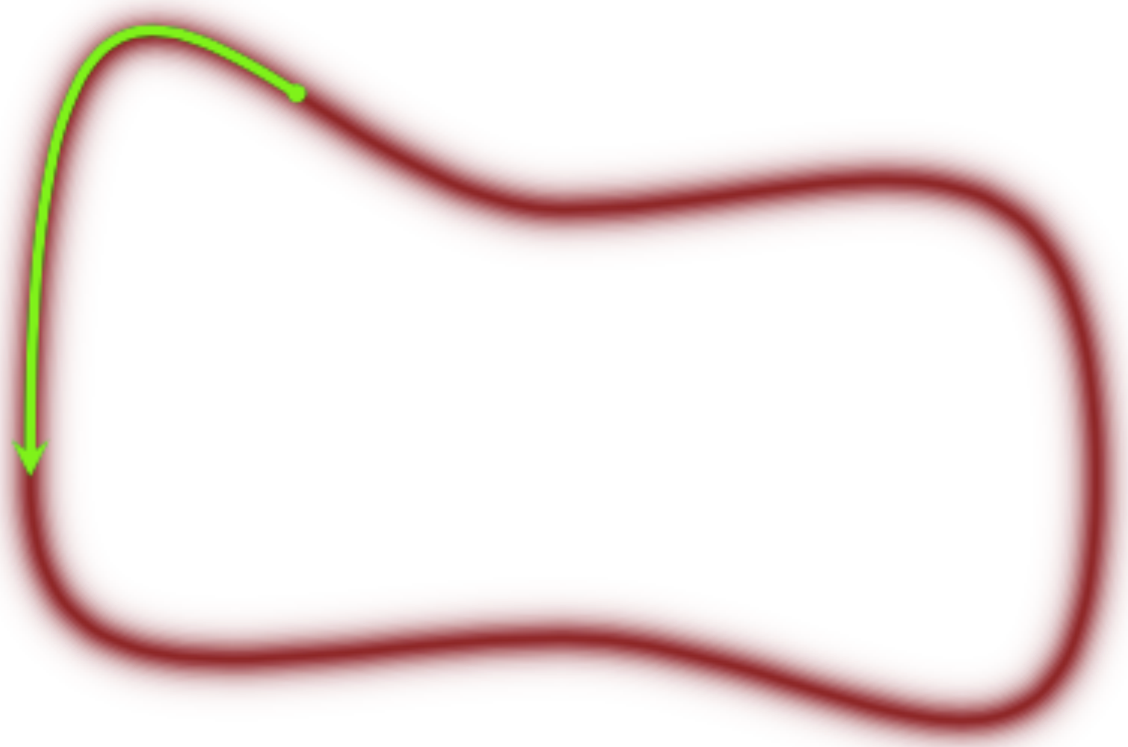
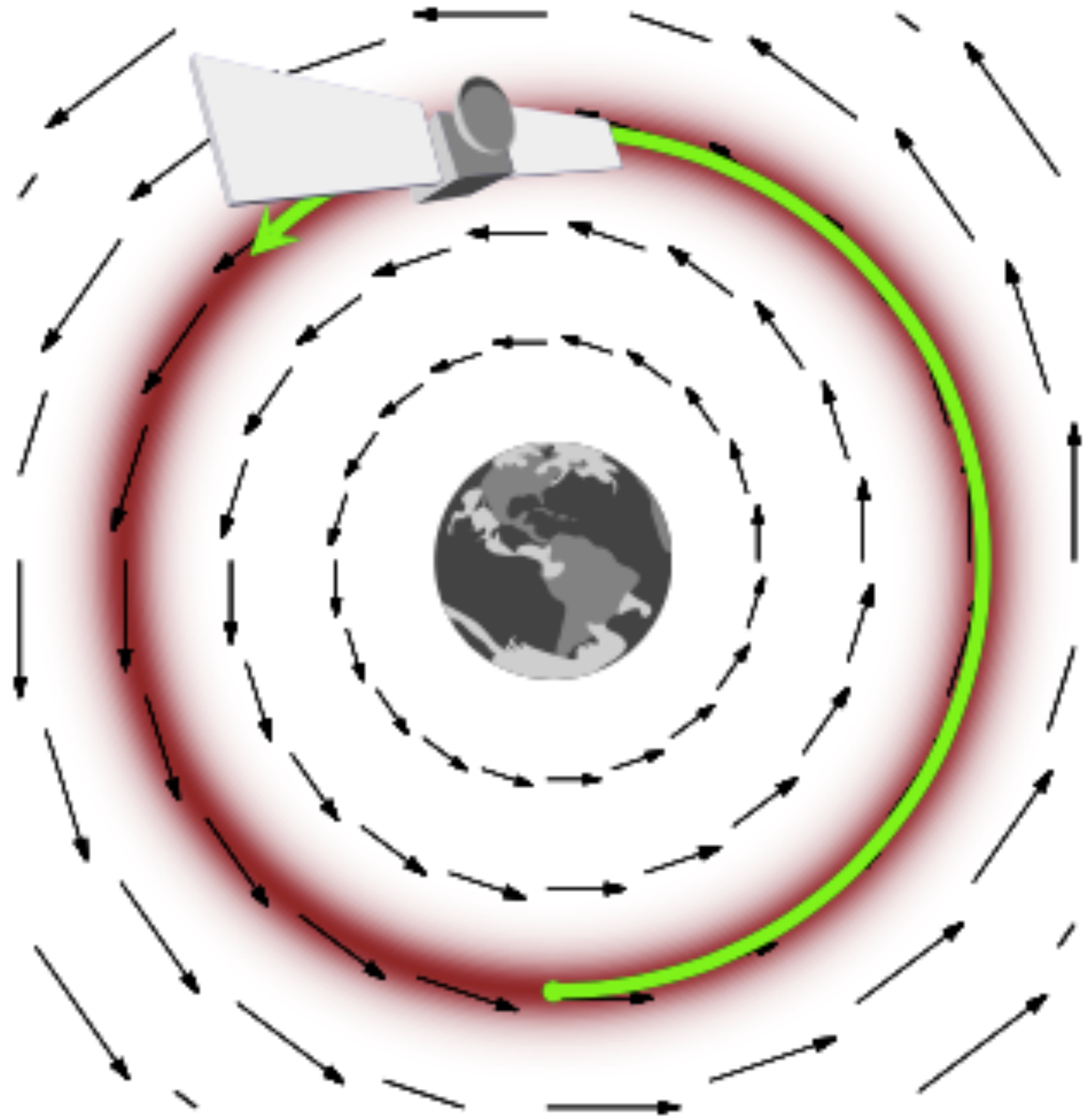
Lecture 17

HMC and

The hierarchical gaussian





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 $p^2/2m$ or more precisely $p^T M^{-1} p$
- write $p(q)$ as $e^{-V(q)}$
- then $p(p, q) = e^{-H(p,q)} = e^{-K(p,q)} e^{-V(q)} = p(p|q)p(q)$

Canonical distribution

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

and thus: $H(p,q) = -log(p(p,q)) = -logp(p|q) - logp(q)$

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

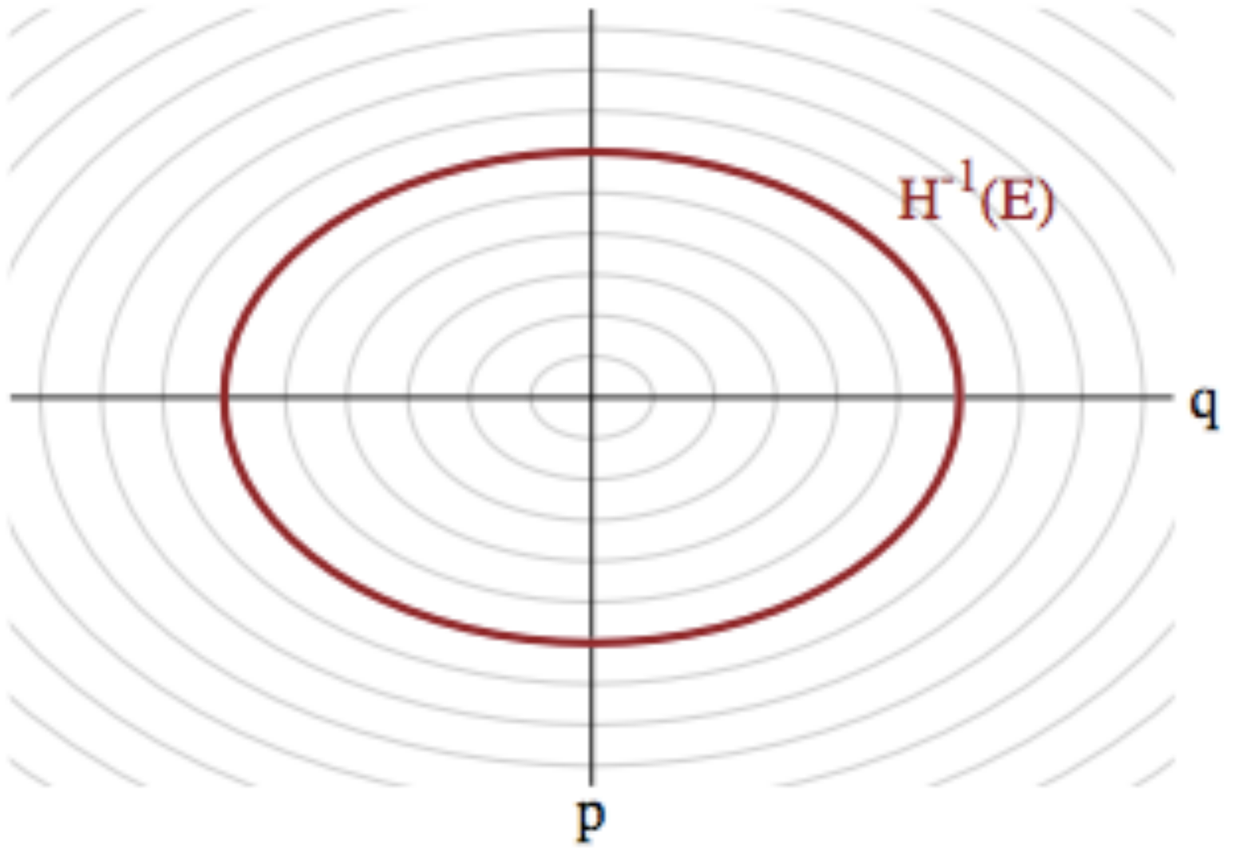
Phase Space level sets: Microcanonical Distribution

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

The energy **Hamiltonian**

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

with E_i constants (constant energies) for each level-set foliate and where the **potential energy** $V(q) = -log(p(q))$ replaces the energy term we had earlier in simulated annealing.



Microcanonical distribution: states for given energy.

Time implicit H: flows **constant energy, vol preserving, reversible**.

The canonical distribution can be written as a product of this microcanonical distribution and a **marginal energy distribution**:

$$p(q,p) = p(\theta_E|E)p(E)$$

where θ_E indexes the position on the level set.

Also need to sample **Marginal Energy Distrib**: probability of level set in the typical set.

Hamiltonian Mechanics

Physics equations of motion in the **Hamiltonian Formalism** set up the "glide" (over a level set).

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q}, \implies , \frac{dq}{dt} = \frac{\partial H}{\partial p}$$

Time independence: $\frac{\partial H}{\partial t} = 0 \implies \frac{dH}{dt} = 0\,,$

$H(t + \Delta t) = H(t)\, \forall t.$

Reversibility

T_s from $(q, p) \rightarrow (q', p')$ to a "later" time $t' = t + s$. Mapping is 1-1, inverse T_{-s}. This can be obtained by simply negating time:

$$\frac{dp}{d(-t)} = -\frac{\partial H}{\partial q}$$

$$\frac{dq}{d(-t)} = \frac{\partial H}{\partial p}$$

Superman transform

If we then transform $p \rightarrow -p$, we have the old equations back:

$$\frac{d(-p)}{d(-t)} = -\frac{\partial H}{\partial q}$$
$$\frac{dq}{d(-t)} = \frac{\partial H}{\partial(-p)}$$

To reverse T_s, flip the momentum, run Hamiltonian equations until you get back to the original position and momentum in phase space at original time t, then flip the momentum again so it is pointing in the right direction.

Volume in phase space is conserved

Jacobian:

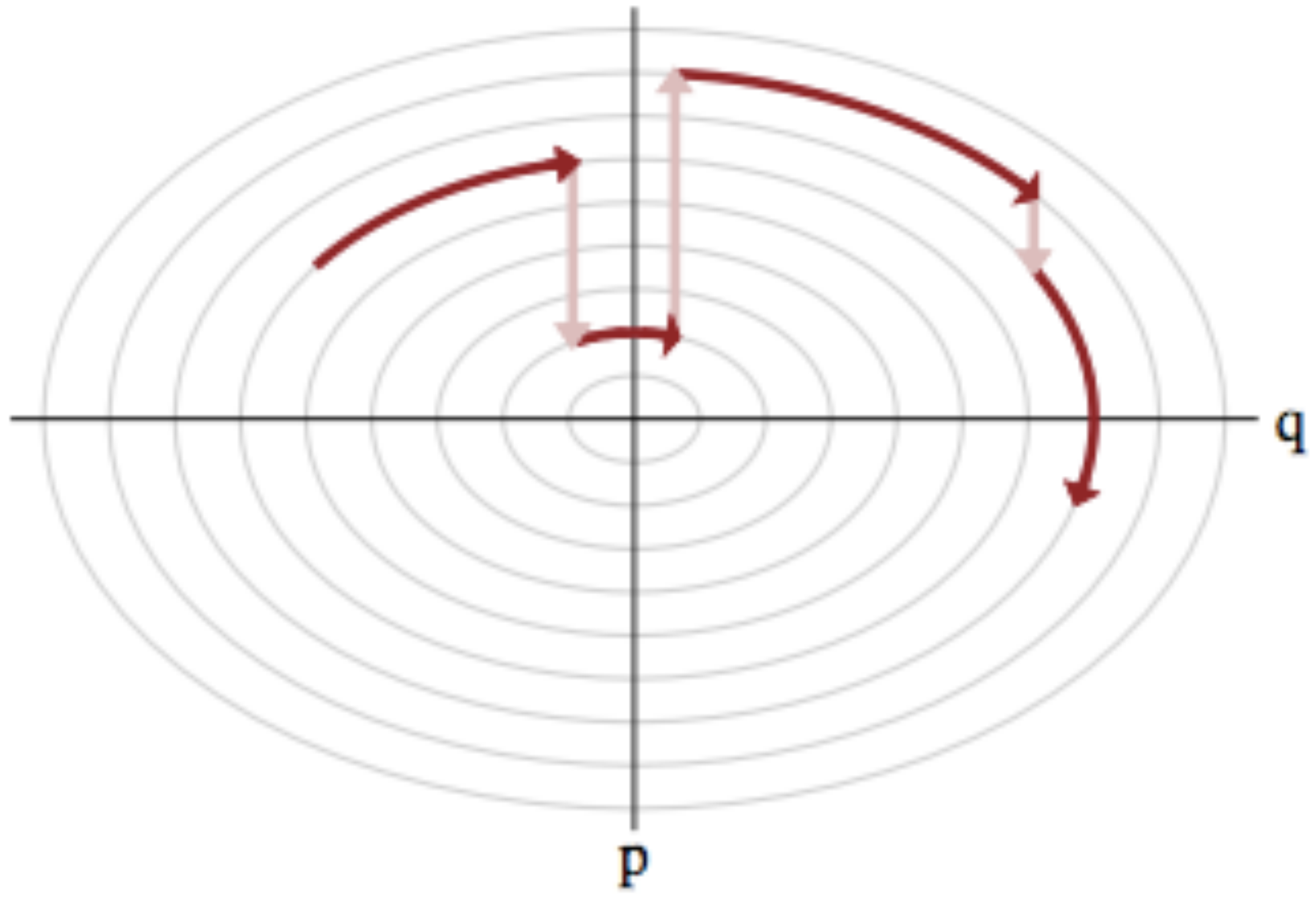
$$det\left(\begin{bmatrix} 1+\delta\frac{\partial^2 H}{\partial q \partial p} & \delta\frac{\partial^2 H}{\partial p^2} \\ \delta\frac{\partial^2 H}{\partial q^2} & 1-\delta\frac{\partial^2 H}{\partial p \partial q} \end{bmatrix}\right) = 1 + O(\delta^2)$$

As a result of this, the momenta we augment our distribution with must be **dual** to our pdf's parameters, transforming in the opposite way so that phase space volumes are invariant.

Summary

- 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 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.

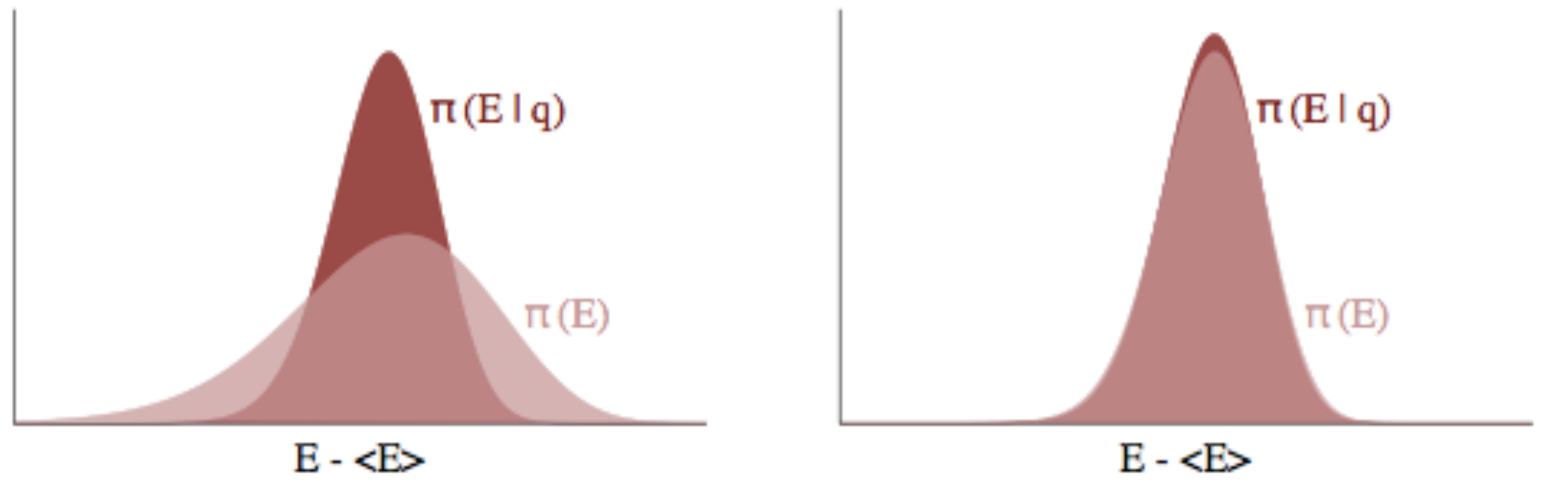
Firing the thruster moves us between level sets!

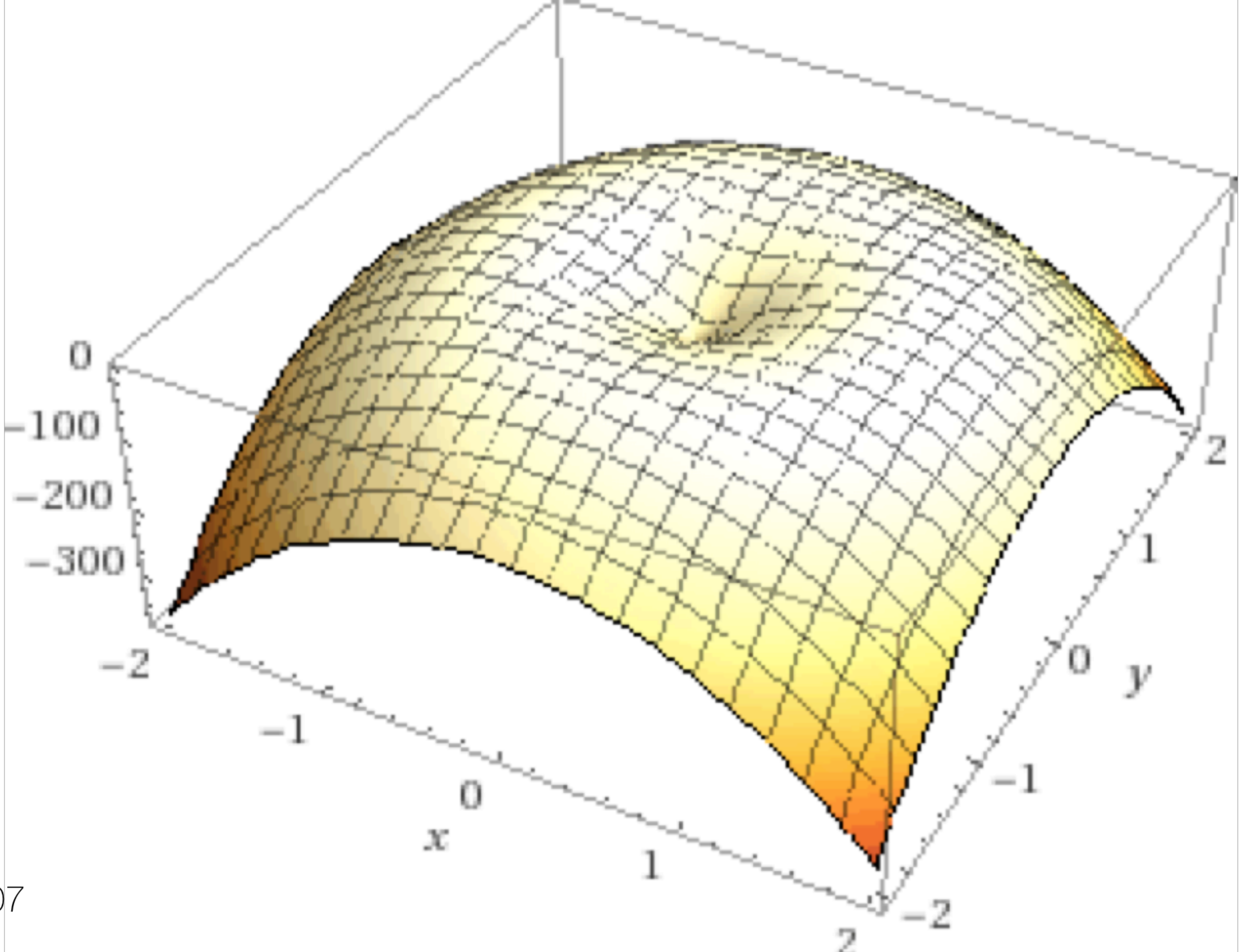
Resampling Efficiency

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.

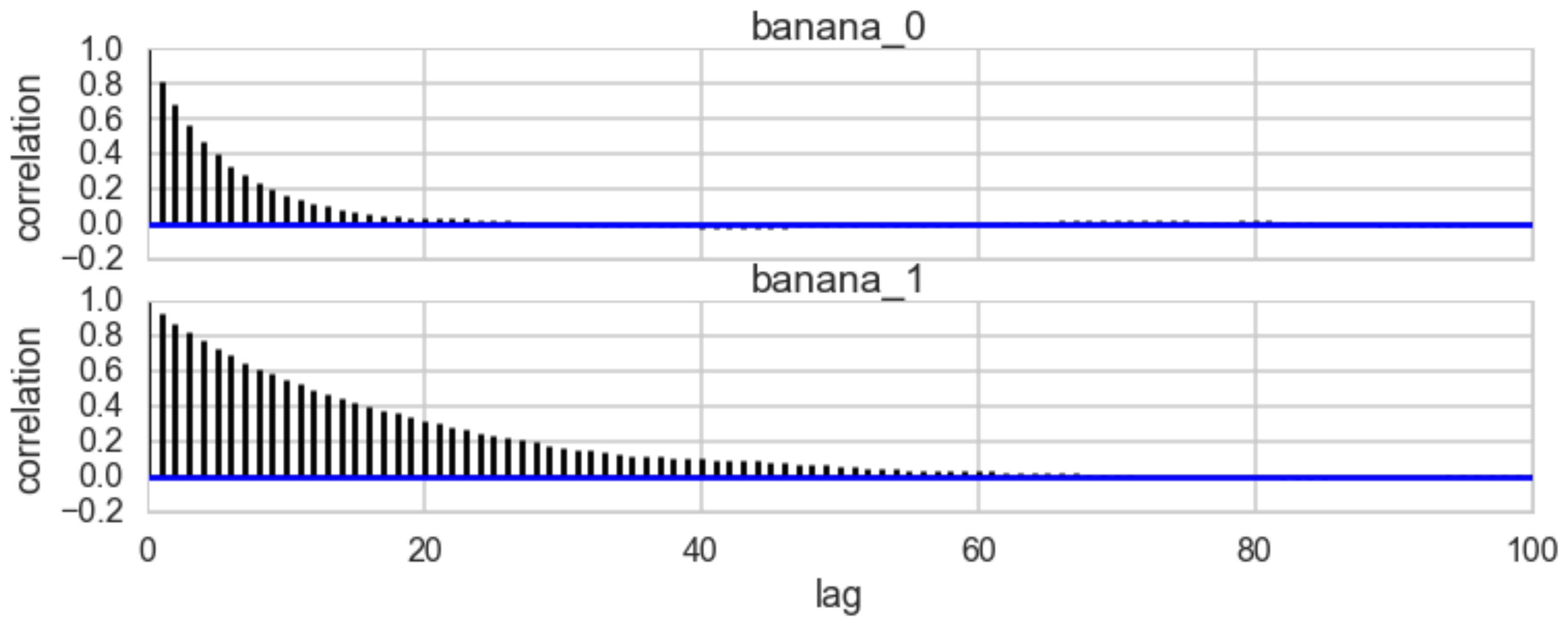
If $p(E|q)$ matches $p(E)$: independent samples generated from the marginal energy distribution very efficiently.

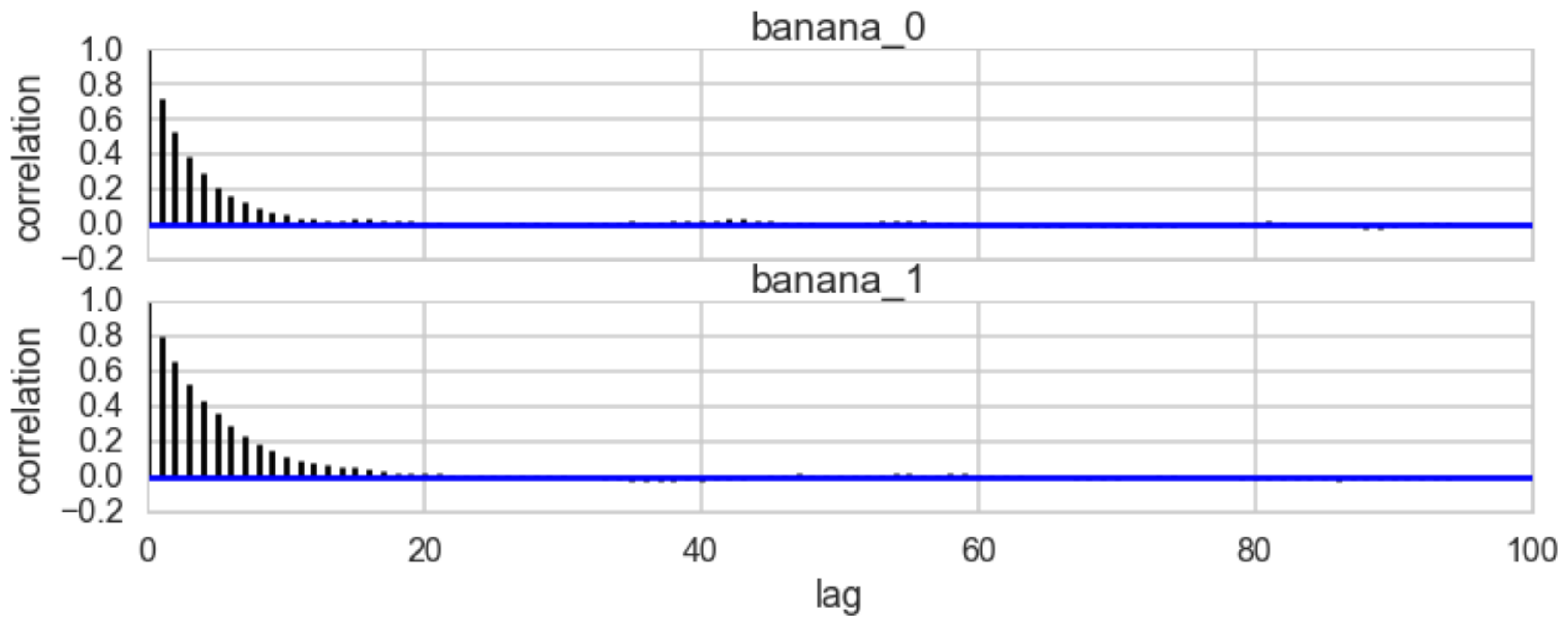



0
−100
−200
−300
−2
−1
0
x
1
2
−2
−1
0
y
1
2

HMC/NUTS in pymc3

```
def clike2(value):
    x = value[0]
    y = value[1]
    val = -100 * (T.sqrt(y**2+x**2)-1)**2 + (x-1)**3 - y -5
    return (val)

with pm.Model() as model:
    banana = pm.DensityDist("custom", clike2, shape=2, testval=[1,1])

with model:
    start = pm.find_MAP()
    stepper=pm.Metropolis()
    trace=pm.sample(100000, step=stepper, start=start)
pm.autocorrplot(trace[20000::5])

with model:
    stepper_nuts=pm.NUTS()
    trace_nuts=pm.sample(100000, step=stepper_nuts)
pm.autocorrplot(trace_nuts[:16000])
```





Tuning: choice of Kinetic energy

- Ideal kinetic energy: microcanonical exploration easy and uniform, marginal exploration matched by the transition distrib.
- In practice we often use $K(p) = \frac{1}{2} p' M^{-1} p = \sum_i p_i^2 / 2m_i$
- Set M^{-1} to the covariance of the target distribution: maximally de-correlate the target. Do in warmup (tune) phase.
- can see this by $p \rightarrow p/\sqrt{M}$, Then $q \rightarrow q\sqrt{M}$

See this for Gaussian:

$$H = \frac{1}{2}p' M^{-1} p + \frac{1}{2}q' \Sigma^{-1} q$$

On transformation

$$H = \frac{1}{2}(p'p + q'q) \text{ if } M^{-1} = \Sigma$$

Thus de-correlate target.

Generalize to arbitrary distributions.

Tuning: integration time

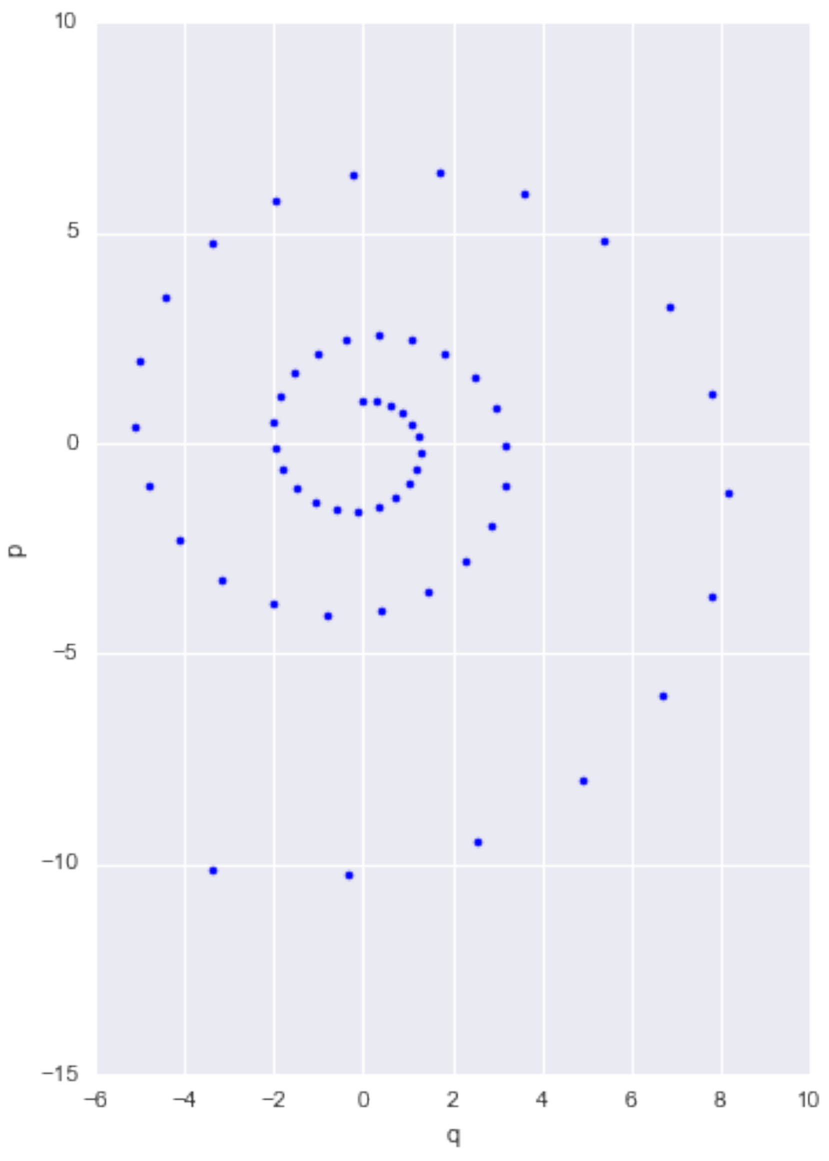
- whats the best integration time?
- should we glide for a long time? then we wont get too may samples
- if our integration was exact we could glide for arbitrary short times
- but integration is not exact and will infact take us off the level set
- thus too many samples/too short time will get us back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

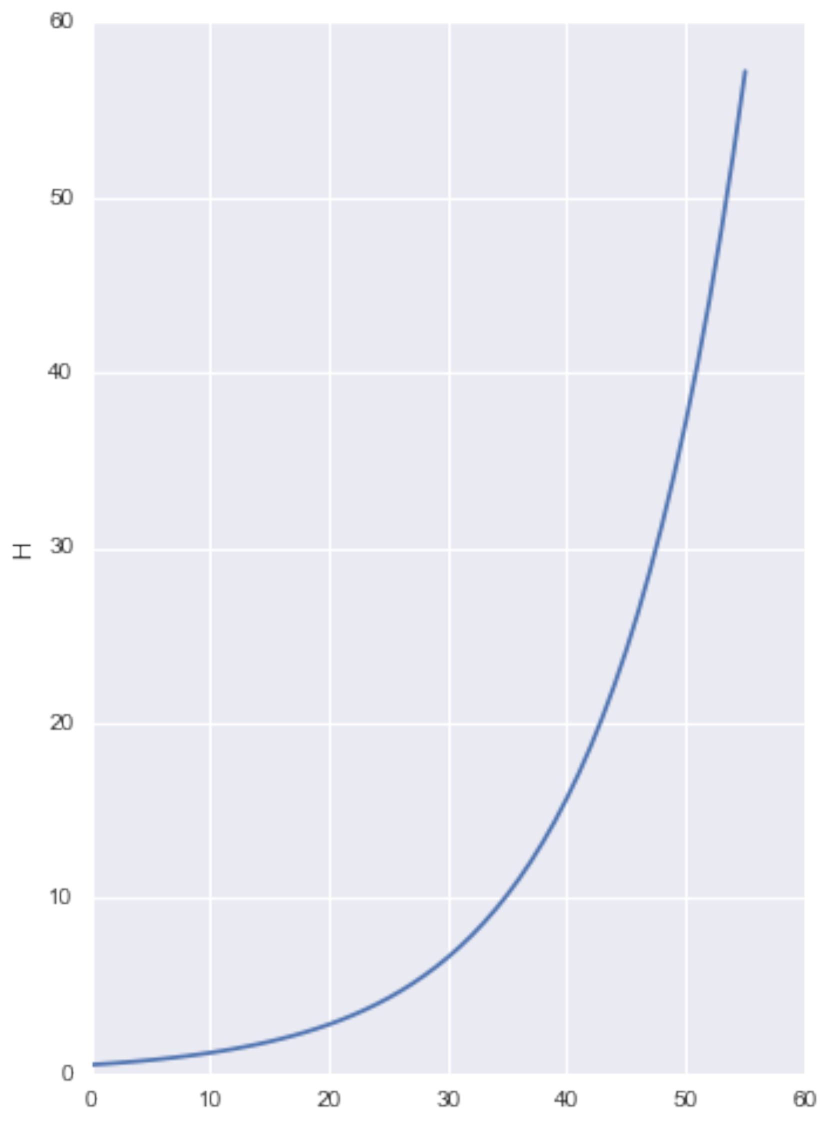
Tuning: integration time

- 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.
- generally static not good, under-samples tails (high-energy micro-canonicals). Estimate dynamically: NUTS (pymc3 and Stan)

Problems

- discretization to solve differential equations and the need for symplecticity
- lack of reversibility even with symplecticity (we are marginally off the level set)

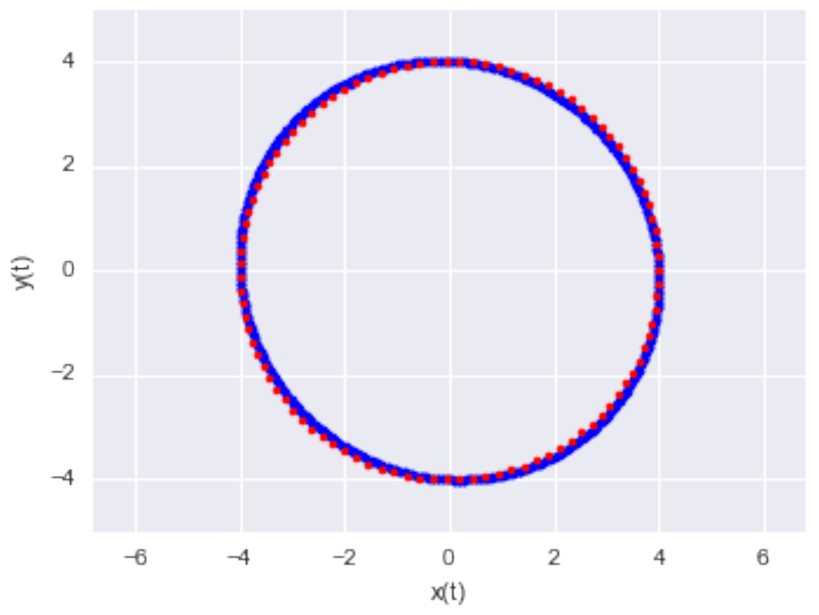


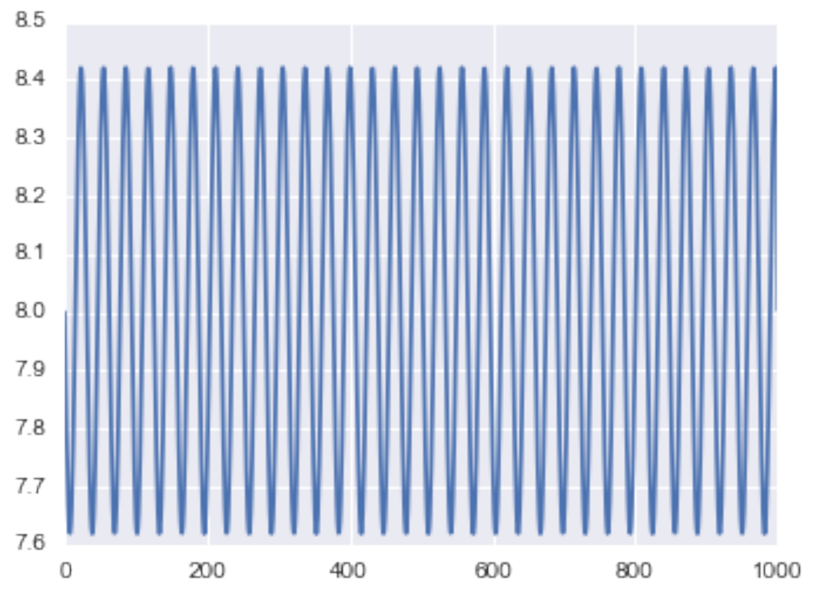


Practical implementation: Discretization and our problems

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

Sympletic Leapfrog



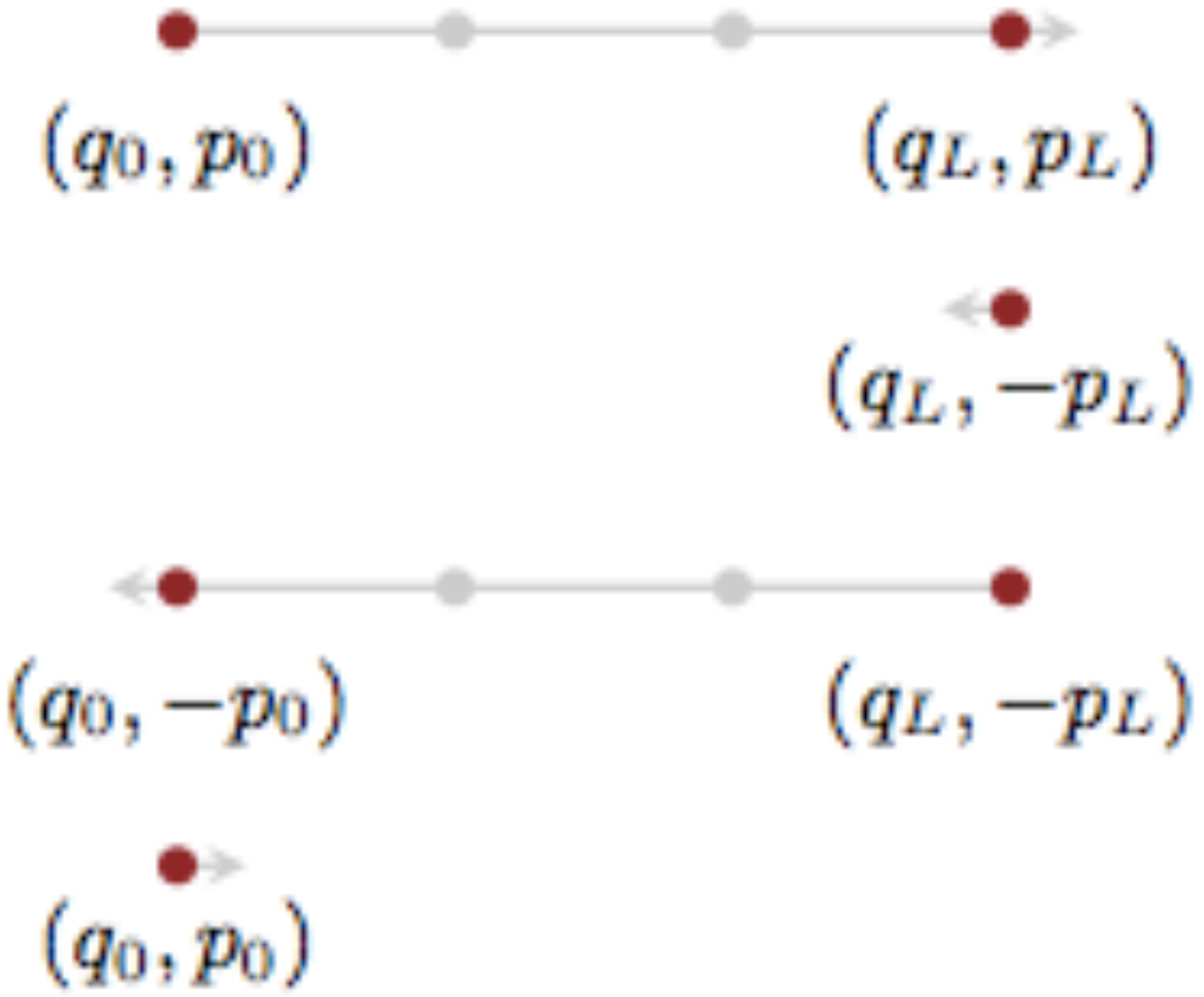


- Only *shear* transforms allowed, will preserve volume.
- $p_i(t + \frac{\epsilon}{2}) = p_i(t) - \frac{\epsilon}{2} \frac{\partial V}{\partial q_i}|_{q(t)}$
- $q_i(t + \epsilon) = q_i(t) + \epsilon \frac{p_i(t + \frac{\epsilon}{2})}{m_i}$
- $p_i(t + \epsilon) = p_i(t + \frac{\epsilon}{2}) - \frac{\epsilon}{2} \frac{\partial V}{\partial q_i}|_{q(t+\epsilon)}$
- still error exists, oscillatory, so reversibility not achieved

Acceptance probability

- might choose $Q(q',p'|q,p) = \delta(q' - q_L)\delta(p' - p_L)$.
- but small symplectic errors means this is only forward in time
- tack on sign change $(q,p) \to (q_L, -p_L)$. Superman to the rescue!
- proposal now: $Q(q',p'|q,p) = \delta(q' - q_L)\delta(p' + p_L)$.
- Acceptance: $A = \min[1, \frac{p(q_L,-p_L)\delta(q_L - q_L)\delta(-p_L + p_L)}{p(q,p)\delta(q-q)\delta(p-p)}]$

- thus:
  $A = \min[1, \exp(-U(q_L) + U(q) - K(p_L) + K(p)]$
- critical thing with HMC is that our **time evolution is on a level set**. So our A always closer to 1, and we have a very efficient sampler. Optimal acceptance can be shown: 65% roughly.
- momentum reversal could be left out if not within a more complex sampling scheme



Stationarity

- want canonical distribution as stationary distribution
- partition phase space into small regions A_k each with small volume V. Let the L leapfrog step image of A_k be B_k
- Detailed Balance: $P(A_i)T(B_j \mid A_i) = P(B_j)T(A_i \mid B_j)$
- $T(X|Y)$ is the conditional probability of proposing and then accepting a move to region X if the current state is in region Y .

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

Stationarity Proof

The probability of the next state being in B_k:

$$P(B_k)R(B_k) + \sum_i P(A_i)T(B_k \mid A_i) = P(B_k)R(B_k) + \sum_i P(B_k)T(A_i \mid B_k)$$

$$= P(B_k)R(B_k) + P(B_k)\sum_i T(A_i \mid B_k)$$

$$= P(B_k)R(B_k) + P(B_k)(1 - R(B_k)) = P(B_k)$$

Ergodicity

- as long as we have no cycles we are good, the hamiltonian flow with momentum resampling will ensure ergodicity
- but if $L\epsilon = 2\pi$ (for oscillator) can get into trouble
- near ergodicty can lead to a bad sampler
- having chosen one, choose the other from a fairly small interval to fix
- in practice not a big problem
- dynamic ergodicity important for sampling efficiency

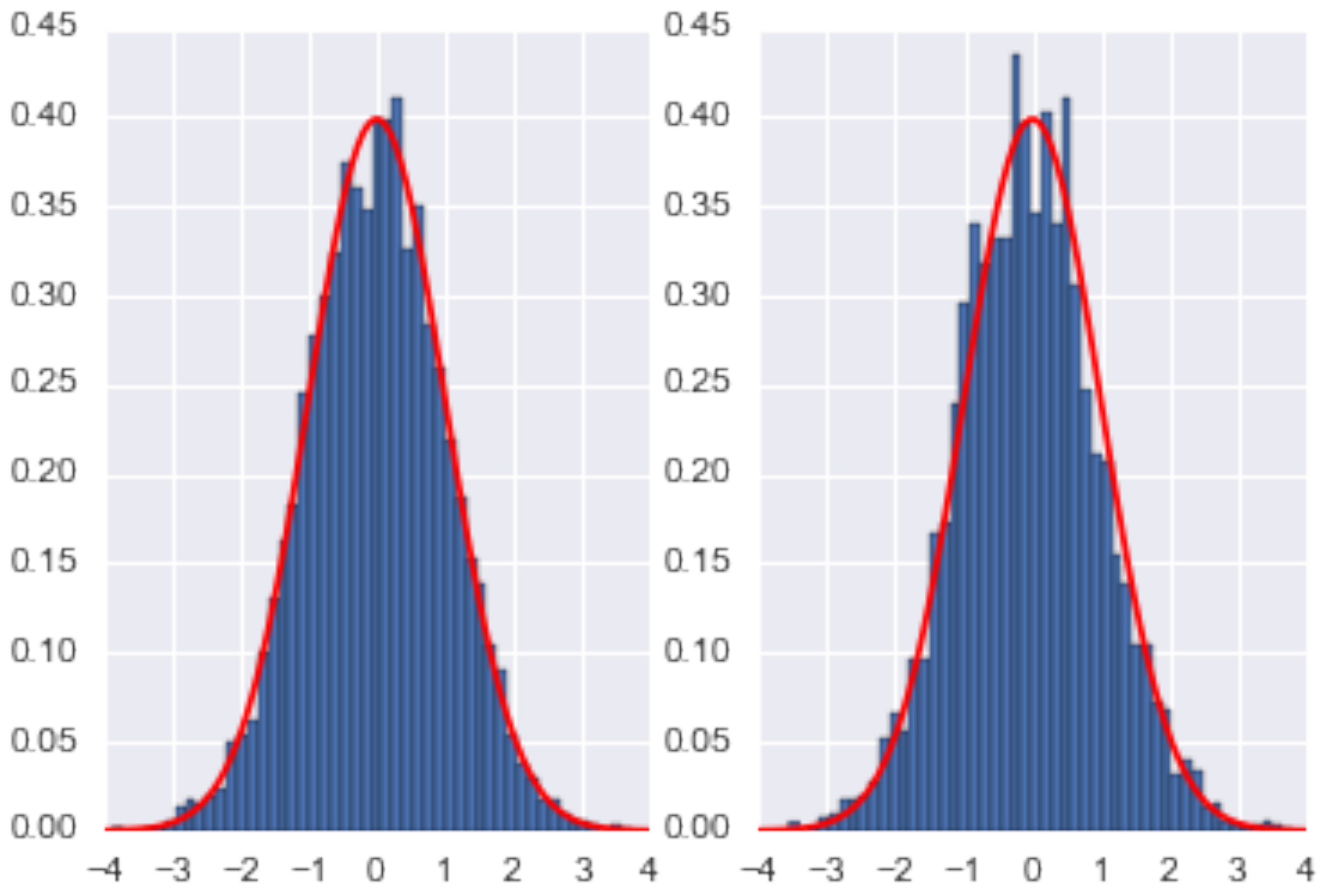
HMC Algorithm

- 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: $p = p - \frac{\epsilon \nabla U(q)}{2}$

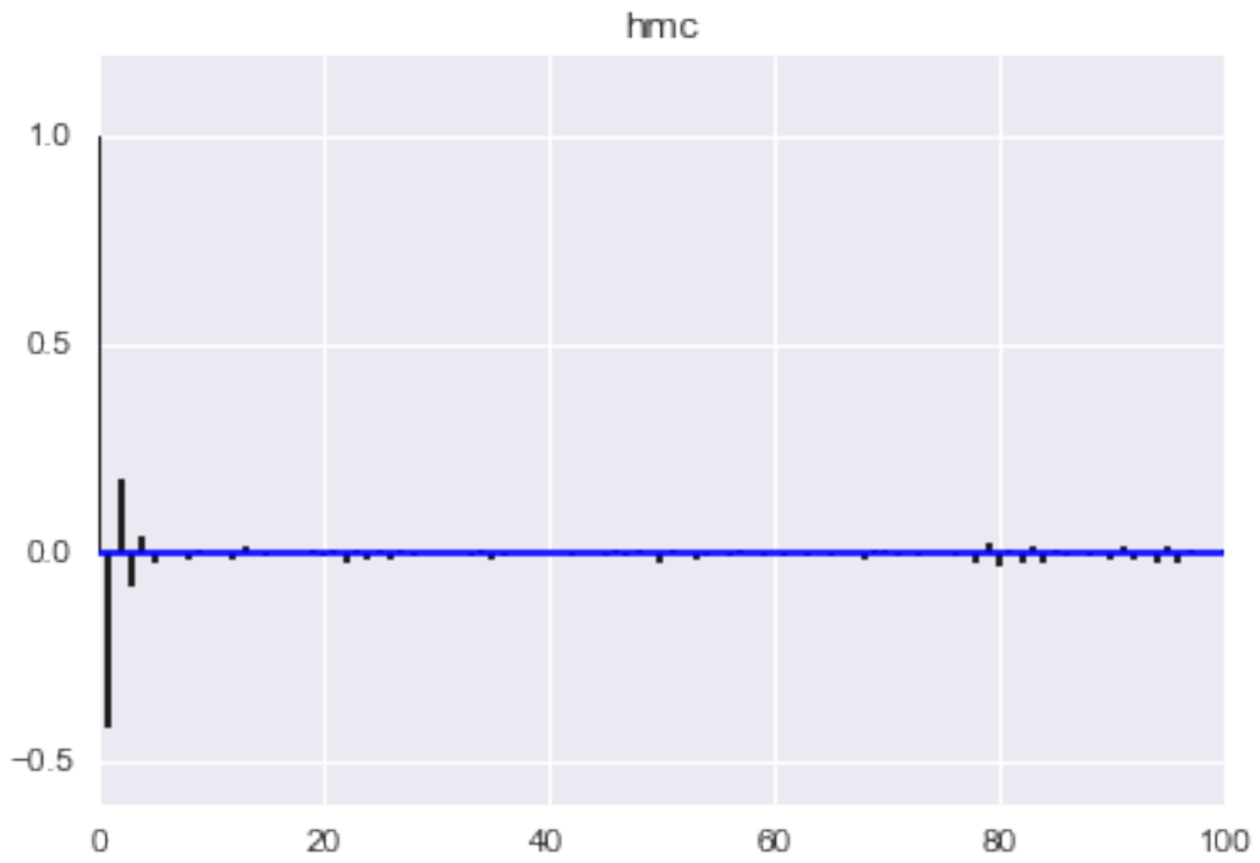
HMC (contd)

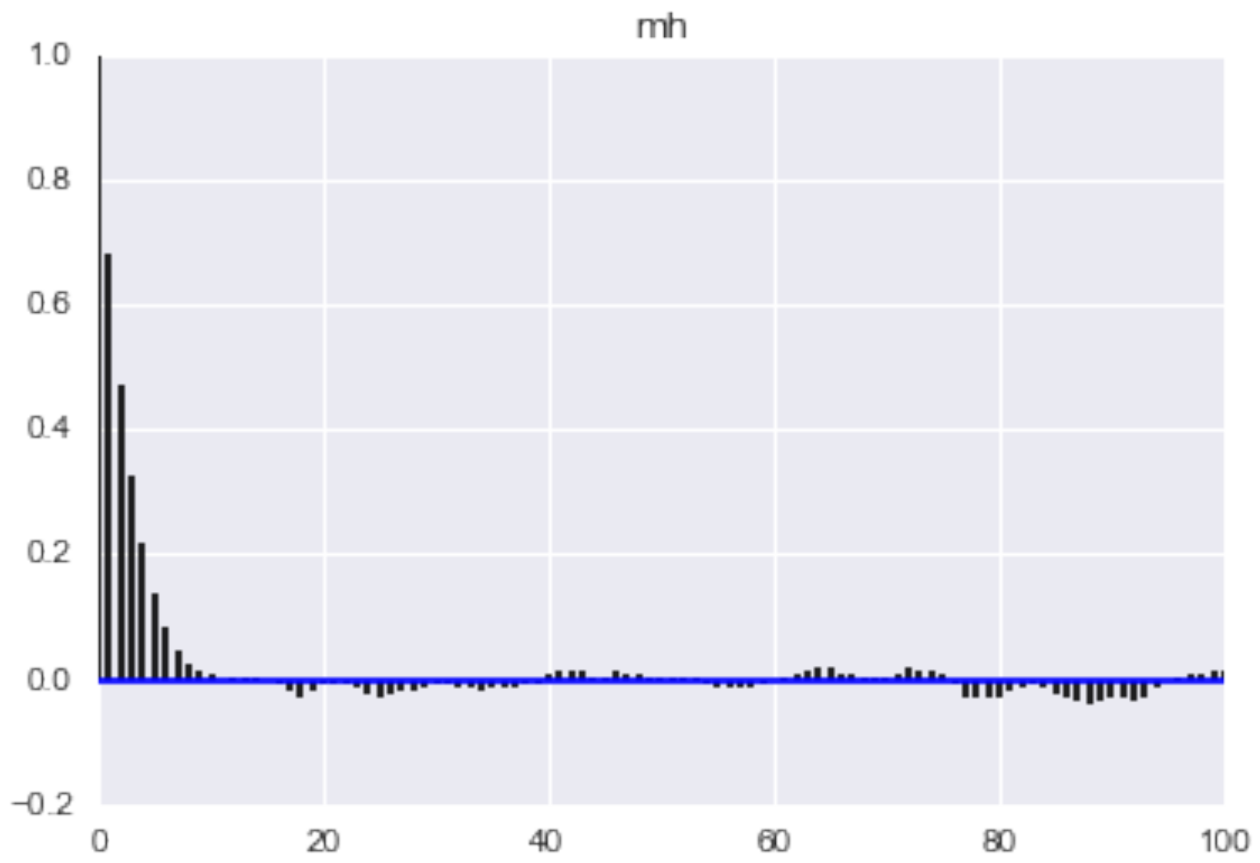
- for i=1:N_samples
  - 7. $p^* = -p$
  - 8. $V_c = V(q_c), \;\; K_c = \frac{p_c^\top M^{-1} p_c}{2}$
  - 9. $V^* = V(q^*), \;\; K^* = \frac{p^{\top *} M^{-1} p^*}{2}$
  - 10. $r \sim \mathrm{Unif}(0, 1)$
  - 11. if $r < e^{(U_c - U^* + K_c - K^*)}$
    - accept $q_i = q^*$
    - otherwise reject

```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





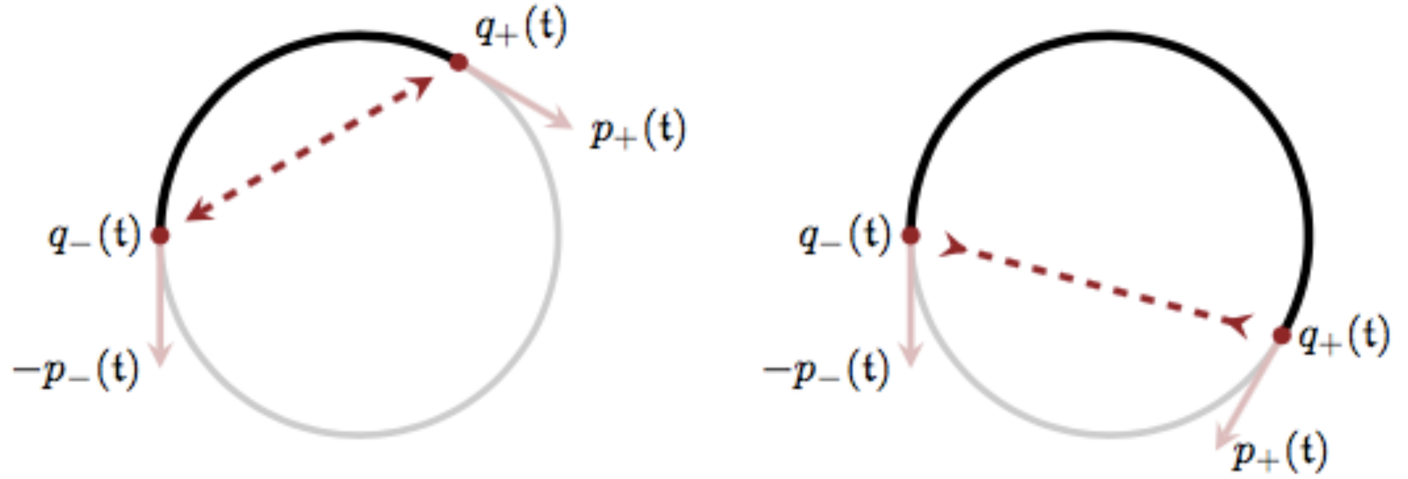
```
H, qall= HMC(U=U,K=K,dUdq=dUdq,N=10000,q_0=0, p_0=-4, epsilon=0.01, L=200)

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

L tuning

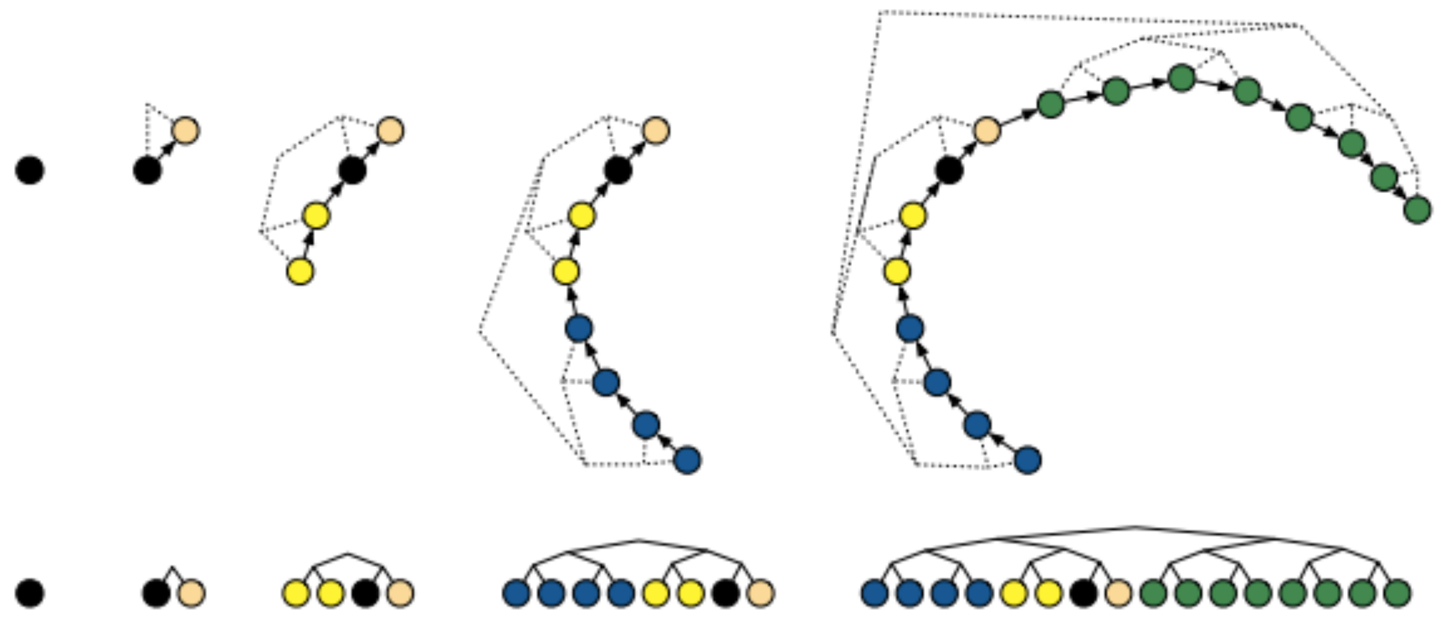
- in HMC, start $L = 100$ increase if for fixed step size, autocorrelation is too much
- Tails correspond to much higher energies, larger level-set surfaces are larger
- fixed length explores a small portion of this set before a momentum resampling takes us off.
- better to set dynamically: NUTS termination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

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 or slice sampling

Tumors in pymc3 with NUTS

```
with Model() as tumor_model:
    # Uniform priors on the mean and variance of the Beta distributions
    mu = Uniform("mu",0.00001,1.)
    nu = Uniform("nu",0.00001,1.)
    # Calculate hyperparameters alpha and beta as a function of mu and nu
    alpha = pm.Deterministic('alpha', mu/(nu*nu))
    beta = pm.Deterministic('beta', (1.-mu)/(nu*nu))
    # Priors for each theta
    thetas = Beta('theta', alpha, beta, shape=N)
    # Data likelihood
    obs_deaths = Binomial('obs_deaths', n=tumorn, p=thetas, observed=tumory)

with tumor_model:
    # Use ADVI for initialization
    mu, sds, elbo = pm.variational.advi(n=100000)
    step = pm.NUTS(scaling=tumor_model.dict_to_array(sds)**2,
                   is_cov=True)
    tumor_trace = pm.sample(5000, step, start=mu)
```

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

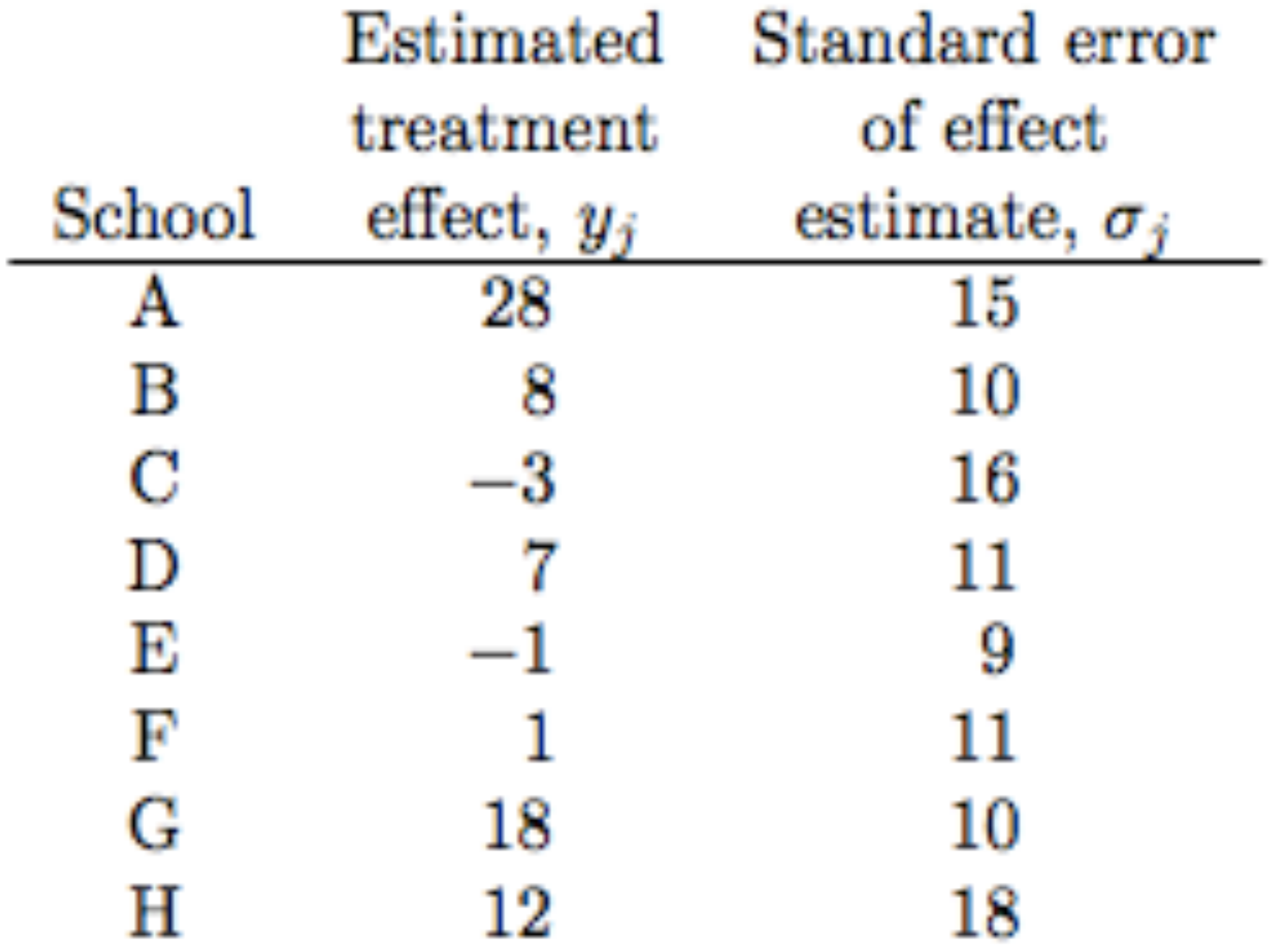
$\bar{y_j} = \frac{1}{n_j} \sum_{i=1}^{n_j} y_{ij}$ with sampling variance

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

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

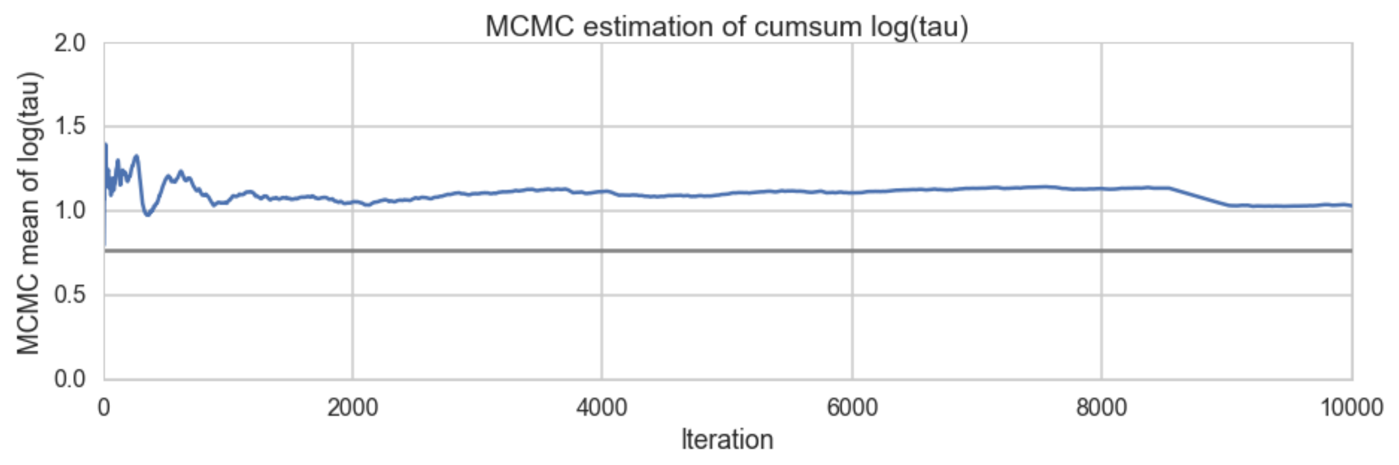
Centered Hierarchical Model

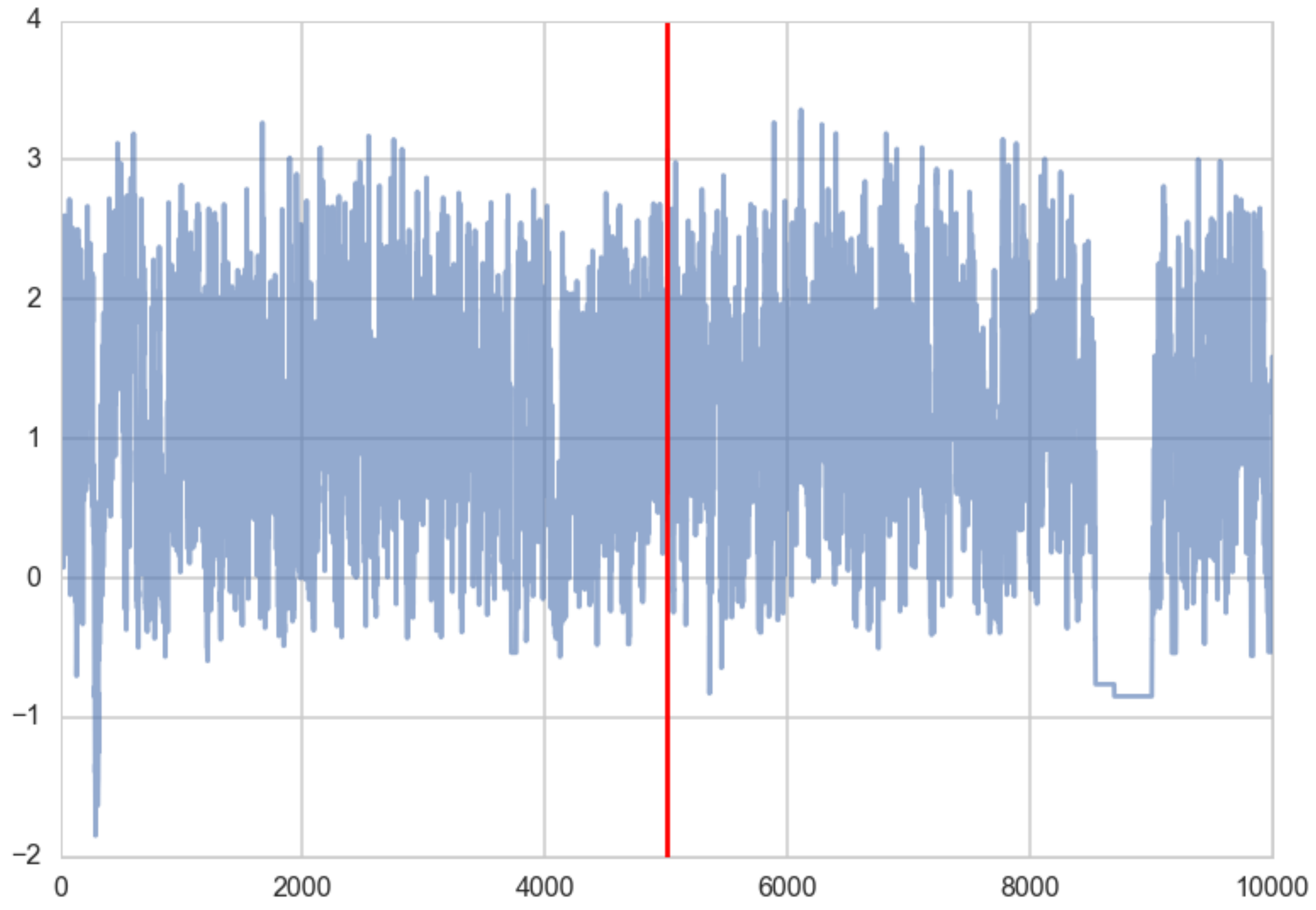
$$\mu \sim \mathcal{N}(0, 5)$$
$$\tau \sim \text{Half-Cauchy}(0, 5)$$
$$\theta_j \sim \mathcal{N}(\mu, \tau)$$
$$\bar{y_j} \sim \mathcal{N}(\theta_j, \sigma_j)$$

```python
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}:

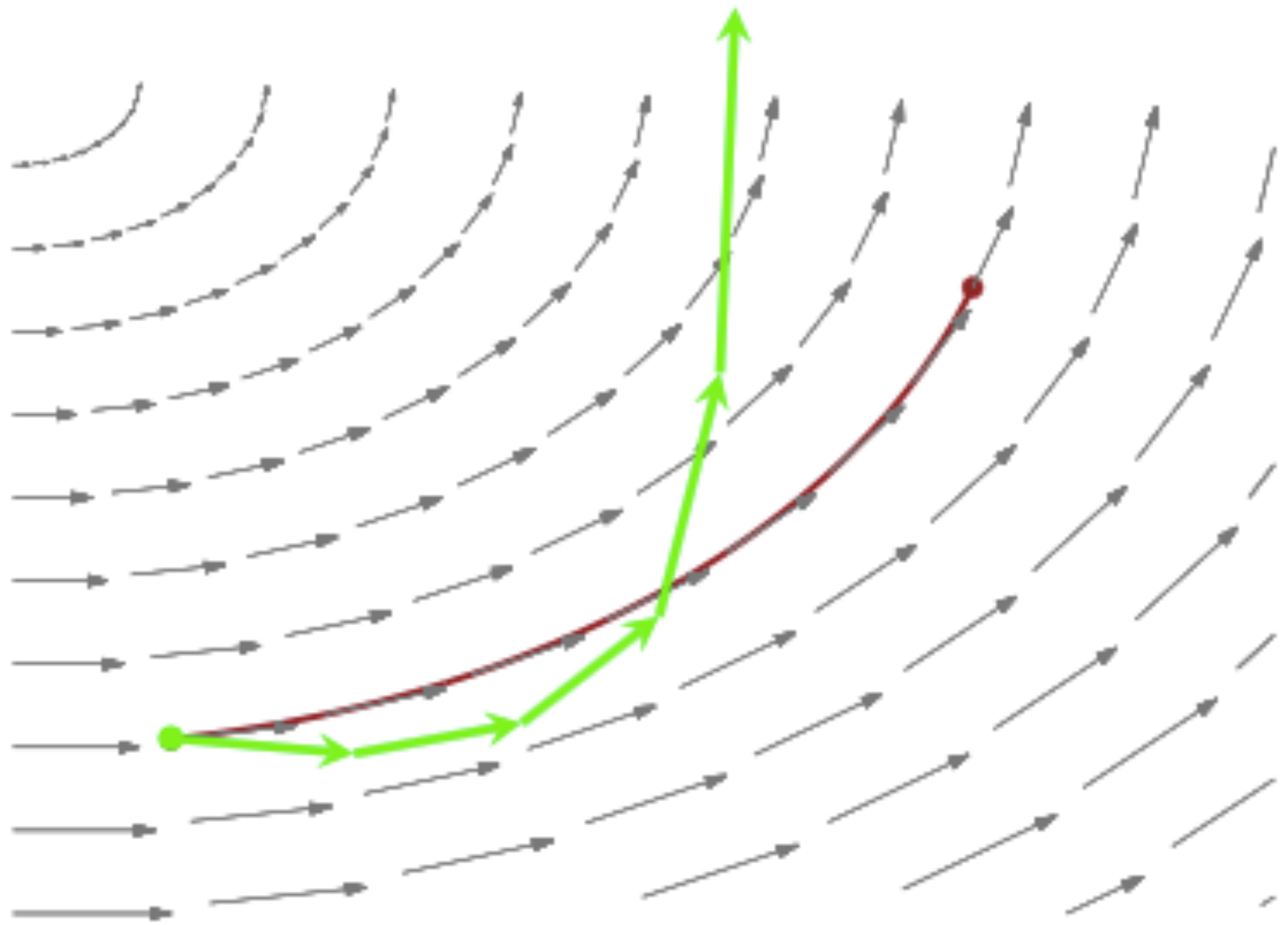
```
{'mu': 101.0,
  'tau': 273.0,
  'tau_log_': 77.0,
  'theta': array([ 169.,   199.,   236.,   193.,   211.,   231.,   139.,   204.])})
```





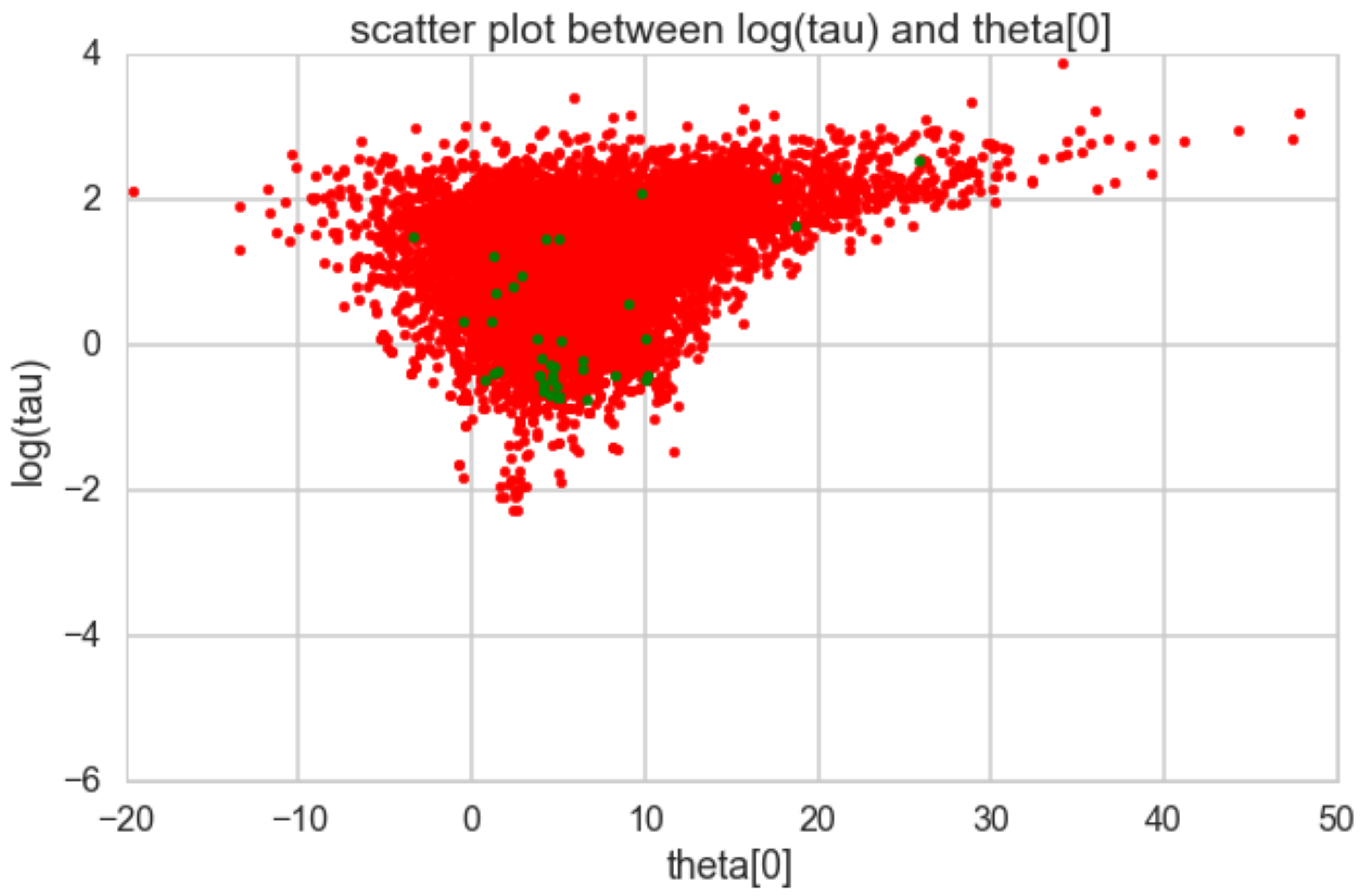
- stickys are actually trying to drive down value of trace
- we are in a region of high curvature

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.



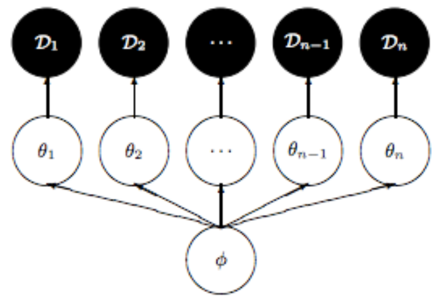
Diagnosed thus:

```
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)
```

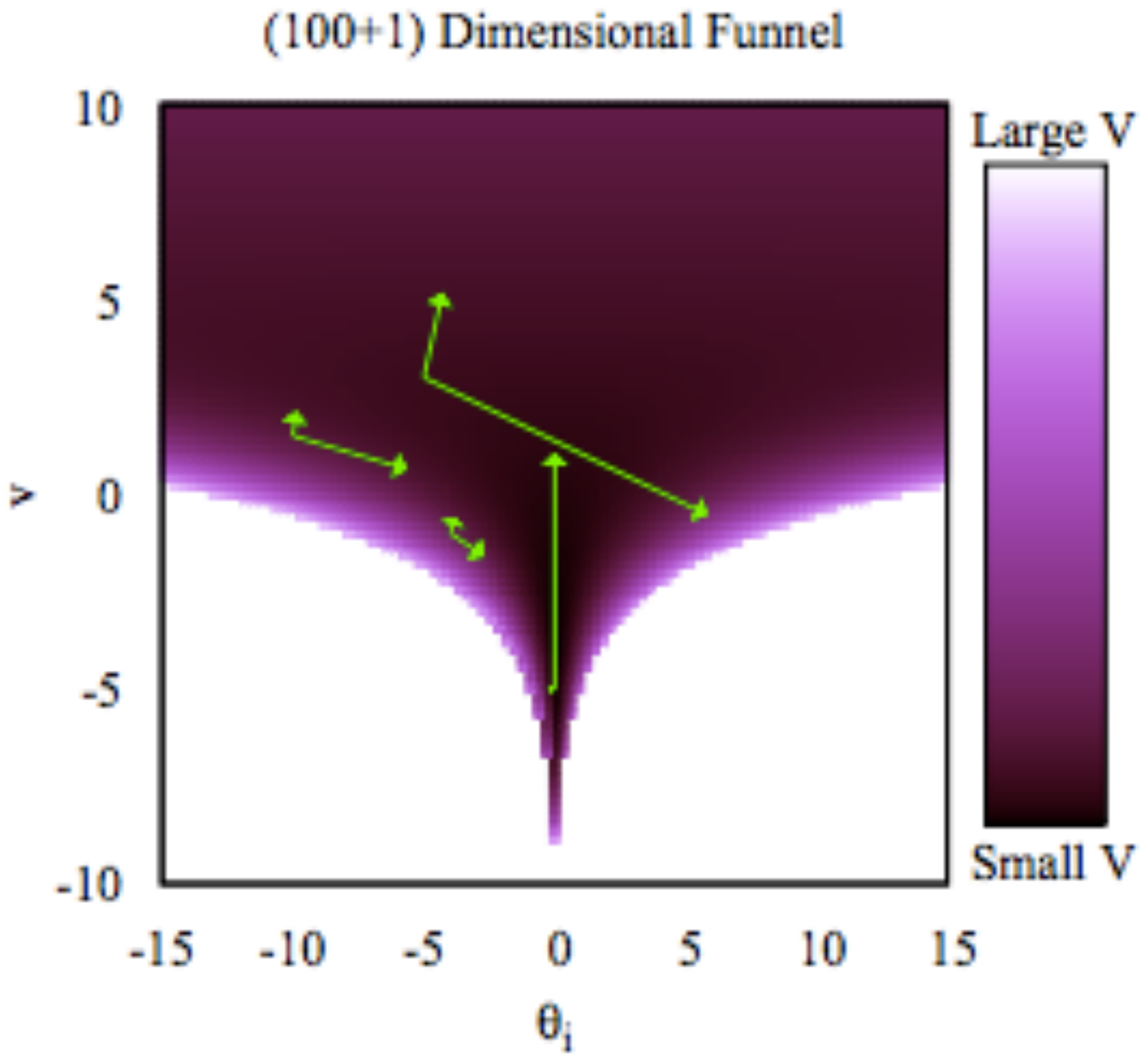
```
Number of Divergent 74
Percentage of Divergent 0.01480
```

- 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



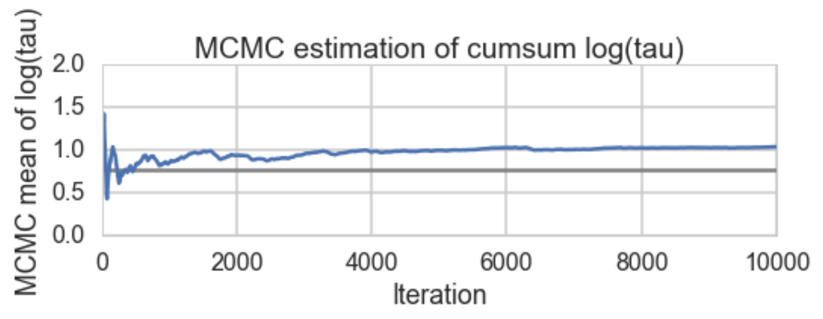
Step size effect

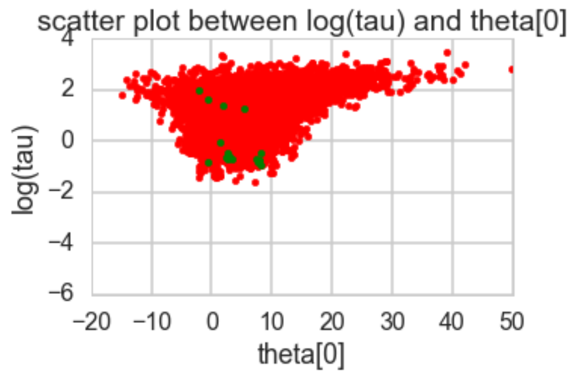
- lower step size ϵ better for symplectic integrators, especially in high curvature regions
- this allows for geometric ergodicity: we go everywhere.
- too small ϵ: return of the random walk.

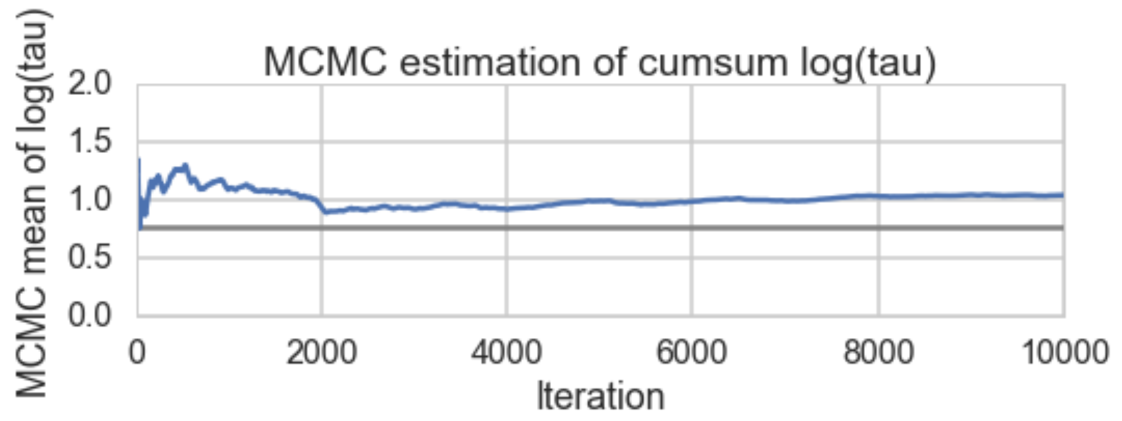
Changing step size

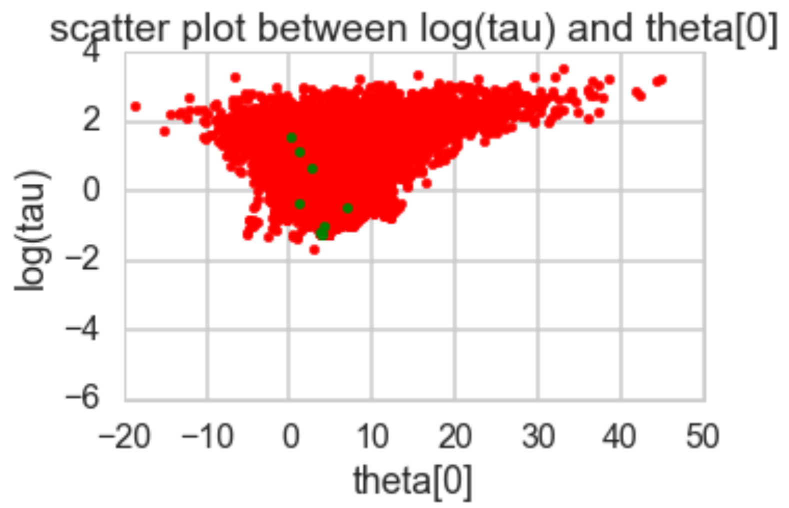
```
with schools1:
    step = pm.NUTS(target_accept=.85)
    trace1_85 = pm.sample(5000, step=step, init=None, njobs=2, tune=1000)
```

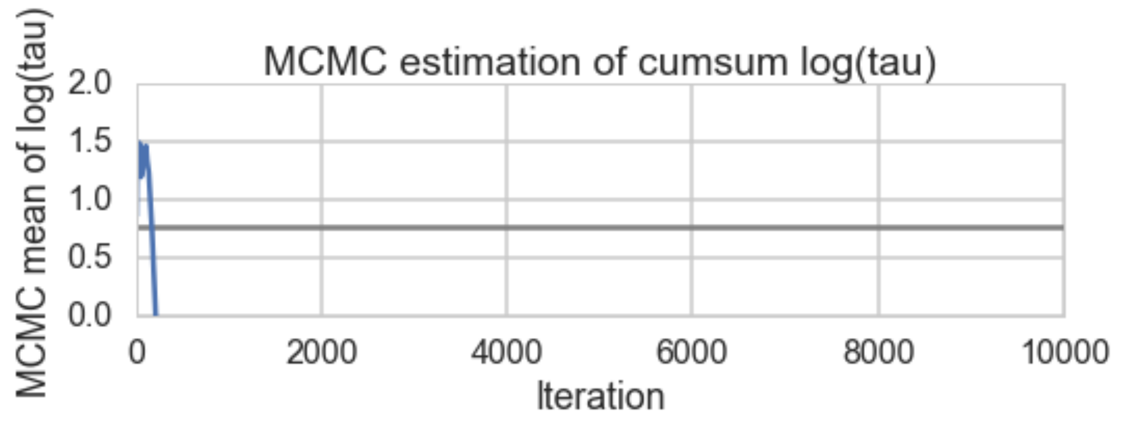
```
85: Acceptance 0.804601458758 Step Size 0.203087336483 Divergence 39
90: Acceptance 0.873340820433 Step Size 0.159223726996 Divergence 18
95: Acceptance 0.923346597897 Step Size 0.126824682121 Divergence 9
99: Acceptance 0.990173791609 Step Size 0.0164237997757 Divergence 5
```

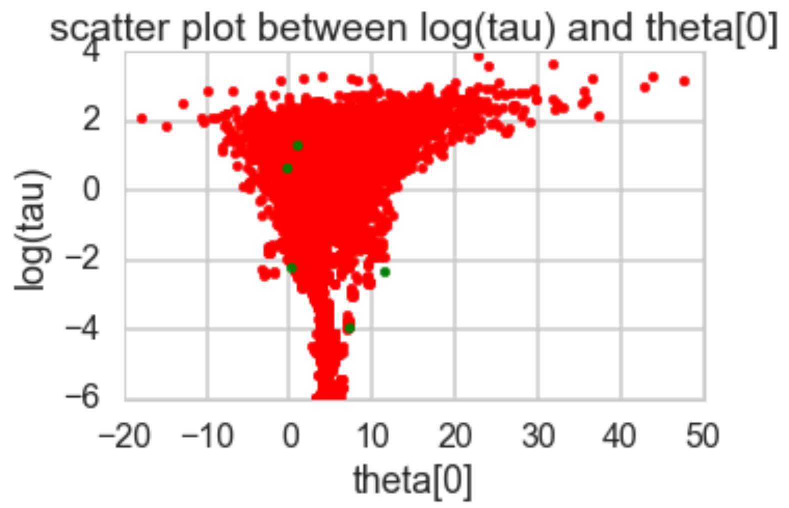








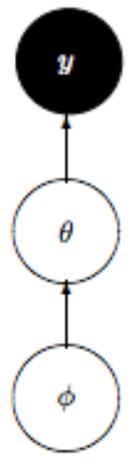


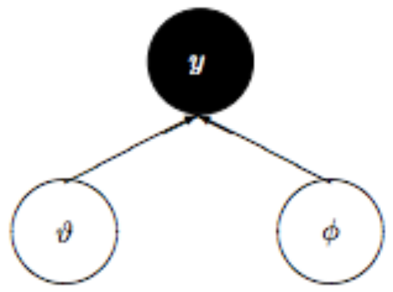


divergences persist. Too curved!

Non-centered model

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

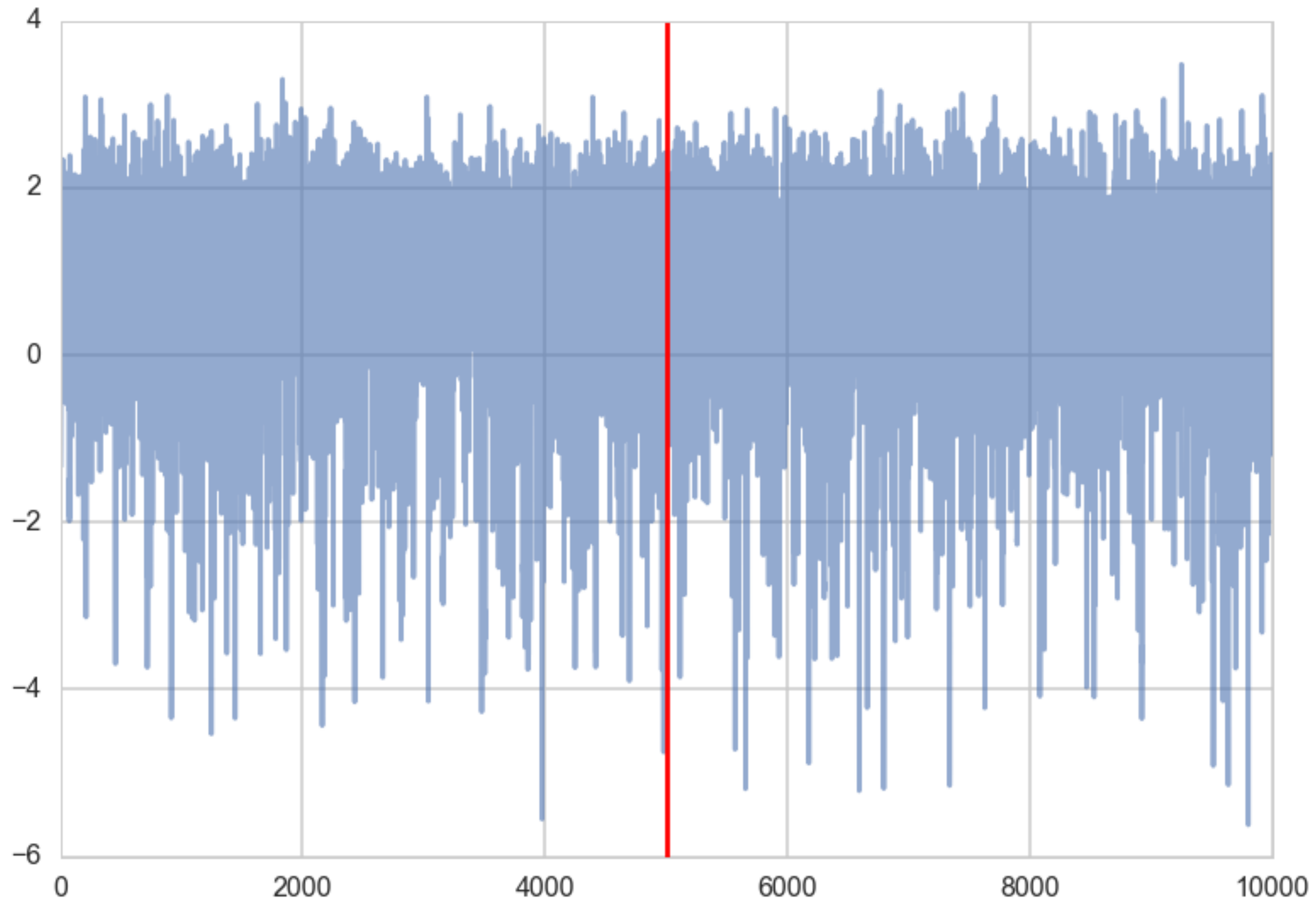




$$\mu \sim \mathcal{N}(0, 5)$$
$$\tau \sim \text{Half-Cauchy}(0, 5)$$
$$\nu_j \sim \mathcal{N}(0, 1)$$
$$\theta_j = \mu + \tau\nu_j$$
$$\bar{y}_j \sim \mathcal{N}(\theta_j, \sigma_j)$$

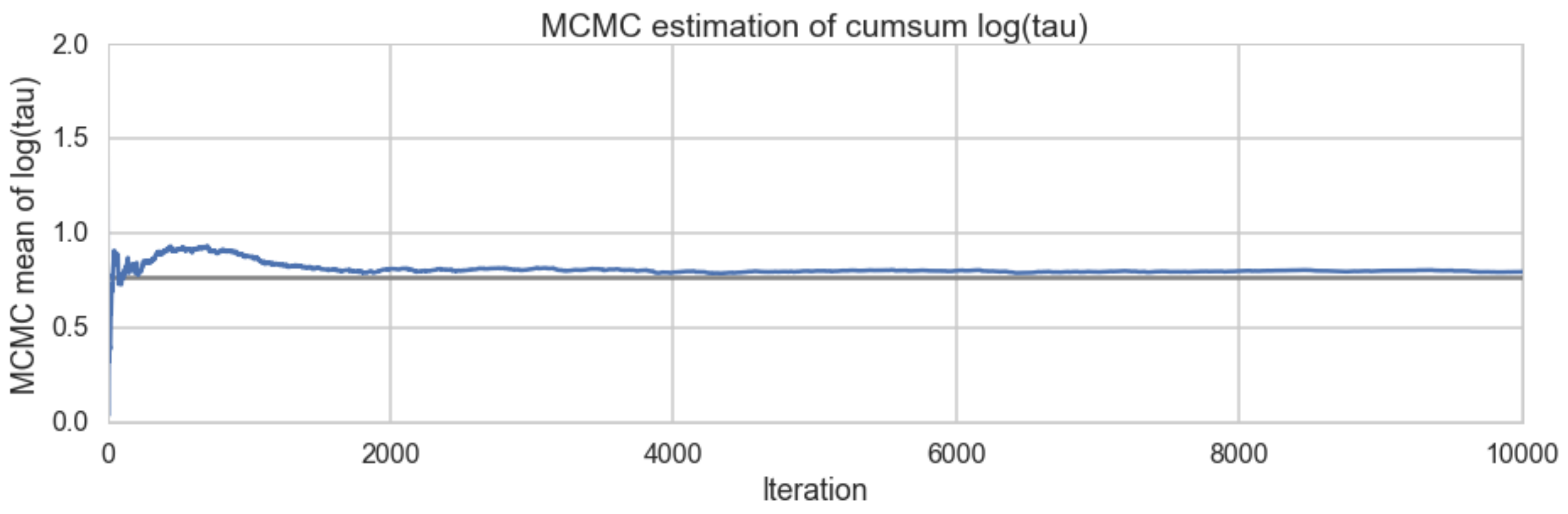
Factor dependency of θ on $\phi = \mu, \tau$ into a deterministic transformation between the layers, leaving the actively sampled variables uncorrelated.

```
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}:

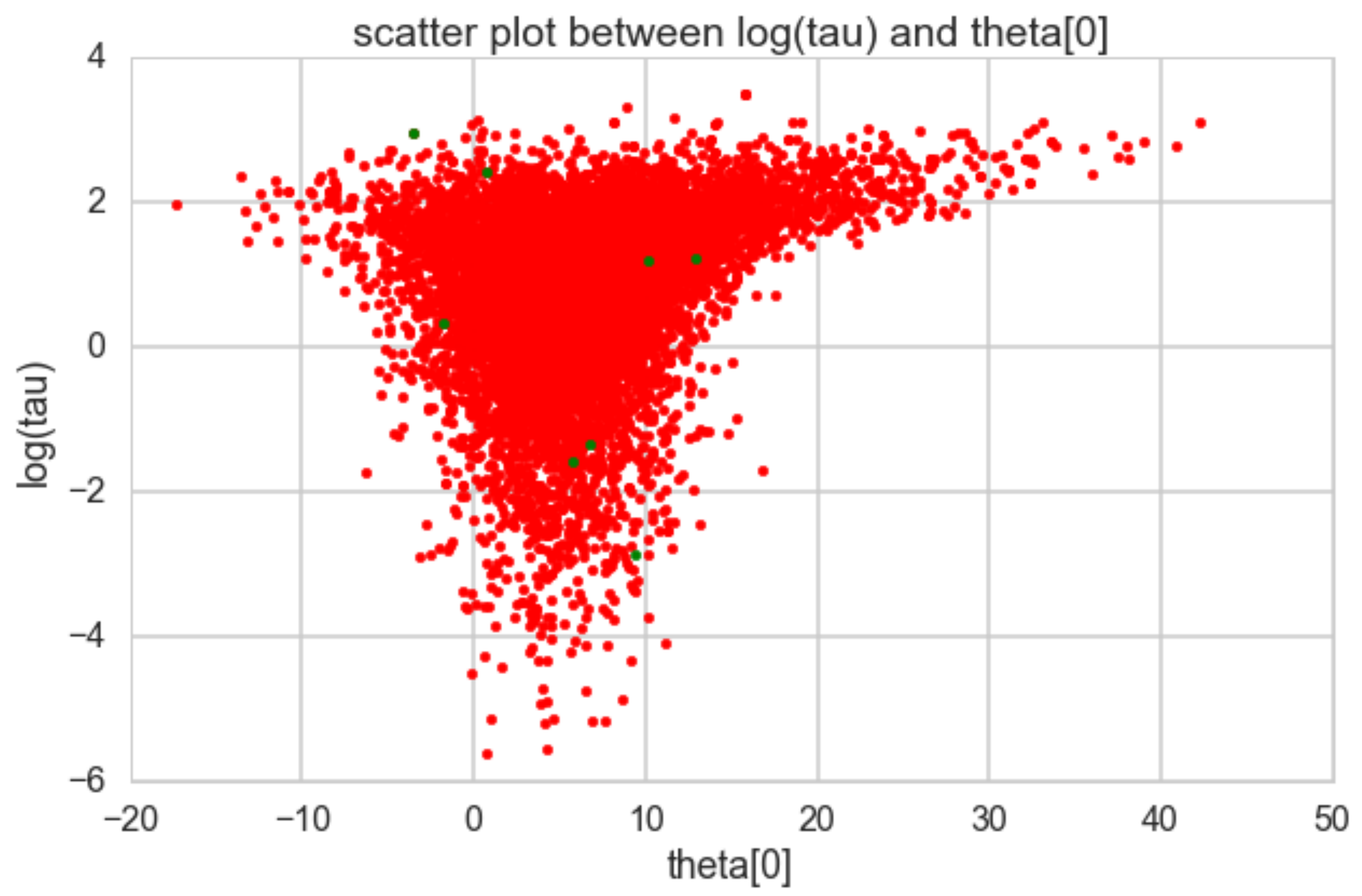
```
{'mu': 10000.0,
  'nu': array([ 10000.,  10000.,  10000.,  10000.,  10000.,  10000.,  10000.,
         10000.]),
  'tau': 6880.0,
  'tau_log_': 5193.0,
  'theta': array([  9624.,  10000.,  10000.,  10000.,  10000.,  10000.,  10000.,
          9829.])}
```

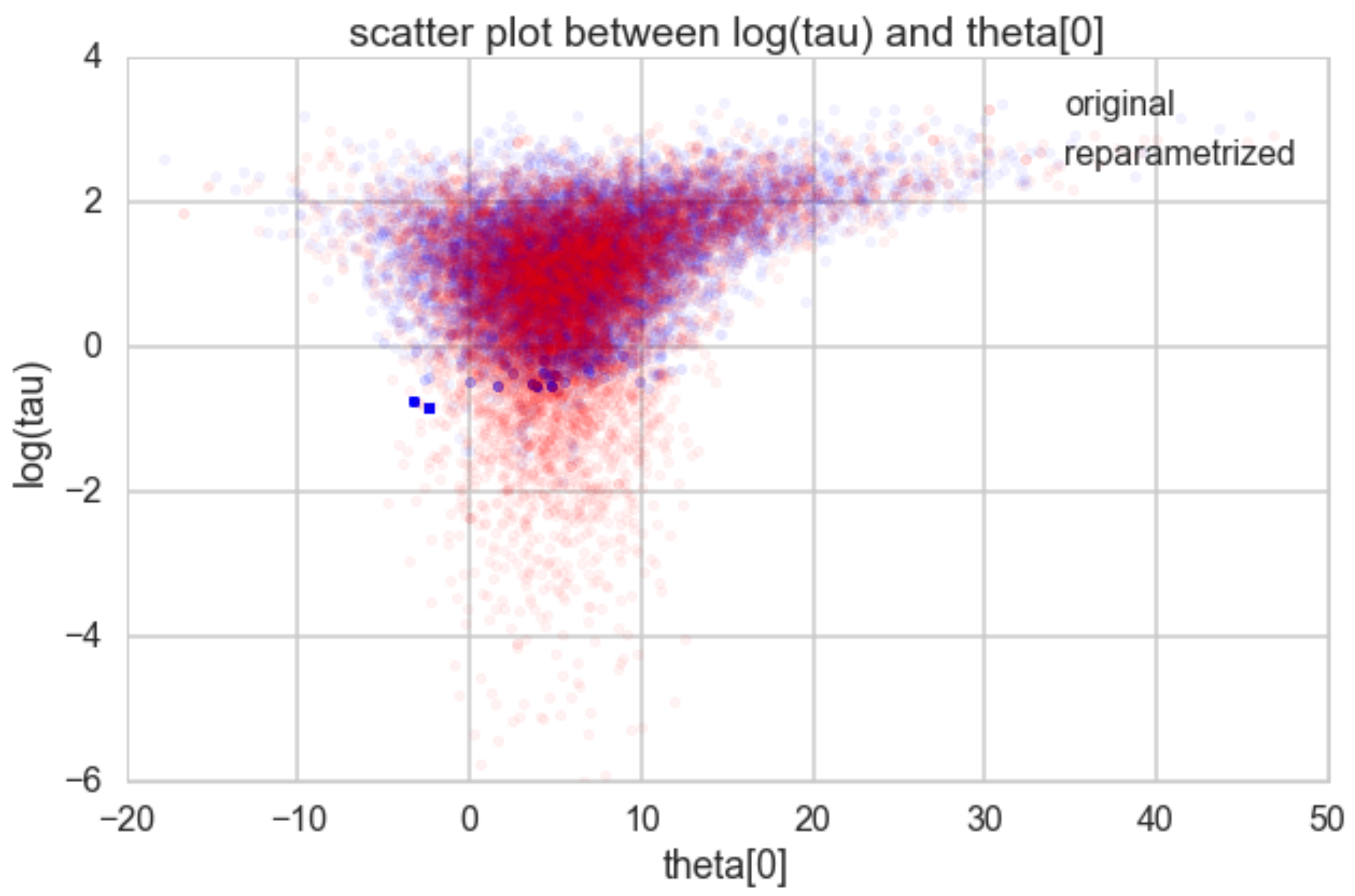


```
divergent = trace2['diverging']
print('Number of Divergent %d' % divergent.nonzero()[0].size)
divperc = divergent.nonzero()[0].size/len(trace2)
print('Percentage of Divergent %.5f' % divperc)
```

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

Divergences and true length of funnel

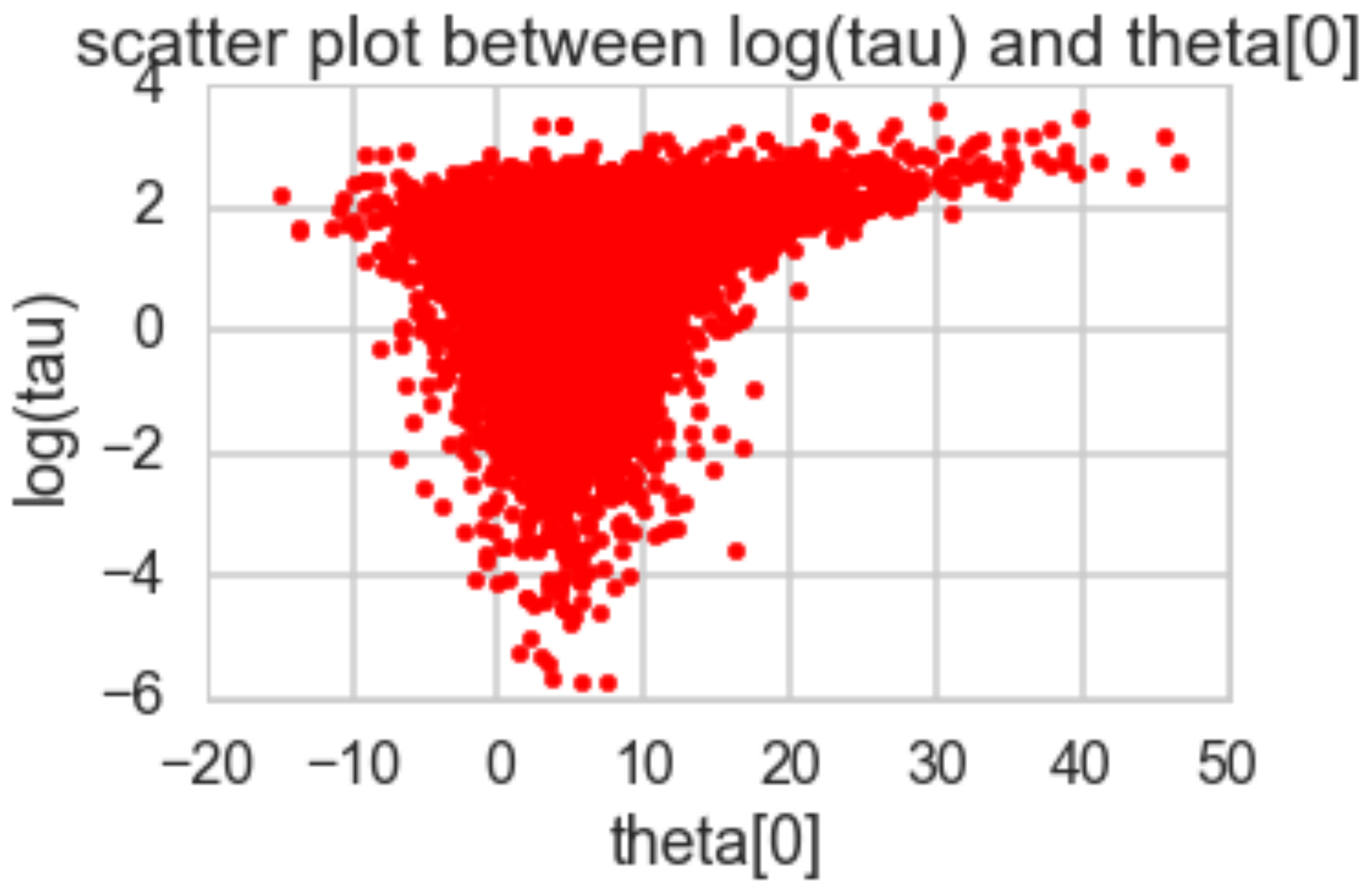




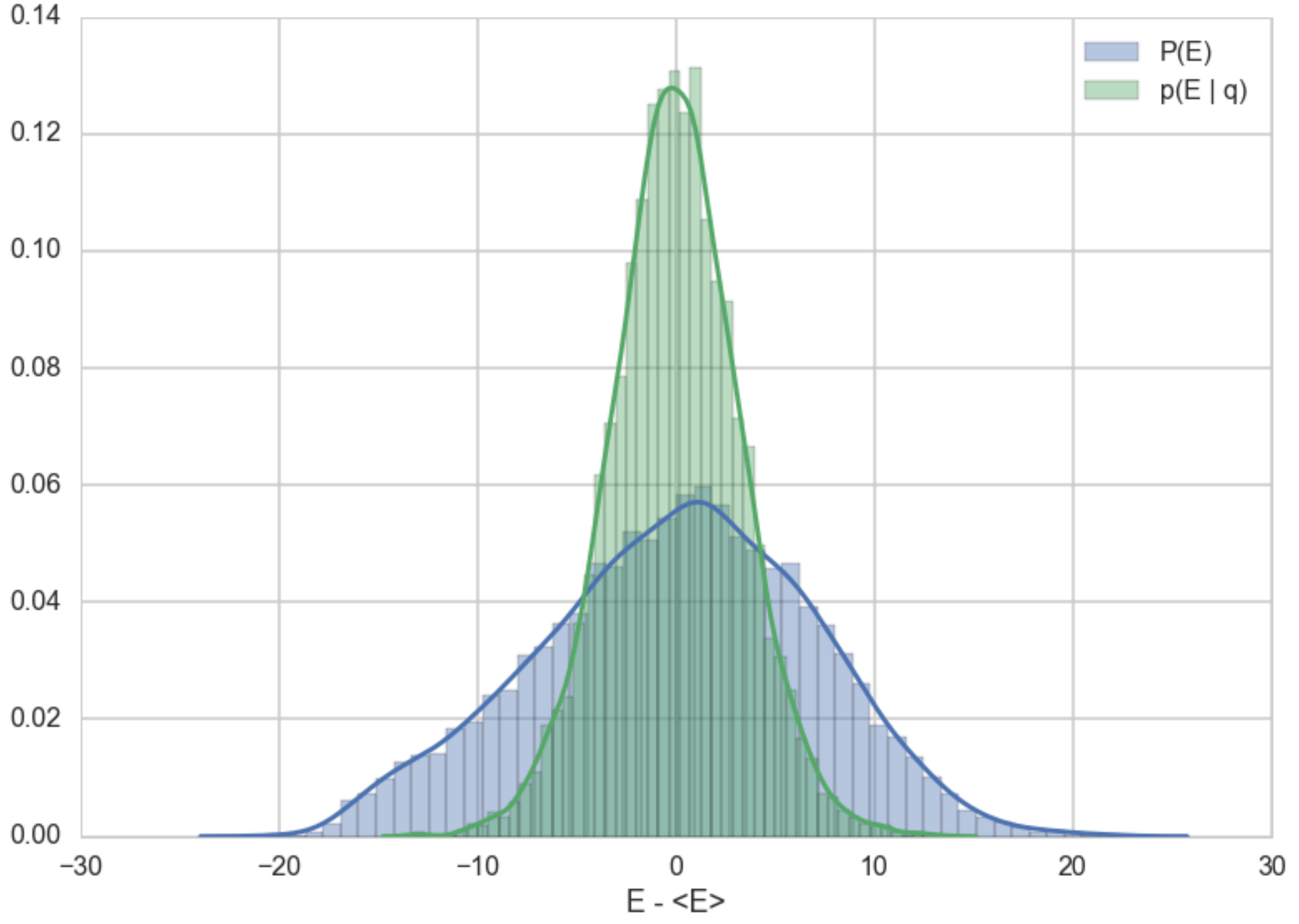
- Divergences infrequent, and all over. Mostly false positives.
- Lowering step sizes should make them go away

```
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

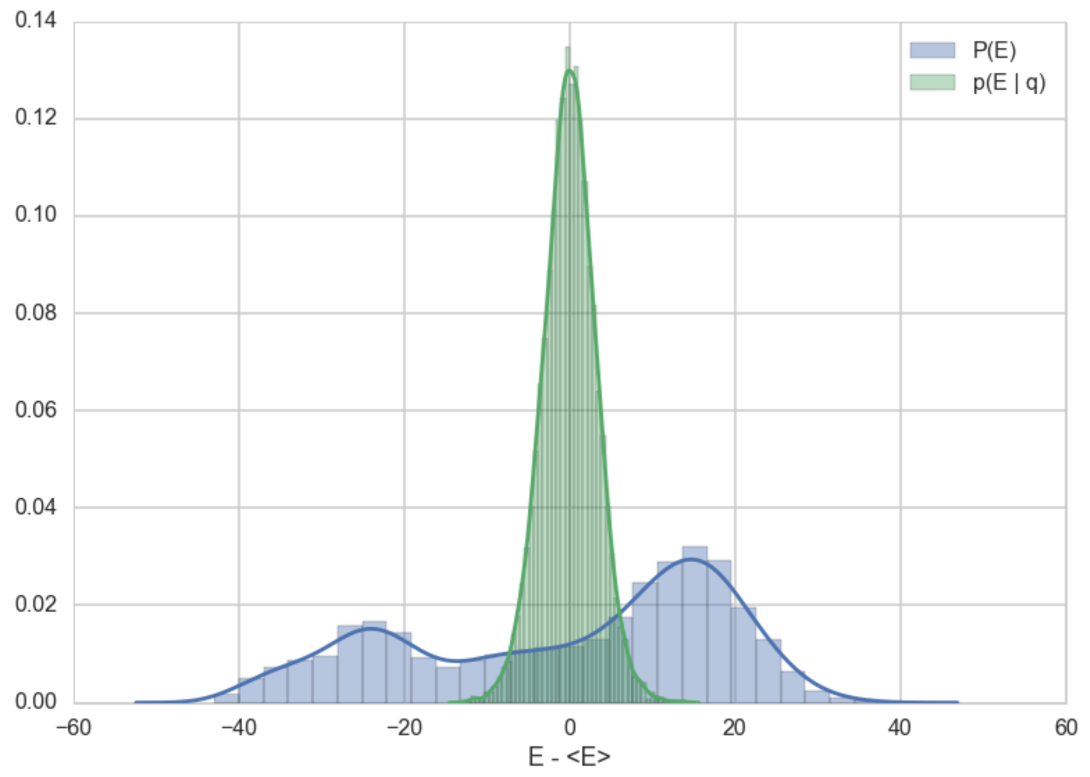


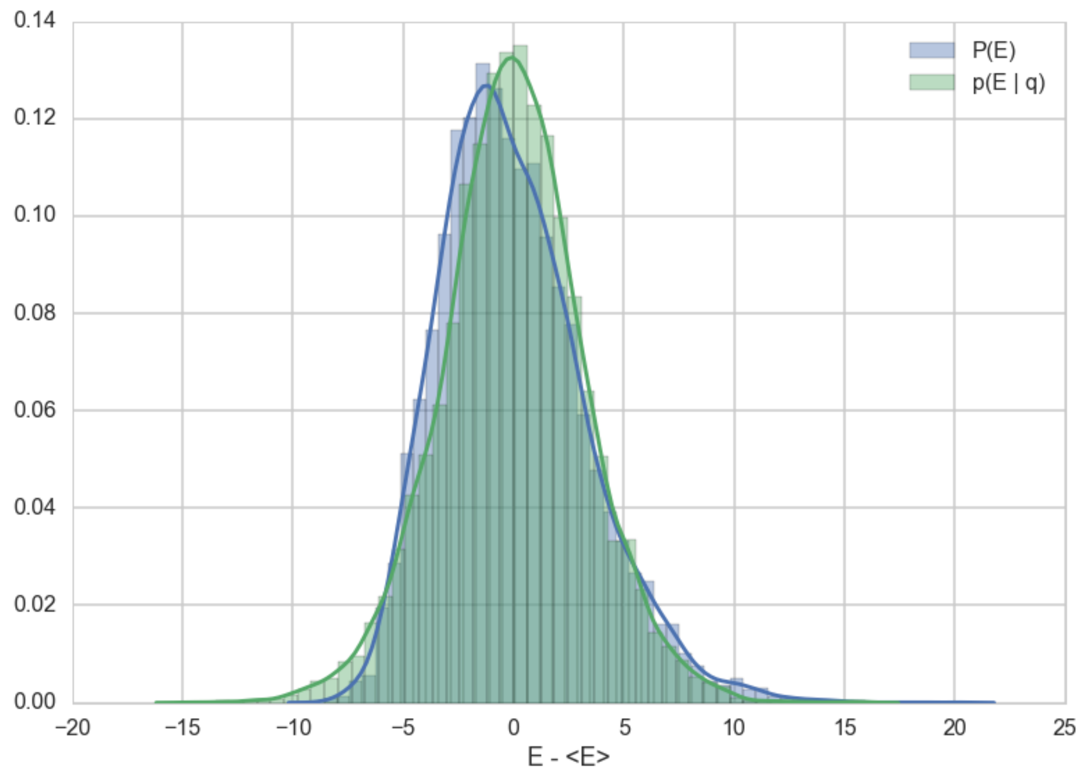
- match transition $p(E|q)$ to marginal $p(E)$

```
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!