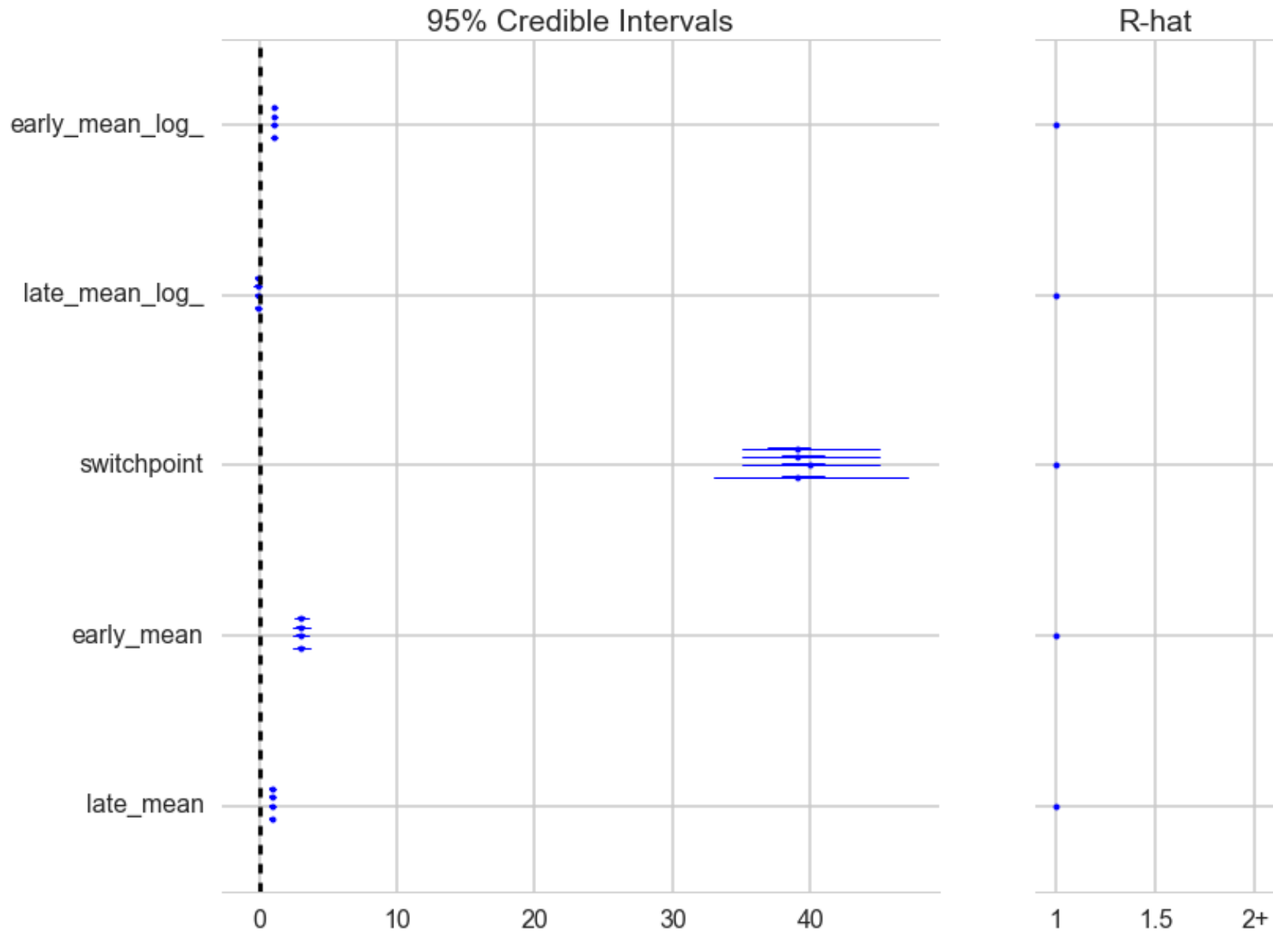
Gelman-Rubin

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)`:

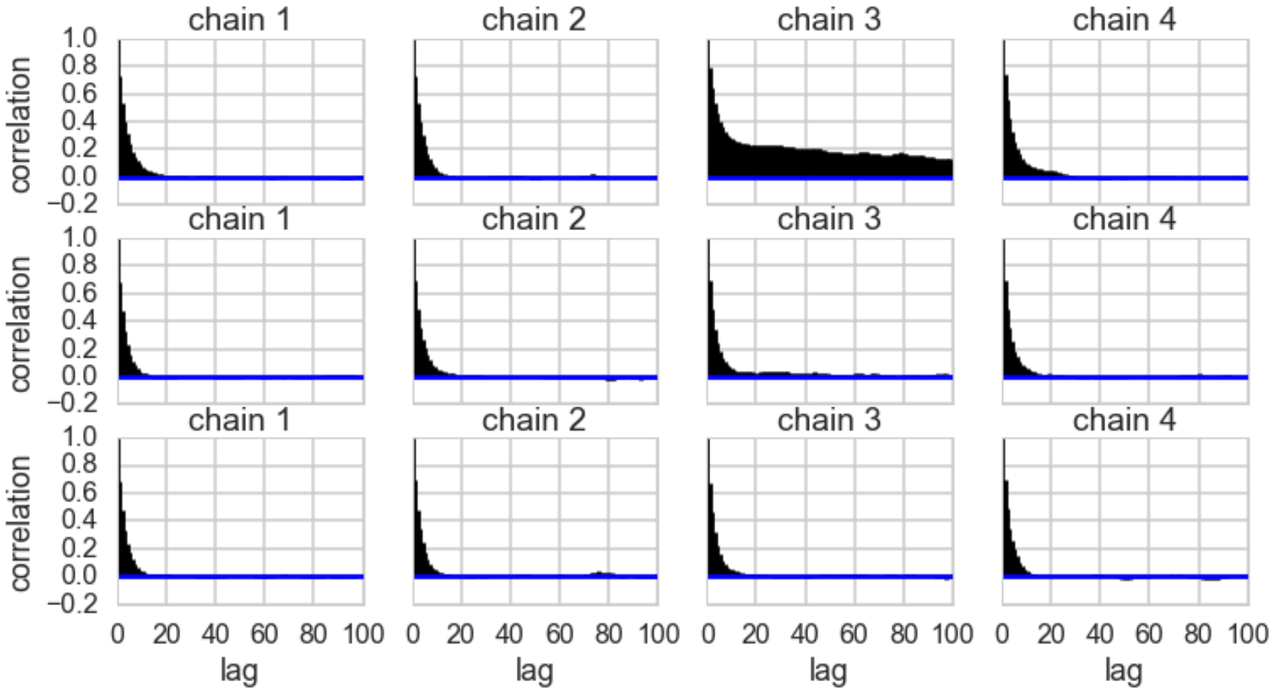
$$\hat{Var}(\theta) = (1 - \frac{1}{n})w + \frac{1}{n}B$$

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: a measure of correlation



IIDness of draws decreases

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

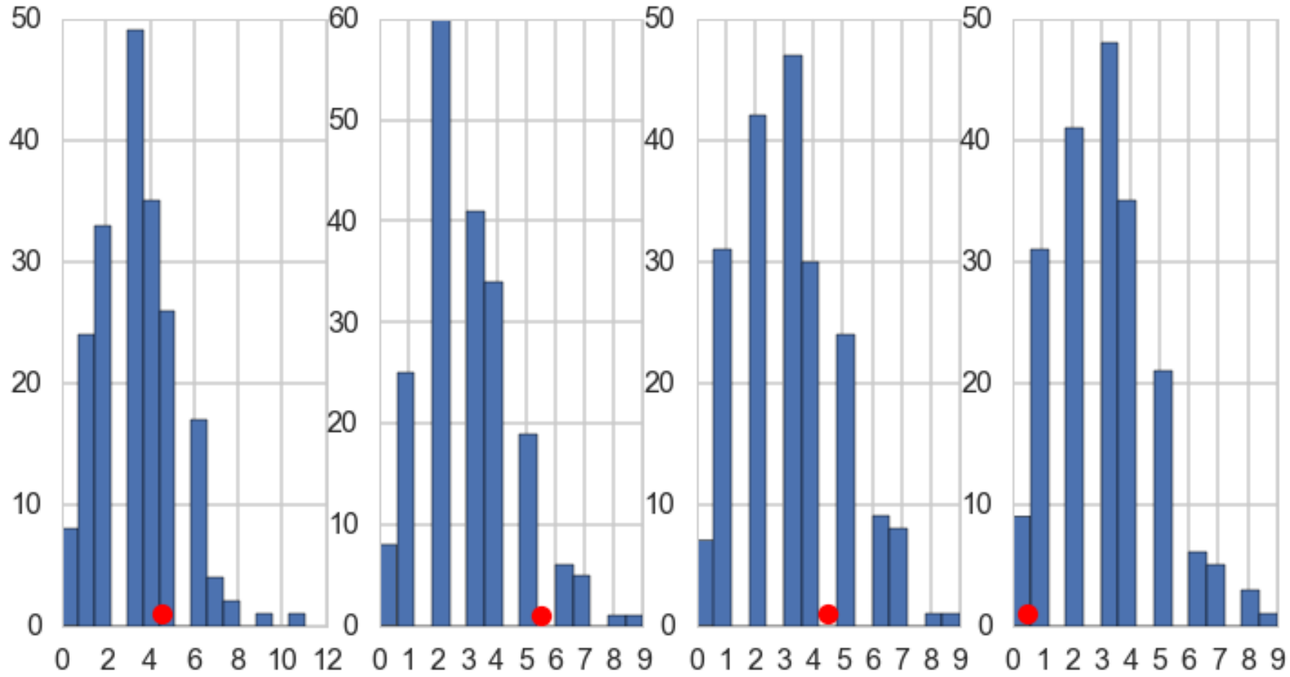
```
{'early_mean': 16857.0,
 'early_mean_log_': 12004.0,
 'late_mean': 27344.0,
 'late_mean_log_': 27195.0,
 'switchpoint': 195.0}
```

(40000 samples)

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

Posterior Predictive Checks

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



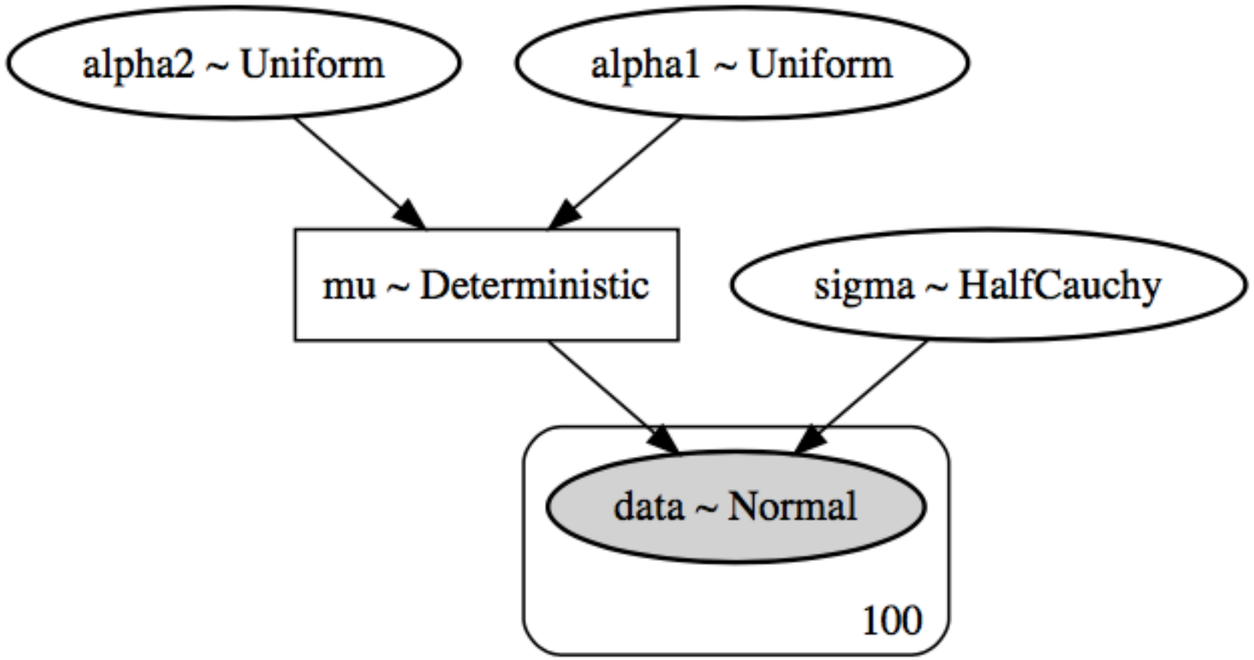
Non-Identifiability and Correlation

Simple Example: generate data from N(0,1).

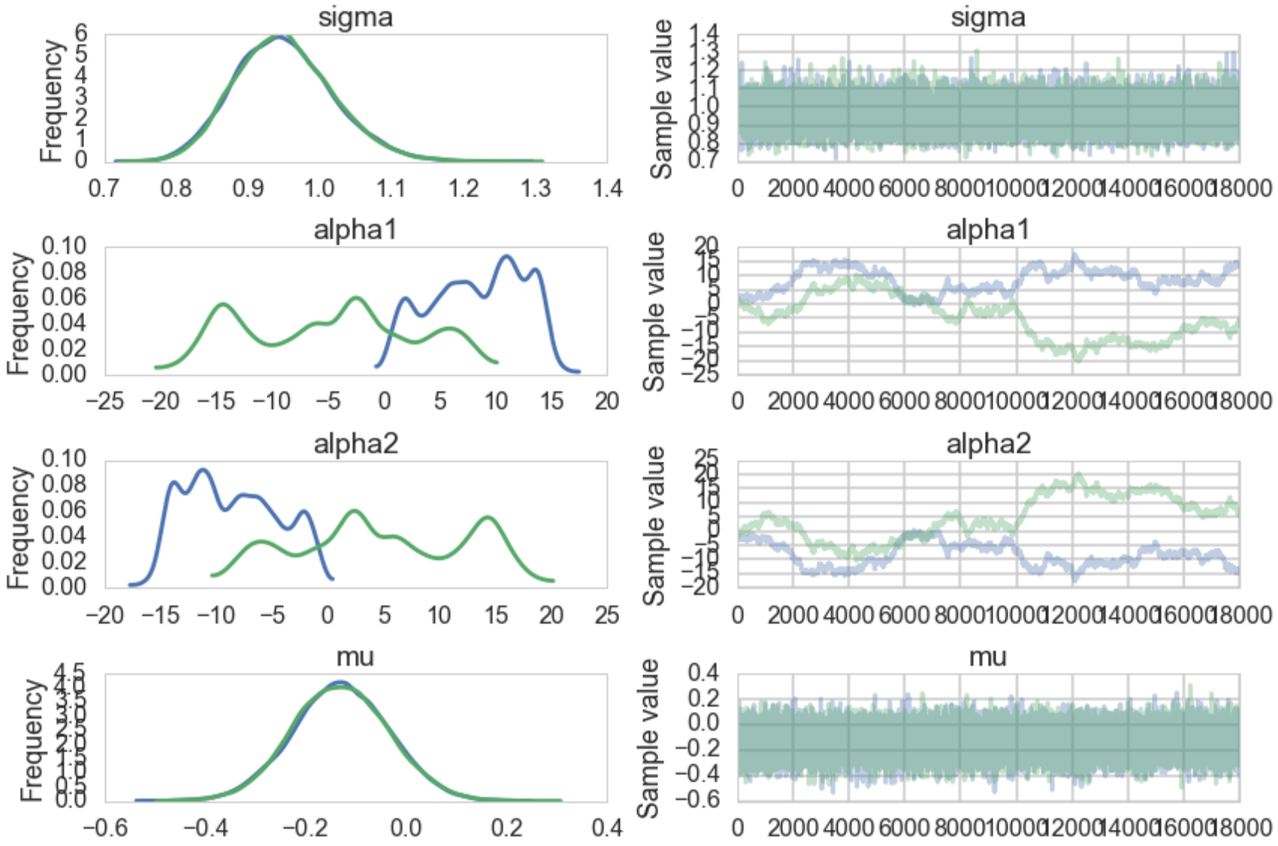
Then fit: $y \sim N(\mu, \sigma)$

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

$$\begin{aligned} \alpha_1 &\sim Unif(-\infty, \infty) \\ \alpha_2 &\sim Unif(-\infty, \infty) \\ \sigma &\sim HalfCauchy(0, 1) \end{aligned}$$



Correlation diagnostic

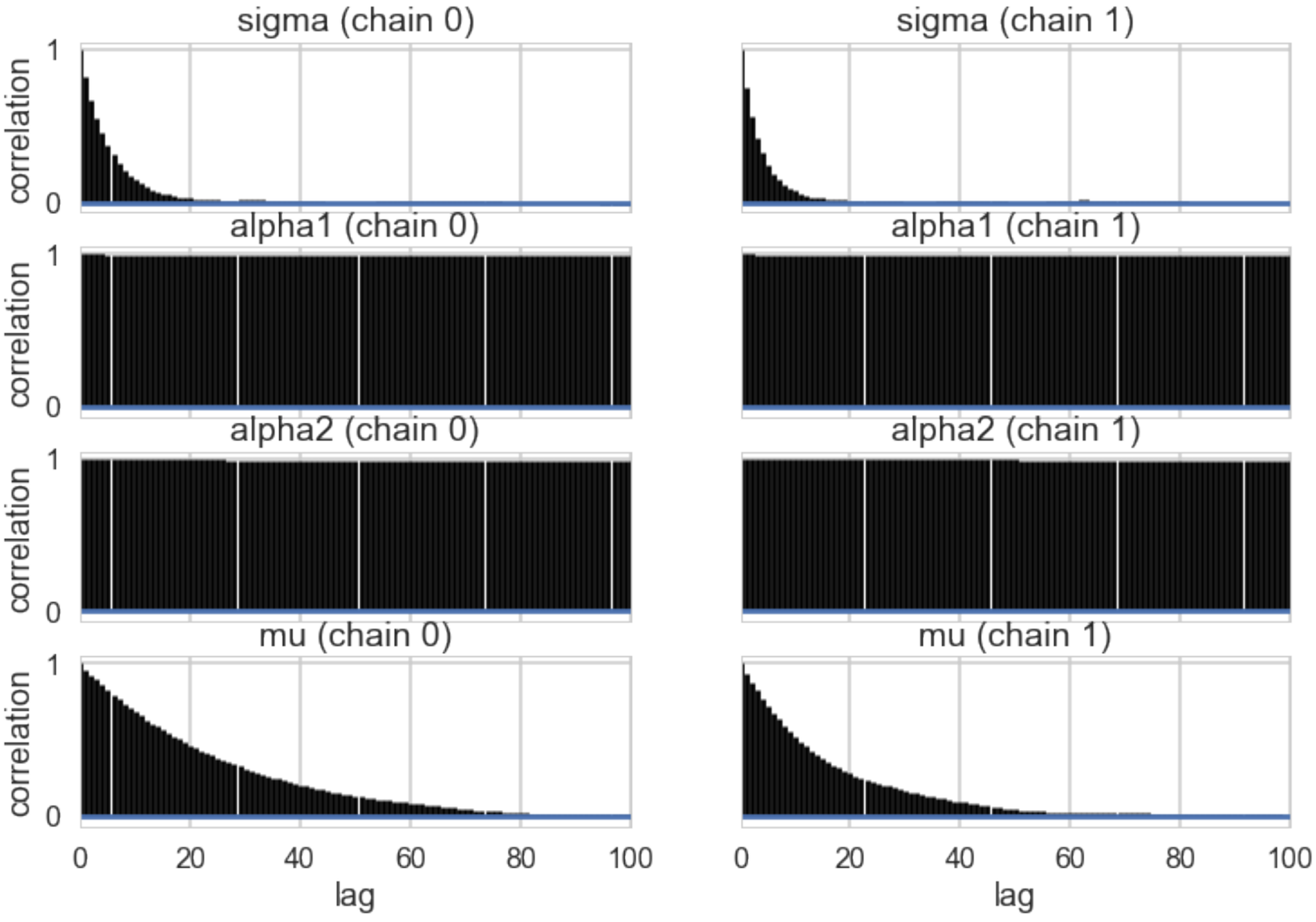



```
sigma = pm.HalfCauchy("sigma", beta=1)
alpha1=pm.Uniform('alpha1', lower=-10**6, upper=10**6)
alpha2=pm.Uniform('alpha2', lower=-10**6, upper=10**6)
mu = pm.Deterministic("mu", alpha1 + alpha2)
y = pm.Normal("data", mu=mu, sd=sigma, observed=data)
```

```
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': 1.7438626593529831,
  'alpha2_interval_': 1.7438626593368223,
  'mu': 0.9999971018206269 5,
  'sigma': 1.0000248056117549,
  'sigma_log_': 1.0000261752214563}
```


sigma (chain 0)
sigma (chain 1)
alpha1 (chain 0)
alpha1 (chain 1)
alpha2 (chain 0)
alpha2 (chain 1)
mu (chain 0)
mu (chain 1)
correlation
1
0
0
20
40
60
80
100
lag

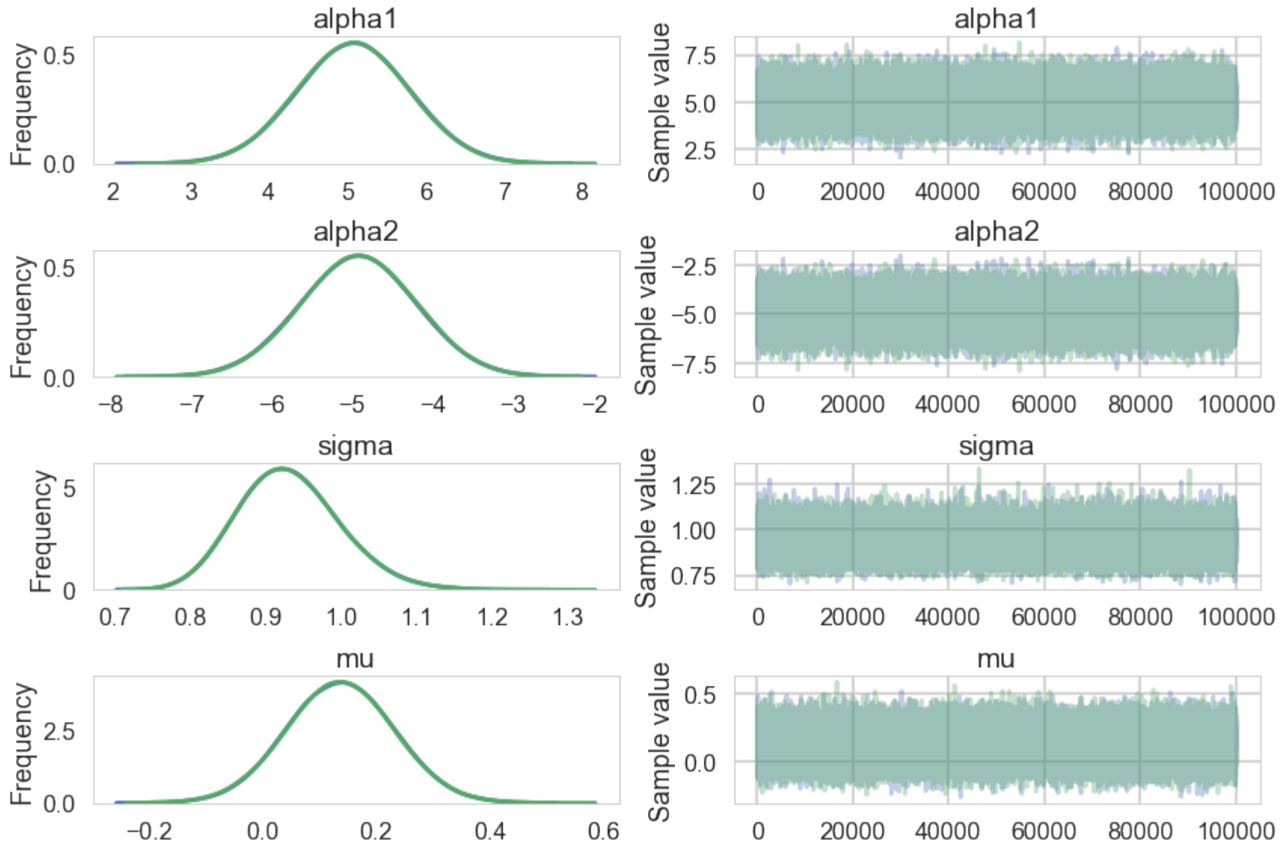
Is autocorrelation bad?

- depends on what you want to do
- this is true for n_{eff} in general
- does not matter much for means
- matters for credible intervals as we need tails

trying to fix

	alpha1	alpha2	sigma	mu
alpha1	1.000000	-0.991369	-0.003822	0.066316
alpha2	-0.991369	1.000000	0.003936	0.065067
sigma	-0.003822	0.003936	1.000000	0.000868
mu	0.066316	0.065067	0.000868	1.000000

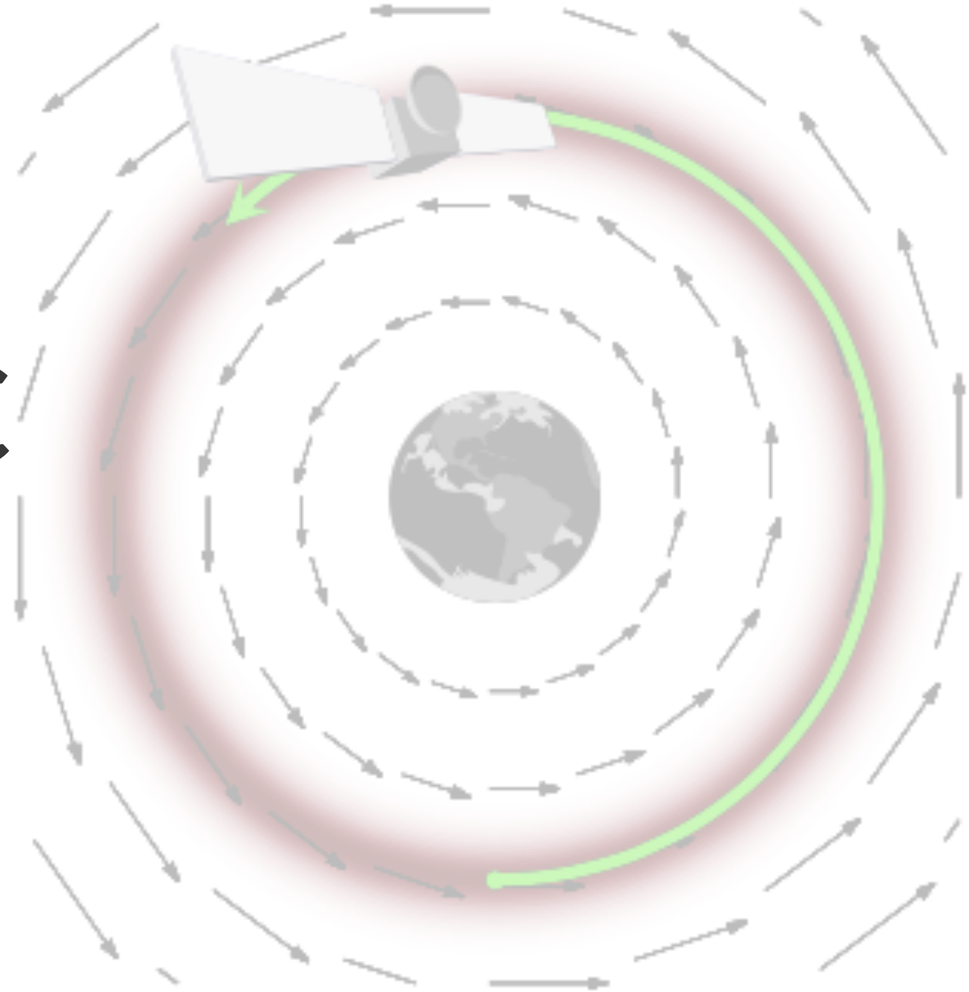
```
with pm.Model() as ni2:
    sigma = pm.HalfCauchy("sigma", beta=1)
    alpha1=pm.Normal('alpha1', mu=5, sd=1)
    alpha2=pm.Normal('alpha2', mu=-5, sd=1)
    mu = pm.Deterministic("mu", alpha1 + alpha2)
    y = pm.Normal("data", mu=mu, sd=sigma, observed=data)
    #stepper=pm.Metropolis()
    #traceni2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

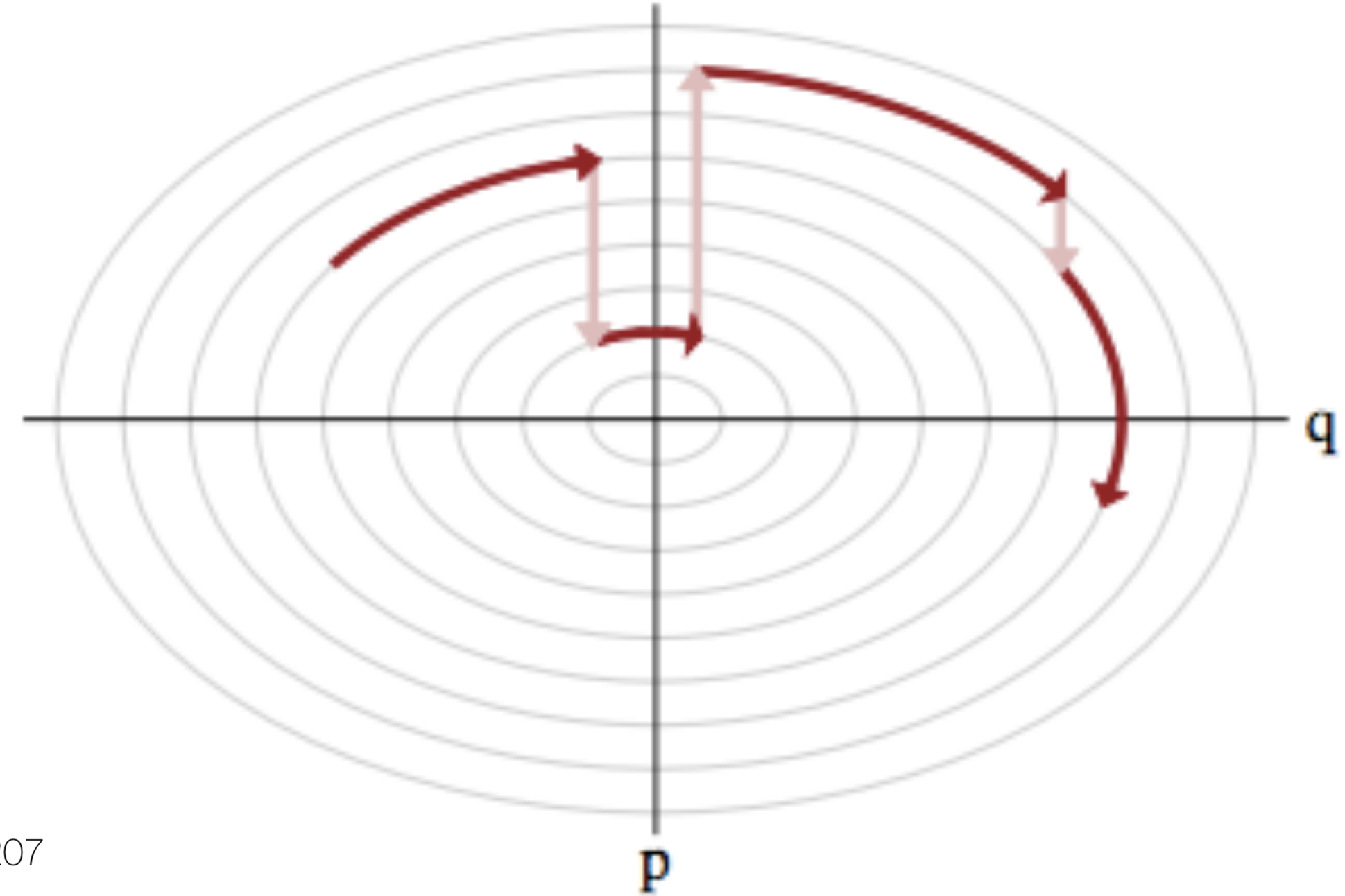


Thoughts on Diagnostics

- be paranoid, you only know you have not converged, not if you have
- what if you missed out an entire lobe? Thus multiple chains and multiple starting points.
- check posterior correlations, trace autocorrelation, effective n, the look of the trace, the acceptance rate
- check gewecke and gelman-rubin

HMC




q
p

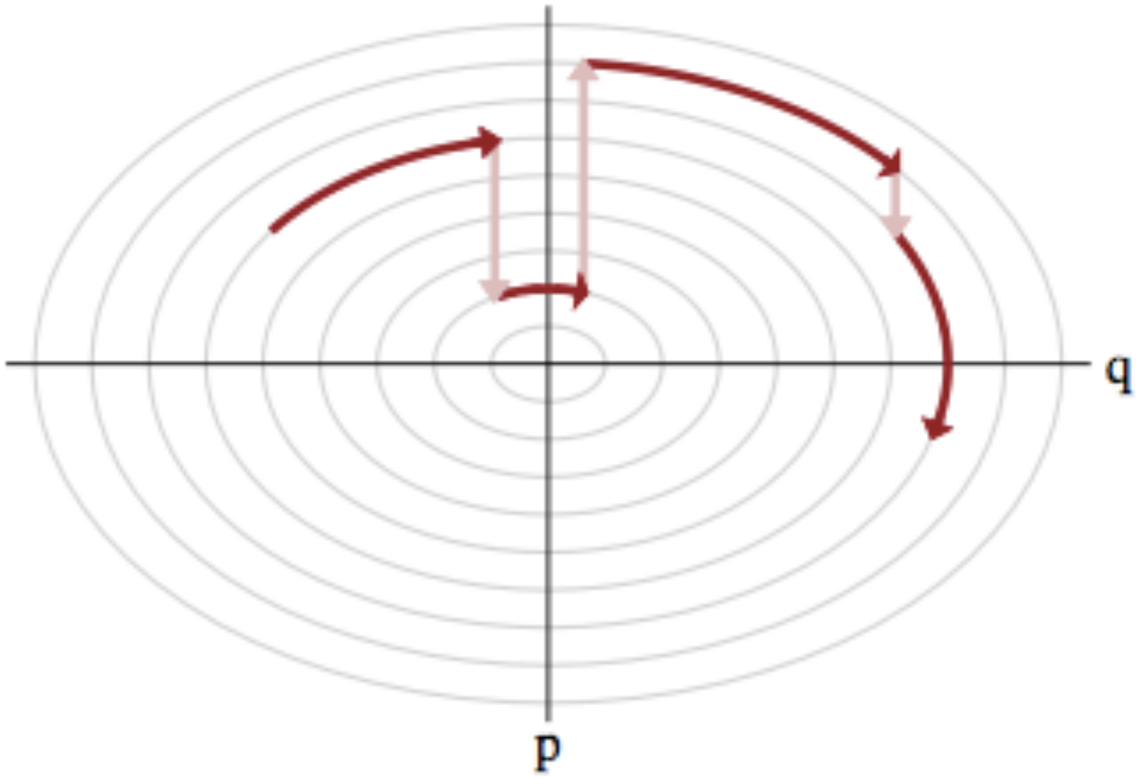
Recap of Hamiltonian Flow ideas

- start with $p(q)$
- augment using momentum to $p(p, q)$
- the momentum comes from a kinetic enegy which looks something like $p^2/2m$
- write $p(q)$ as $e^{-V(q)}$
- then $p(p, q) = e^{-H(p,q)} = e^{-K(p,q)}e^{-V(q)} = p(p|q)p(q)$

Basic Idealized Idea

1. Move on a level set of the Hamiltonian $H(p, q)$. Pick up samples at will with acceptance probability 1. Why? Reversibility, Flow.
2. Fire thrusters, that is sample p, from kinetic energy distribution, to move to another level set. Why? Cover whole Typical set
3. Repeat

Momentum resampling



Draw p from a distribution that is determined by the distribution of momentum, i.e. $p \sim N(0, \sqrt{M})$ for example, and attempt to explore the level sets.

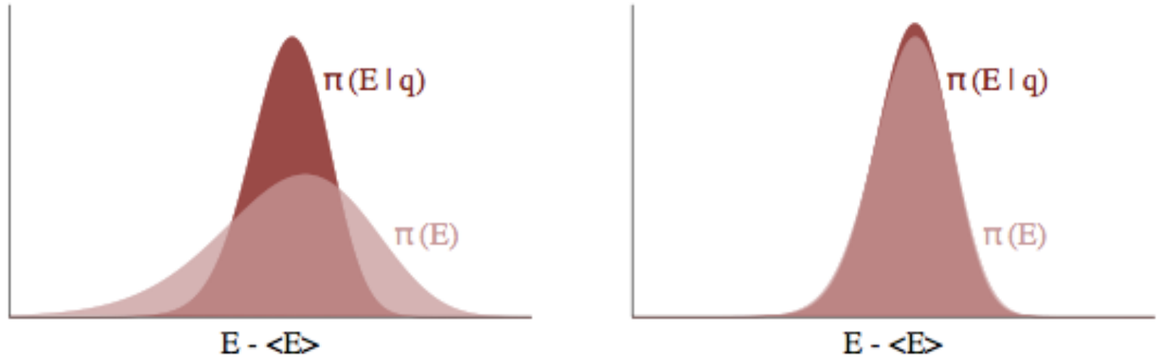
Firing the thruster moves us between level sets!

Resampling Efficiency

Let $p(E|q)$ as the transition distribution of energies induced by a momentum resampling using $p(p|q) = -log\, K(p, q)$ at a given position q.

If $p(E|q)$ narrow compared to the marginal energy distribution $p(E)$: random walk amongst level sets proceeds slowly.

If $p(E|q)$ matches $p(E)$: independent samples generated from the marginal energy distribution very efficiently.



Tuning: choice of Kinetic energy

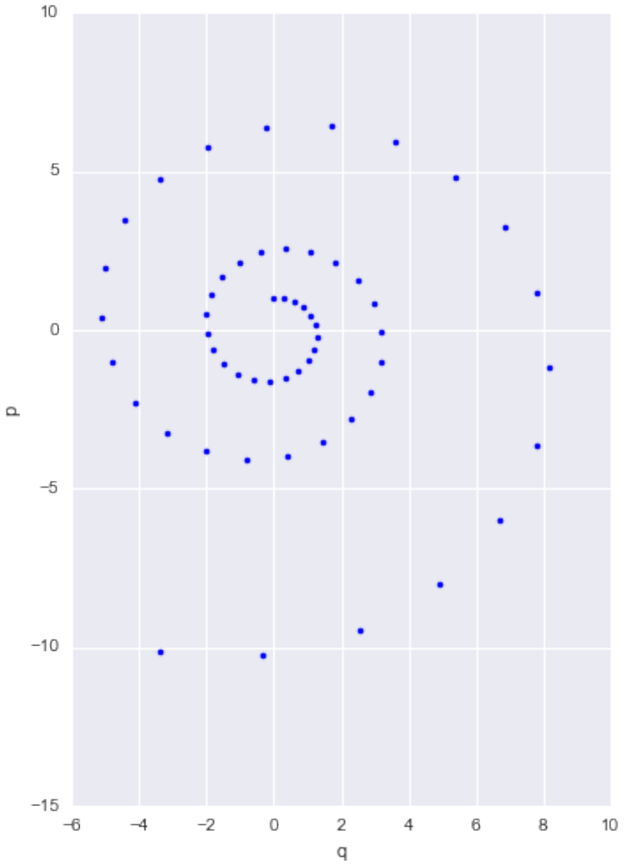
- Set M^{-1} to the covariance of the target distribution: maximally de-correlate the target. Do in warmup (tune) phase.
- can see this by $p \to p/\sqrt{M}$, Then $q \to q\sqrt{M}$

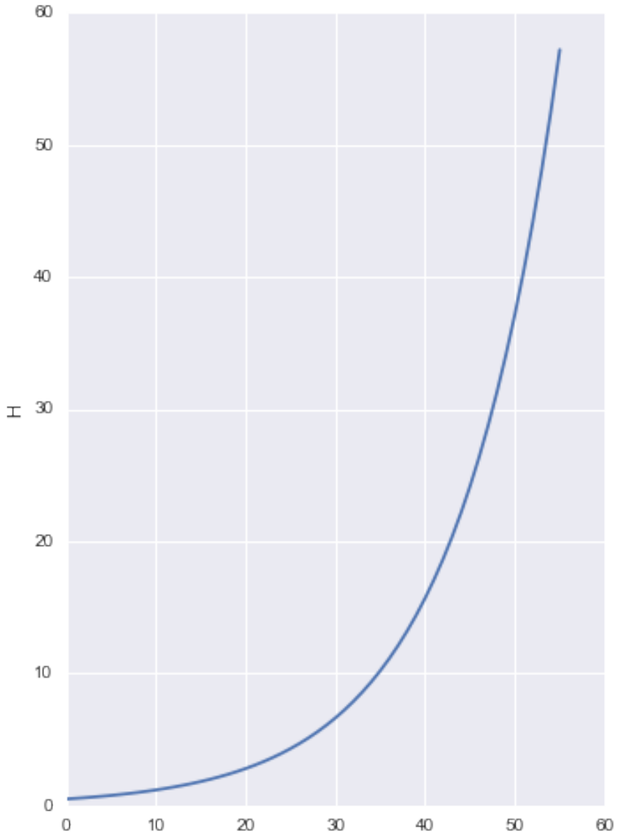
$$H = \frac{1}{2}p^T M^{-1} p + \frac{1}{2}q^T \Sigma^{-1} q \text{ becomes } H = \frac{1}{2}\left(p'^T p + q'^T q\right)$$

if $M^{-1} = \Sigma$

Thus de-correlate target. Generalize to arbitrary distributions.

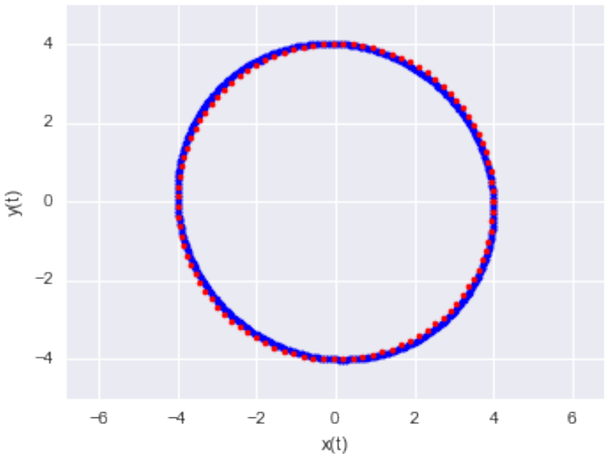
Discretization problems

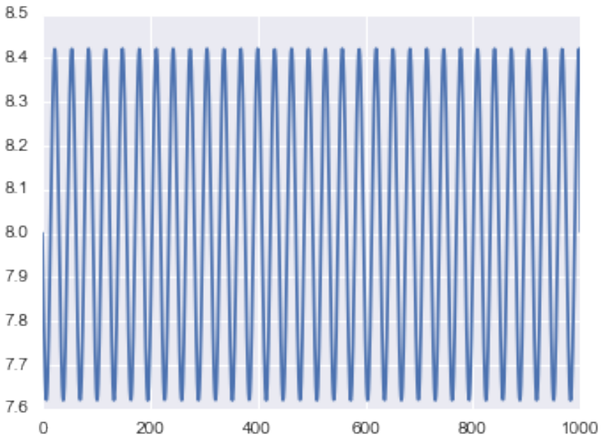




- $p_i(t+\epsilon) = p_i(t) - \epsilon \frac{\partial U}{\partial q_i}|_{q(t)}$
- $q_i(t+\epsilon) = q_i(t) + \epsilon \frac{p_i(t)}{m_i}$
- off-diagonal terms of size ϵ makes volume not preserved
- leads to drift over time
- use "leapfrog" instead

Sympletic Leapfrog (why volume needs conservation 1)





- Only *shear* transforms allowed, will preserve volume.
- $p_i(t+\frac{\epsilon}{2}) = p_i(t) - \frac{\epsilon}{2}\frac{\partial V}{\partial q_i}|_{q(t)}$
- $q_i(t+\epsilon) = q_i(t) + \epsilon\frac{p_i(t+\frac{\epsilon}{2})}{m_i}$
- $p_i(t+\epsilon) = p_i(t+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{\partial V}{\partial q_i}|_{q(t+\epsilon)}$
- still error exists, oscillatory, so reversibility not achieved
- use superman transform. Works even when we are off level set.

WE ARE MARGINALLY OFF THE LEVEL SETS!

So must consider: acceptance probability

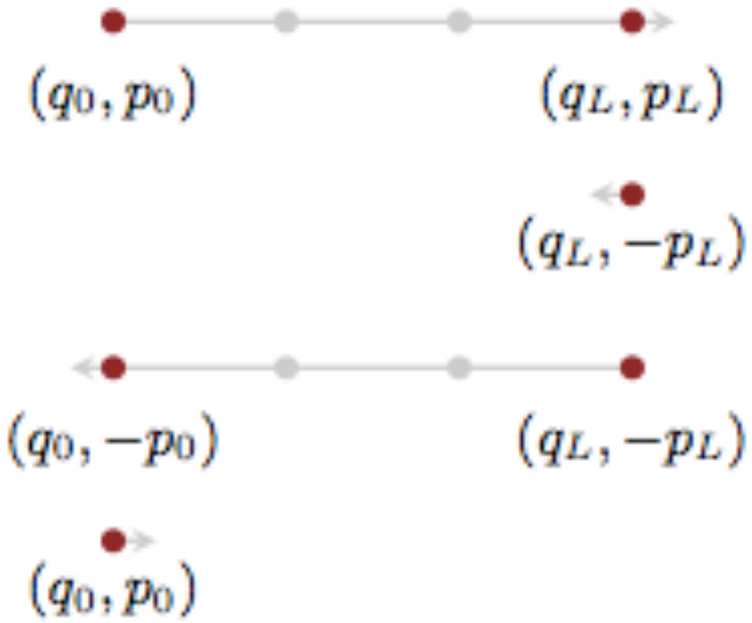
$$A = \min[1, \frac{p(q',p')Q(q',p'|q,p)}{p(q,p)Q(q,p)|q',p')}]$$

What should we choose as our proposal?

Superman choice

- tack on sign change $(q, p) \rightarrow (q_L, -p_L)$. Superman to the rescue!
- proposal now: $Q(q', p'|q, p) = \delta(q' - q_L)\delta(p' + p_L)$.
- Acceptance:

$$A = \min[1, \frac{p(q_L, -p_L)\delta(q_L - q_L)\delta(-p_L + p_L)}{p(q, p)\delta(q - q)\delta(p - p)}]$$



$$A = \min[1, \exp(-U(q_L) + U(q) - K(p_L) + K(p)]$$

$$= \min[1, \exp(-H_L + H)]$$

critical thing with HMC is that our **time evolution is always close to being on a level set** if we have no problems with our sympelctic integrator. So our A always closer to 1, and we have a very efficient sampler.

Second reason for Volume Conservation

From Neal's paper:

The significance of volume preservation for MCMC is that we need not account for any change in volume in the acceptance probability for Metropolis updates. If we proposed new states using some arbitrary, non-Hamiltonian, dynamics, we would need to compute the determinant of the Jacobian matrix for the mapping the dynamics defines, which might well be infeasible.

Detailed Balance

- obvious for $i \neq j$, but for $i = j$, call it k:
- in limit of regions becoming smaller, H can be thought of as constant inside the region, and thus the canonical densities and transition probs become constant too:

$$\frac{V}{Z} exp(-H_{A_k}) min[1, exp(-H_{B_k} + H_{A_k})] = \frac{V}{Z} exp(-H_{B_k}) min[1, exp(-H_{A_k} + H_{B_k})]$$

true

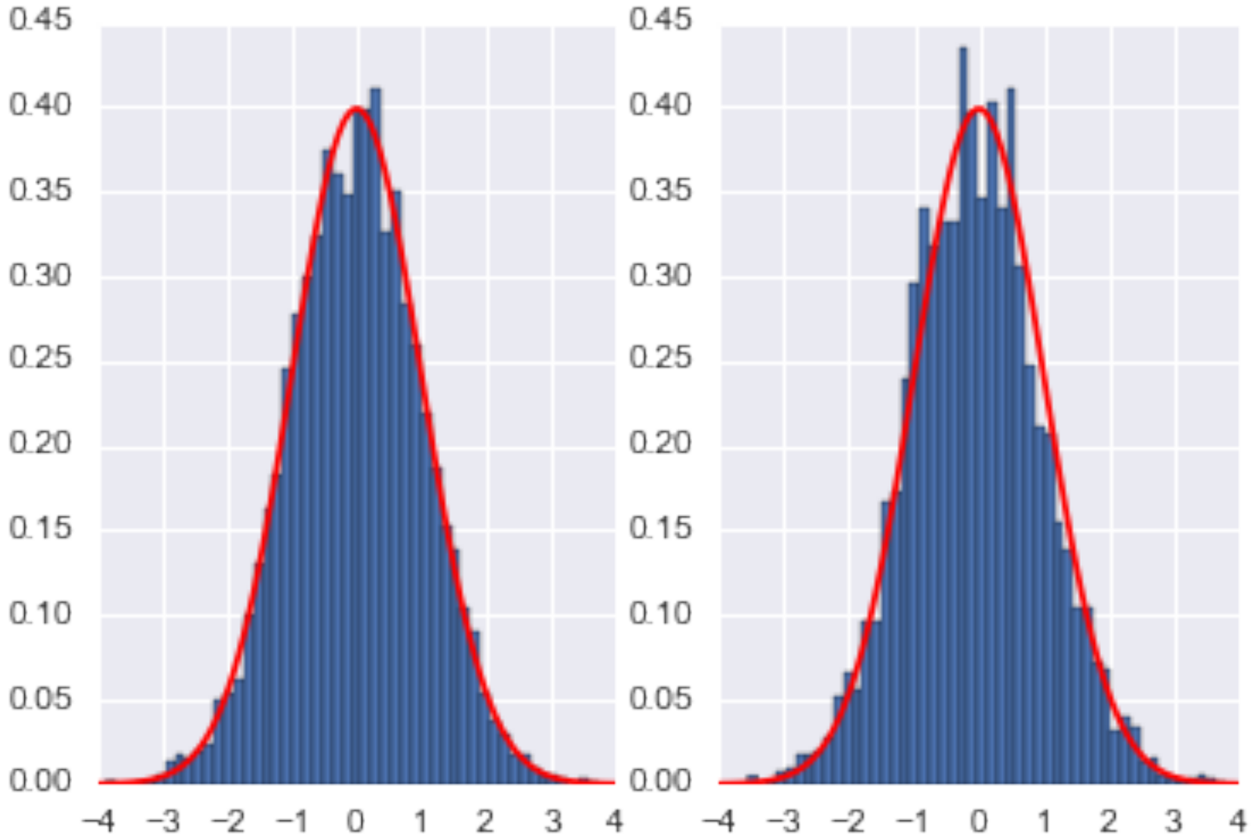
HMC Algorithm (momentum reversal could be left out if not within a more complex sampling scheme)

- for i=1:N_samples
  - 1. Draw $p \sim N(0, M)$
  - 2. Set $q_c = q^{(i)}$ where the subscript c stands for current
  - 3. $p_c = p$
  - 4. Update momentum before going into LeapFrog stage: $p^* = p_c - \frac{\epsilon * \nabla U(q_c)}{2}$
  - 5. LeapFrog to get new proposals. For j=1:L (first/third steps together)
    - $q^* = q^* + \epsilon p$
    - if not the last step, $p = p - \epsilon \nabla U(q)$
  - 6. Complete leapfrog: $p = p - \frac{\epsilon \nabla U(q)}{2}$

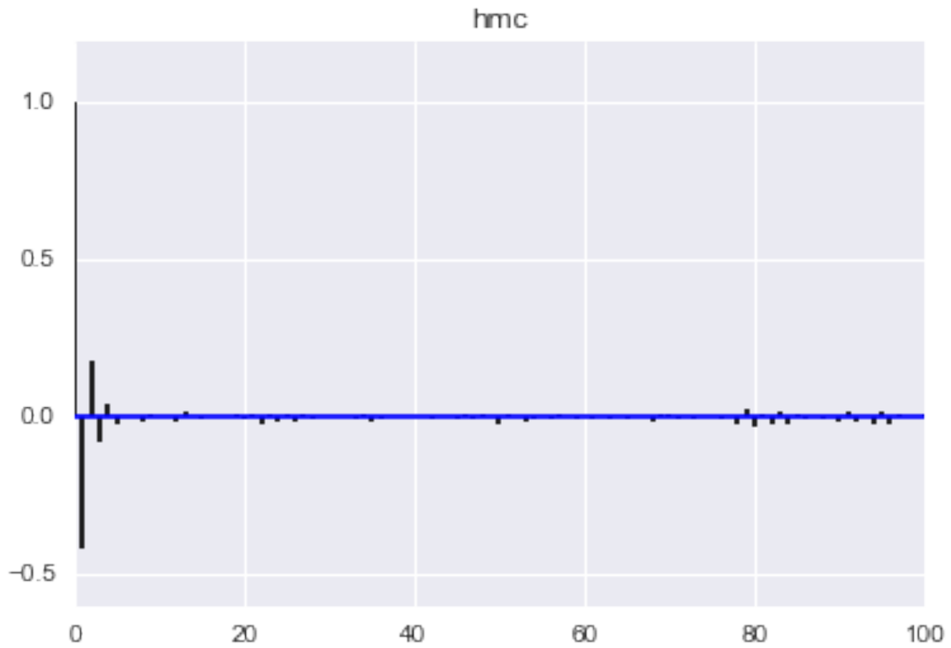
HMC (contd)

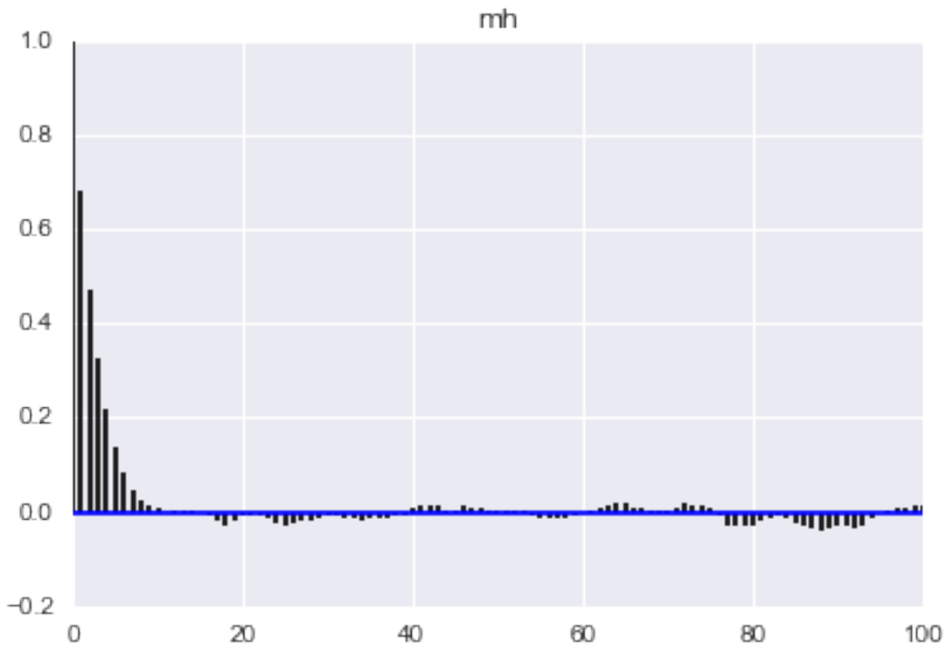
- for i=1:N_samples
  - 7. $p^* = -p$
  - 8. $V_c = V(q_c), \;\; K_c = \frac{p_c^\top M^{-1} p_c}{2}$
  - 9. $V^* = V(q^*), \;\; K^* = \frac{p^{\top *} M^{-1} p^*}{2}$
  - 10. $r \sim \mathbf{Unif}(0,1)$
  - 11. if $r < e^{(U_c - U^* + K_c - K^*)}$
    - accept $q_i = q^*$
    - otherwise reject

```python
def HMC(U,K,dUdq,N,q_0, p_0, epsilon=0.01, L=100):
    current_q = q_0
    current_p = p_0
    H = np.zeros(N)
    qall = np.zeros(N)
    accept=0
    for j in range(N):
        q = current_q
        p = current_p
        #draw a new p
        p = np.random.normal(0,1)
        current_p=p
        # leap frog
        # Make a half step for momentum at the beginning
        p = p - epsilon*dUdq(q)/2.0
        # alternate full steps for position and momentum
        for i in range(L):
            q = q + epsilon*p
            if (i != L-1):
                p = p - epsilon*dUdq(q)
        #make a half step at the end
        p = p - epsilon*dUdq(q)/2.
        # negate the momentum
        p= -p;
        current_U = U(current_q)
        current_K = K(current_p)
        proposed_U = U(q)
        proposed_K = K(p)
        A=np.exp( current_U-proposed_U+current_K-proposed_K)
        # accept/reject
        if np.random.rand() < A:
            current_q = q
            qall[j]=q
            accept+=1
        else:
            qall[j] = current_q
        H[j] = U(current_q)+K(current_p)
    print("accept=",accept/np.double(N))
    return H, qall
```



Autocorrelation: HMC vs MH





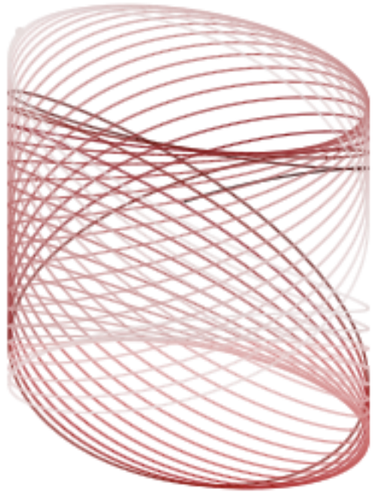

```
H, qall= HMC(U=U,K=K,dUdq=dUdq,N=10000,q_0=0, p_0=-4, epsilon=0.01, L=200)

samples_mh = MH_simple(p=P, n=10000, sig=4.0, x0=0)
```

Tuning: integration time

- whats the best integration time?
- should we glide for a long time? then we wont get too may samples
- if our integration was exact we could glide for arbitrary short times
- but integration is not exact and will infact take us off the level set
- thus too many samples/too short time will get us back to MH

Dynamic Ergodicity

Because orbits fill in first, orbital average become "spatial" expectations

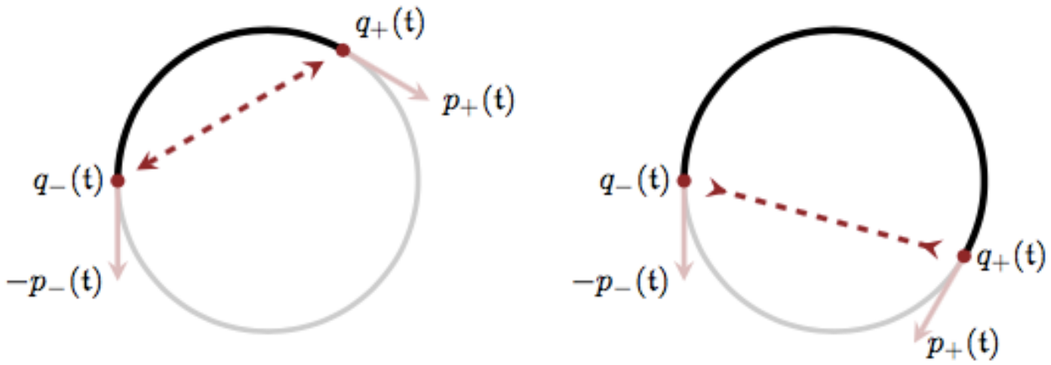


Tuning: L and ϵ

- find the point at which the orbital expectations converge to the spatial expectations..a sort of ergodicity
- L, number of iterations for which we run the Hamiltonian dynamics, and ϵ which is the (small) length of time each iteration is run.

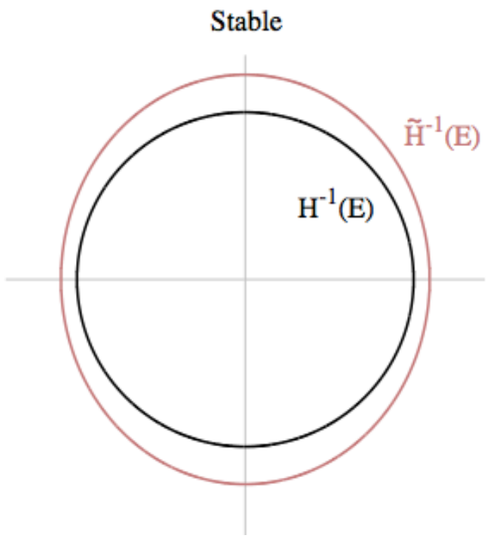
L tuning

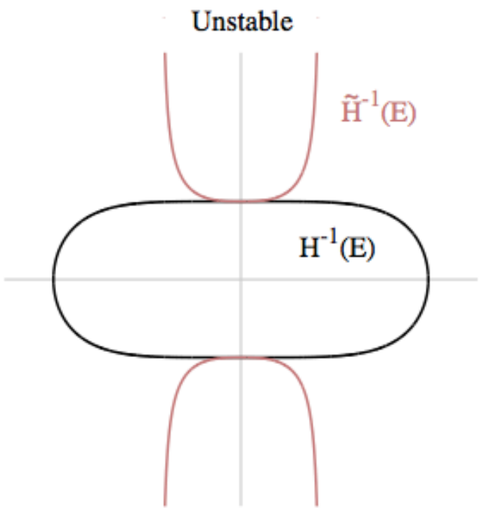
- in HMC, start $L = 100$ increase if for fixed step size, autocorrelation is too much
- Tails correspond to much higher energies, larger level-set surfaces are larger
- fixed length explores a small portion of this set before a momentum resampling takes us off.
- better to set dynamically: NUTS termination criterion

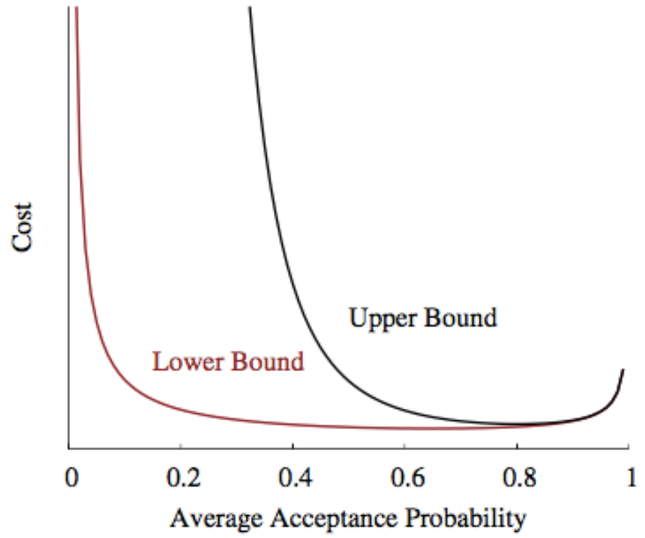


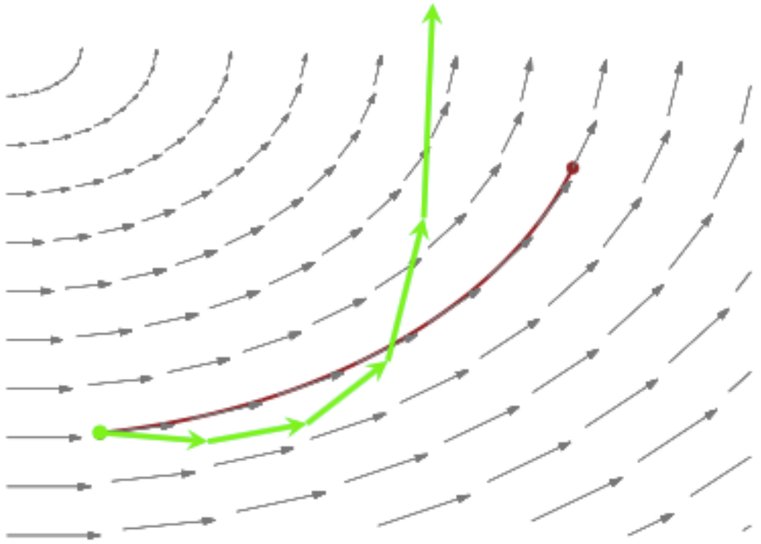
ϵ tuning

- if too small, accurate trajectories but too much time
- if too large, we will go off more and thus reject most of the time
- optimal ϵ is determined by the "shadow hamiltonian"
- want acceptance to be between 60 and 80 percent in most cases to have lower bounds of shadow and upper bounds of shadow close to each other


Stable
$\tilde{H}^{-1}(E)$
$H^{-1}(E)$

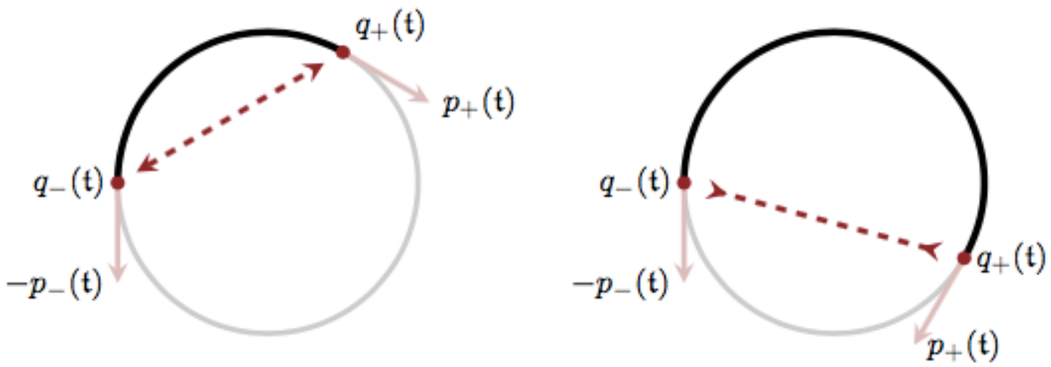

Unstable
$\tilde{H}^{-1}(E)$
$H^{-1}(E)$


Cost
Upper Bound
Lower Bound
0
0.2
0.4
0.6
0.8
1
Average Acceptance Probability

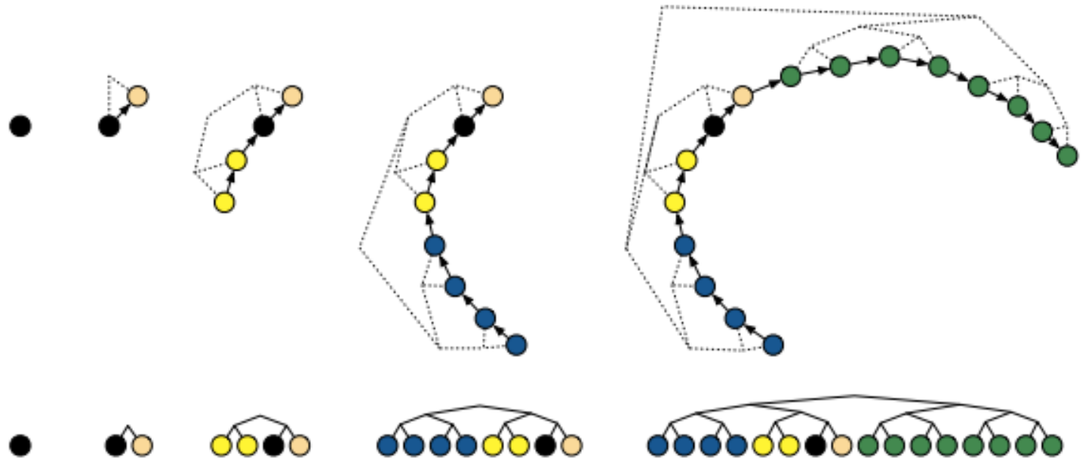


From HMC to HMC++

- one idea maybe to average over all points in orbit of length L
- To autotune L it is better to sample from orbit rather than get last point only: dynamic ergodicity: time average is orbit average
- NUTS: sample trajectories containing initial point and then sample point from them with trajectory canonical weights
- need a criterion for when to stop doing this



NUTS in a nutshell



- termination criterion destroys detailed balance, must rebuild
- sample from trajectory not just endpoint
- sample backwards and forwards in time until u-turn
- choose a sample with boltzmann weights over the trajectory using multinomial sampling

Hierarchicals with NUTS

Normal-Normal Hierarchical Model

J independent experiments, experiment j estimating the parameter θ_j from n_j independent normally distributed data points, y_{ij}, each with known error variance σ^2; that is,

$$y_{ij}|\theta_j \sim N(\theta_j, \sigma^2),\ i = 1, \ldots, n_j; j = 1, \ldots, J.$$

Gelman 8-schools problem: estimated coaching effects $\bar{y}_j$ to improve SAT scores for school j, with sampling variances, σ_j^2.

Sample mean of each group j

$\bar{y}_j = \frac{1}{n_j}\sum_{i=1}^{n_j} y_{ij}$ with sampling variance

$\sigma_j^2 = \sigma^2/n_j.$

Likelihood for θ_j using suff-stats, $\bar{y}_j$:

$$\bar{y}_j|\theta_j \sim N(\theta_j, \sigma_j^2).$$

Notation flexible in allowing a separate variance σ_j^2 for the mean of each group j.

Appropriate when the variances differ for reasons other than number of data pts.

School	Estimated treatment effect, y_j	Standard error of effect estimate, σ_j
A	28	15
B	8	10
C	−3	16
D	7	11
E	−1	9
F	1	11
G	18	10
H	12	18

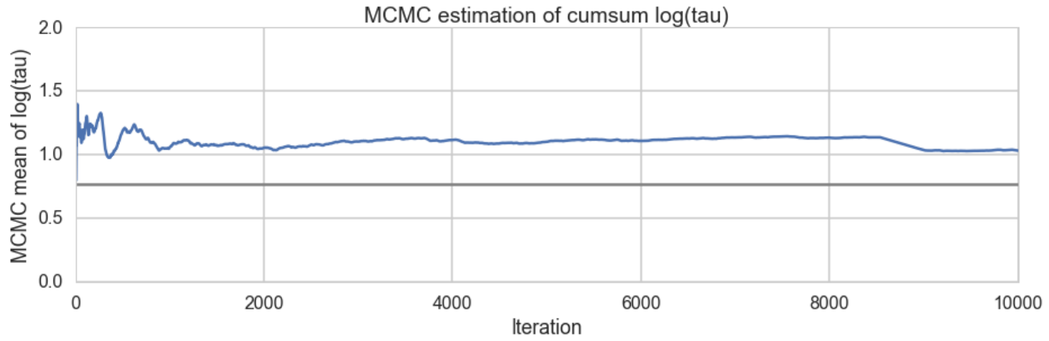
Centered Hierarchical Model

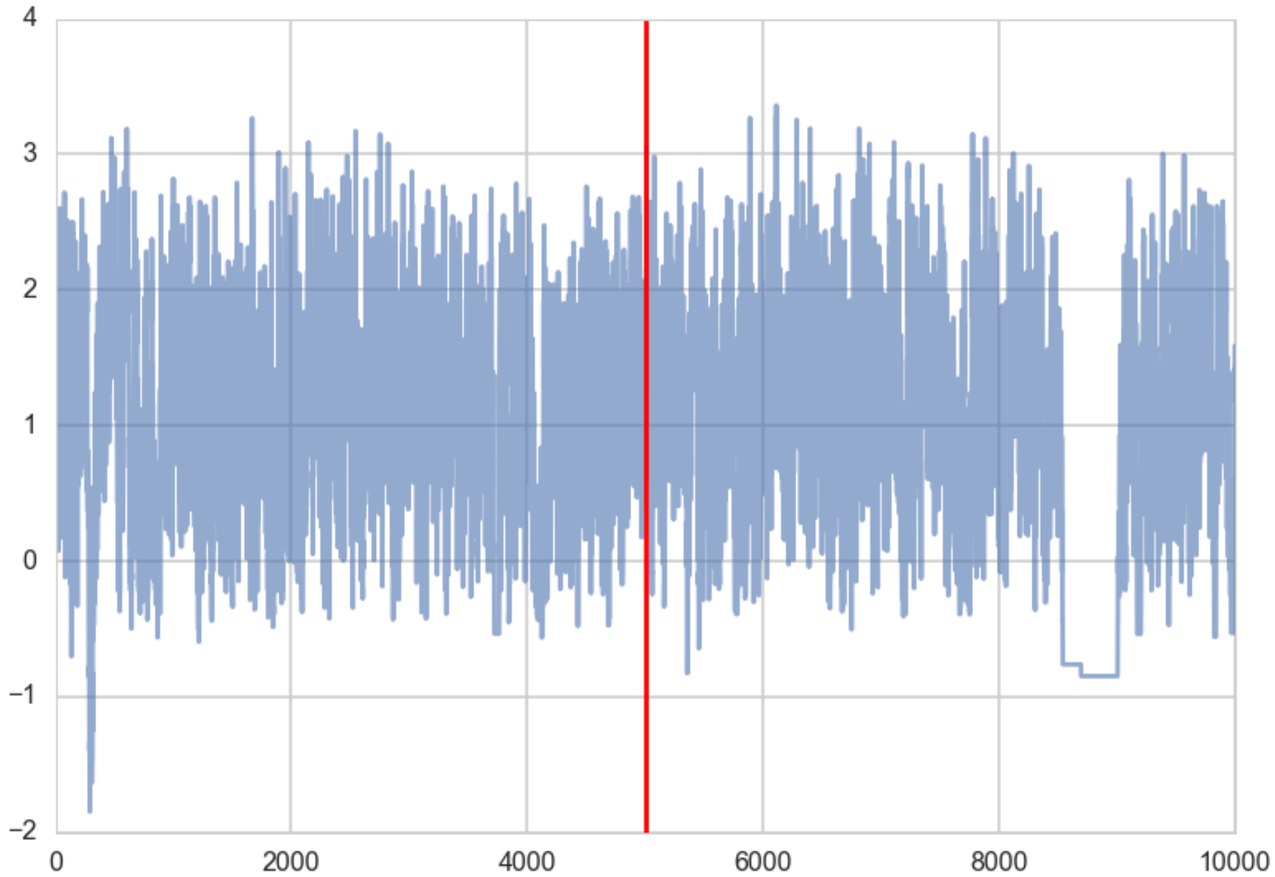
$$\mu \sim \mathcal{N}(0, 5)$$
$$\tau \sim \text{Half-Cauchy}(0, 5)$$
$$\theta_j \sim \mathcal{N}(\mu, \tau)$$
$$\bar{y}_j \sim \mathcal{N}(\theta_j, \sigma_j)$$

```
with pm.Model() as schools1:

    mu = pm.Normal('mu', 0, sd=5)
    tau = pm.HalfCauchy('tau', beta=5)
    theta = pm.Normal('theta', mu=mu, sd=tau, shape=J)
    obs = pm.Normal('obs', mu=theta, sd=sigma, observed=y)

with schools1:
    trace1 = pm.sample(5000, init=None, njobs=2, tune=500)
```

Small n_{eff}:

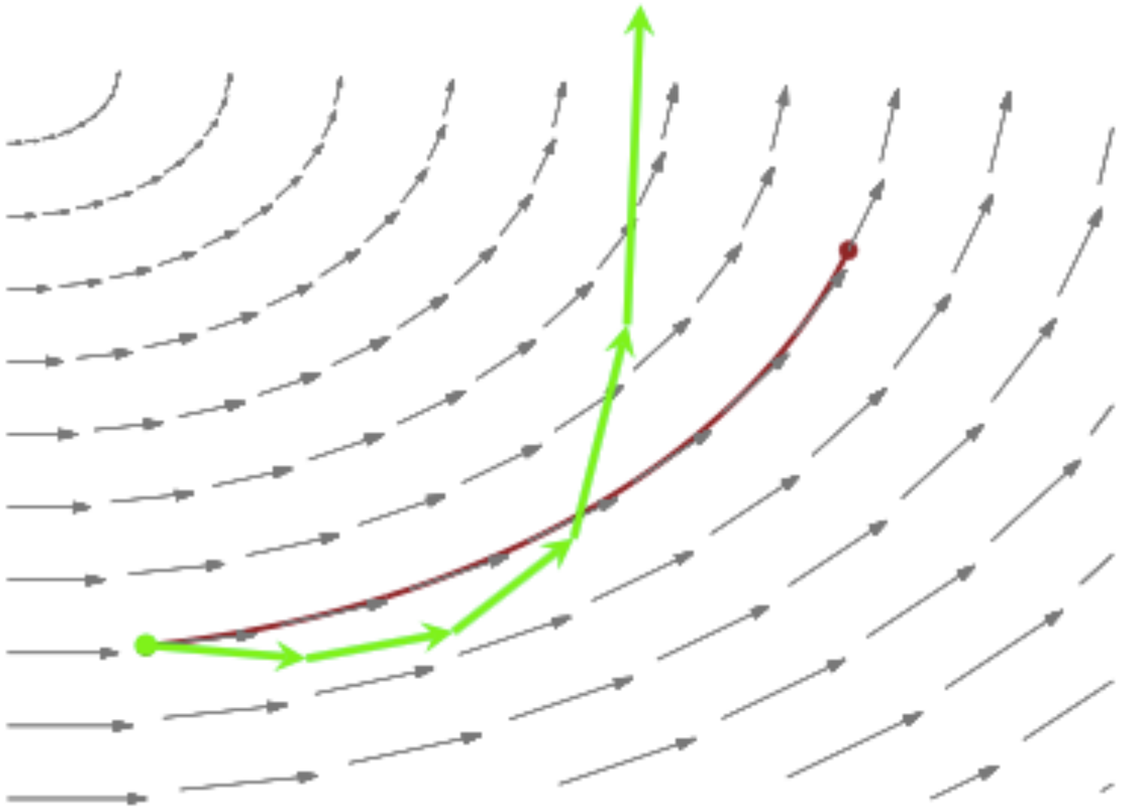
```
{'mu': 101.0,
  'tau': 273.0,
  'tau_log_': 77.0,
  'theta': array([ 169.,  199.,  236.,  193.,  211.,  231.,  139.,  204.])})
```





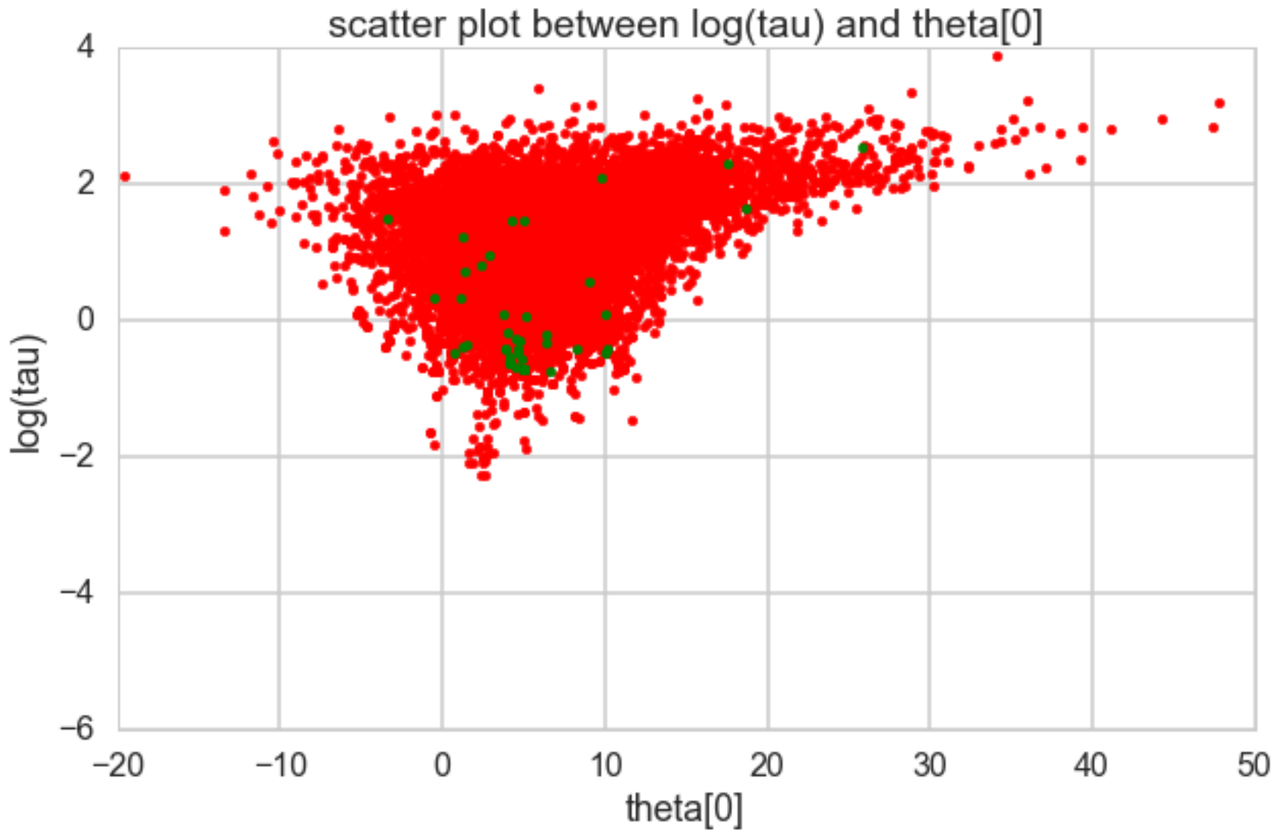
- stickys are actually trying to drive down value of trace
- we are in a region of high curvature

High Curvature Issues

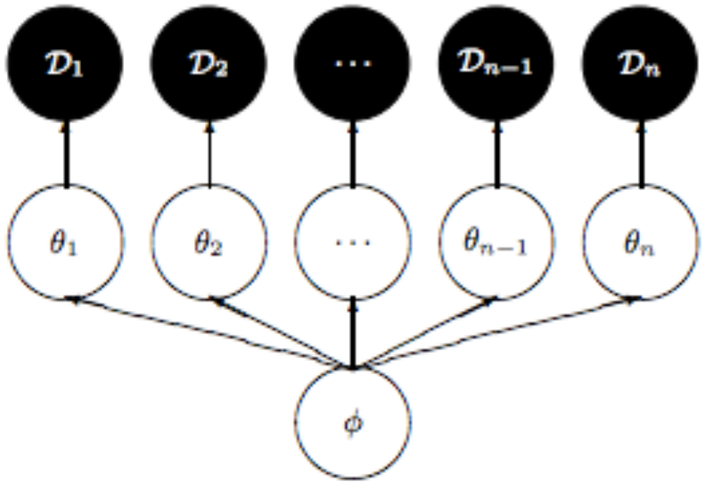


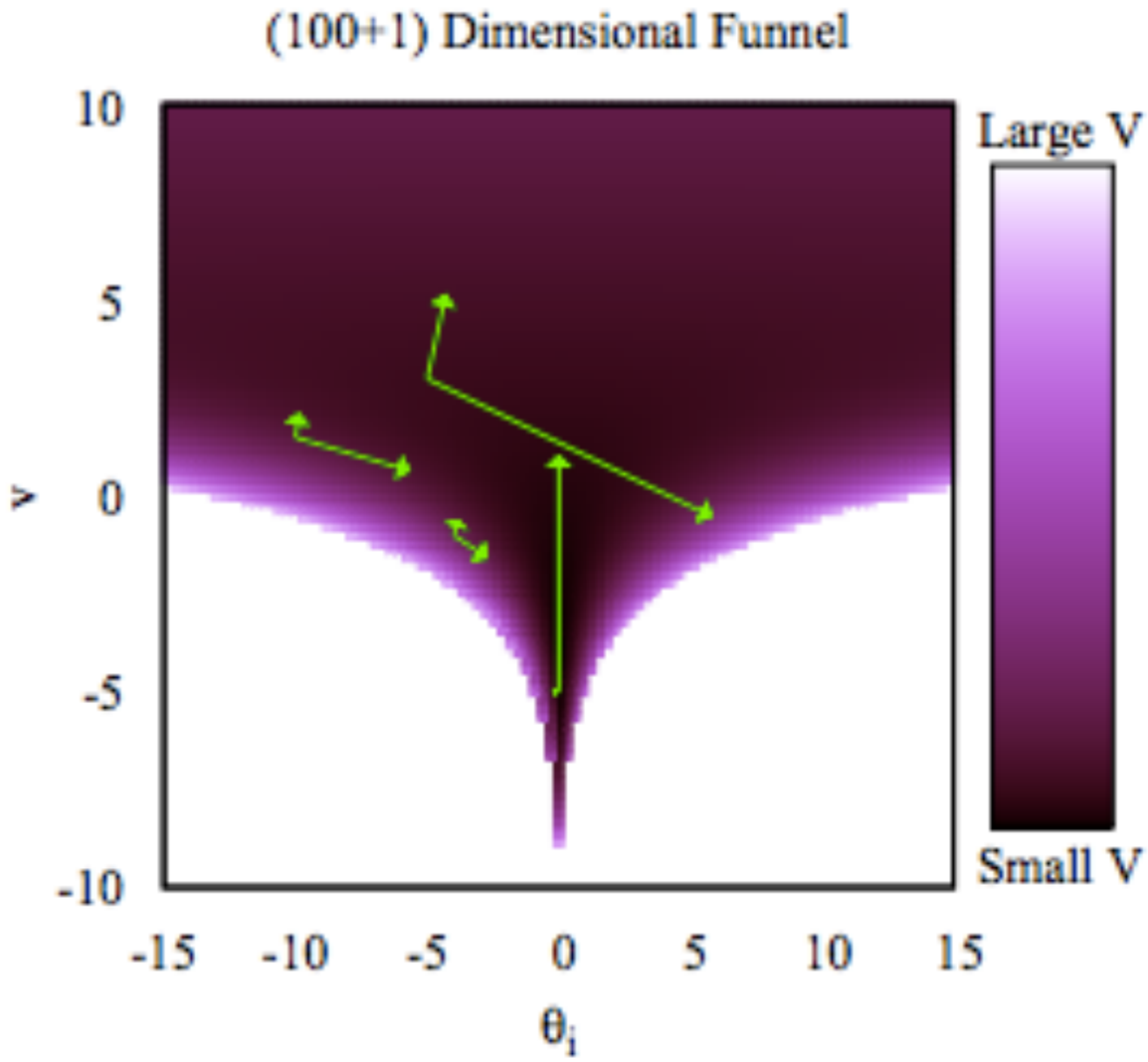
High Curvature Issues

- symplectic integration diverges: good diagnostic. False positives from heuristic.
- sampler needs to have real small steps to not diverge, but then becomes sticky
- regions of high curvature often have high energy differences, causing trouble for microcanonical jump transitions.



Hierarchical Models have high curvature





- characteristic funnel, also there in MH and gibbs
- reflects high correlation between levels in tree
- divergences occur in neck

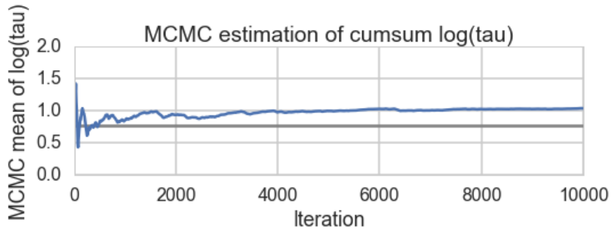
Step size effect

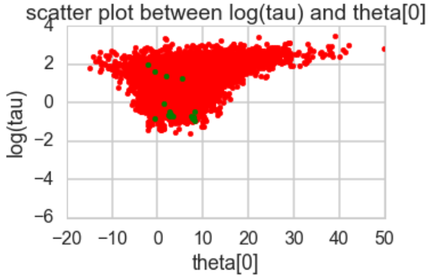
- lower step size ϵ better for symplectic integrators, especially in high curvature regions
- this allows for geometric ergodicity: we go everywhere.
- too small ϵ: return of the random walk.

Changing step size

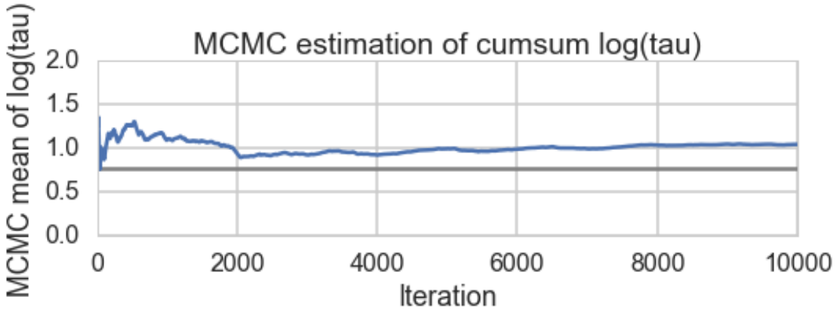
```
with schools1:
    step = pm.NUTS(target_accept=.85)
    trace1_85 = pm.sample(5000, step=step, init=None, njobs=2, tune=1000)
```

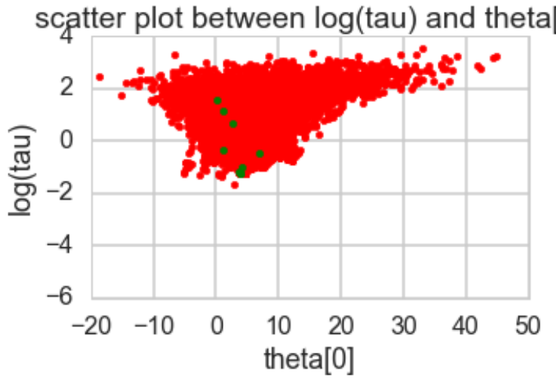
```
85: Acceptance 0.804601458758 Step Size 0.203087336483 Divergence 39
90: Acceptance 0.873340820433 Step Size 0.159223726996 Divergence 18
95: Acceptance 0.923346597897 Step Size 0.126824682121 Divergence 9
99: Acceptance 0.990173791609 Step Size 0.0164237997757 Divergence 5
```

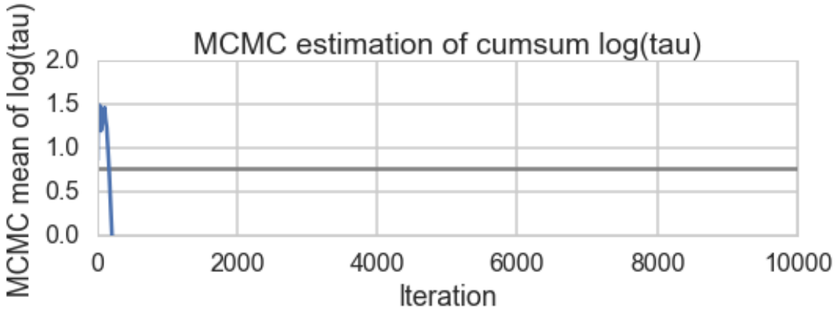


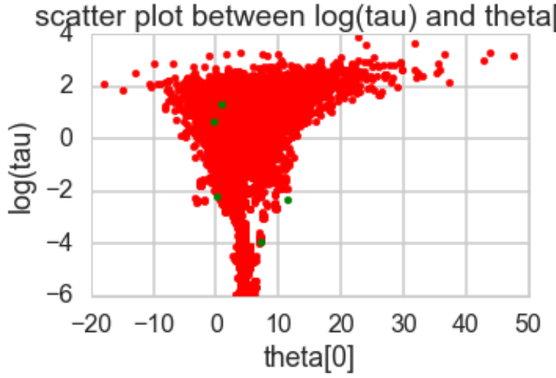


divergences persist. Too curved!



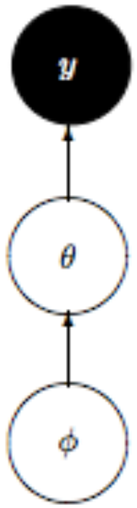


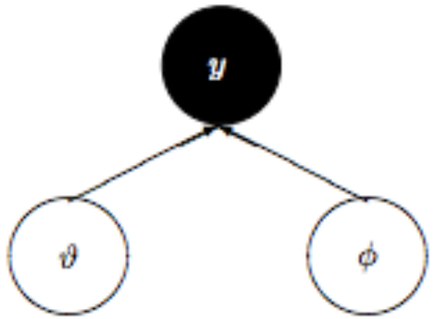




Non-centered model

- could change kinetic energy (riemannian HMC) to make mass matrix dependent upon position
- simpler: reparametrize to reduce levels in hierarchy

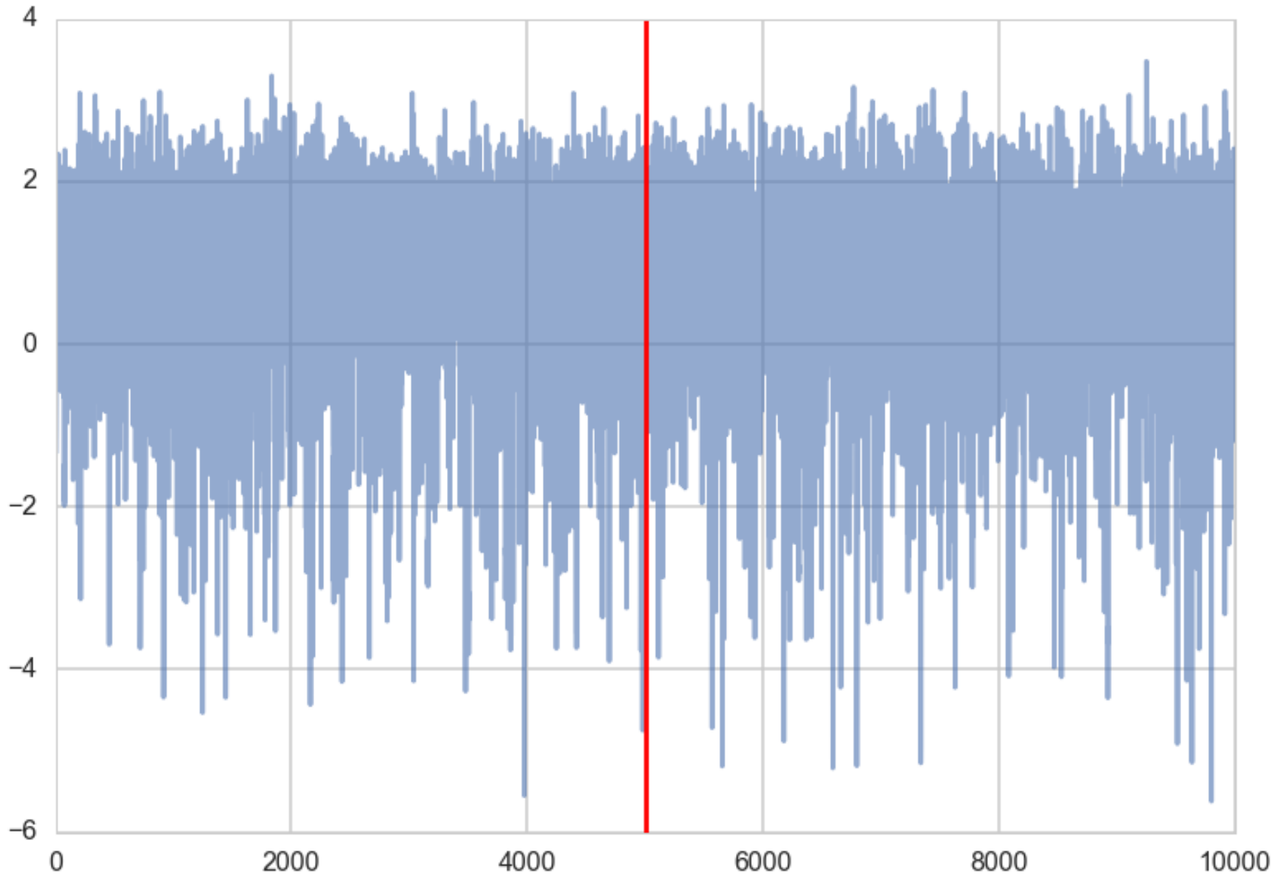


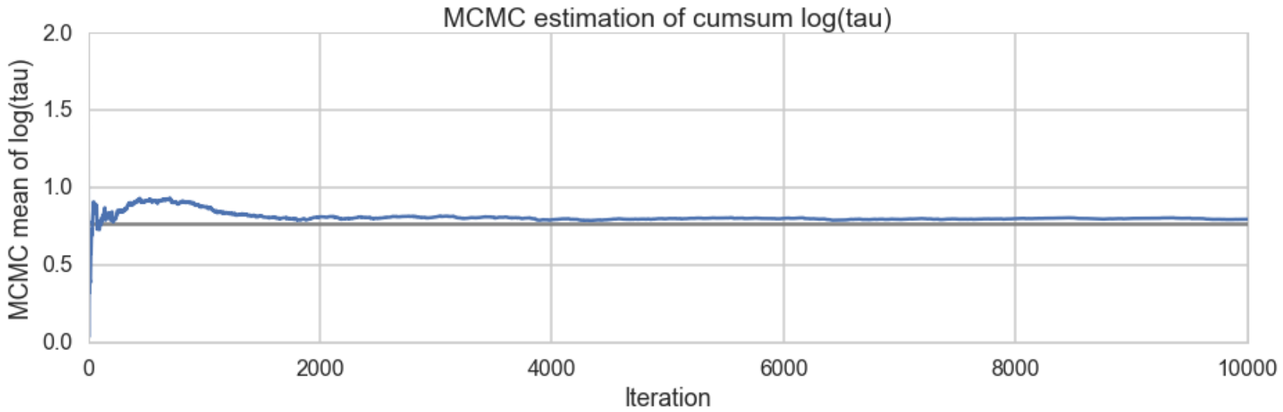


$$\mu \sim \mathcal{N}(0, 5)$$
$$\tau \sim \text{Half-Cauchy}(0, 5)$$
$$\nu_j \sim \mathcal{N}(0, 1)$$
$$\theta_j = \mu + \tau\nu_j$$
$$\bar{y}_j \sim \mathcal{N}(\theta_j, \sigma_j)$$

Factor dependency of θ on $\phi = \mu, \tau$ into a deterministic transformation between the layers, leaving the actively sampled variables uncorrelated.

```
with pm.Model() as schools2:
    mu = pm.Normal('mu', mu=0, sd=5)
    tau = pm.HalfCauchy('tau', beta=5)
    nu = pm.Normal('nu', mu=0, sd=1, shape=J)
    theta = pm.Deterministic('theta', mu + tau * nu)
    obs = pm.Normal('obs', mu=theta, sd=sigma, observed=y)
    trace2 = pm.sample(5000, init=None, njobs=2, tune=500)
```





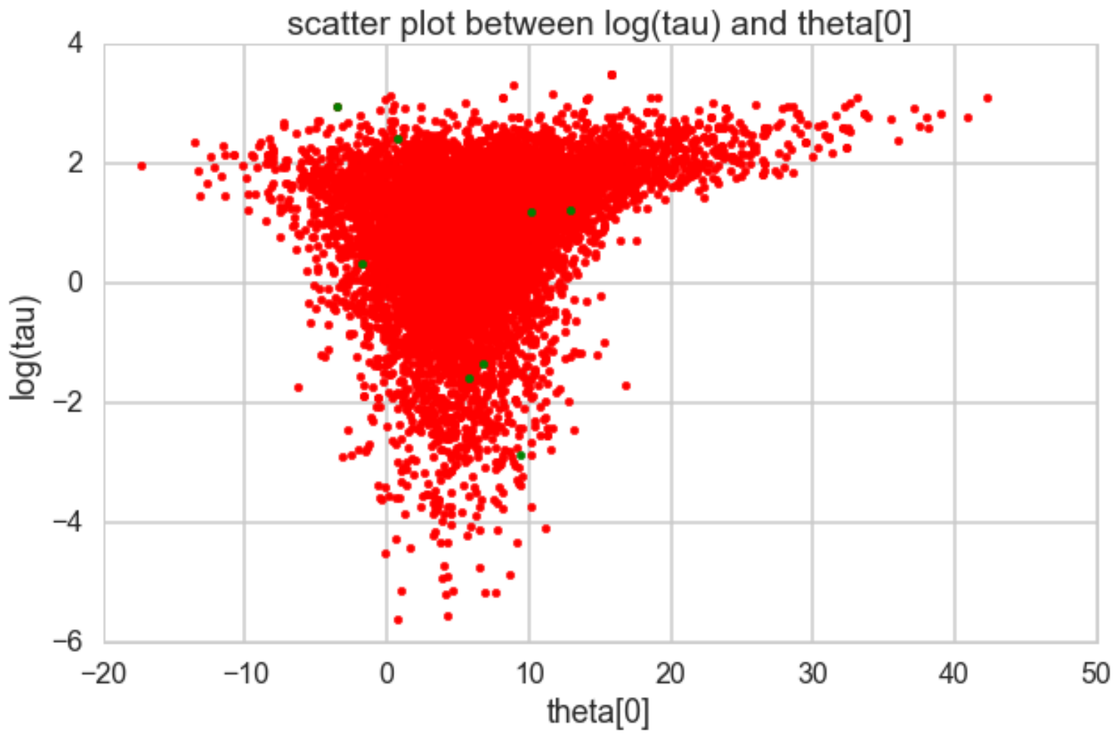
n_{eff}:

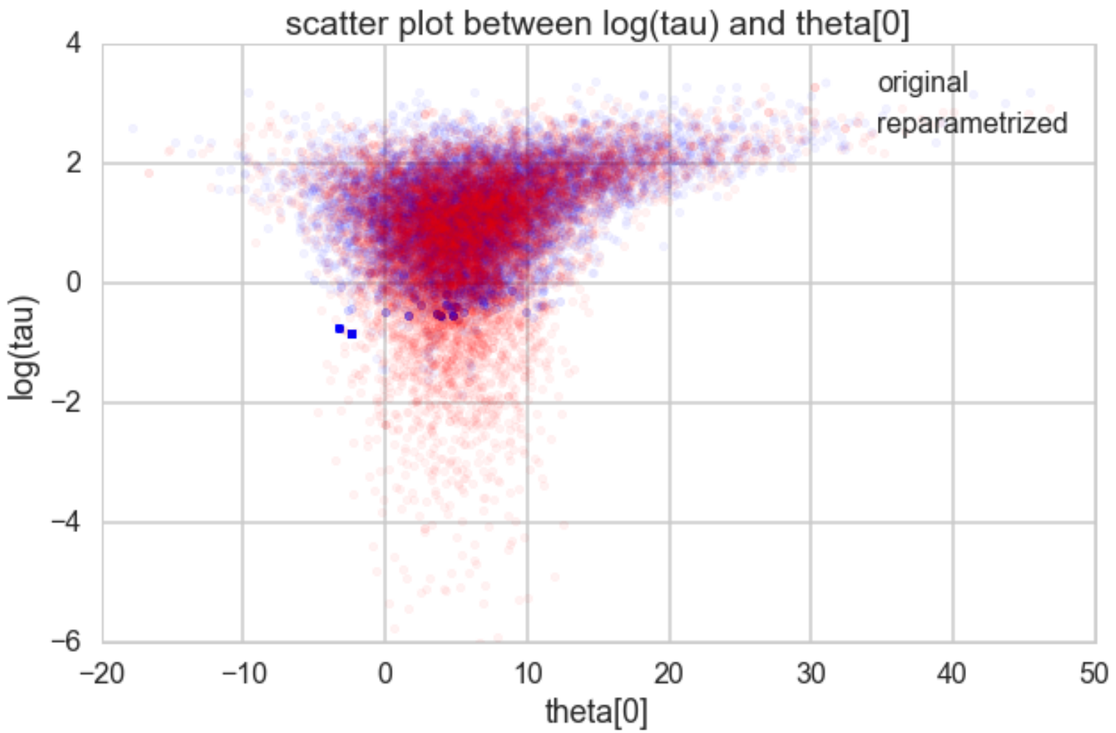
```
{'mu': 10000.0,
  'nu': array([ 10000.,  10000.,  10000.,  10000.,  10000.,  10000.,  10000.,
         10000.]),
  'tau': 6880.0,
  'tau_log_': 5193.0,
  'theta': array([  9624.,  10000.,  10000.,  10000.,  10000.,  10000.,  10000.,
          9829.])}
```

```
divergent = trace2['diverging']
print('Number of Divergent %d' % divergent.nonzero()[0].size)
divperc = divergent.nonzero()[0].size/len(trace2)
print('Percentage of Divergent %.5f' % divperc)
```

```
Number of Divergent 8
Percentage of Divergent 0.00160
```

Divergences and true length of funnel

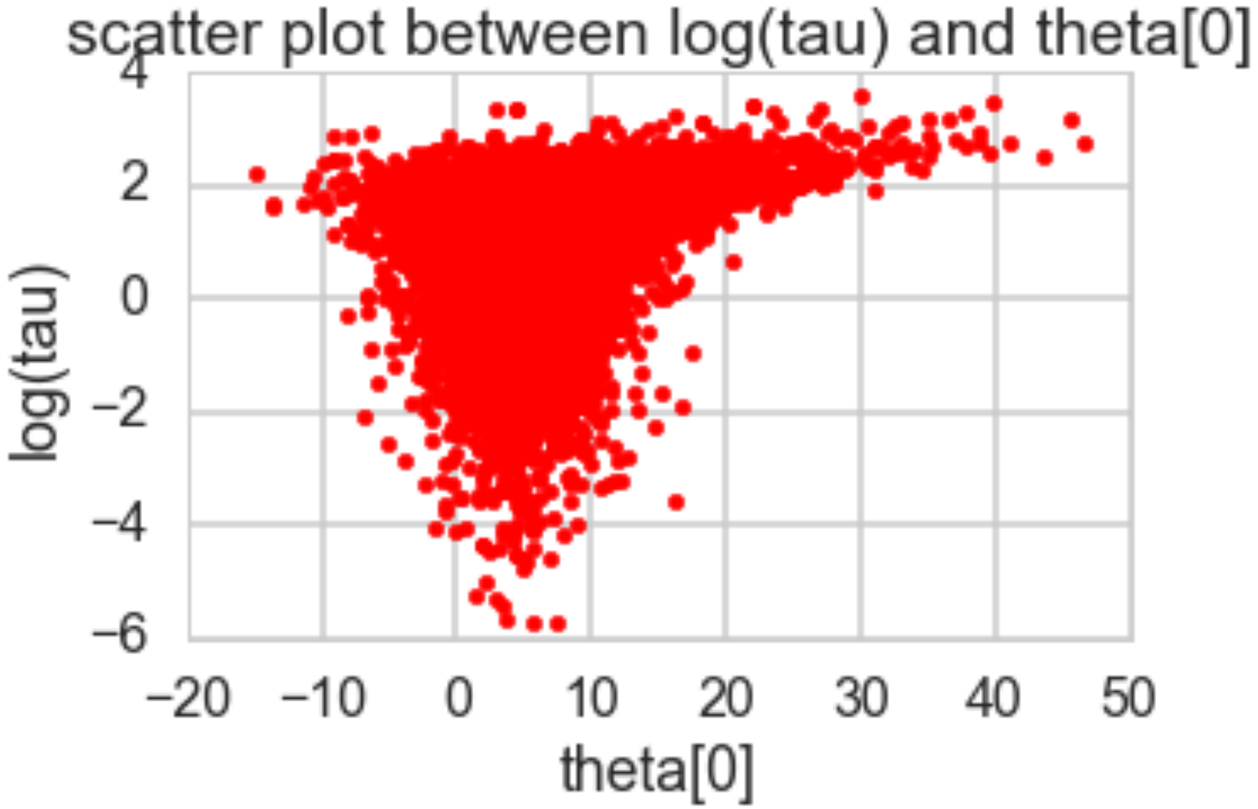


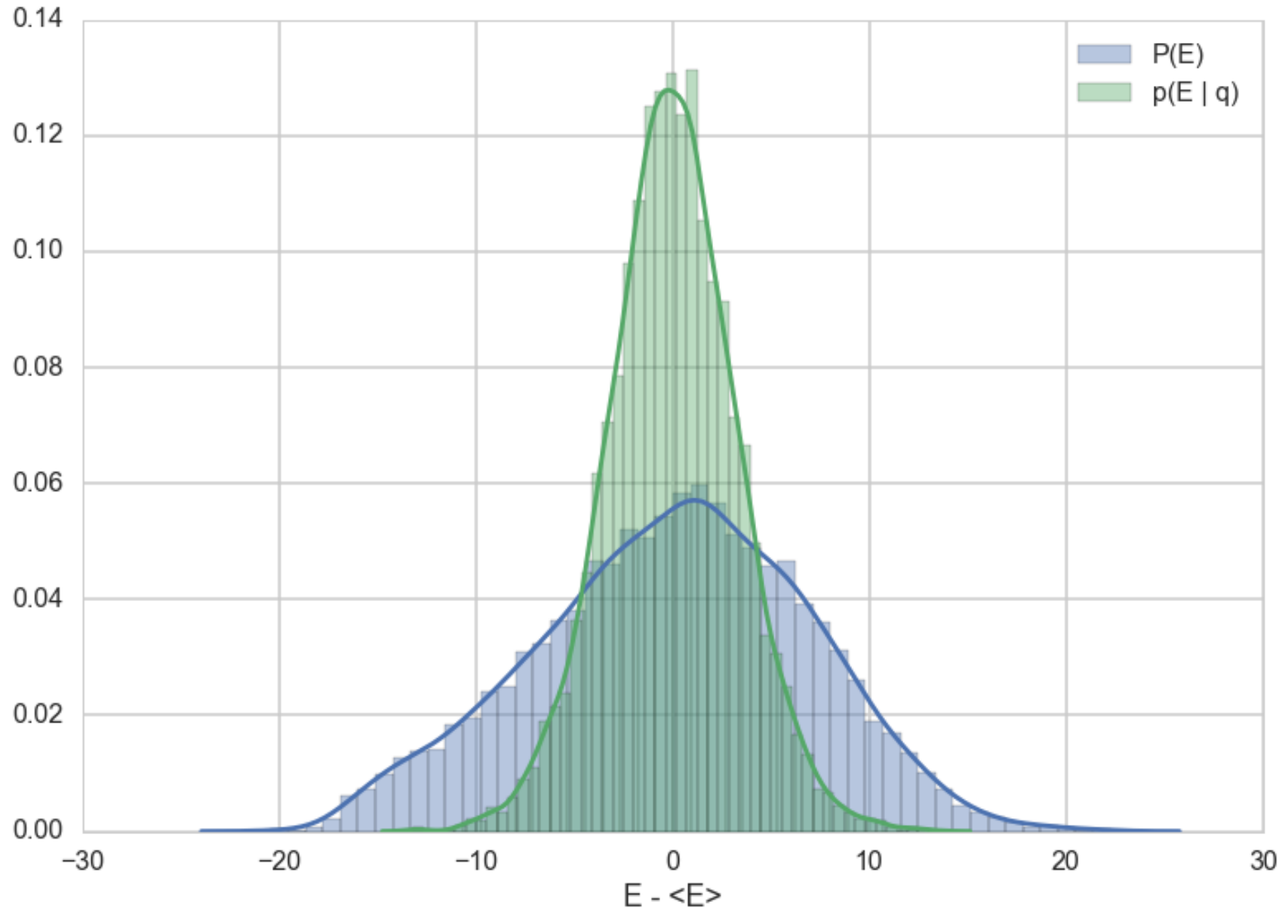


- Divergences infrequent, and all over. Mostly false positives.
- Lowering step sizes should make them go away

```
with schools2:
    step = pm.NUTS(target_accept=.95)
    trace2_95 = pm.sample(5000, step=step, init=None, njobs=2, tune=1000)
```

- lower curvature ensures geometric ergodicity deep in our funnel
- see Betancourt for big discussion





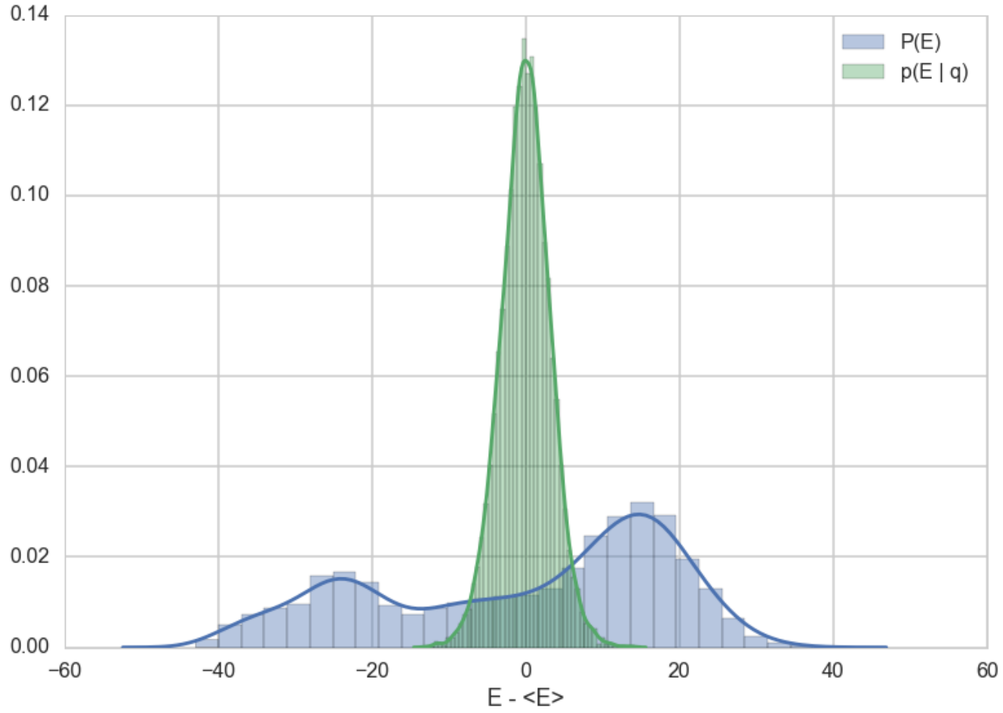
Momentum resampling Efficiency

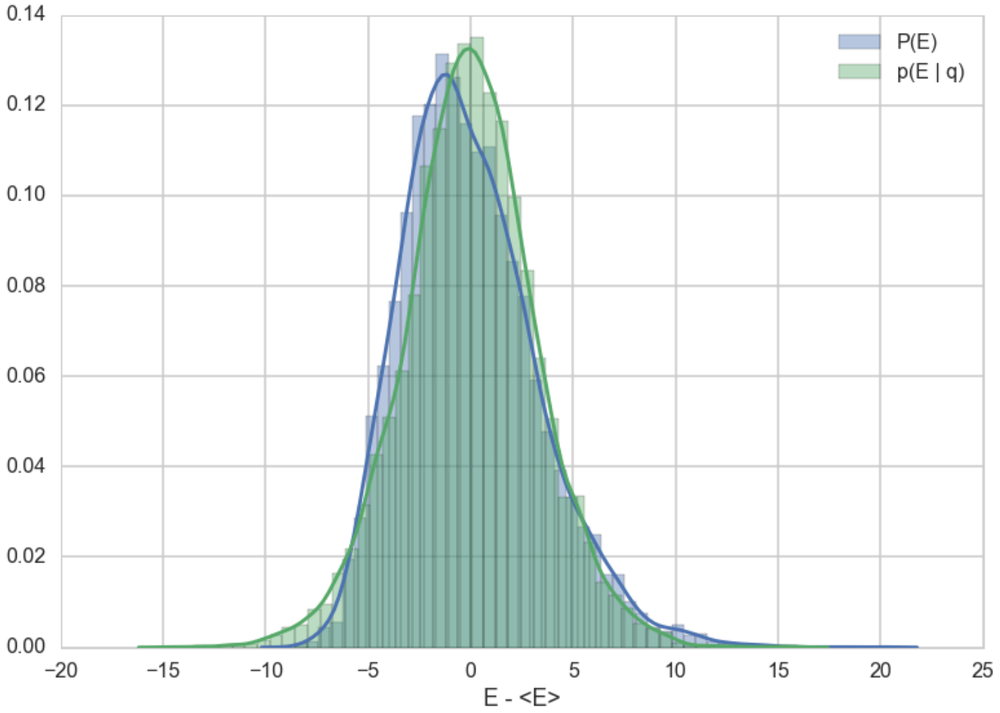
- match transition $p(E|q)$ to marginal $p(E)$

```
def resample_plot(t):
    sns.distplot(t['energy']-t['energy'].mean(), label="P(E)")
    sns.distplot(np.diff(t['energy']), label = "p(E | q)")
    plt.legend();
    plt.xlabel("E - <E>")
```

- if marginal has bigger tails we are in trouble
- indicative here of big energy changes in high-curvature regions not possible to boost to.

centered, small step size vs Non-centered





On left, centered, your sampler is not exploring, so make sure what you are diagnosing. On right, nice match!