Lecture 18

Posterior Geometry and Sampling

Model Checking and glms

Canonical distribution

$$p(p,q) = p(p|q)p(q) = e^{-H(p,q)} = e^{-K(p,q)}e^{-V(q)}$$

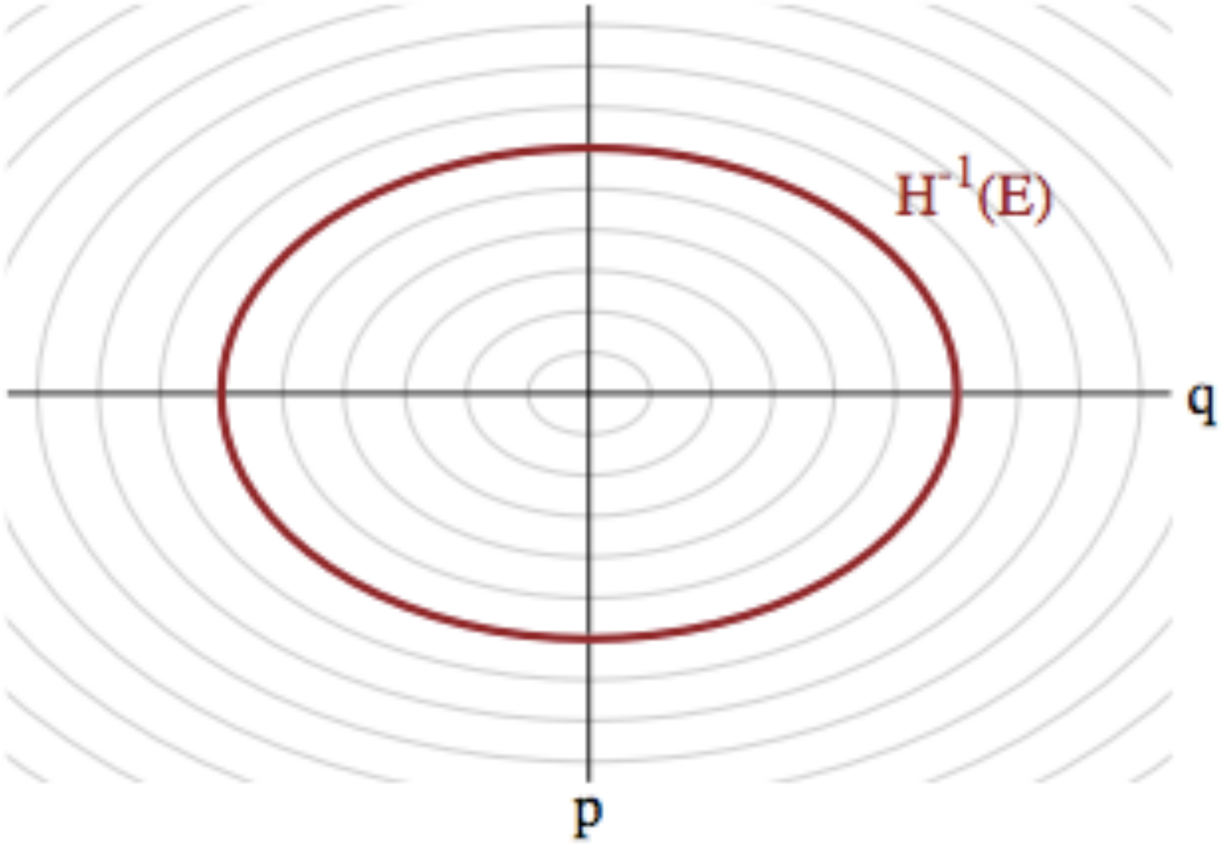
$$\int dpp(p,q) = \int dpp(p|q)p(q) = p(q)\int p(p|q)dp = p(q)$$

$$H(p,q) = \frac{p^2}{2m} + V(q) = E_i\,,$$

Phase Space level sets: Microcanonical Distribution

Typical Set decomposes into level sets of constant probability(energy)

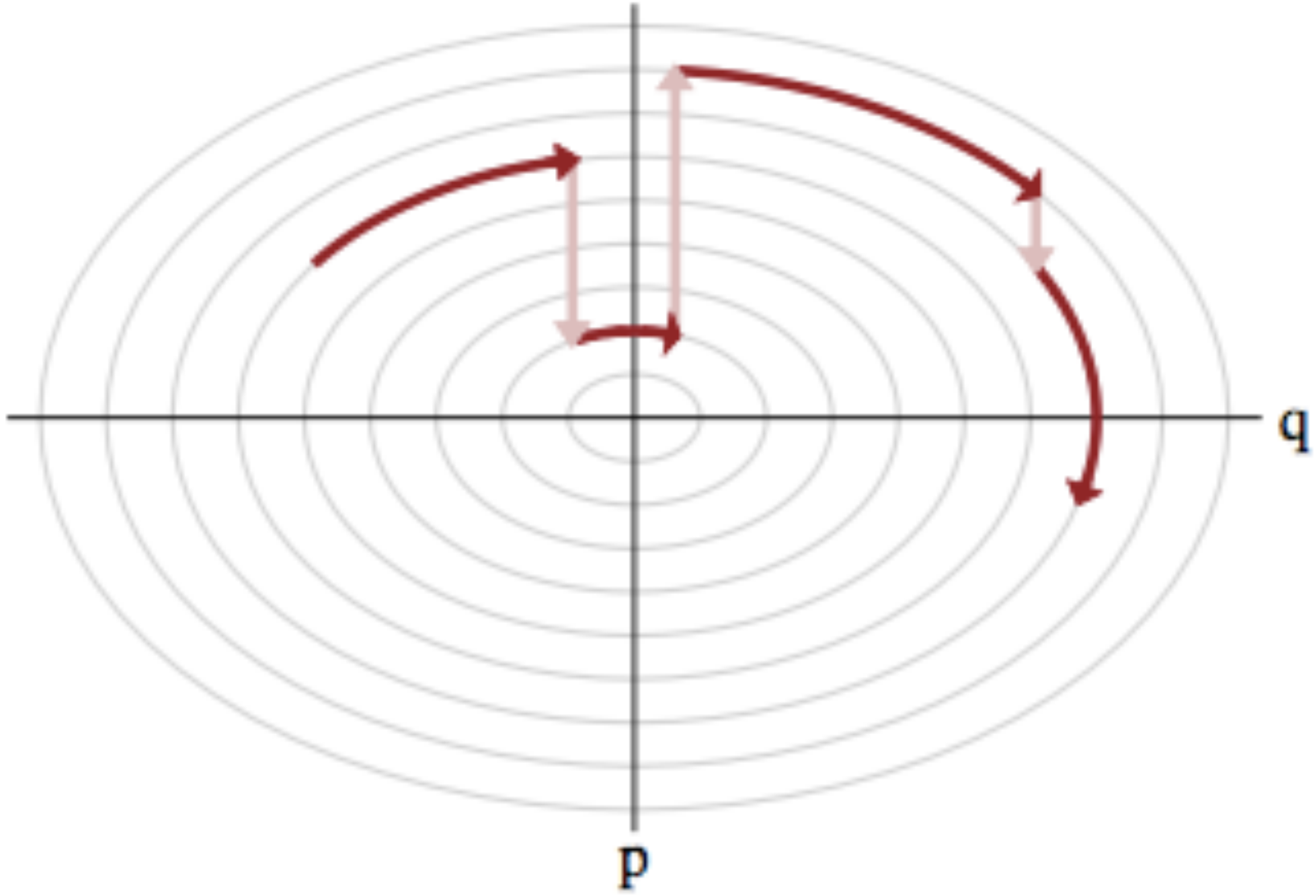
Run Hamiltonian Mechanics to glide and sample from Microcanonical and jump between level sets.



Characteristics

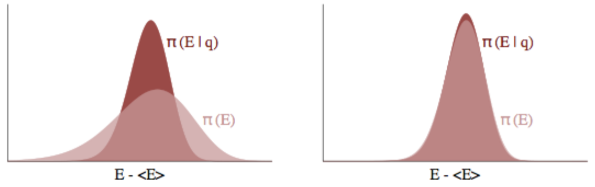
- Superman transform reversibility: run, flip, run back, flip
- Volume in phase space is conserved, use symplectic integration
- thus momenta are **dual**, can use covariance as inverse mass matrix

Momentum resampling



Draw $p \sim N(0, \sqrt{M})$ for example, and attempt to explore the level sets.

Microcanonical exploration (approximate) followed by momentum resampling



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

Tuning

- use mass as inverse covariance to decorrelate target
- too small ϵ means slow random walk, too much cannot cover curvature.
- for L, find the point at which the orbital expectations converge to the spatial expectations..a sort of ergodicity
- generally static L not good, under-samples tails (high-energy micro-canonicals). Estimate dynamically: NUTS (pymc3 and Stan)

Acceptance probability

- symplectic integration keeps you only approximately microcanonical. Efficient sampler. Optimal acceptance can be shown: > 65% roughly.
- thus: $A = \min[1, \exp(-U(q_L) + U(q) - K(p_L) + K(p)]$
- 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

HMC Algorithm

- 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. $r \sim \text{Unif}(0,1)$
    - 11. if $r < e^{(U_c - U^* + K_c - K^*)}$
        - accept $q_i = q^*$
        - otherwise reject

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

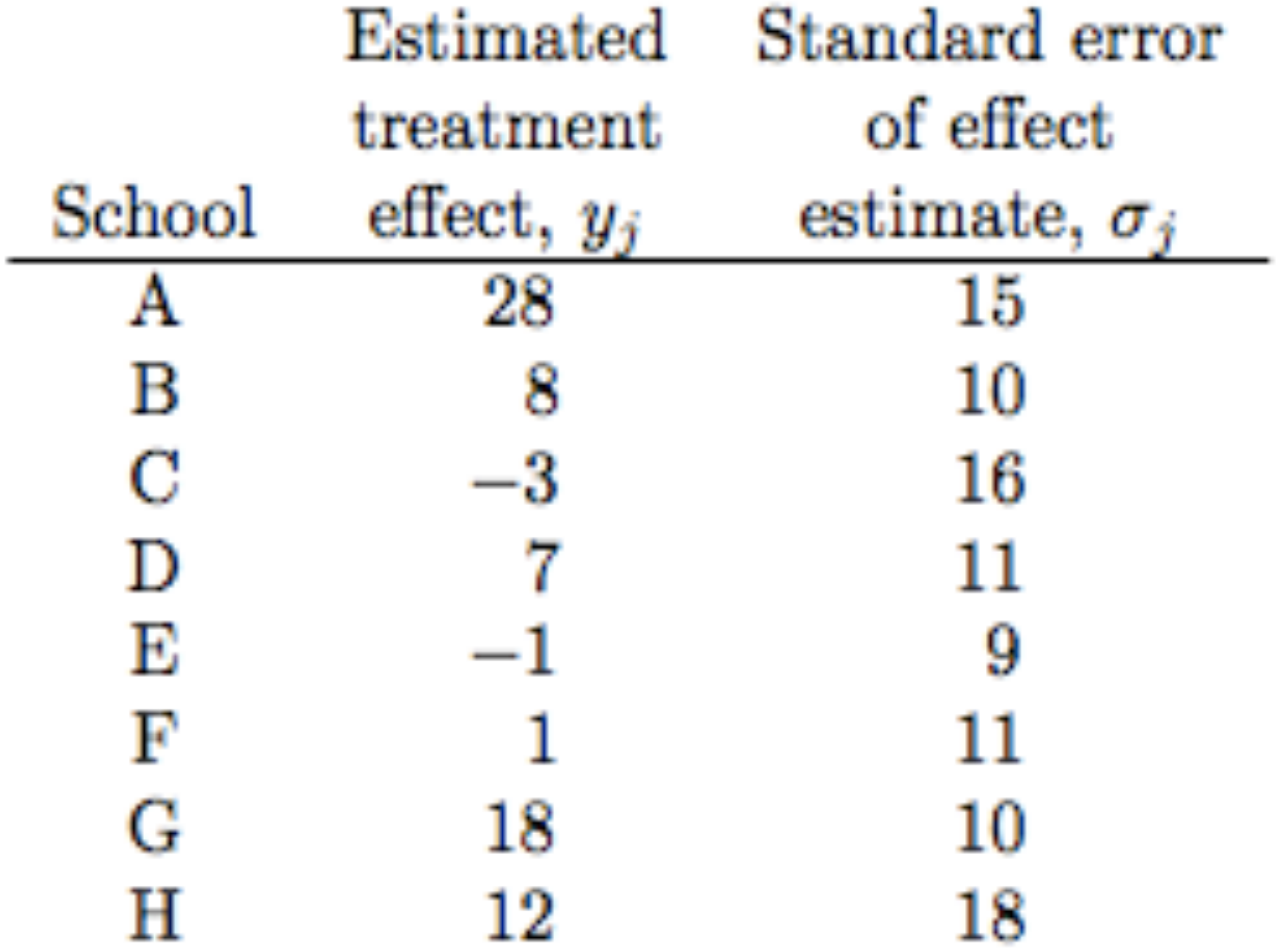
$\bar{y_j} = \frac{1}{n_j} \sum_{i=1}^{n_j} y_{ij}$ with sampling variance

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

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

Installation

```
pip install theano==0.9
pip install pymc3==3.1rc2
```

```
pm.__version__
'3.1.rc2'
```

should as of yesterday be possible using conda directly

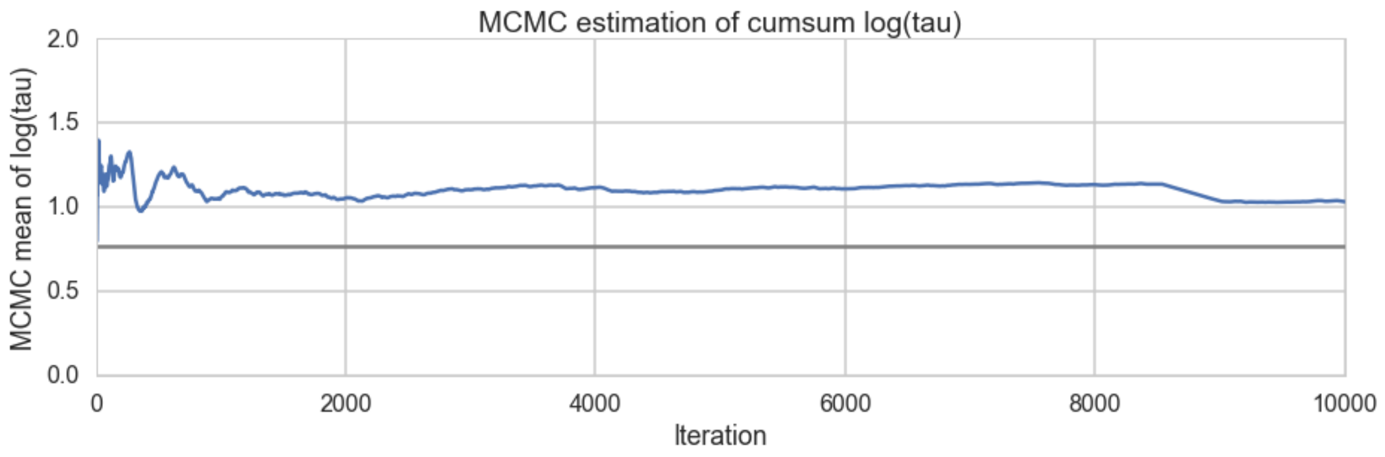
Centered Hierarchical Model

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

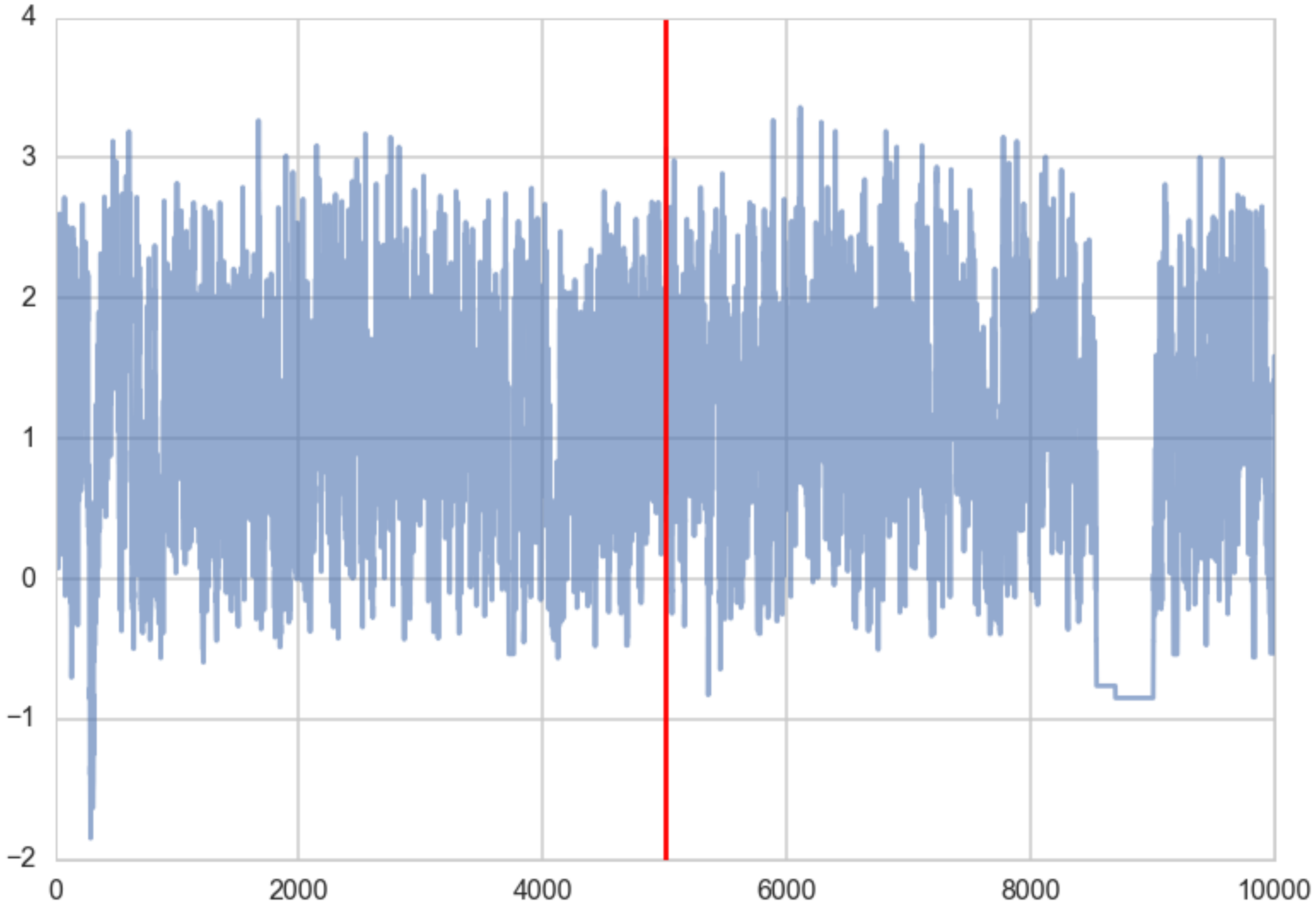
```
with pm.Model() as schools1:

    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.])})
```

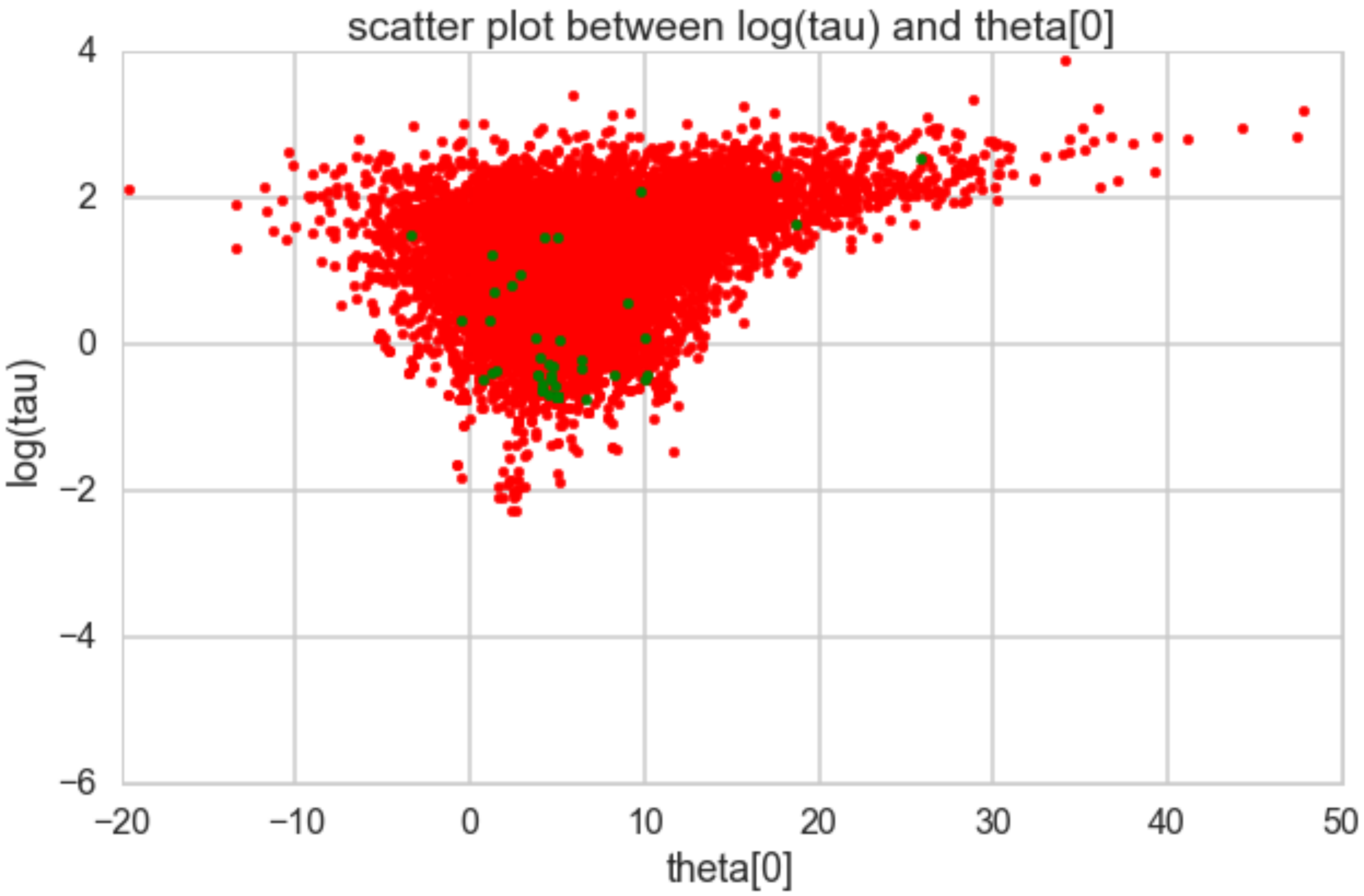


- stickys are actually trying to drive down value of trace
- we are in a region of high curvature



High Curvature Issues

- 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.



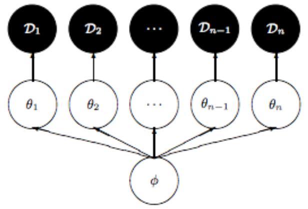
Diagnosed thus:

```
divergent = trace1['diverging']
print('Number of Divergent %d' % divergent.nonzero()[0].size)
divperc = divergent.nonzero()[0].size/len(trace1)
print('Percentage of Divergent %.5f' % divperc)
```

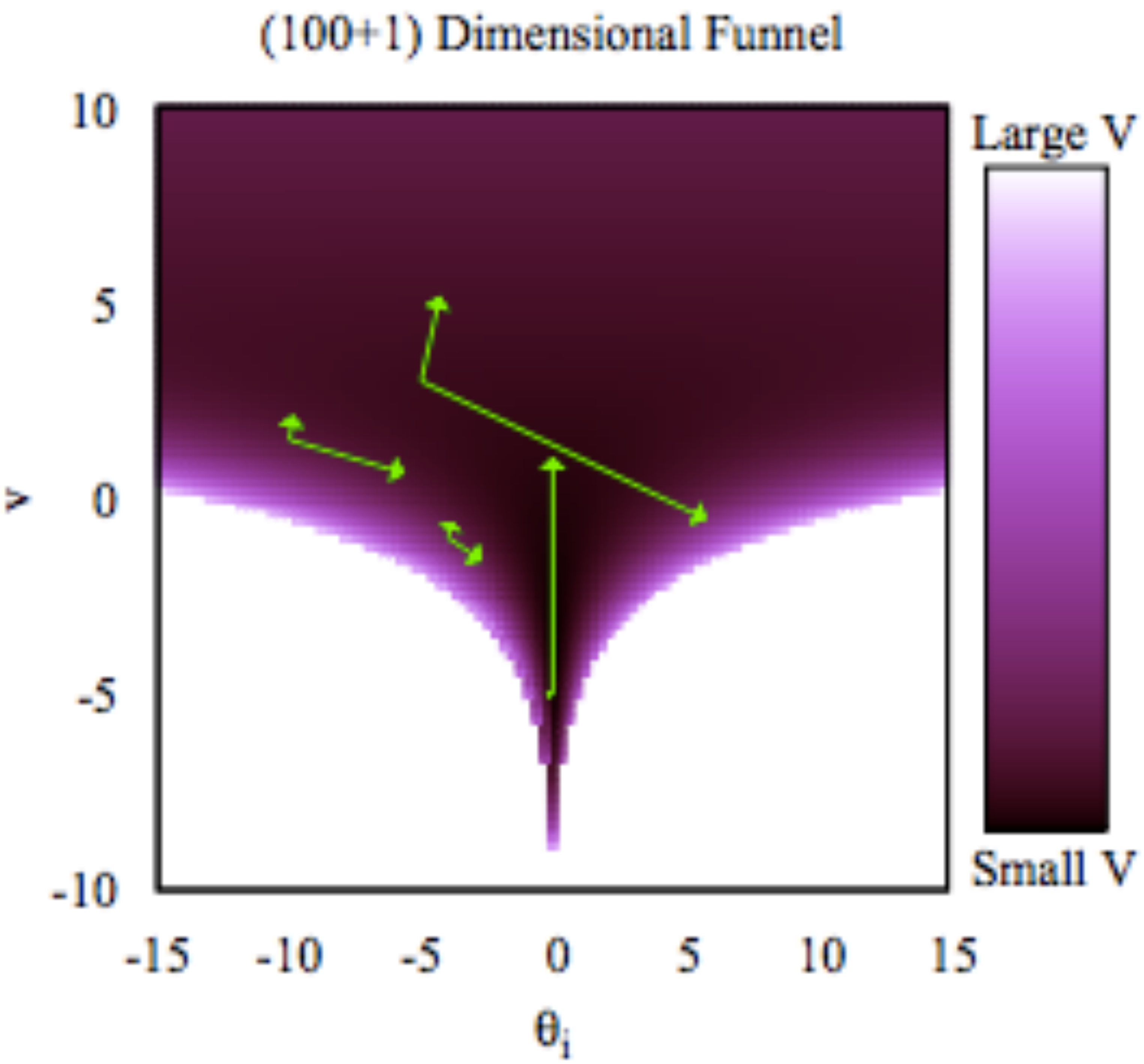
```
Number of Divergent 74
Percentage of Divergent 0.01480
```

- Not characterizing neck well
- No confidence in postrior in this region

Hierarchical Models have high curvature



- characteristic funnel, also there in MH and gibbs
- reflects high correlation between levels in tree
- divergences occur in neck, others may be false positives



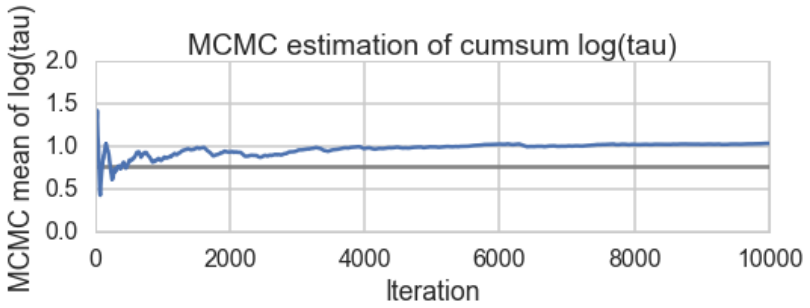
Step size effect

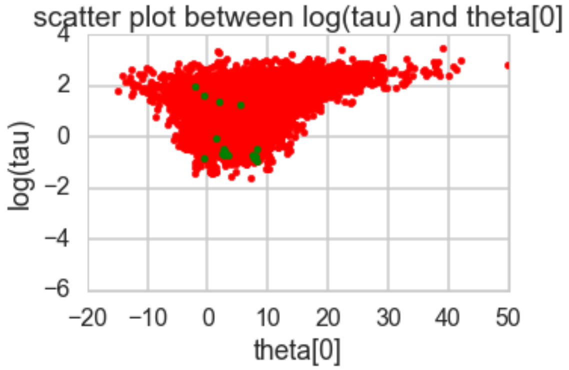
- lower step size ϵ better for symplectic integrators, especially in high curvature regions
- this allows for geometric ergodicity: we go everywhere.
- too small ϵ: return of the random walk.

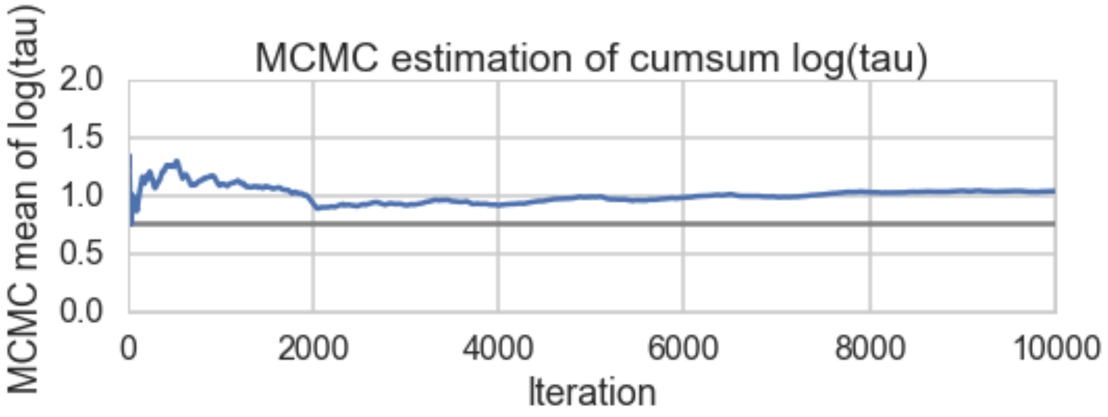
Changing step size

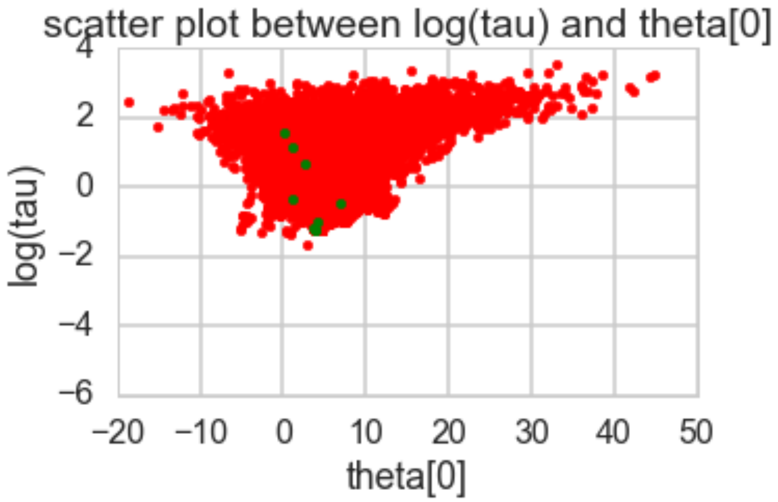
```
with schools1:
    step = pm.NUTS(target_accept=.85)
    trace1_85 = pm.sample(5000, step=step, init=None, njobs=2, tune=1000)
```

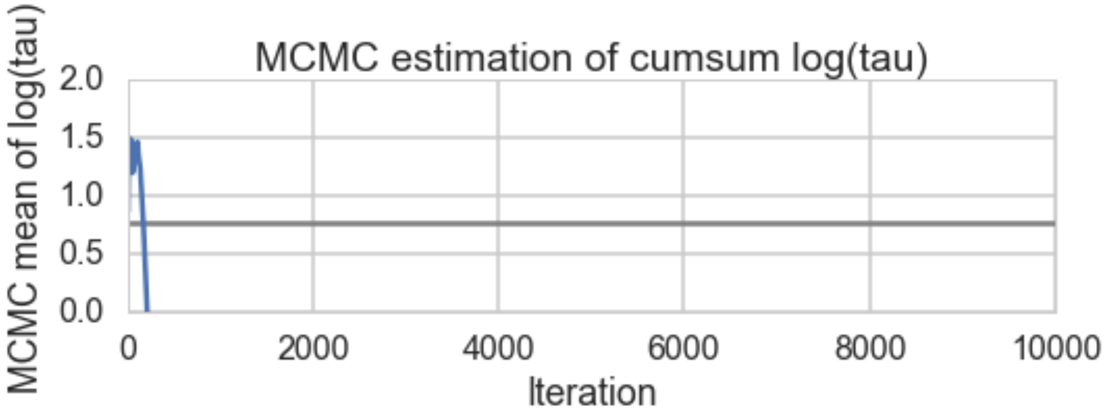
```
85: Acceptance 0.804601458758 Step Size 0.203087336483 Divergence 39
90: Acceptance 0.873340820433 Step Size 0.159223726996 Divergence 18
95: Acceptance 0.923346597897 Step Size 0.126824682121 Divergence 9
99: Acceptance 0.990173791609 Step Size 0.0164237997757 Divergence 5
```

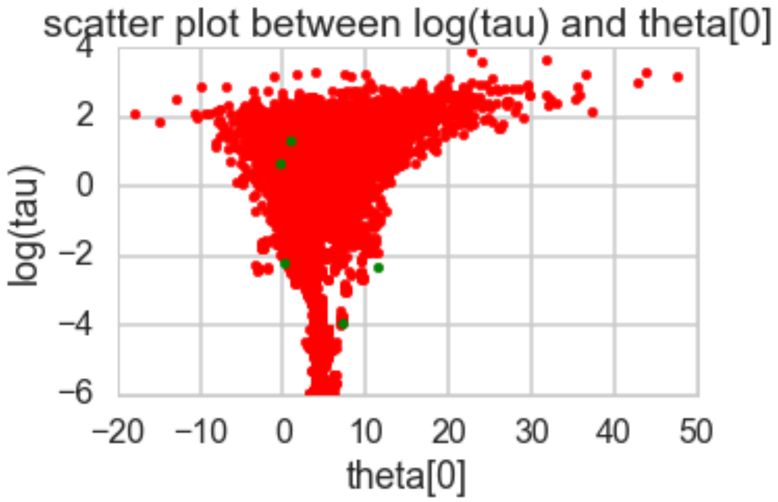








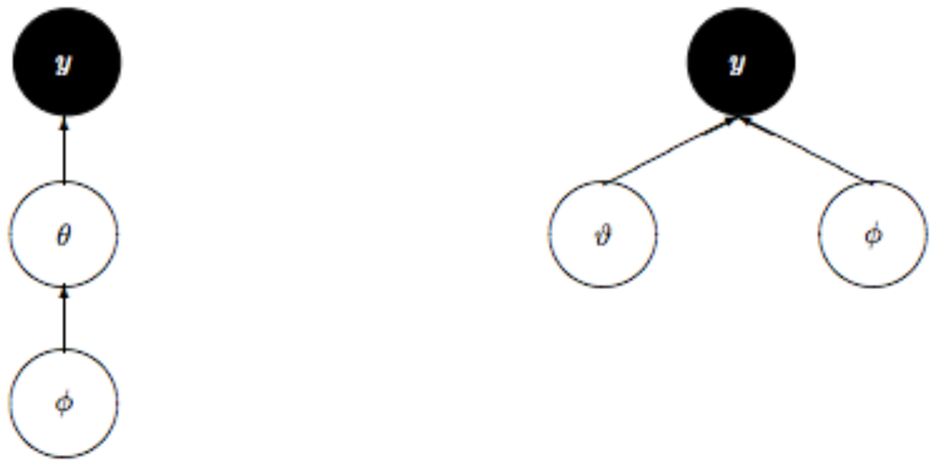




divergences persist. Too curved!

Non-centered model

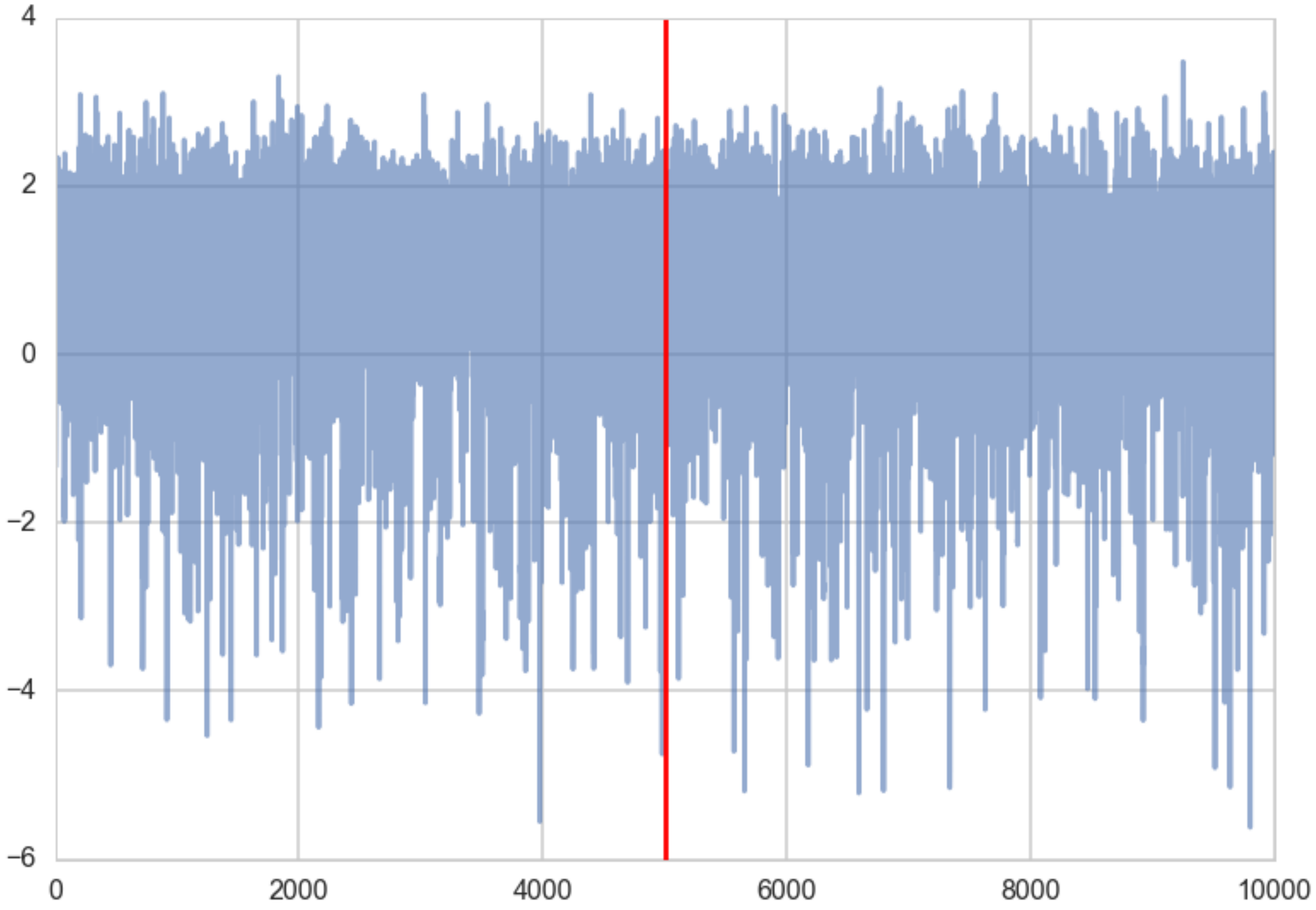
- could change kinetic energy (riemannian HMC) to make mass matrix dependent upon position
- simpler: reparametrize to reduce levels in hierarchy



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

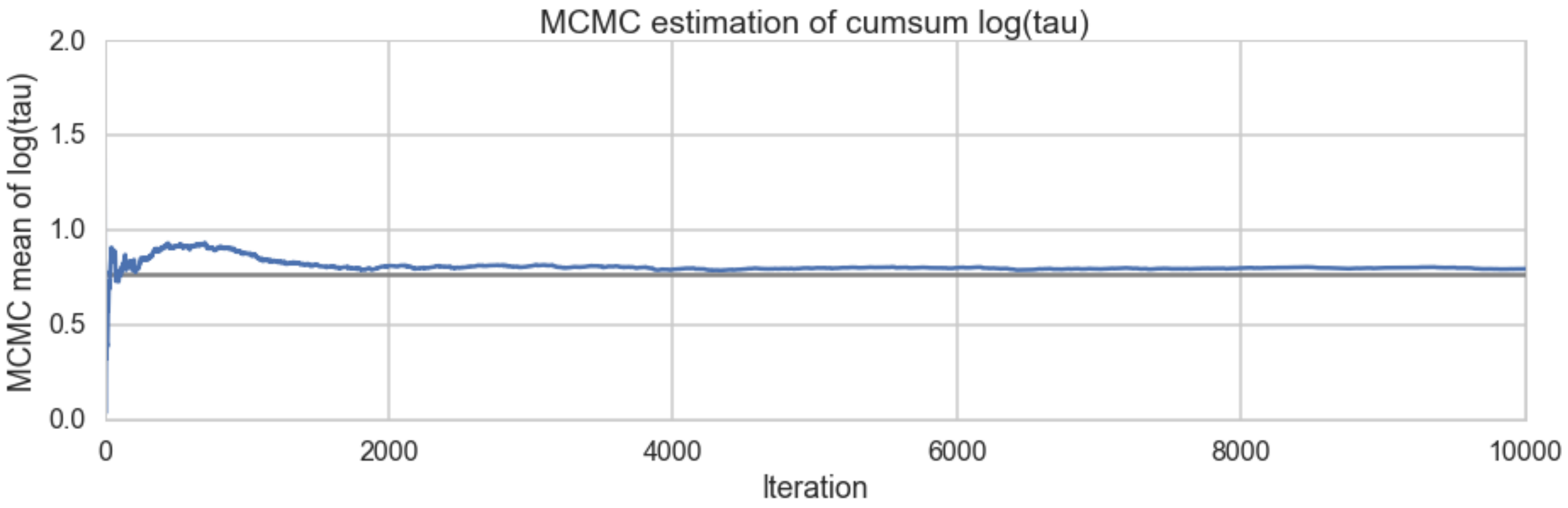
Factor dependency of θ on $\phi = \mu, \tau$ into a deterministic transformation between the layers, leaving the actively sampled variables uncorrelated.

```
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}:

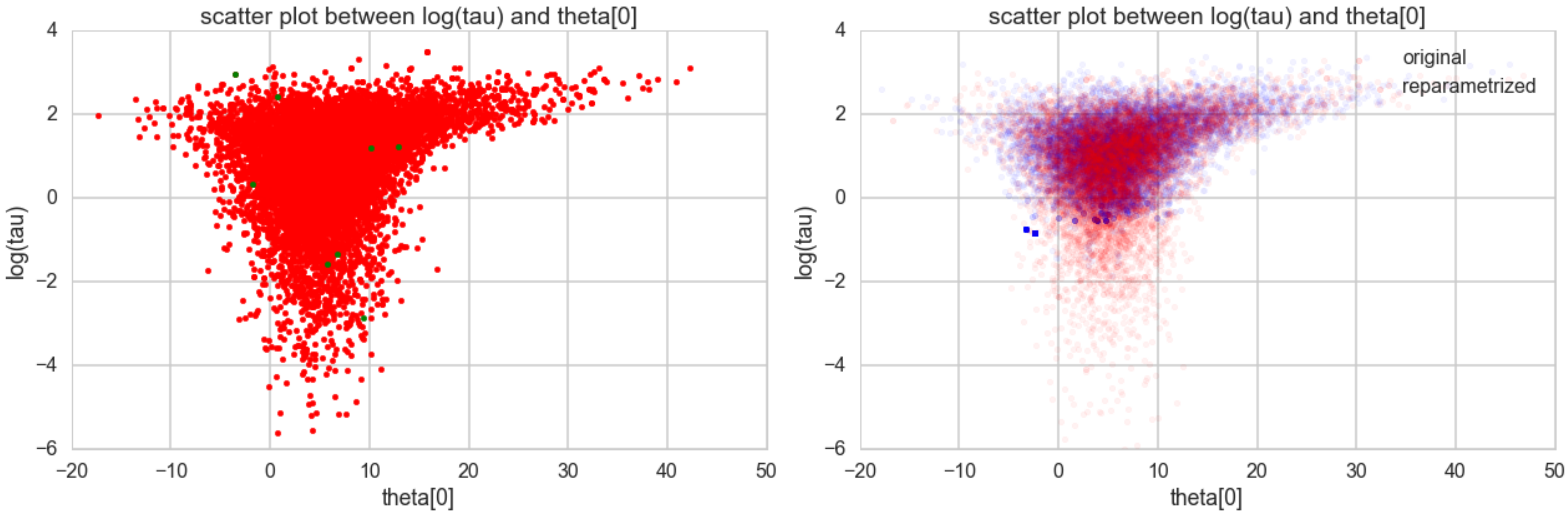
```
{'mu': 10000.0,
  'nu': array([ 10000.,  10000.,  10000.,  10000.,  10000.,  10000.,  10000.,
         10000.]),
  'tau': 6880.0,
  'tau_log_': 5193.0,
  'theta': array([  9624.,  10000.,  10000.,  10000.,  10000.,  10000.,  10000.,
          9829.])}
```


MCMC estimation of cumsum log(tau)

```
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
```

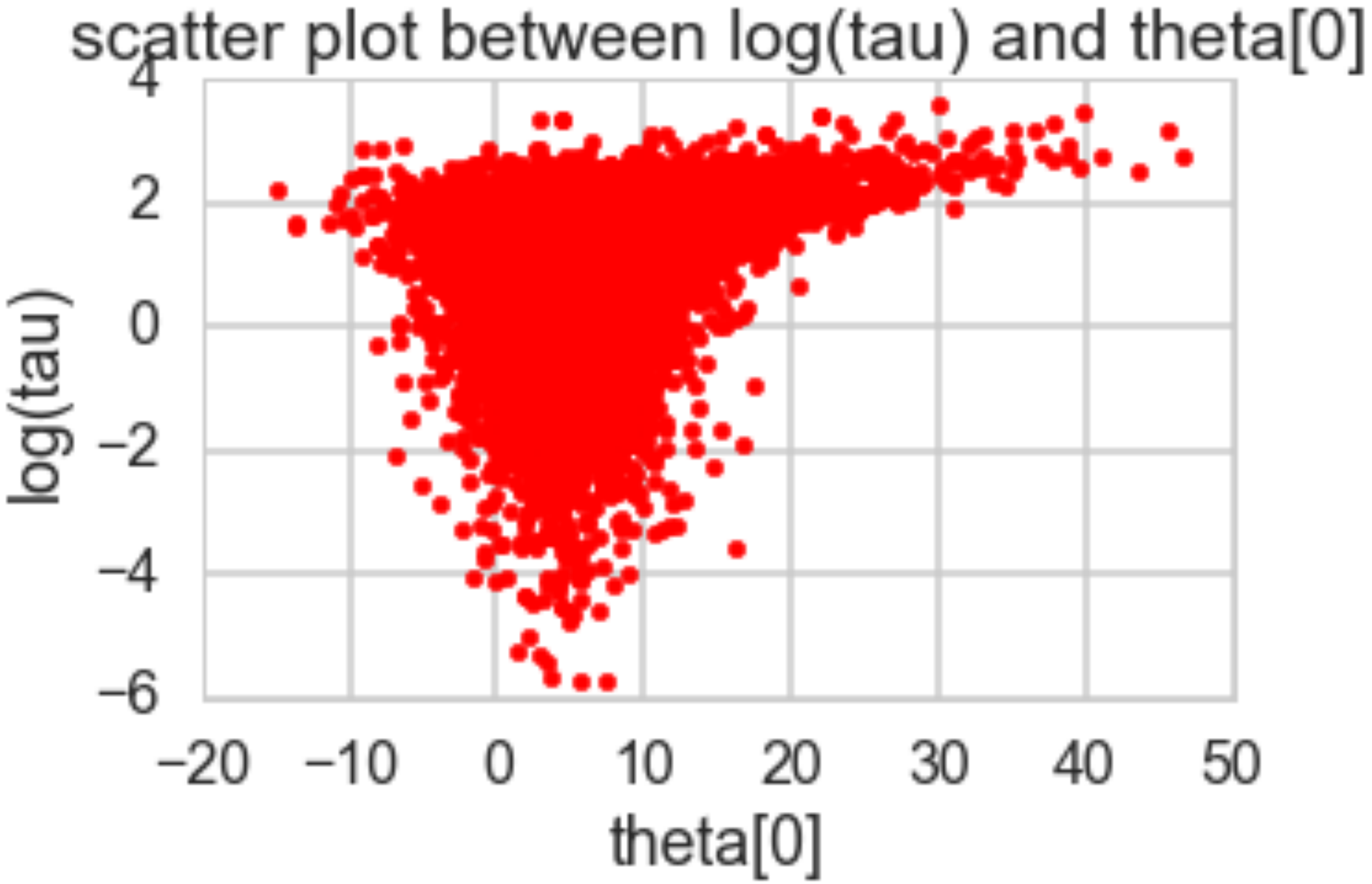
Divergences and true length of funnel



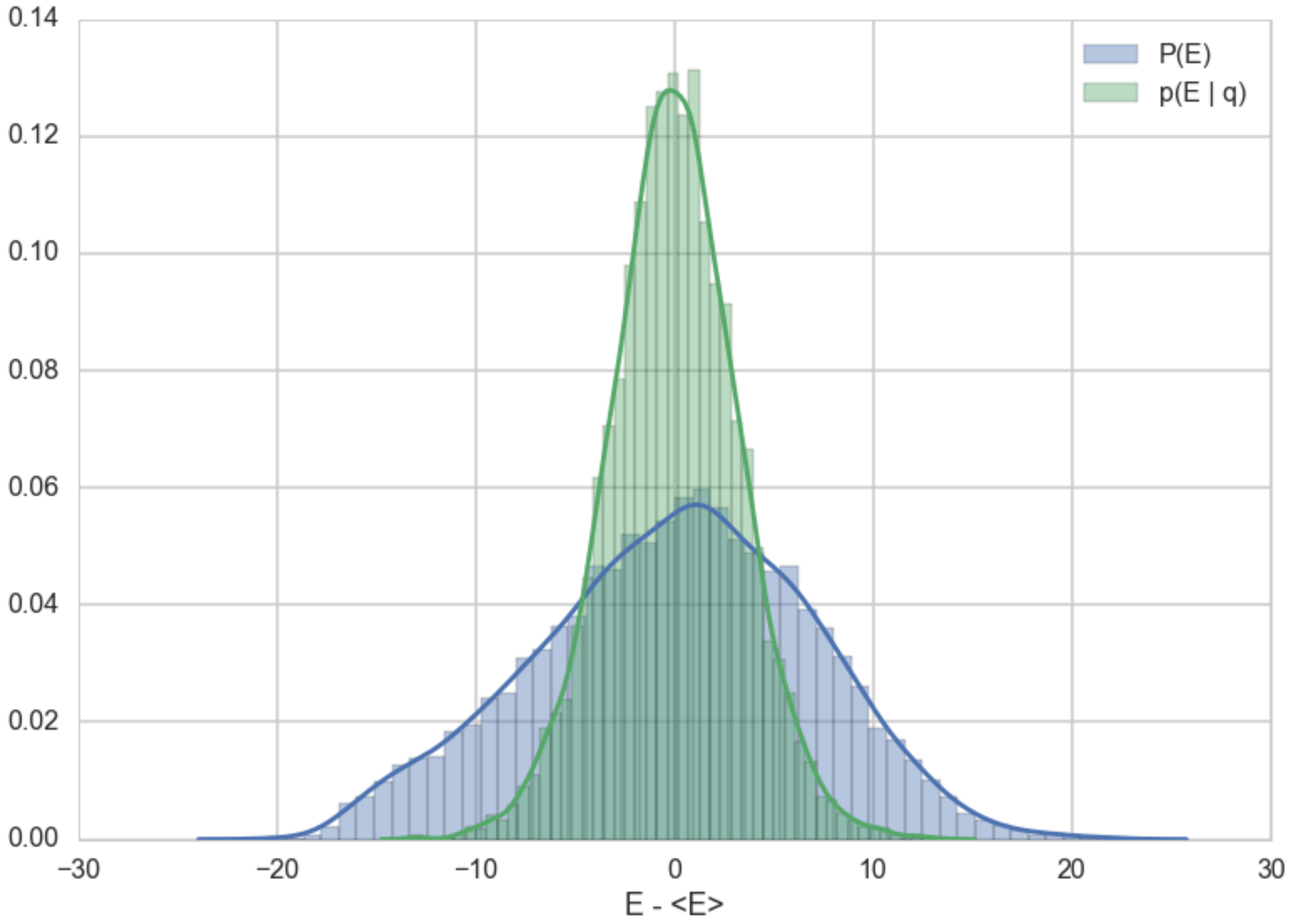
- Divergences infrequent, and all over. Mostly false positives.
- Lowering step sizes should make them go away

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

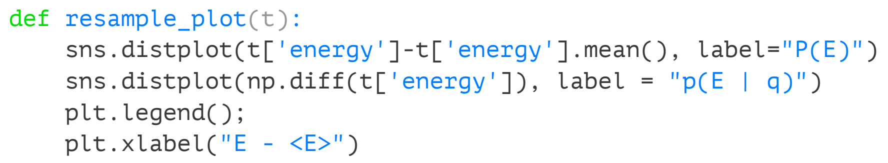
- lower curvature ensures geometric ergodicity deep in our funnel
- see Betancourt for big discussion



Momentum resampling Efficiency



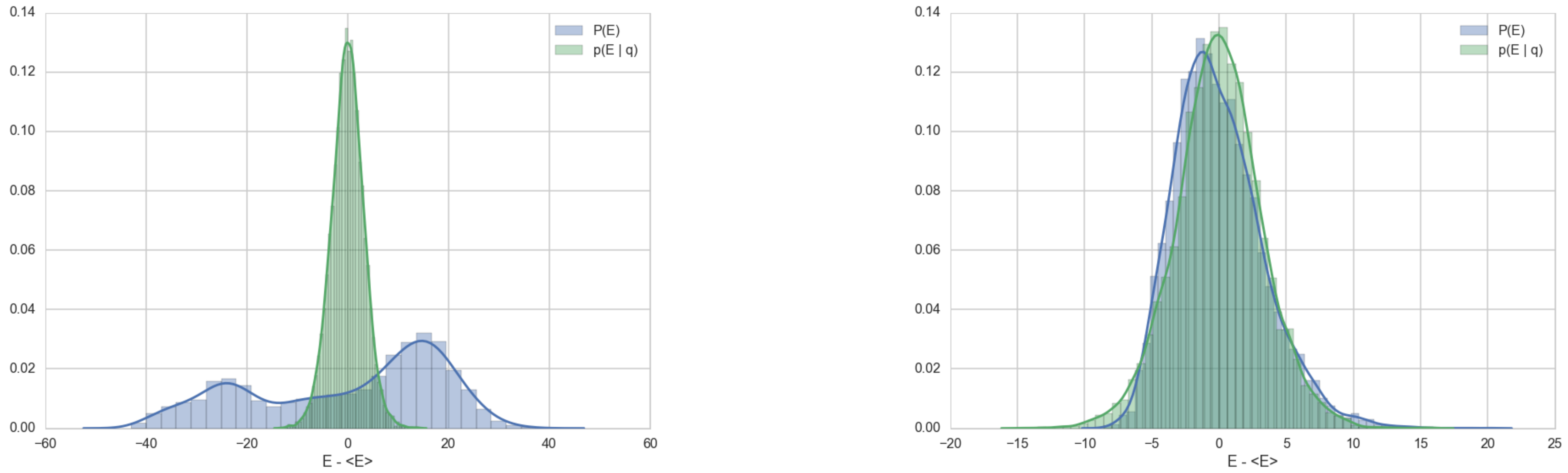
- match transition $p(E|q)$ to marginal $p(E)$



```
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.

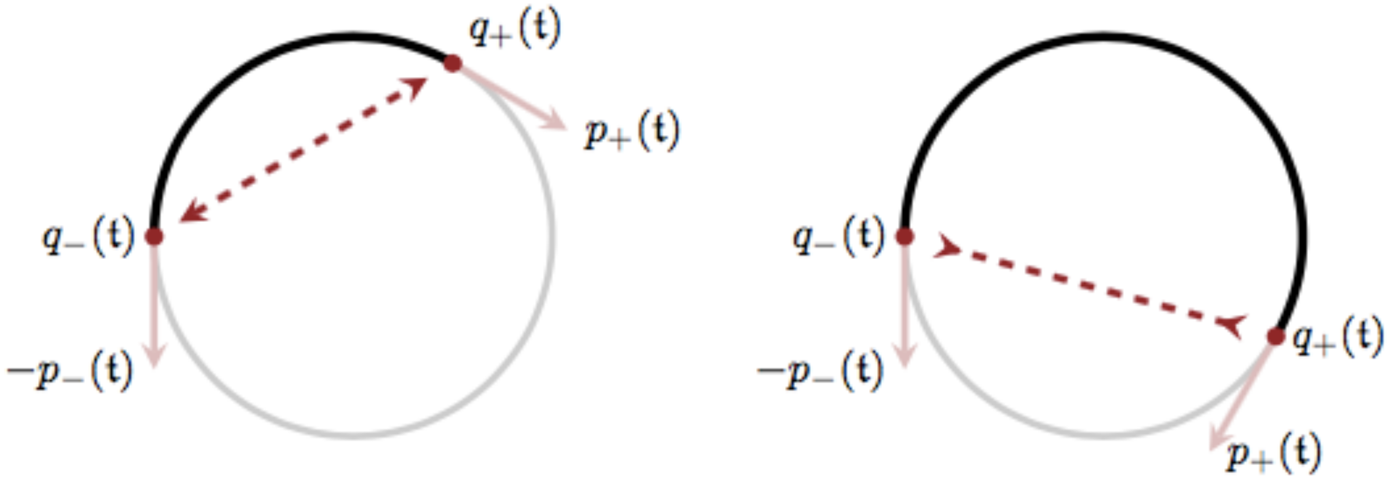
centered, small step size vs Non-centered



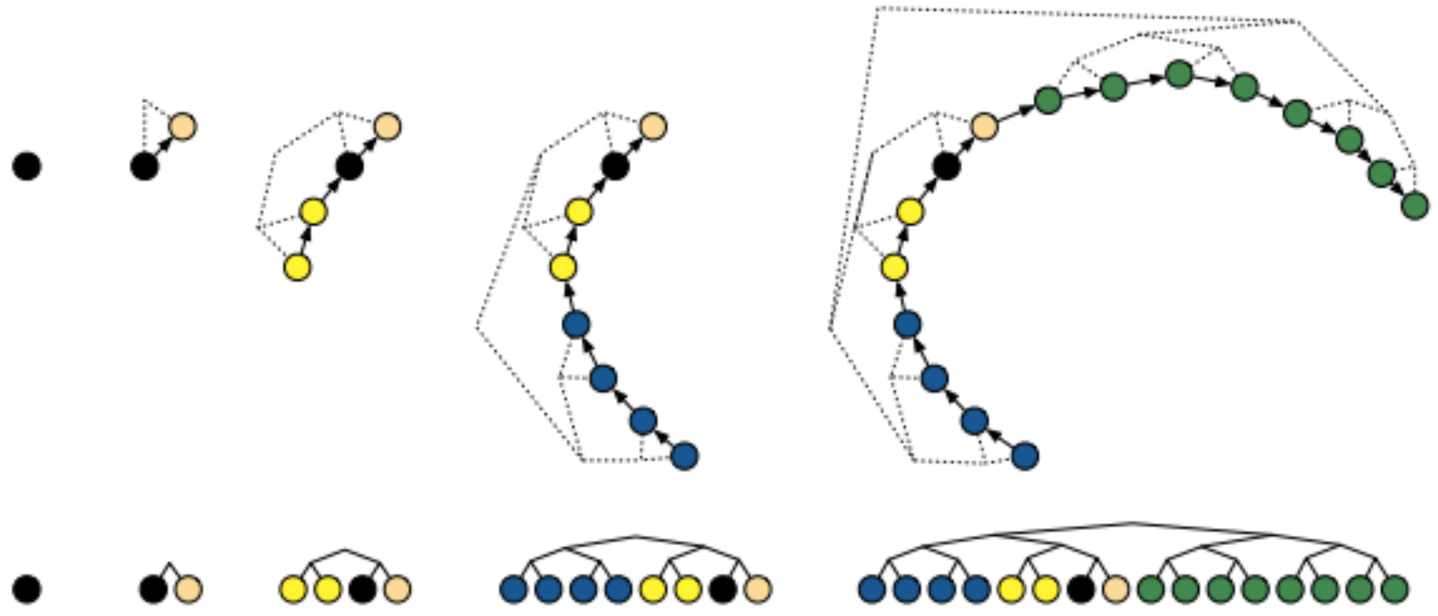
On left, centered, your sampler is not exploring, so make sure what you are diagnosing. On right, nice match!

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



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 usining multinomial or slice sampling

glms

MAXENT

- gaussian likelihood for linear regression maxent choice
- poor choice for constraints such as the outcome being counts, or being only positive.
- use all the information we have about the constraints on an outcome variable to choose a likelihood, typically in the exponential family, that is a maxent distribution.

LINK

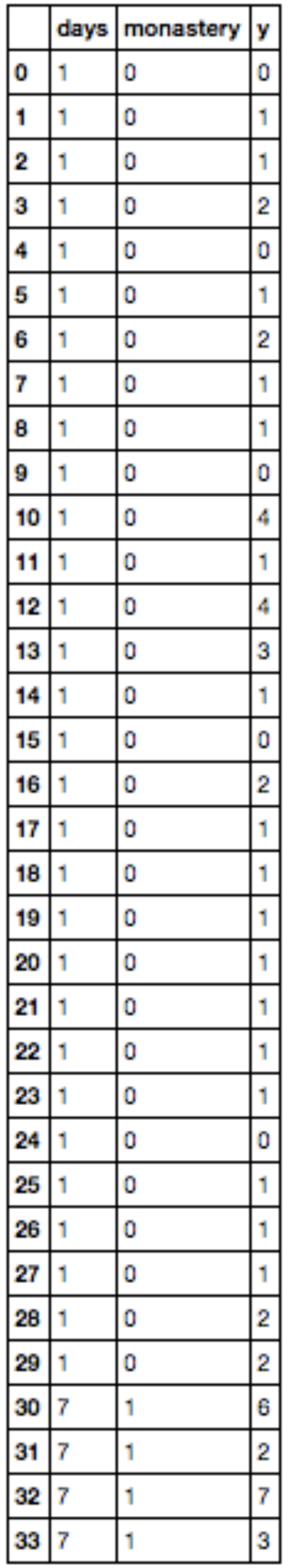
$f(p_i) = \alpha + \beta x_i$ where p_i is the parameter at the ith data point.

Bioassay: f is the logit, and the parameter p_i is the probability in the ith experiment, so that we have

$$logit(p_i) = \alpha + \beta x_i,$$

And where the likelihood used is $Binom(n_i, p_i)$.

For most GLMs, the common links we use are the *logit* link, already used by you in the bioassay Binomial GLM to model the space of probabilities, and the *log* link which you will use here to enforce positiveness on a parameter in poisson regression.



	days	monastery	y
0	1	0	0
1	1	0	1
2	1	0	1
3	1	0	2
4	1	0	0
5	1	0	1
6	1	0	2
7	1	0	1
8	1	0	1
9	1	0	0
10	1	0	4
11	1	0	1
12	1	0	4
13	1	0	3
14	1	0	1
15	1	0	0
16	1	0	2
17	1	0	1
18	1	0	1
19	1	0	1
20	1	0	1
21	1	0	1
22	1	0	1
23	1	0	1
24	1	0	0
25	1	0	1
26	1	0	1
27	1	0	1
28	1	0	2
29	1	0	2
30	7	1	6
31	7	1	2
32	7	1	7
33	7	1	3

$$y_i \sim Poisson(\lambda_i)$$

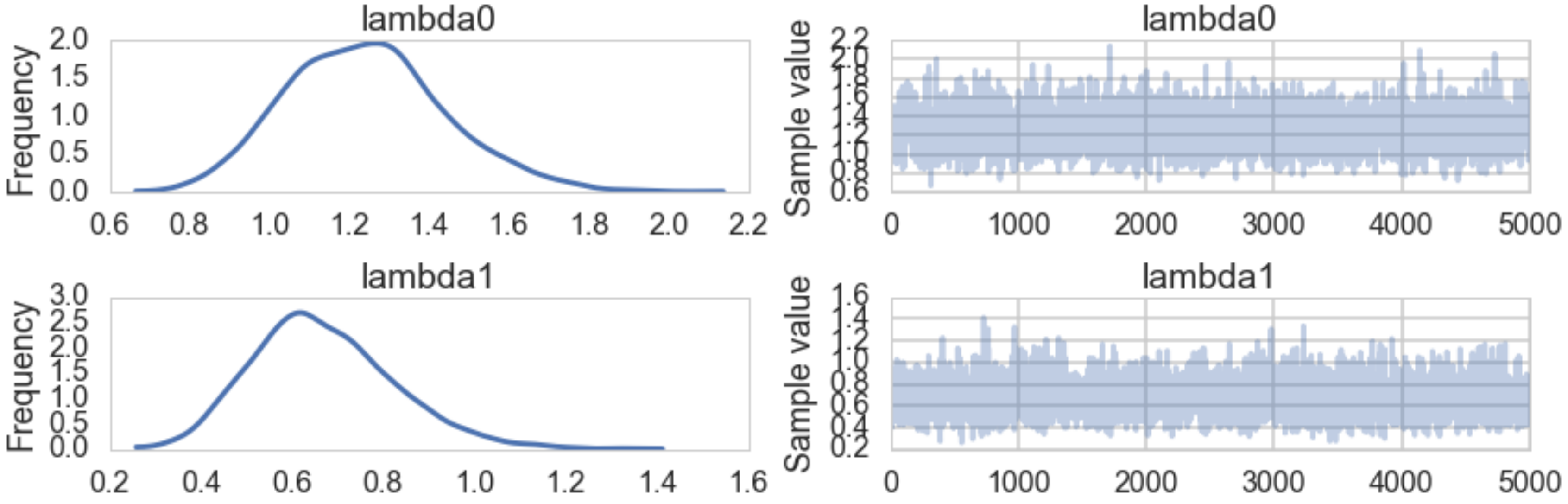
$$log(\lambda_i) = log(\frac{\mu_i}{\tau_i}) = \alpha + \beta x_i$$

λ_i is rate, μ_i is counts, τ_i is exposure.

μ_i or λ_i constrained to be positive.

```
import theano.tensor as t
with pm.Model() as model1:
    alpha=pm.Normal("alpha", 0,100)
    beta=pm.Normal("beta", 0,1)
    logmu = t.log(df.days)+alpha+beta*df.monastery
    y = pm.Poisson("obsv", mu=t.exp(logmu), observed=df.y)
    lambda0 = pm.Deterministic("lambda0", t.exp(alpha))
    lambda1 = pm.Deterministic("lambda1", t.exp(alpha + beta))
```

```
lambda0:

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

  1.243            0.199            0.004            [0.869, 1.635]

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

  0.889           1.100           1.234           1.365          1.671


lambda1:

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

  0.669            0.155            0.003            [0.394, 0.988]

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

  0.407           0.561           0.655           0.765          1.008
```



Posterior Predictive Checking, Informal

Speed of light experiment

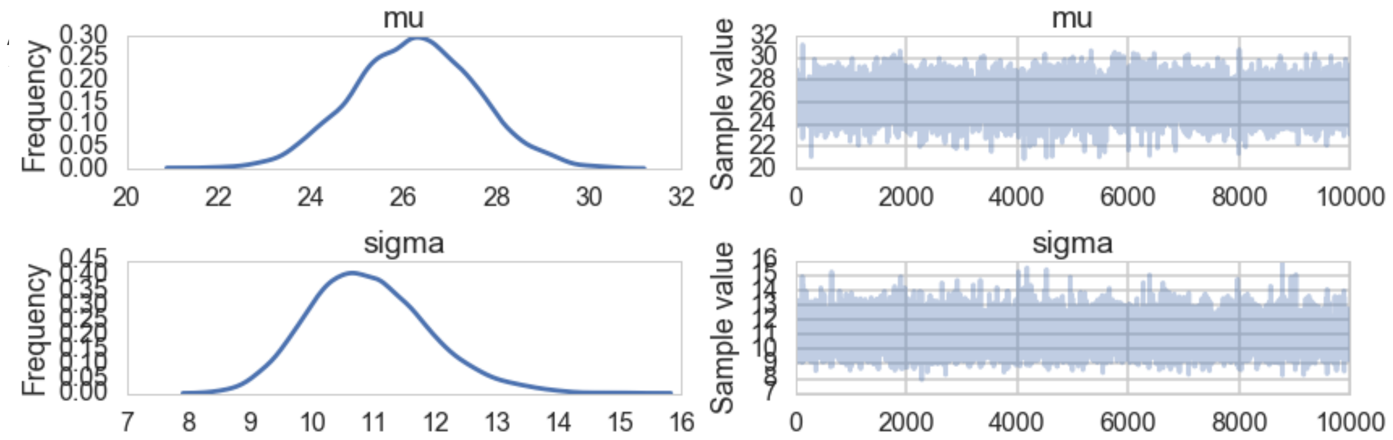
- Simon Newcomb, 1882, times required for light to travel 7442 metres, recorded as deviations from 24,800 nanoseconds

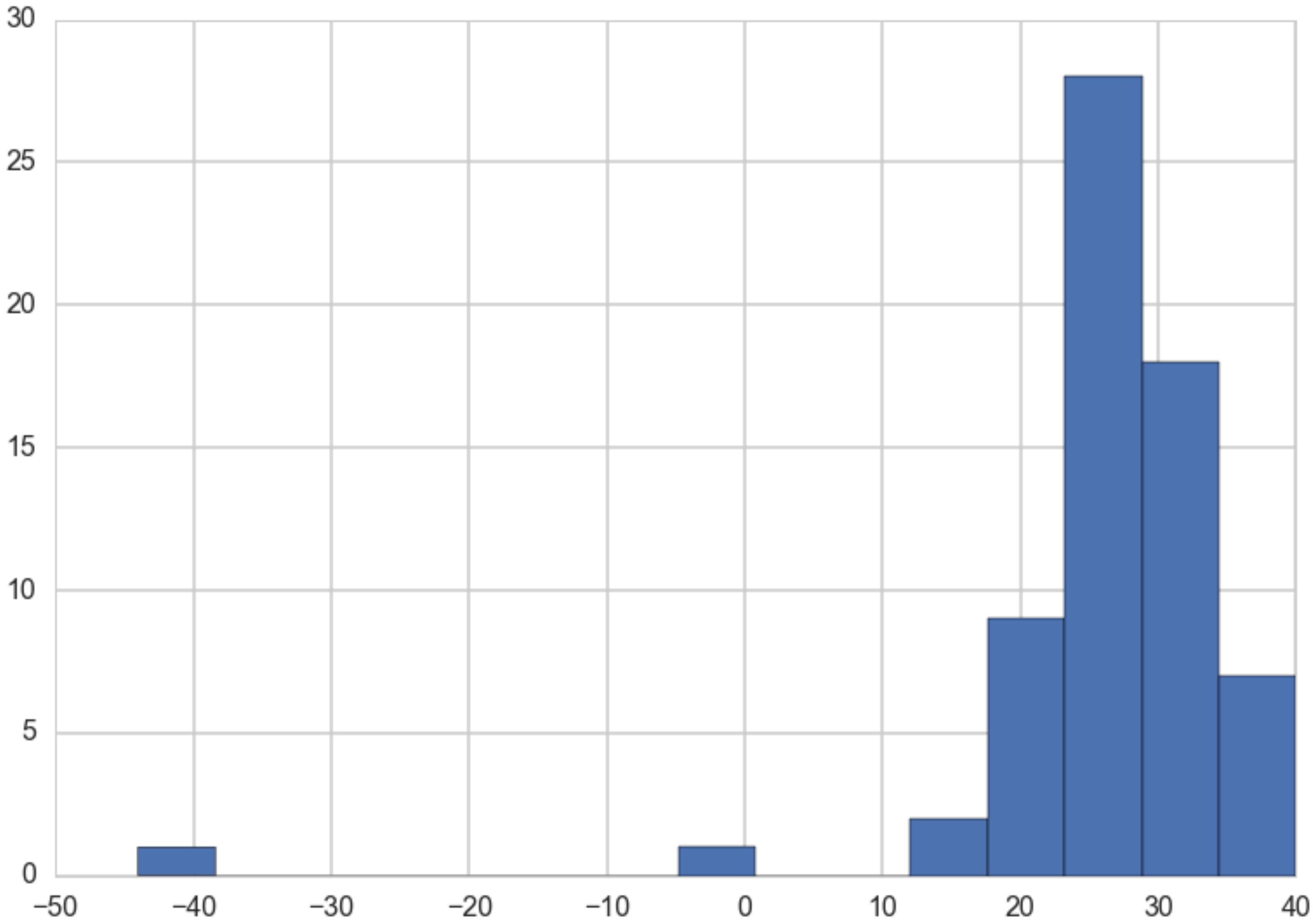
```
light_speed = np.array([28, 26, 33, 24, 34, -44, 27, 16, 40, -2, 29, 22, 24, 21, 25,
                        30, 23, 29, 31, 19, 24, 20, 36, 32, 36, 28, 25, 21, 28, 29,
                        37, 25, 28, 26, 30, 32, 36, 26, 30, 22, 36, 23, 27, 27, 28,
                        27, 31, 27, 26, 33, 26, 32, 32, 24, 39, 28, 24, 25, 32, 25,
                        29, 27, 28, 29, 16, 23])
```

Use Normal model with weakly informative priors to model

```
with pm.Model() as light_model:
    mu = pm.Uniform('mu', lower=-1000,
                    upper=1000.0)
    sigma = pm.Uniform('sigma', lower=0.1, upper=1000.0)
    obsv = pm.Normal('obsv', mu=mu, sd=sigma, observed=light_speed)
```

```
with light_model:
    trace = pm.sample(10000)
```





Some big outliers in data

Multiple replications of the posterior predictive

$$p(\{y^*\}) = \int p(\{y^*\}|\theta)p(\theta|\mathcal{D})d\theta, \text{ observed data: } \mathcal{D} = \{y\}$$

Replicated Data: $\{y_r\}$: data seen tomorrow if experiment replicated with same model and value of θ producing todays data $\{y\}$.

$\{y_r\}$ comes from posterior predictive, and if there are covariates $\{x^*\}$, then $\{y_r\}$ is calculated at those covariates only (`sample_ppc`).

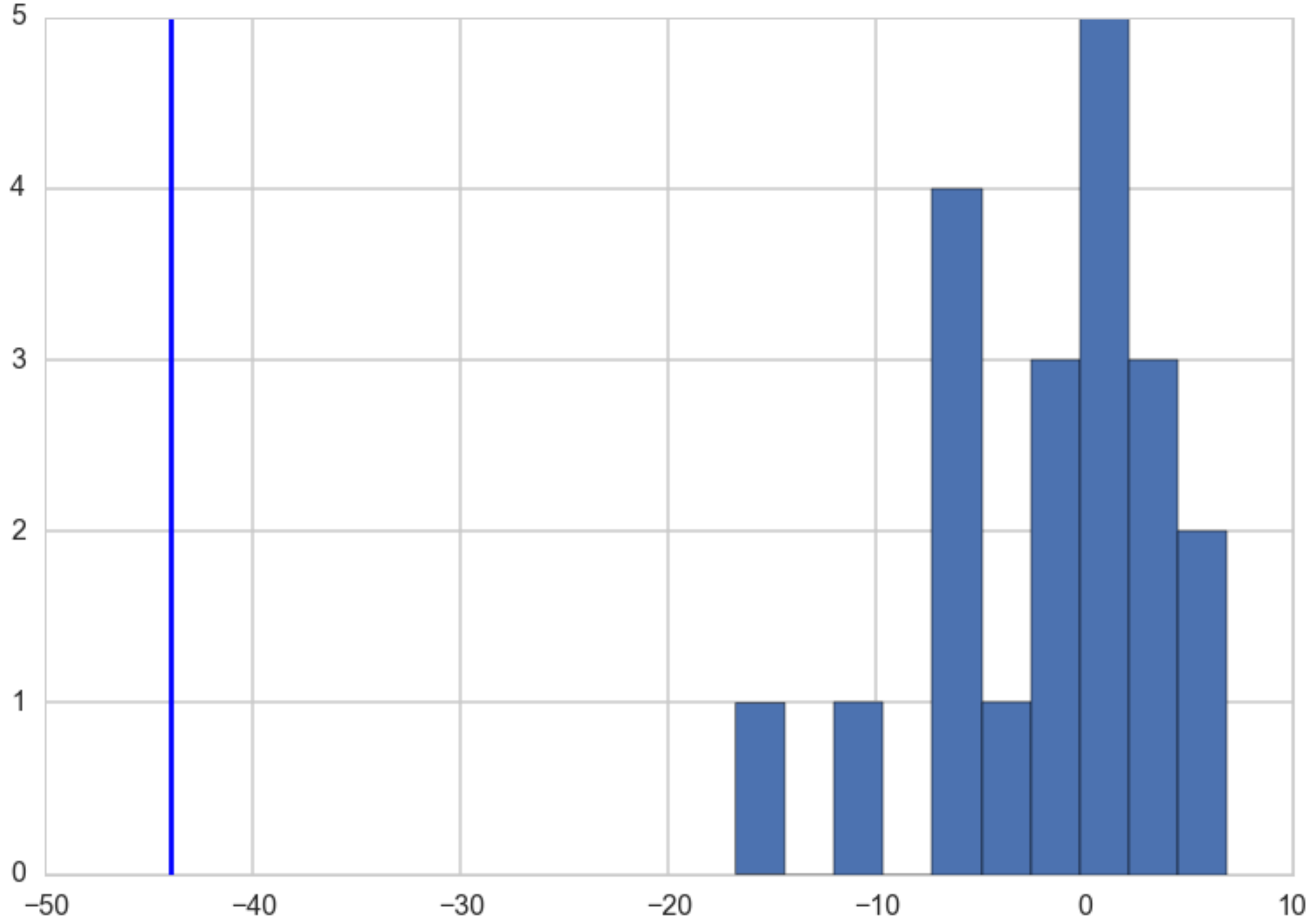
Departure from usual predictive sampling

Sample an entire $\{y_r\}$ at each θ from trace.

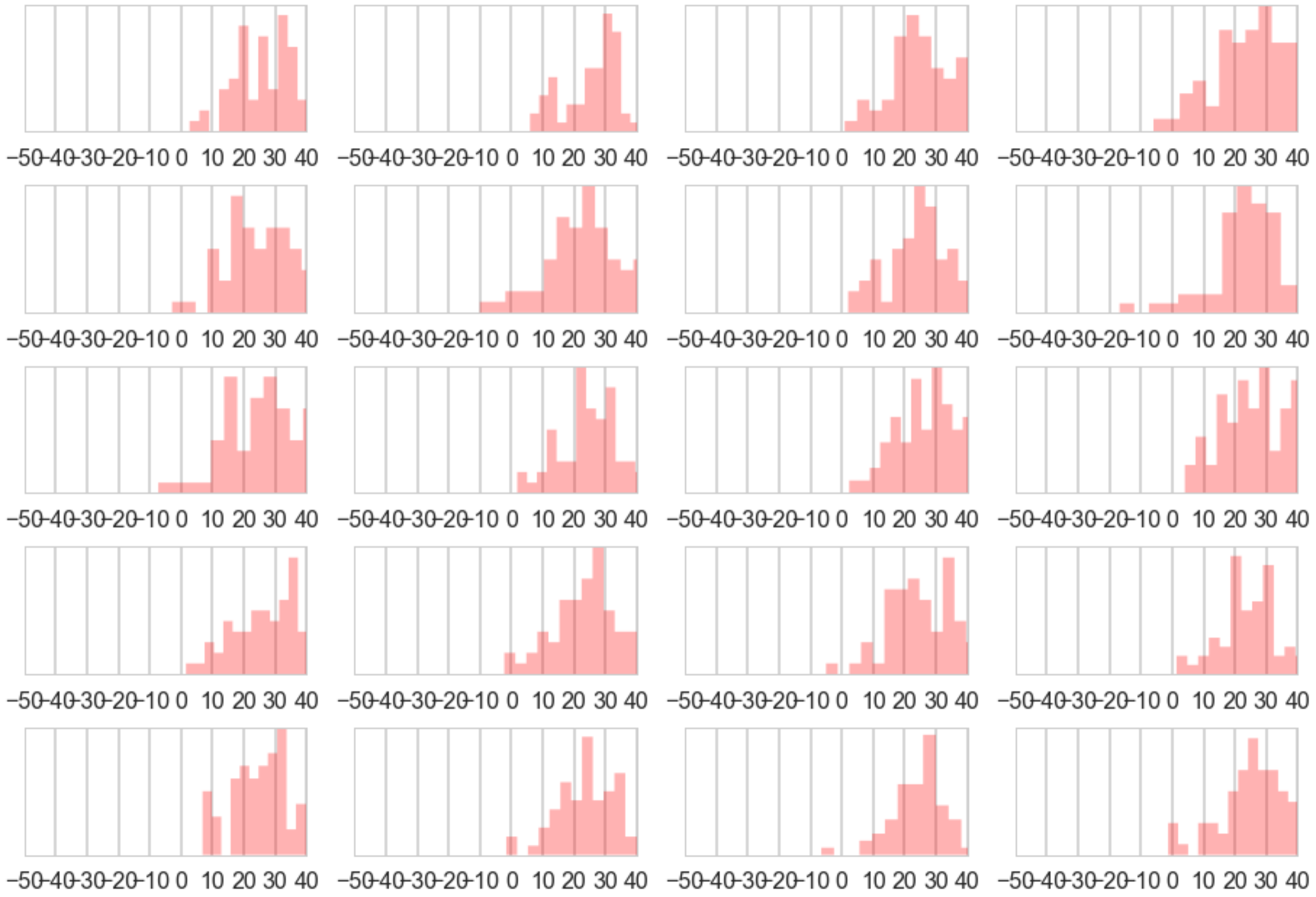
This allows to compute distributions from the posterior predictive replications.

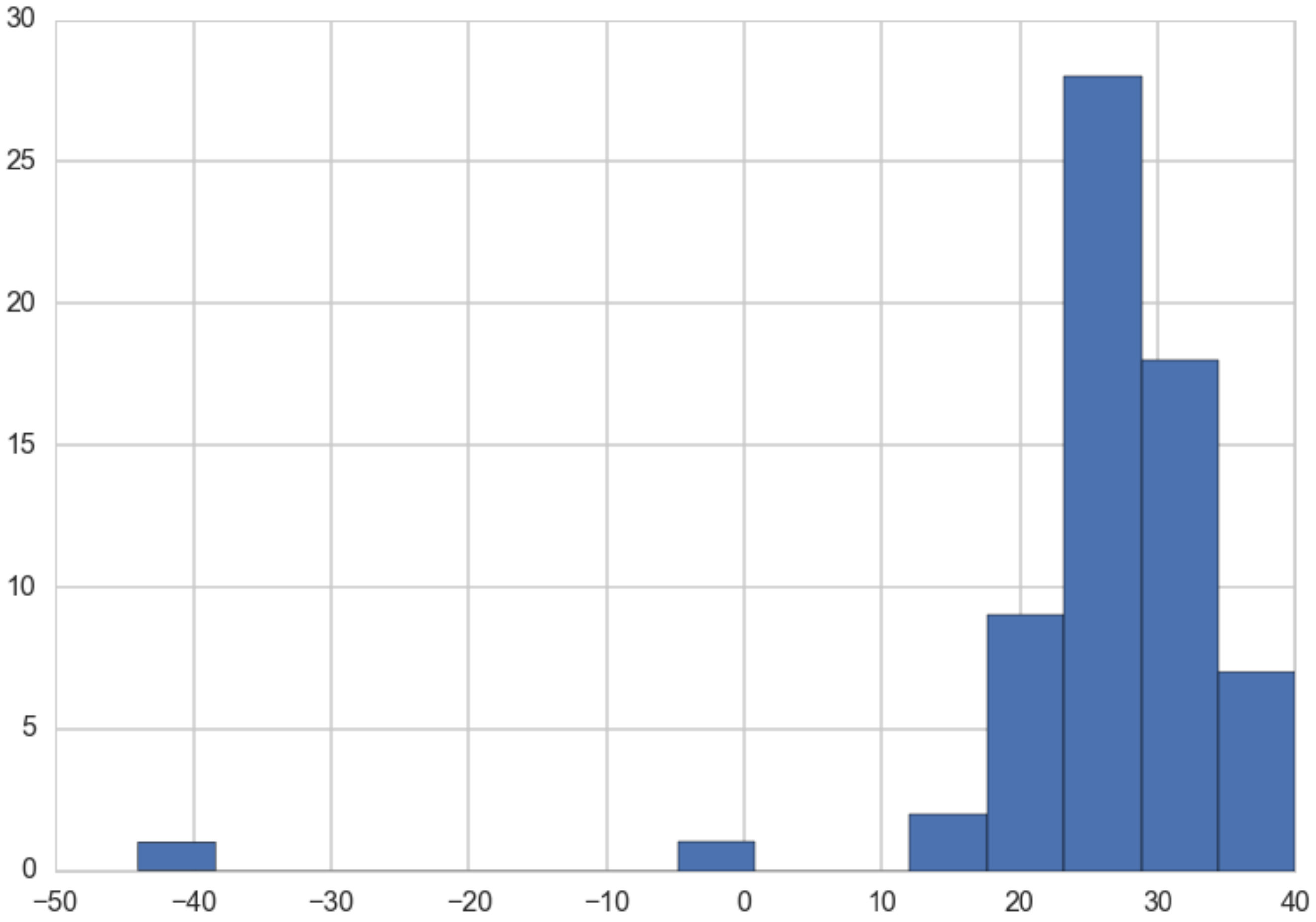
For example the minimum value of speed of light in 20 predictive replications.

An informal test statistic.



Visual Checking





Do these even