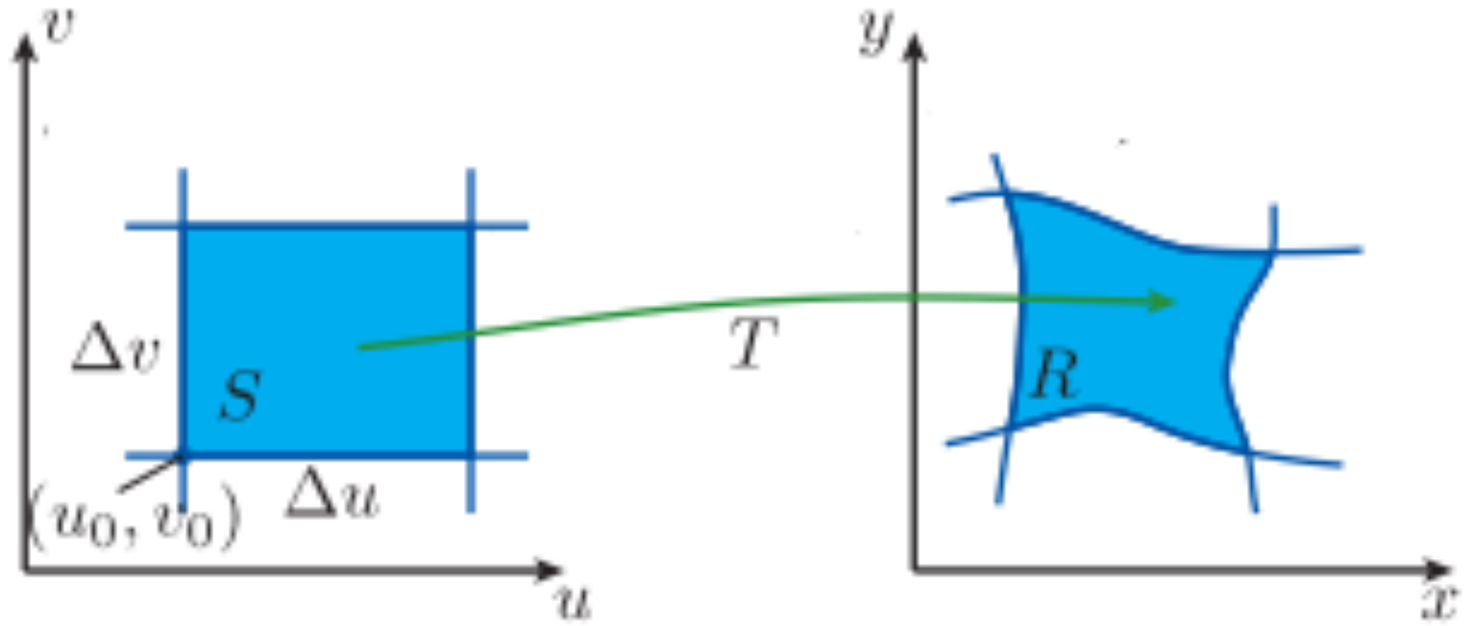
Lecture 18

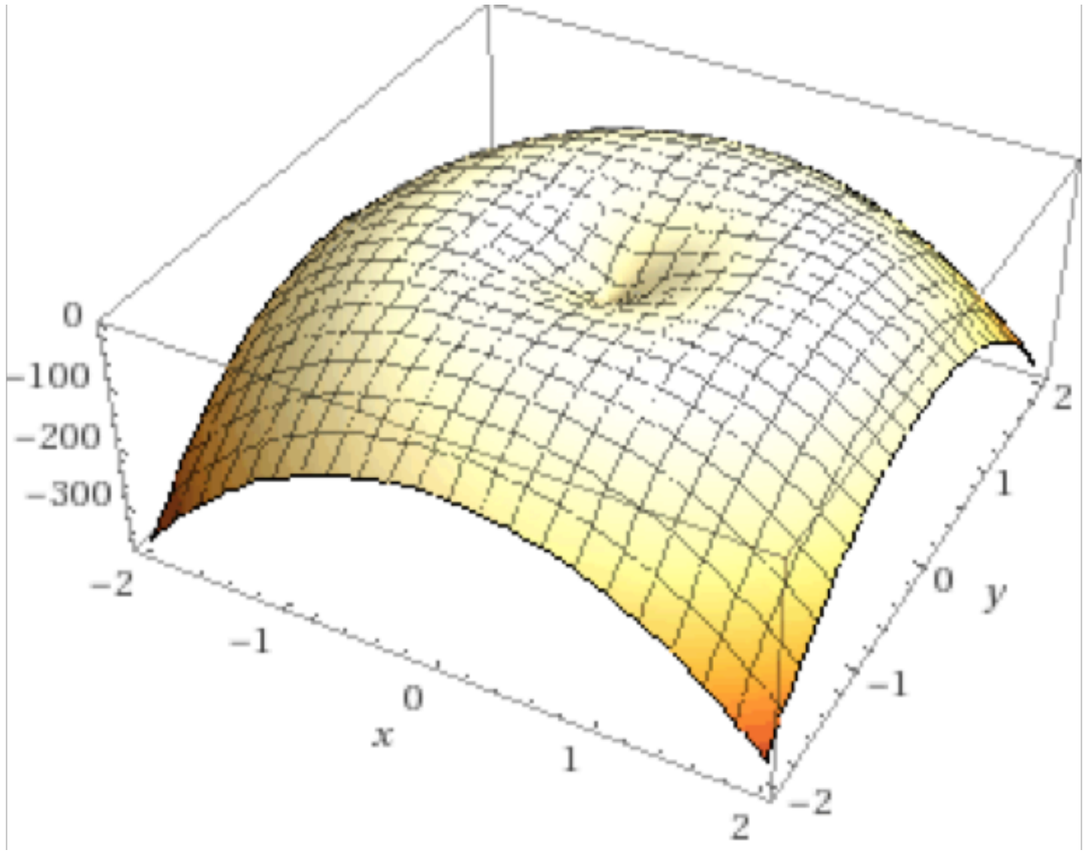
HMC and Formal Tests

Jacobian



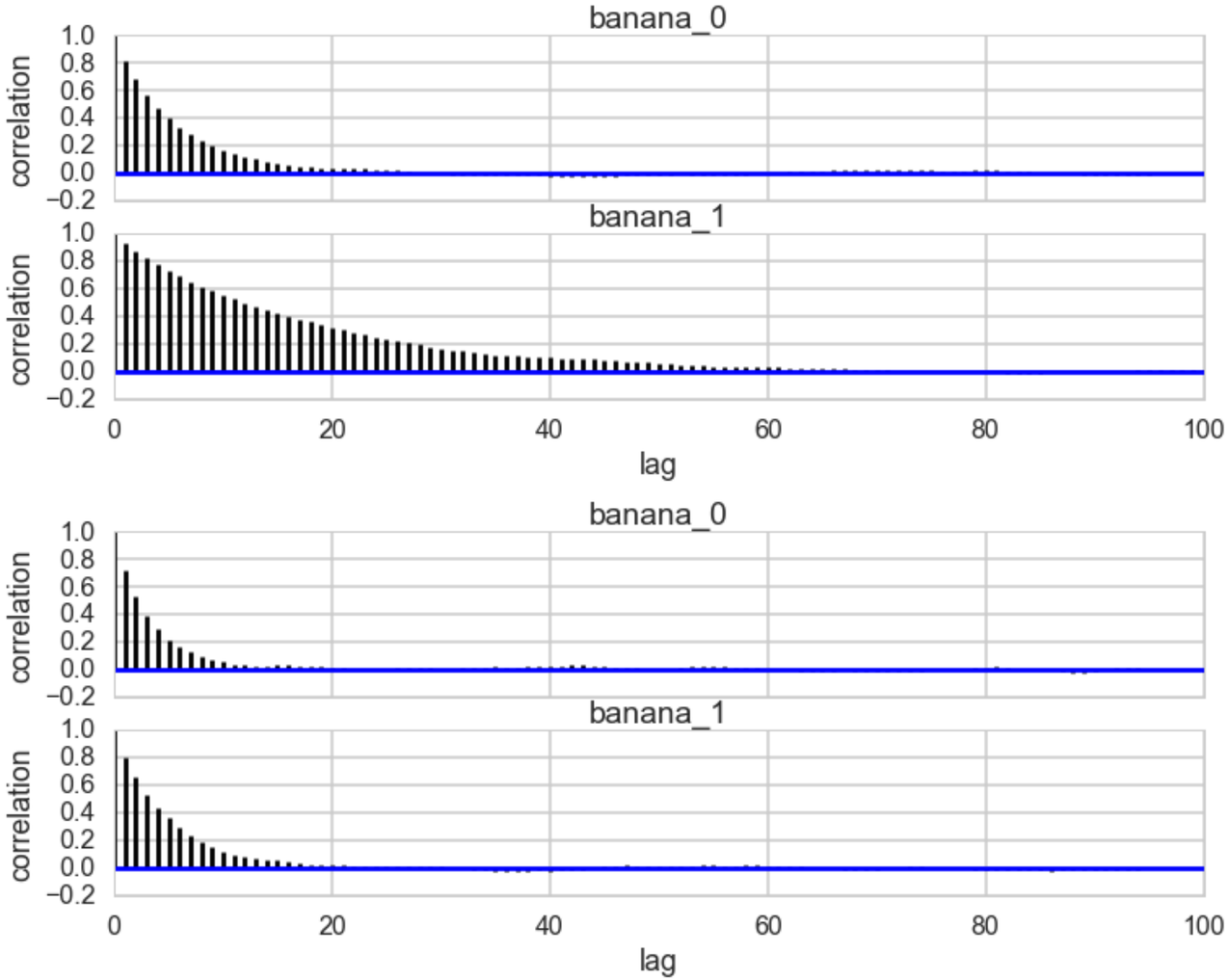
(from https://www.projectrhea.org/rhea/index.php/Jacobian)

ADVERT

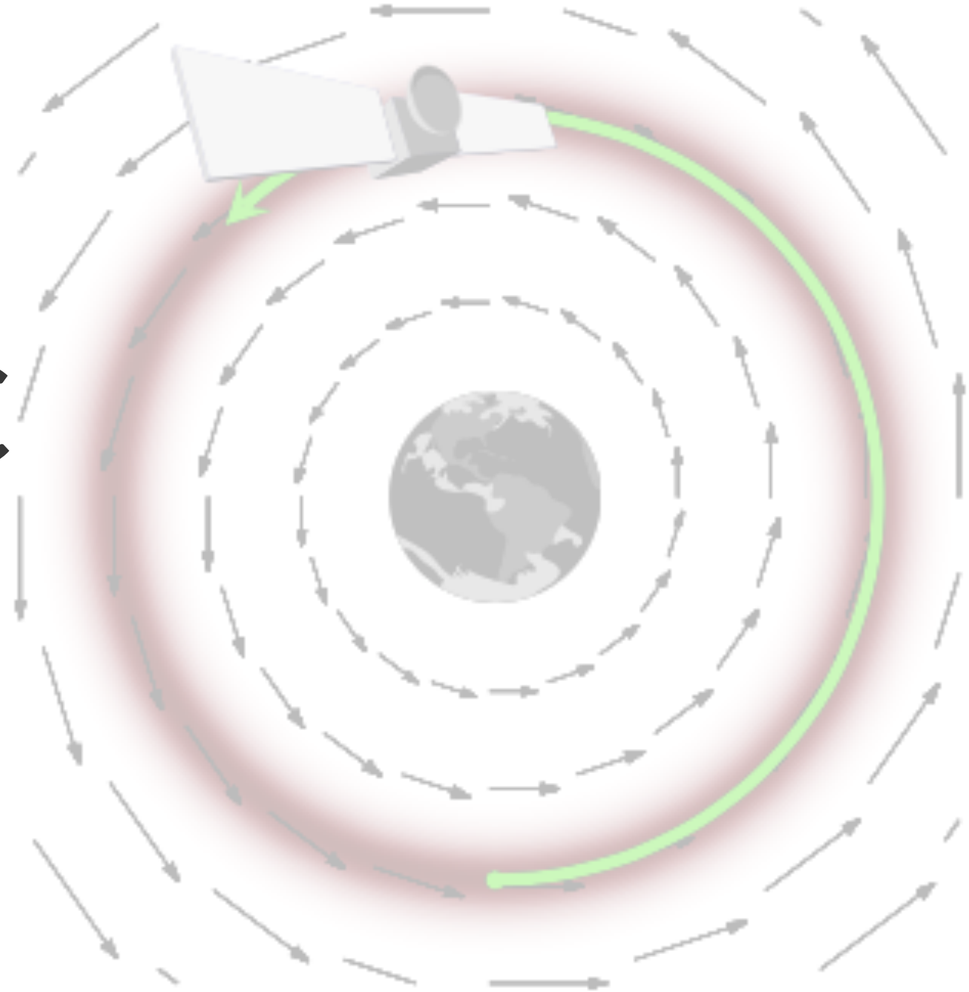


HMC/NUTS in pymc3

```
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])
```



HMC



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 $e^{-V(q)}$ V(q) = potential energy
- 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: $H(p,q) = -log(p(p,q)) = -logp(p|q) - logp(q)$

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

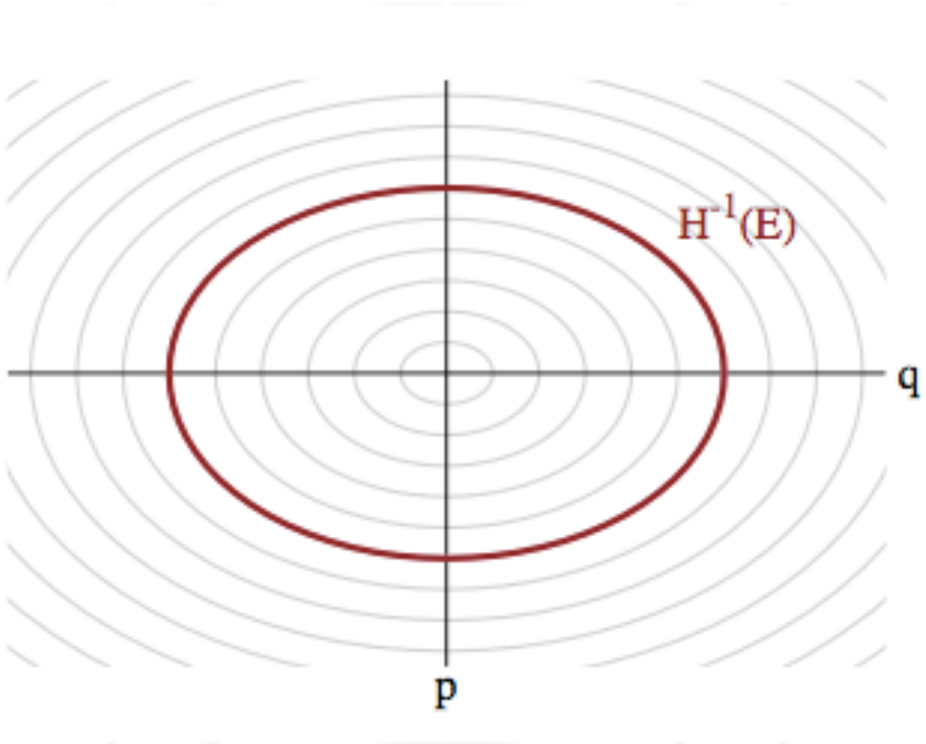
Phase Space level sets: Microcanonical Distribution

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 $(q, p) \to (q', p')$ 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.

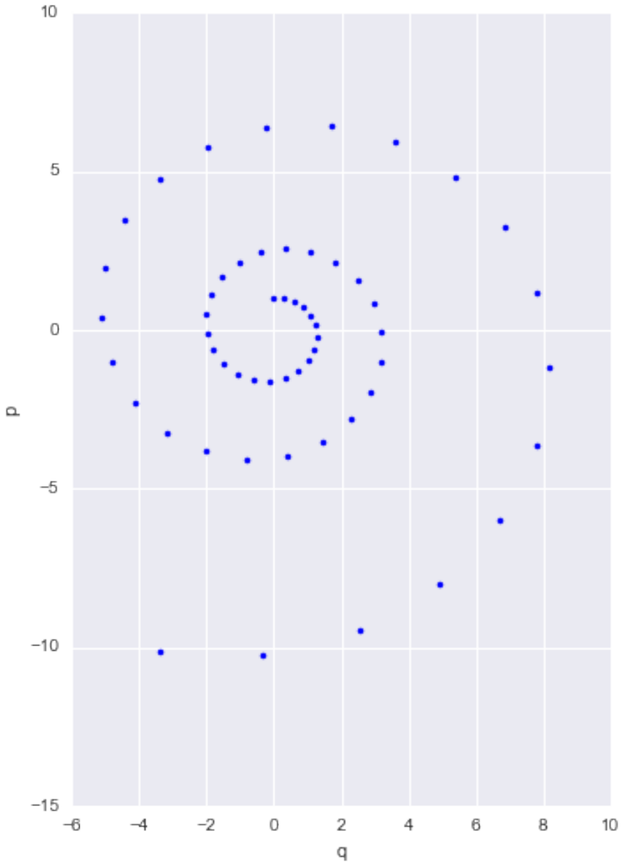
Basic Idea

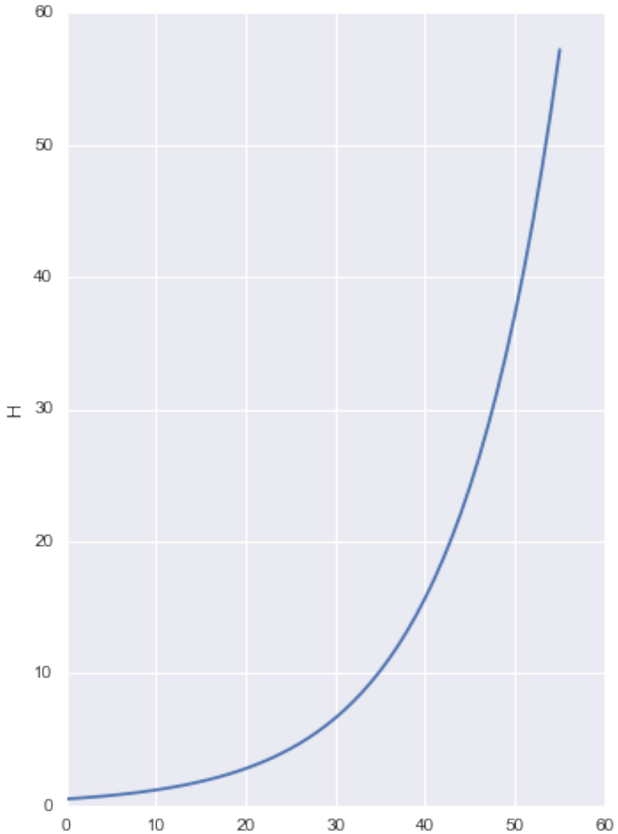
1. Move on a level set of the Hamiltonian $H(p, q)$. Pick up samples at will with acceptance probability 1
2. Fire thrusters, that is sample p, from kinetic energy distribution, to move to another level set
3. Repeat

More Problems arise

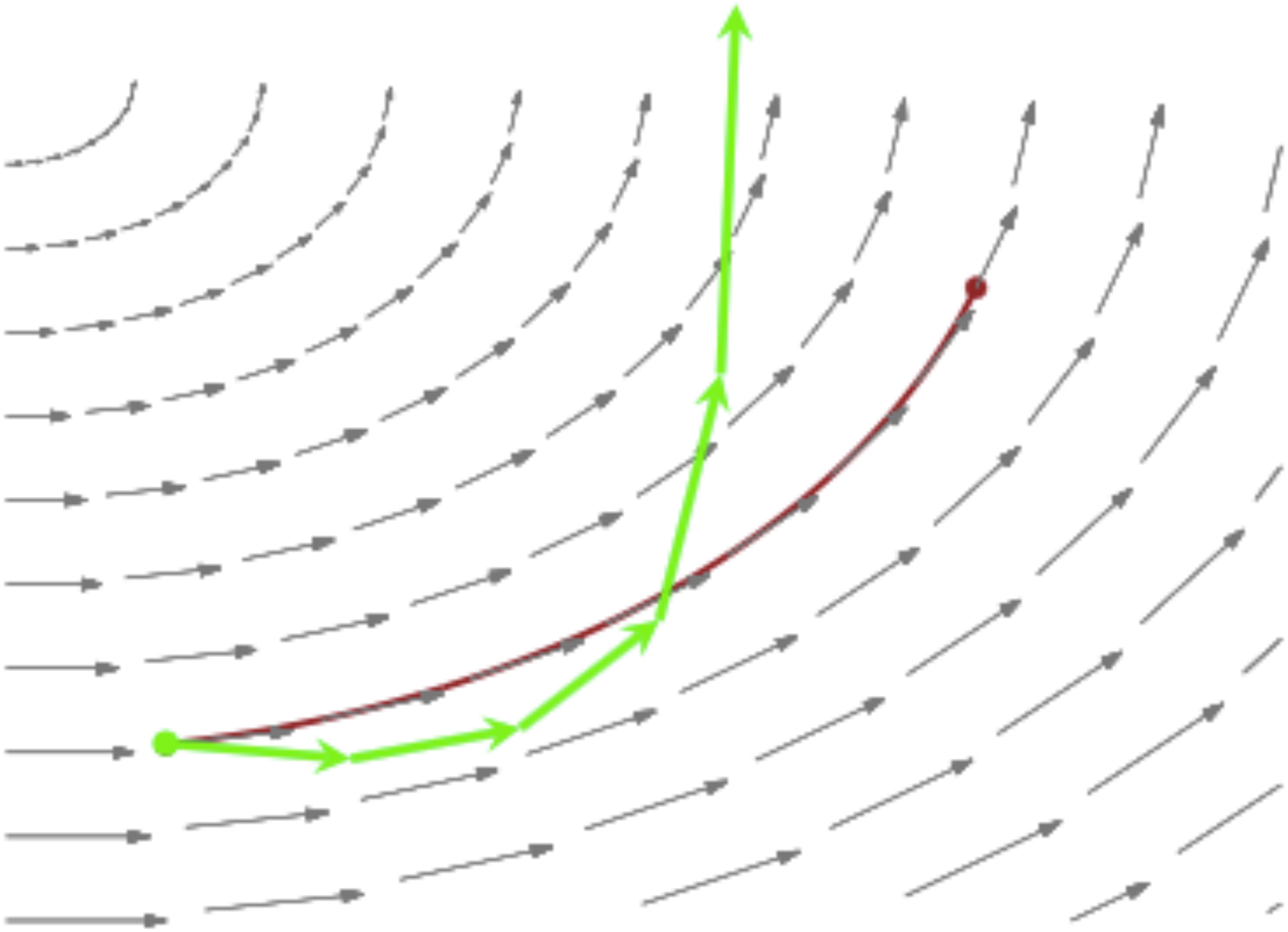
- discretization to solve differential equations can lead to infinities. Thus the need for symplecticity
- lack of reversibility even with symplecticity (we are marginally off the level set)
- this means that the acceptance probability with no longer be 1
- how long should we go?
- whats a good kinetic energy?

Discretization problems

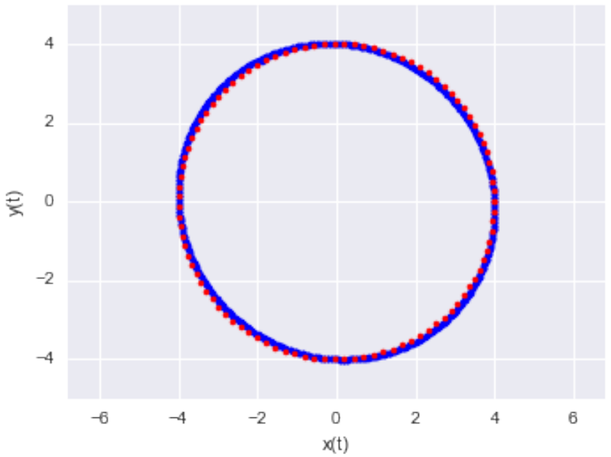


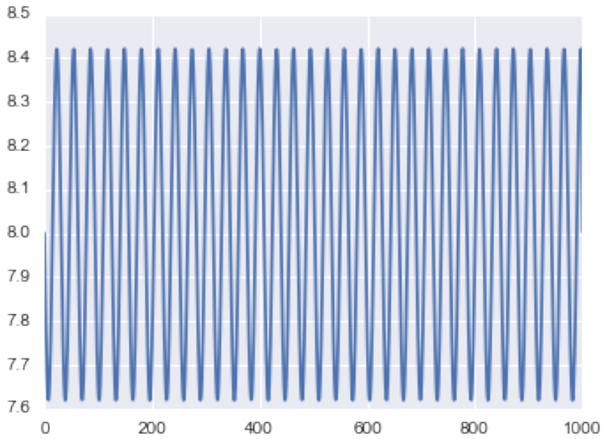


- $p_i(t+\epsilon) = p_i(t) - \epsilon \frac{\partial U}{\partial q_i}|_{q(t)}$
- $q_i(t+\epsilon) = q_i(t) + \epsilon \frac{p_i(t)}{m_i}$
- off-diagonal terms of size ϵ makes volume not preserved
- leads to drift over time
- use "leapfrog" instead



Sympletic Leapfrog



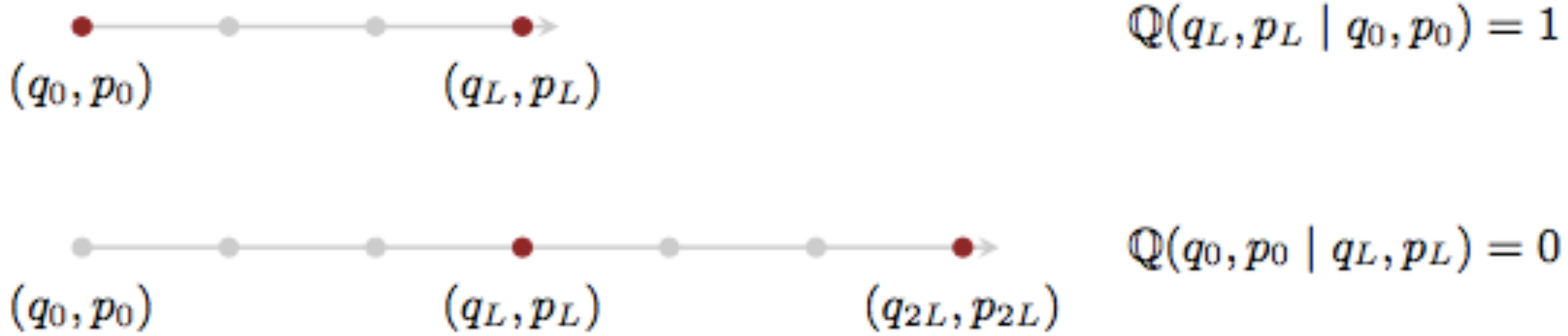


- Only *shear* transforms allowed, will preserve volume.
- $p_i(t+\frac{\epsilon}{2}) = p_i(t) - \frac{\epsilon}{2}\frac{\partial V}{\partial q_i}|_{q(t)}$
- $q_i(t+\epsilon) = q_i(t) + \epsilon\frac{p_i(t+\frac{\epsilon}{2})}{m_i}$
- $p_i(t+\epsilon) = p_i(t+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{\partial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

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?

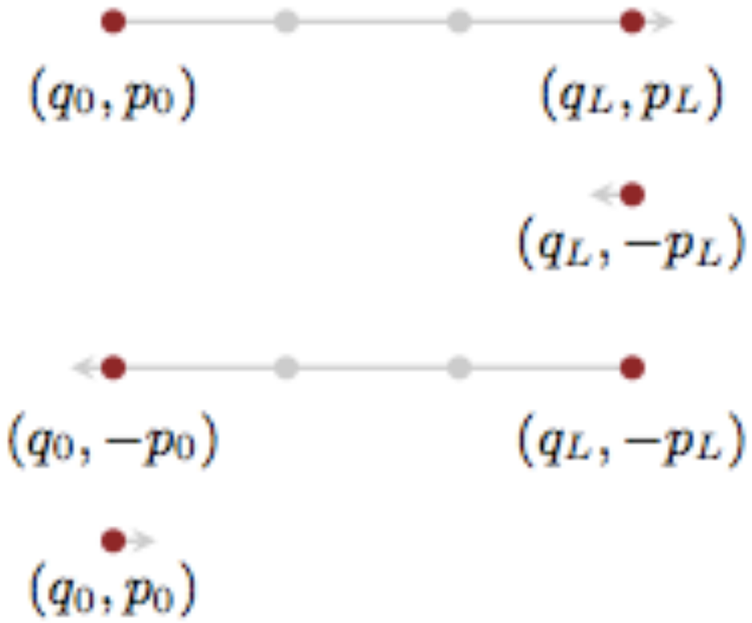


- might choose $Q(q', p'|q, p) = \delta(q' - q_L)\delta(p' - p_L)$.
- but small symplectic errors means this is only forward in time

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. Optimal acceptance can be shown: 65% roughly.

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.

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 .
- Due to the reversibility of the leapfrog steps, the B_k will also partition the space, and since the leapfrog steps preserve volume (as does negation), each B_k will also have volume V.

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

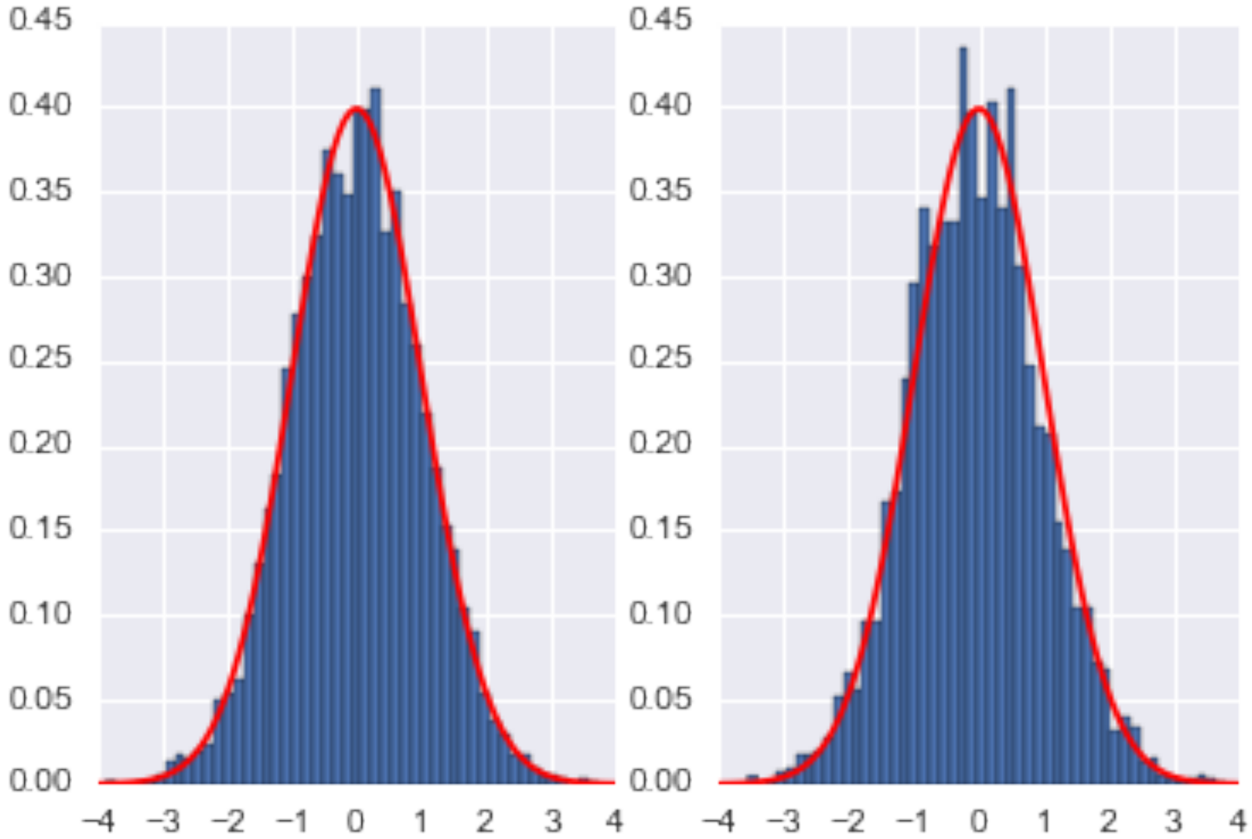
HMC Algorithm (momentum reversal could be left out if not within a more complex sampling scheme)

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

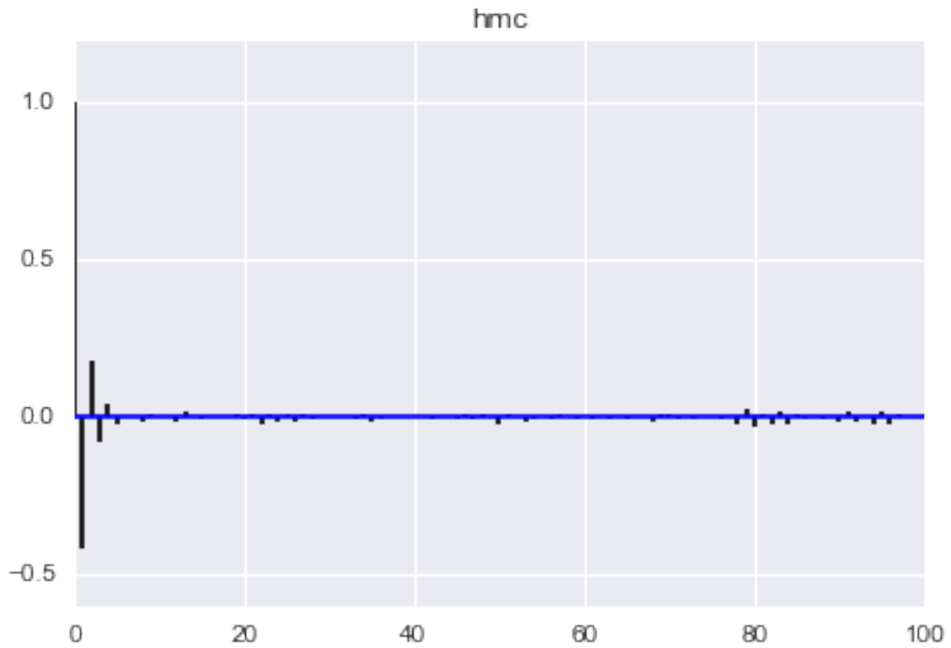
HMC (contd)

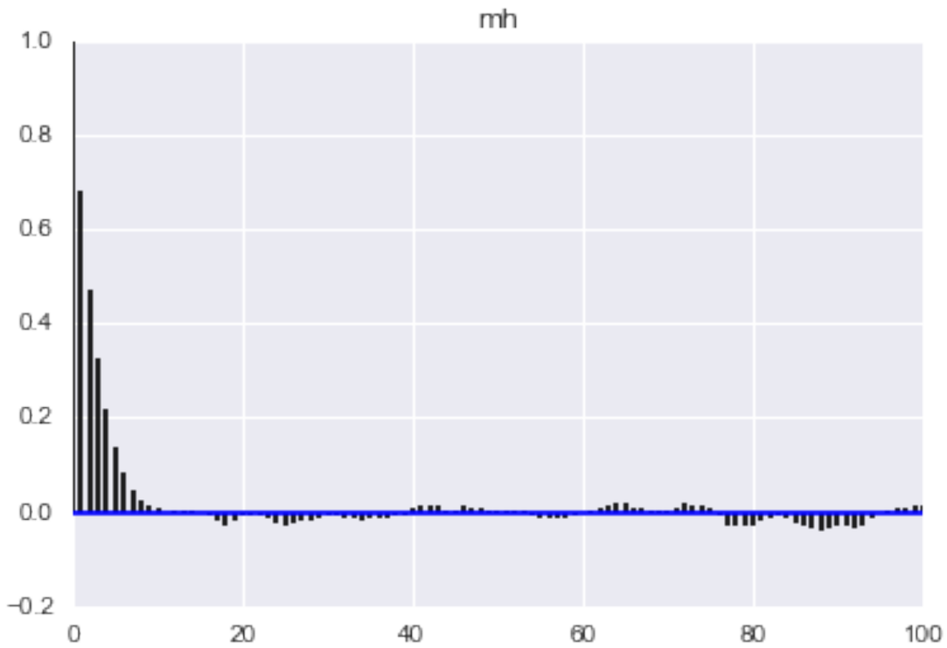
- for i=1:N_samples
  - 7. $p^* = -p$
  - 8. $V_c = V(q_c), \;\; K_c = \frac{p_c^\top M^{-1} p_c}{2}$
  - 9. $V^* = V(q^*), \;\; K^* = \frac{p^{\top *} M^{-1} p^*}{2}$
  - 10. $r \sim \mathbf{Unif}(0,1)$
  - 11. if $r < e^{(U_c - U^* + K_c - K^*)}$
    - accept $q_i = q^*$
    - otherwise reject

```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
```



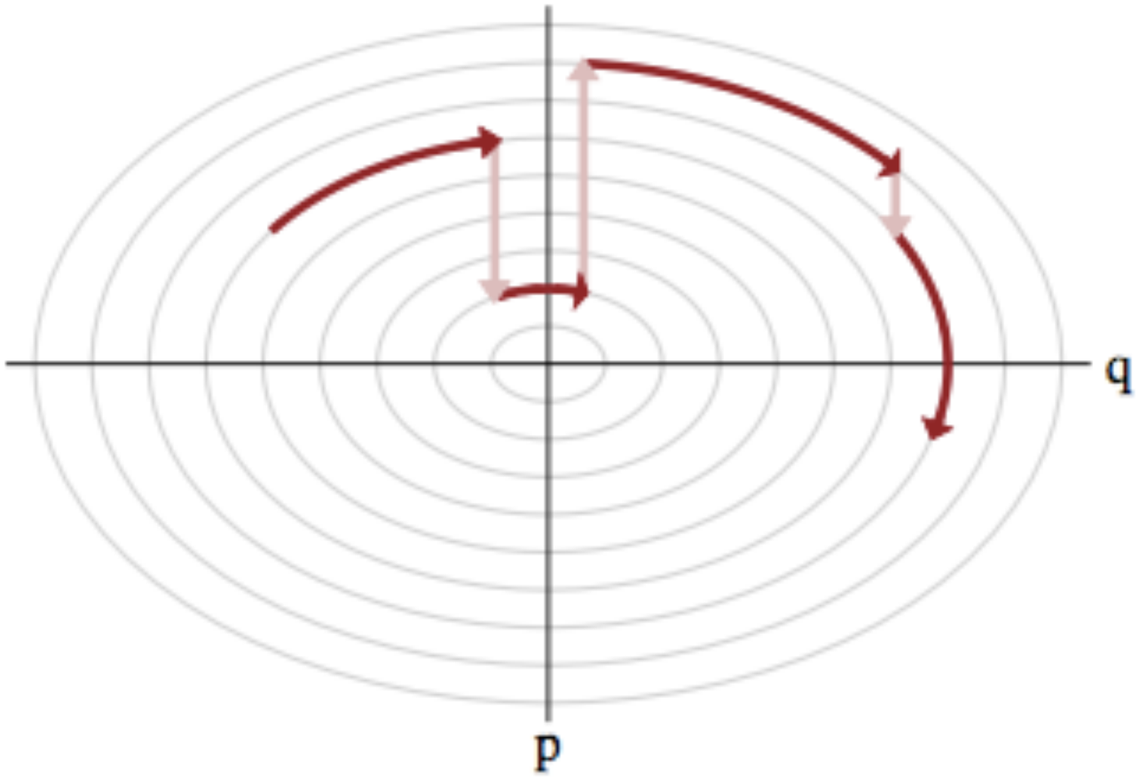
Autocorrelation: HMC vs MH






```
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)
```

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.

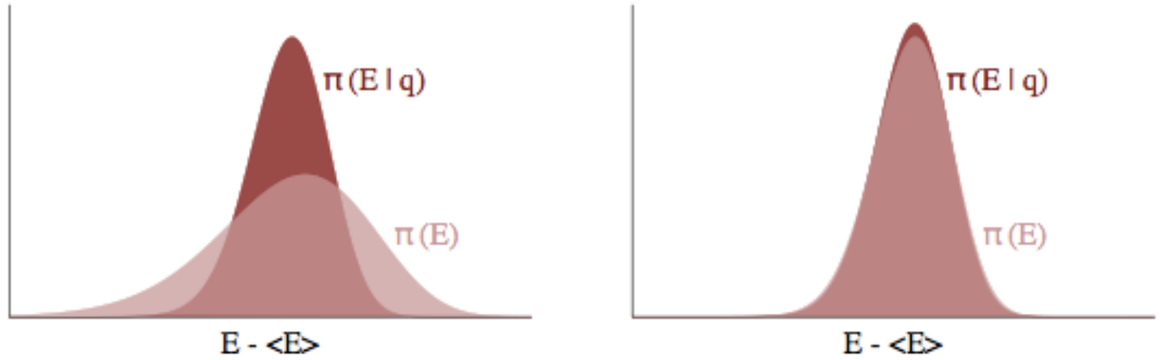
Firing the thruster moves us between level sets!

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

- thus momenta are **dual**, can use covariance as inverse mass matrix
- 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 $p \to p/\sqrt{M}$, Then $q \to q\sqrt{M}$

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

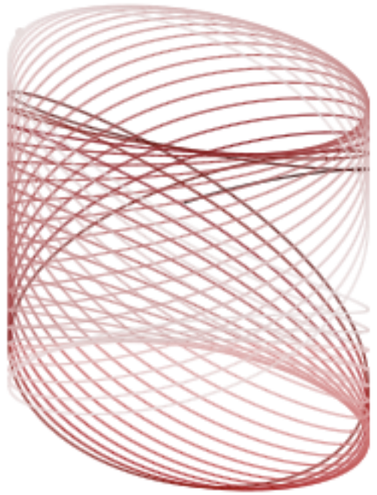
- whats the best integration time?
- should we glide for a long time? then we wont get too may samples
- if our integration was exact we could glide for arbitrary short times
- but integration is not exact and will infact take us off the level set
- thus too many samples/too short time will get us back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

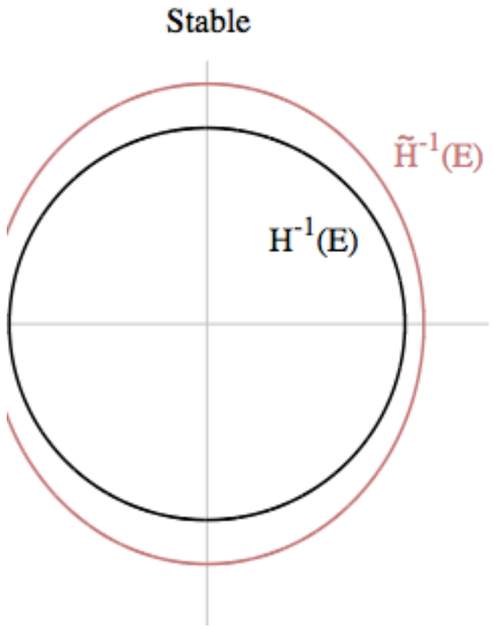
L tuning

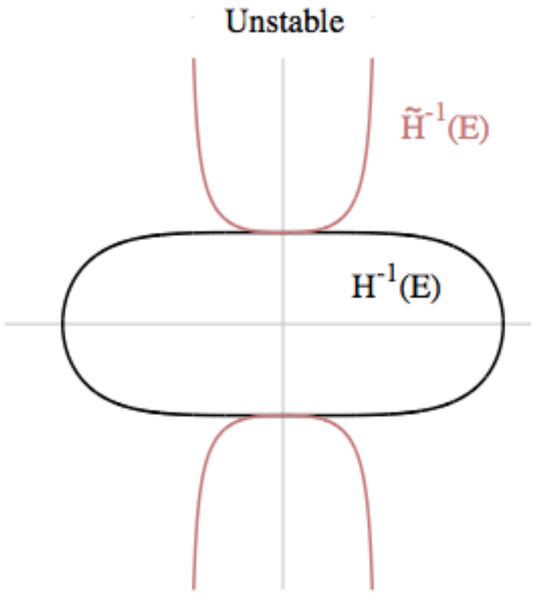
- in HMC, start $L = 100$ increase if for fixed step size, autocorrelation is too much
- Tails correspond to much higher energies, larger level-set surfaces are larger
- fixed length explores a small portion of this set before a momentum resampling takes us off.
- better to set dynamically: NUTS termination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

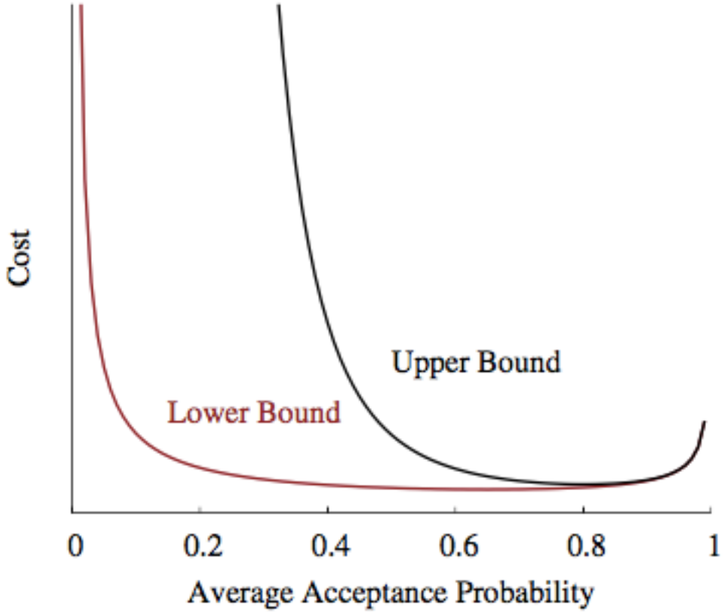


ϵ tuning

- 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

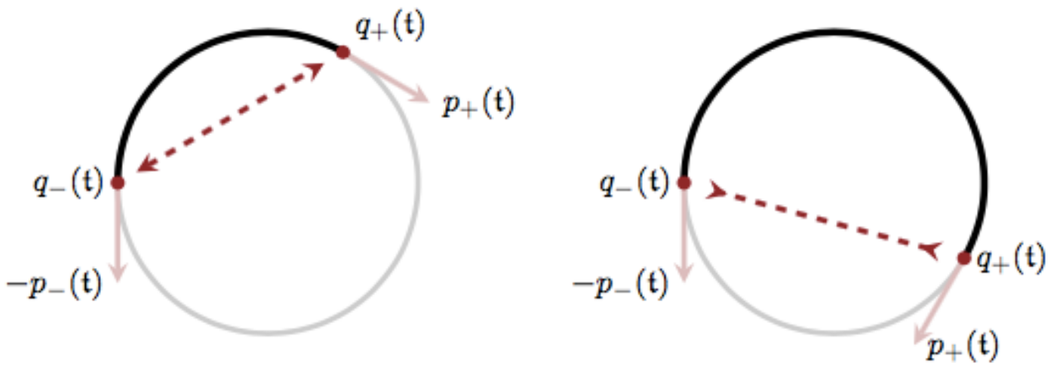

Stable
$\tilde{H}^{-1}(E)$
$H^{-1}(E)$


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

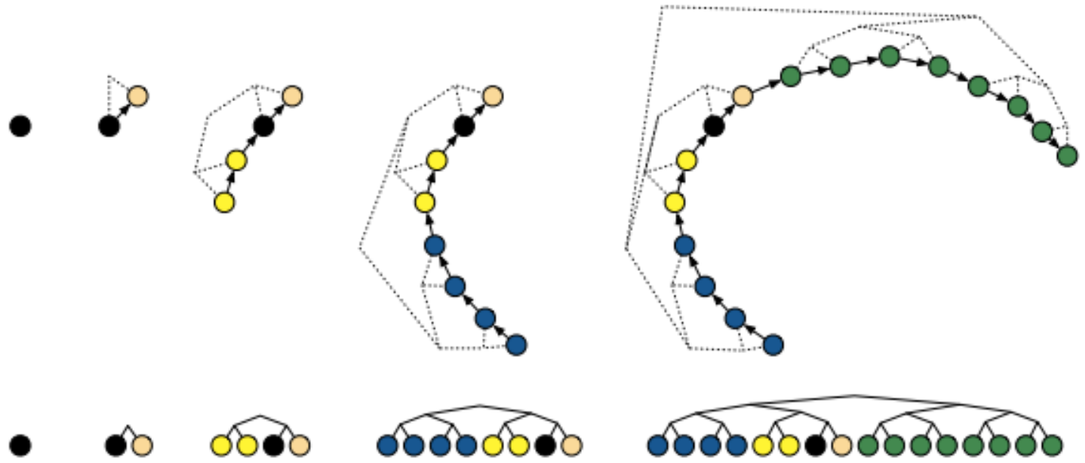

Cost
Upper Bound
Lower Bound
0
0.2
0.4
0.6
0.8
1
Average Acceptance Probability

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



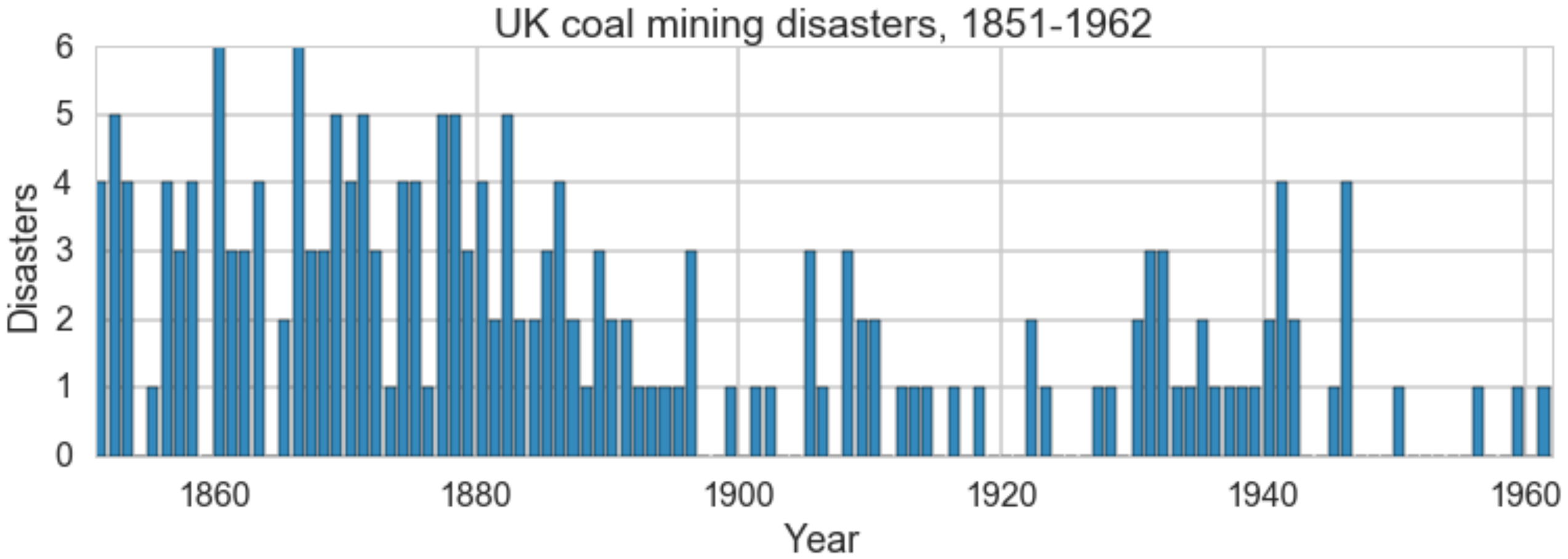
- termination criterion destroys detailed balance, must rebuild
- sample from trajectory not just endpoint
- sample backwards and forwards in time until u-turn
- choose a sample with boltzmann weights over the trajectory using multinomial or slice sampling

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)
```

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

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

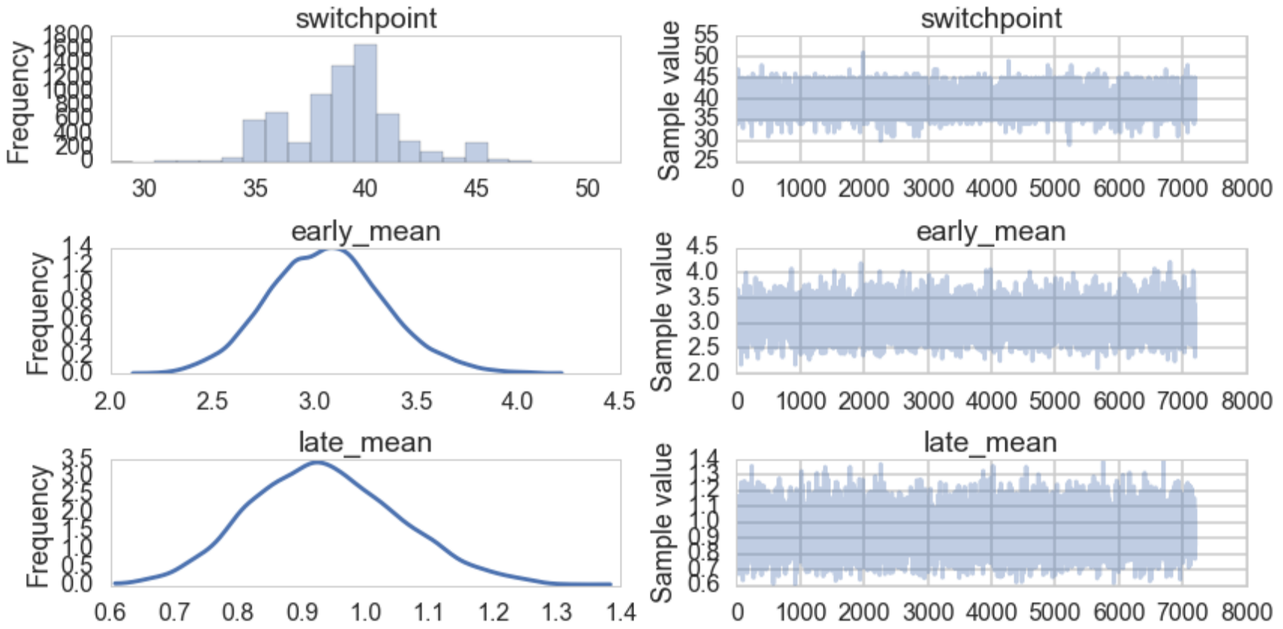
$$r_t = \lambda_1 \text{ if } t < \tau \text{ else } \lambda_2 \text{ for } t \in [t_l, t_h]$$
$$\tau \sim DiscreteUniform(t_l, t_h)$$

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

```
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]
```



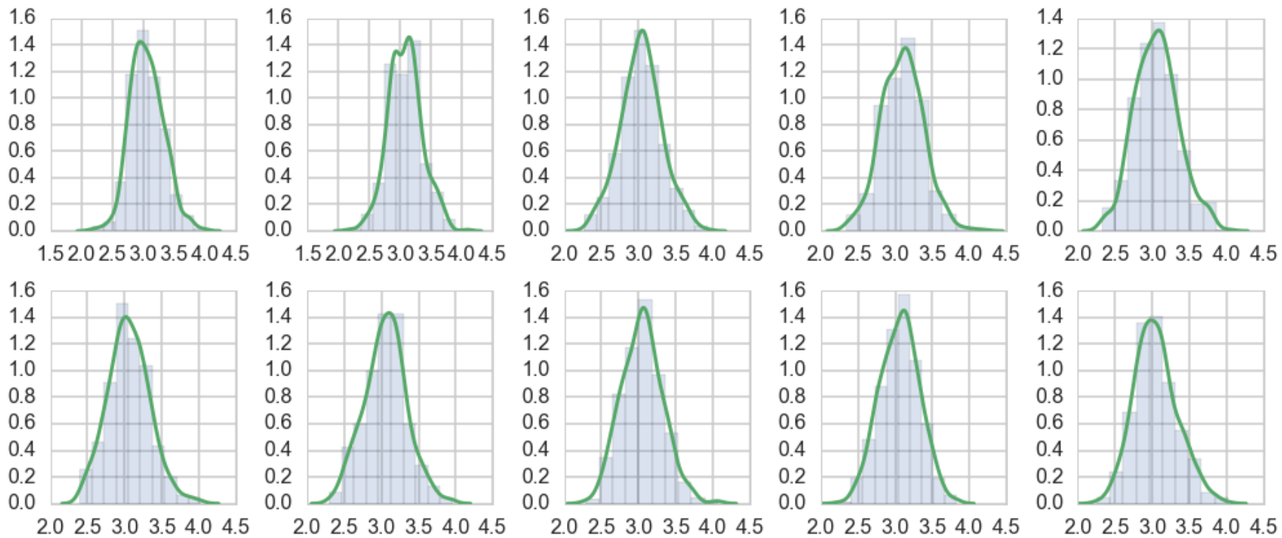
```
>>>coaldis1.vars #stochastics
[early_mean_log_, late_mean_log_, switchpoint]
>>>coaldis1.deterministics #deterministics
[early_mean, late_mean]
>>>coaldis1.observed_RVs
[disasters]
>>>ed=pm.Exponential.dist(1)
<class 'pymc3.distributions.continuous.Exponential'>
>>>ed.random(size=10)
array([ 1.18512233,  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(-4.71849887129509 4)
```

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

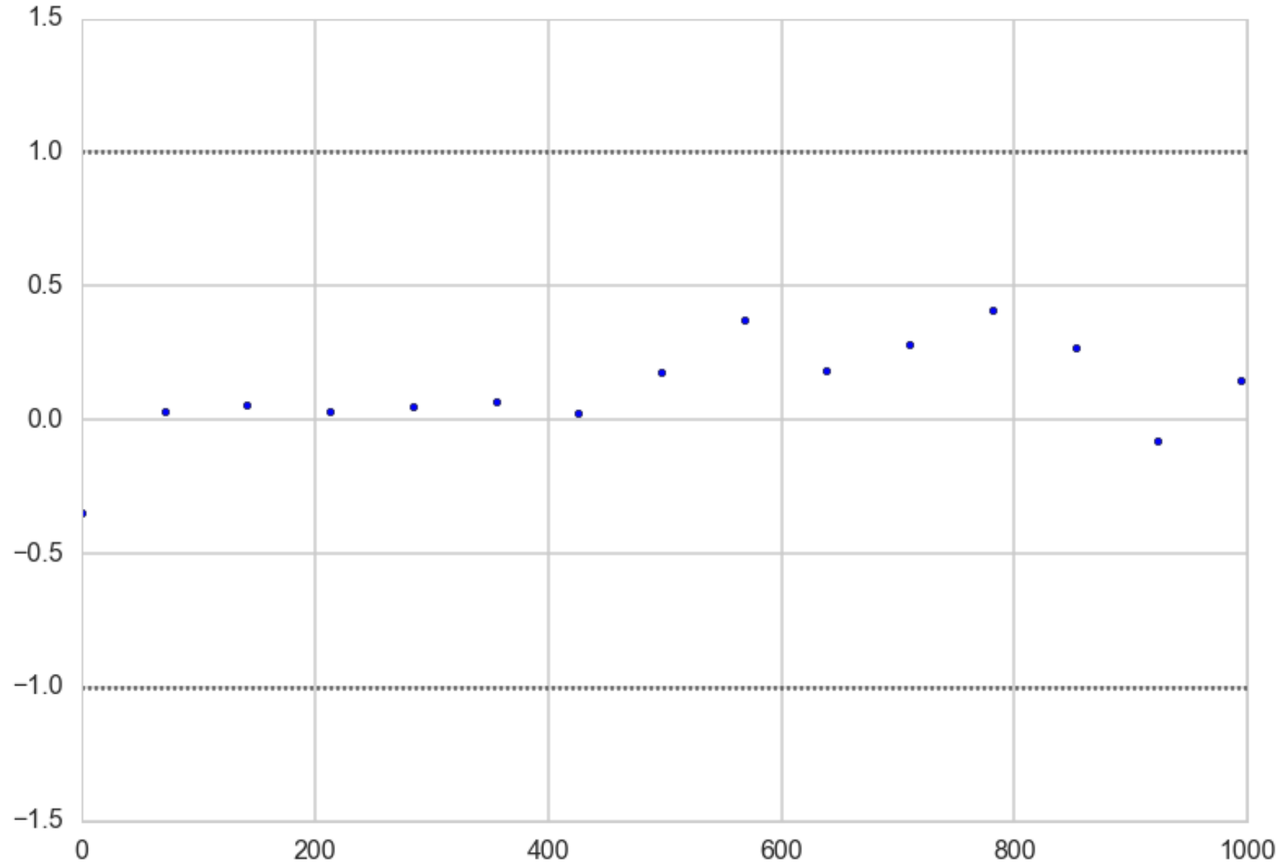


Gewecke: difference of means

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

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

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




```
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)
```

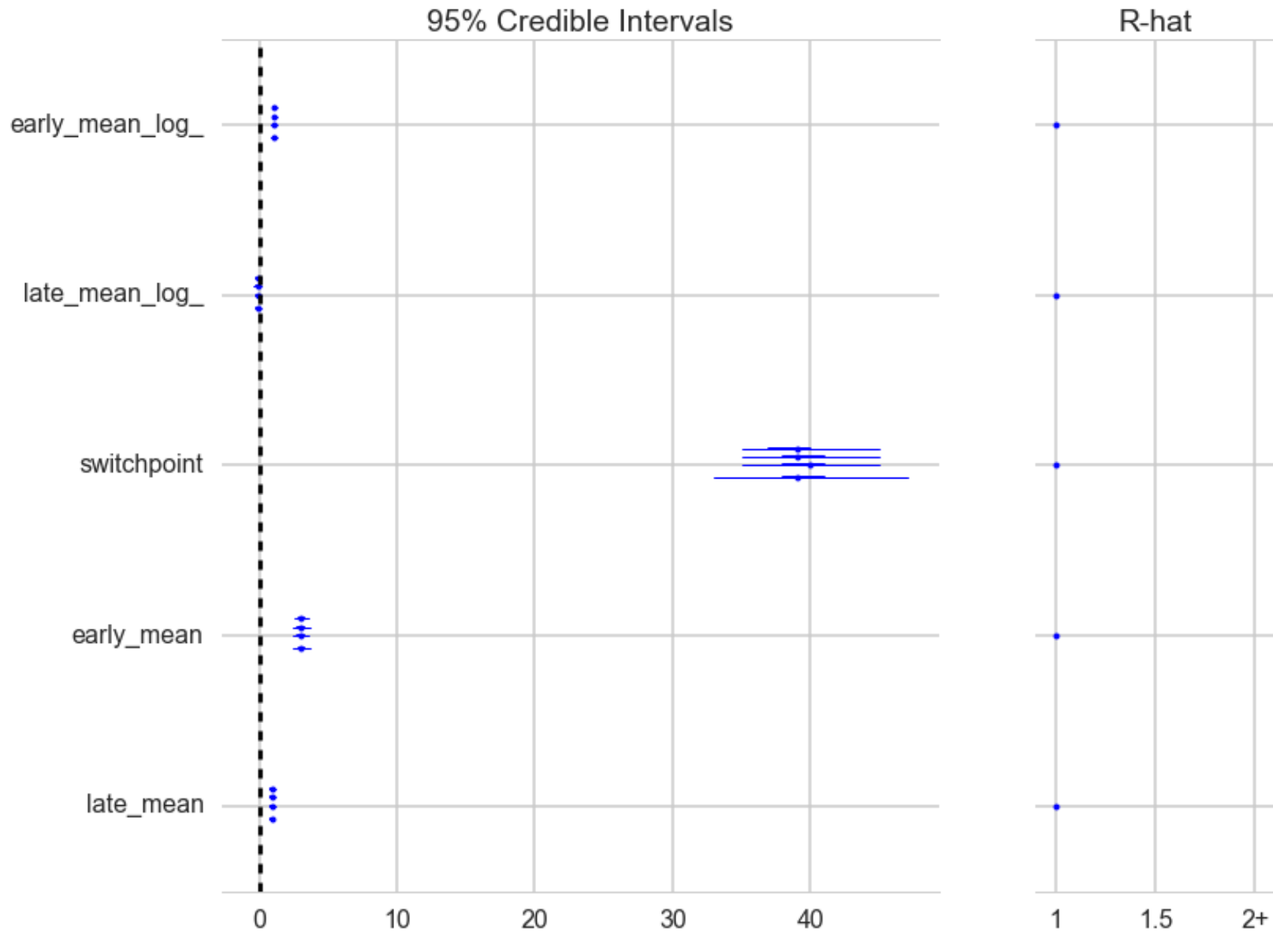
Gelman-Rubin

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)`:

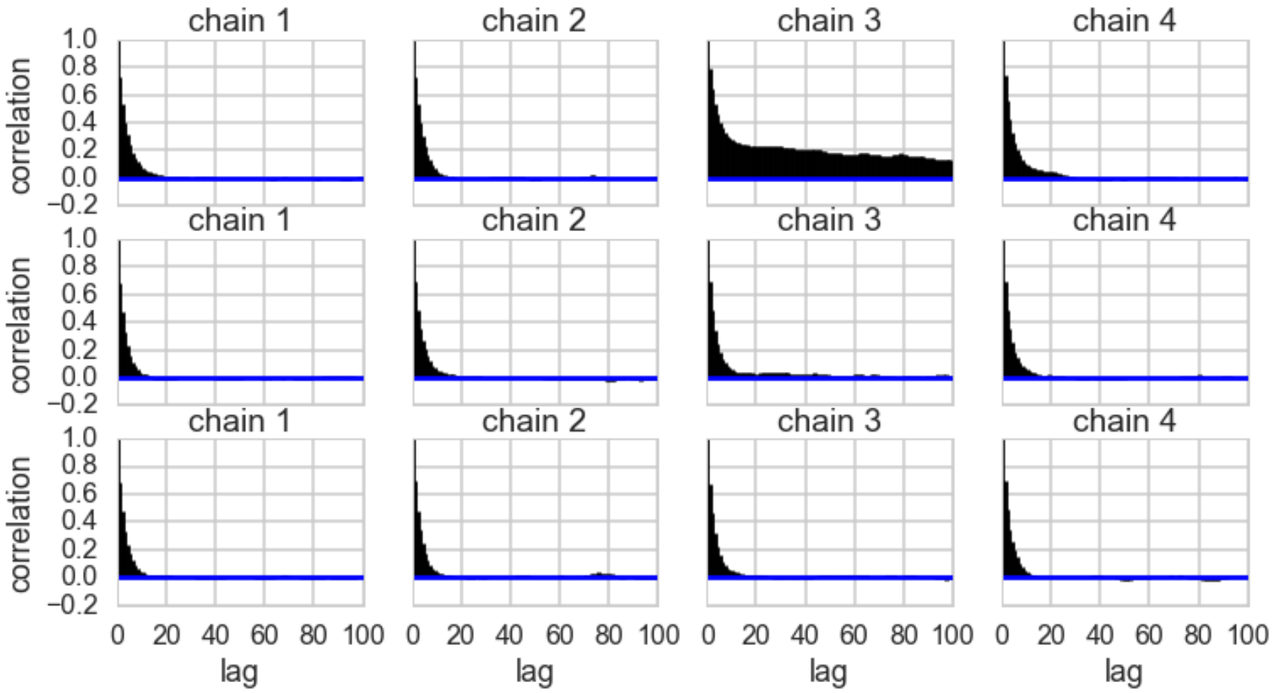
$$\hat{Var}(\theta) = (1 - \frac{1}{n})w + \frac{1}{n}B$$

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)

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