Lecture 15

Recap (contd)

Slice Sampling, and

HMC

Simplest form of a DA algo:

1. Draw $Y \sim p_{Y|X}(.|x)$ and call the observed value y

2. Draw $X_{n+1} \sim p_{X|Y}(.|y)$

3. Histo the X

Pymc3

Coal disasters Model:

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

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

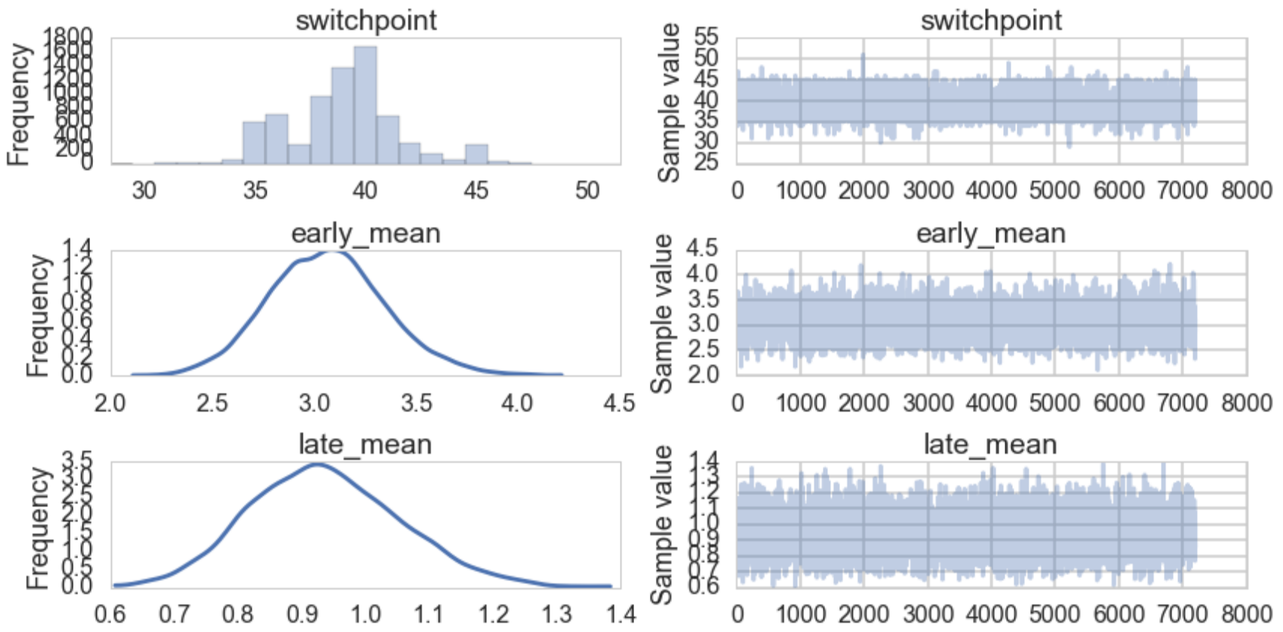
$$\lambda_1 \sim Exp(a)$$
$$\lambda_2 \sim Exp(b)$$

```
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)
```

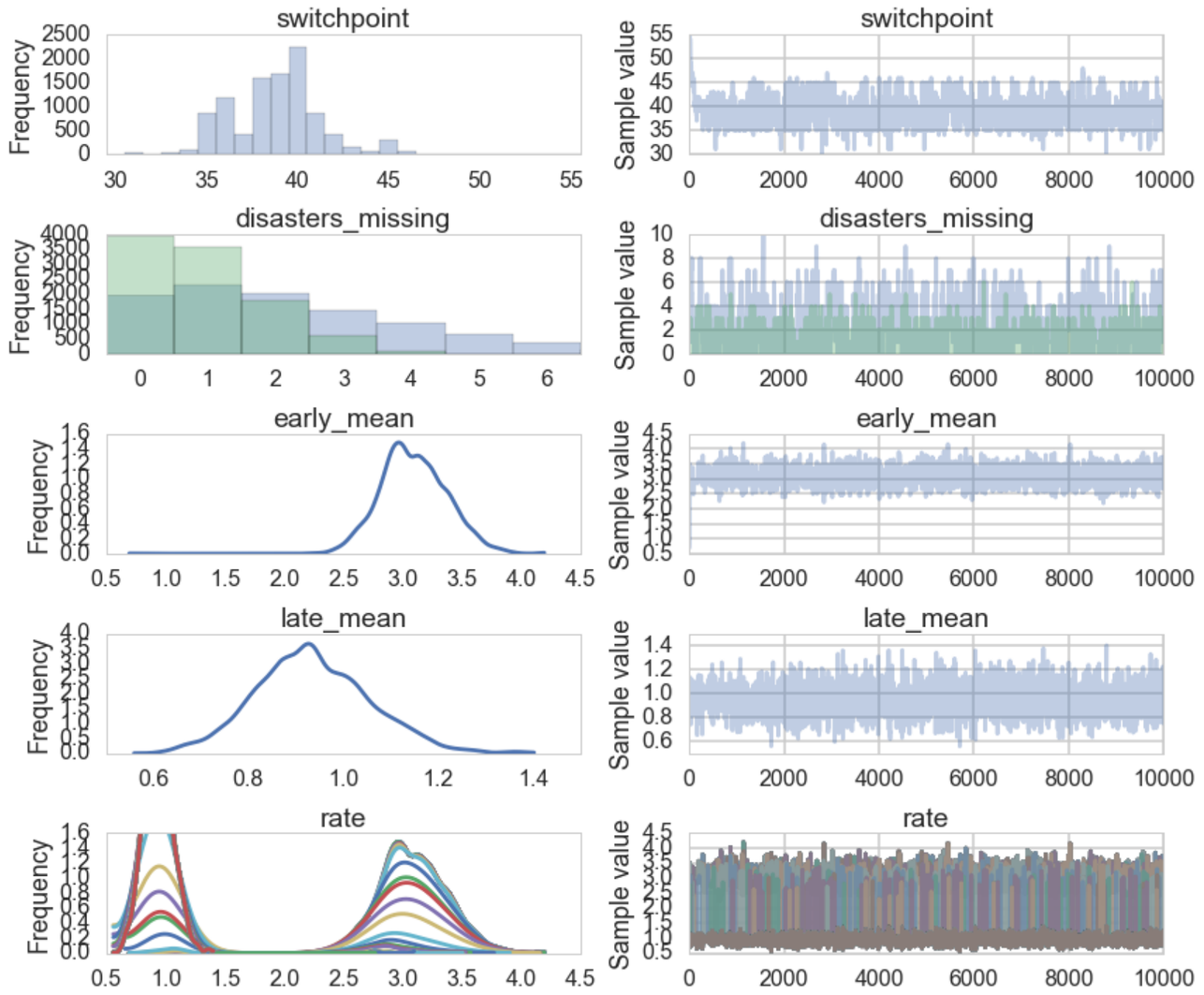
```
100%|██████████| 40000/40000 [00:12<00:00, 3326.53it/s] | 229/40000 [00:00<00:17, 2289.39it/s]
```

```
>>>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(-4.718498871295094)
```

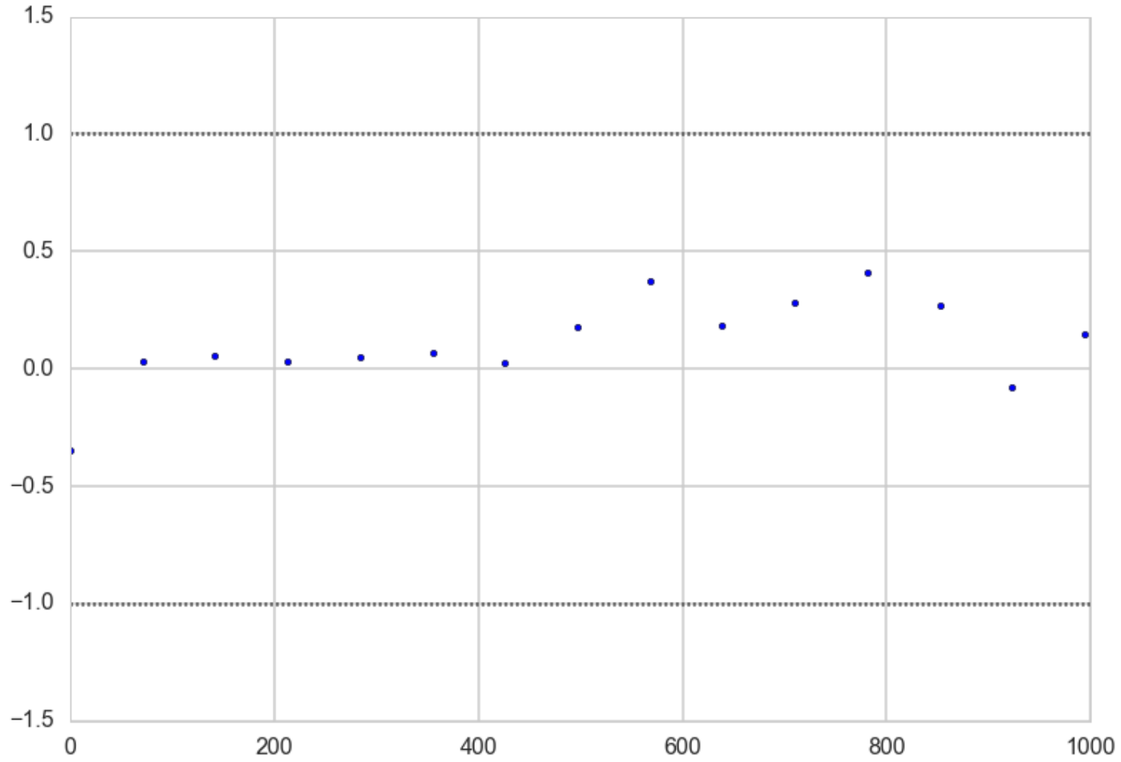


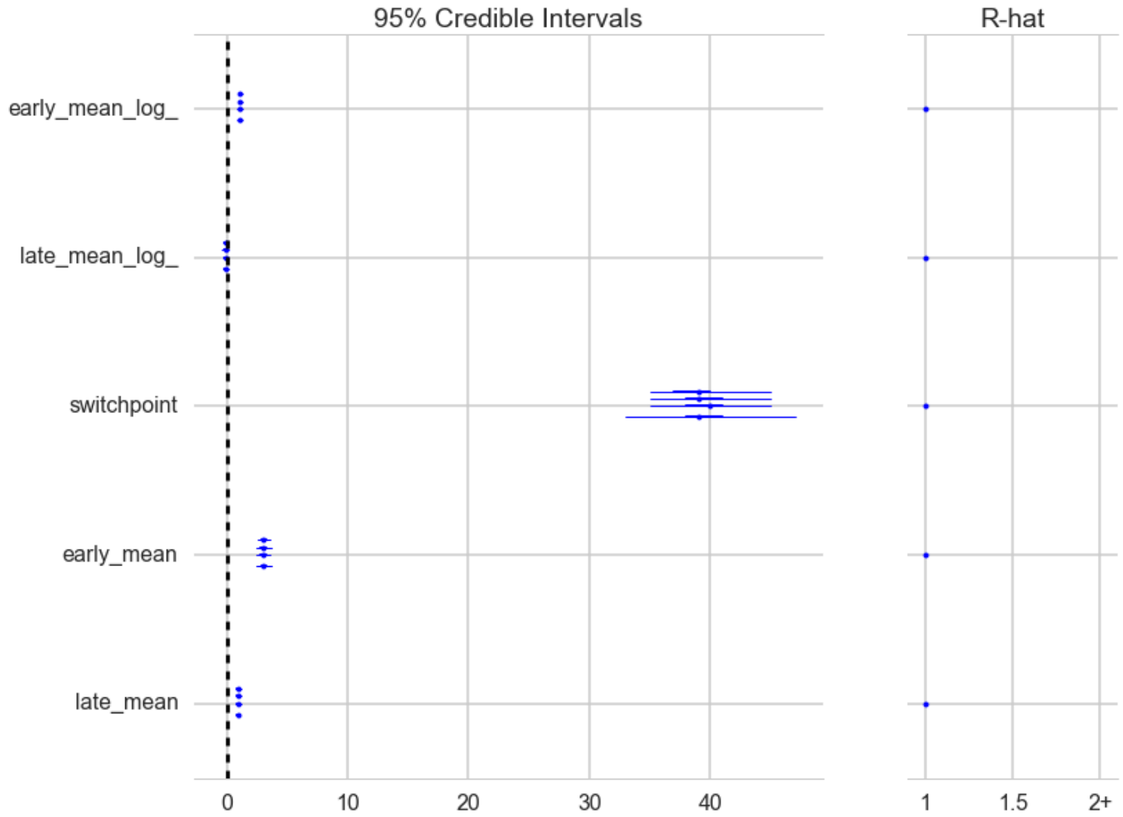
```python
with pm.Model() as missing_data_model:
    switchpoint = pm.DiscreteUniform('switchpoint', lower=0, upper=len(disasters_masked))
    early_mean = pm.Exponential('early_mean', lam=1.)
    late_mean = pm.Exponential('late_mean', lam=1.)
    idx = np.arange(len(disasters_masked))
    rate = pm.Deterministic('rate',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



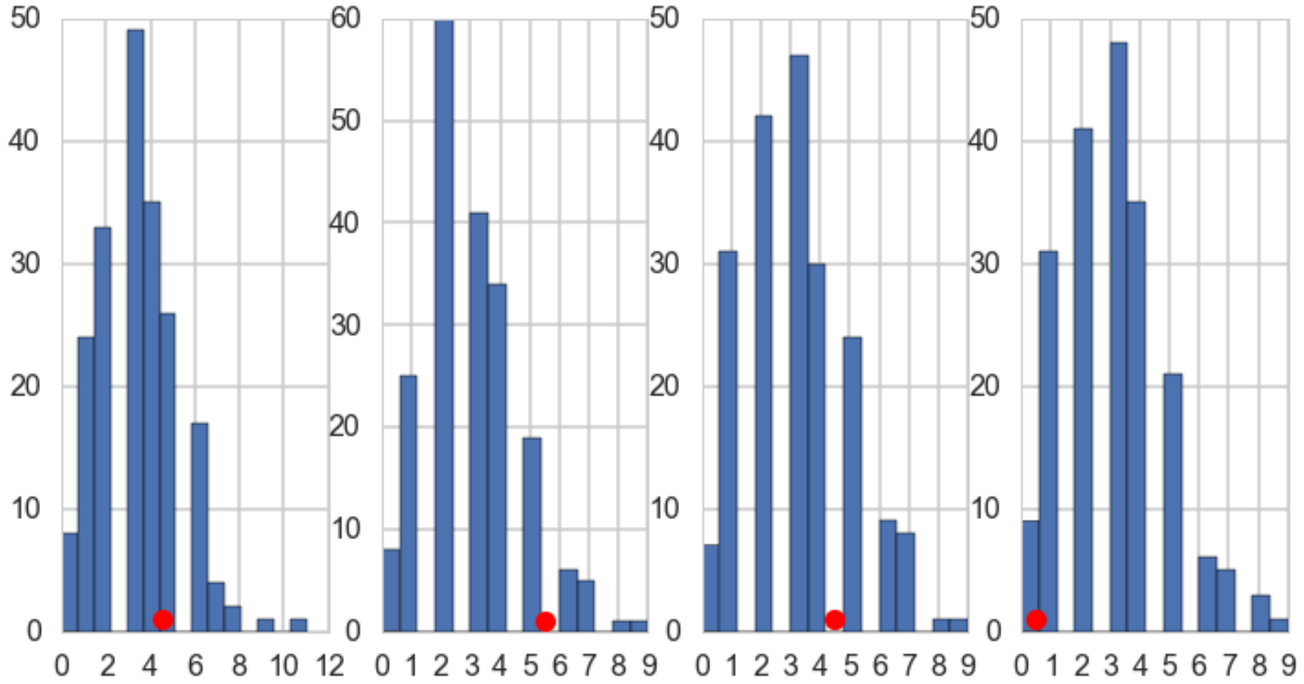
Gewecke and Gelman Rubin



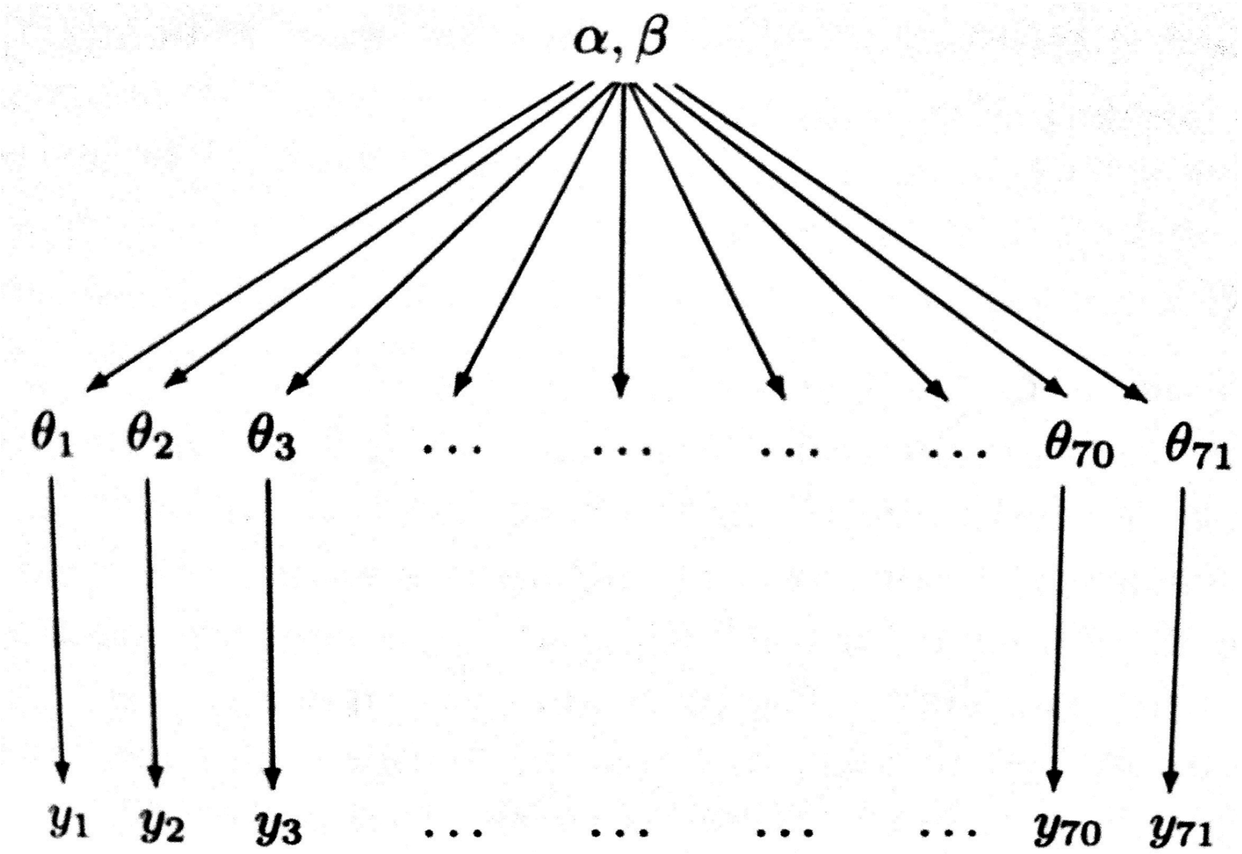


Posterior Predictive Checks

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



Hierarchical models



Key Idea: Share statistical strength

- Some **units** (experiments) statistically more robust
- Non-robust experiments have smaller samples or outlier like behavior
- Borrow strength from all the data as a whole through the estimation of the hyperparameters
- **regularized partial pooling model** in which the "lower" parameters (θs) tied together by "upper level" hyperparameters.

First idea: estimate directly from data

Posterior-predictive distribution, as a function of upper level parameters $\eta = (\alpha, \beta)$.

$$p(y^*|D, \eta) = \int d\theta \, p(y^*|\theta) \, p(\theta|D, \eta)$$

A likelihood with parameters η and simply use maximum-likelihood with respect to η to estimate these η using our "data" y^*

Full Sampling

- Fix α and β, we have a Gibbs step for all of the θ_is
- For α and β, everything else fixed, use stationary metropolis step, as conditionals not simply sampled distributions
- when we sample for α, we will propose a new value using a normal proposal, while holding all the θs and β constant at the old value. ditto for β.
- OR just specify in pymc and go!

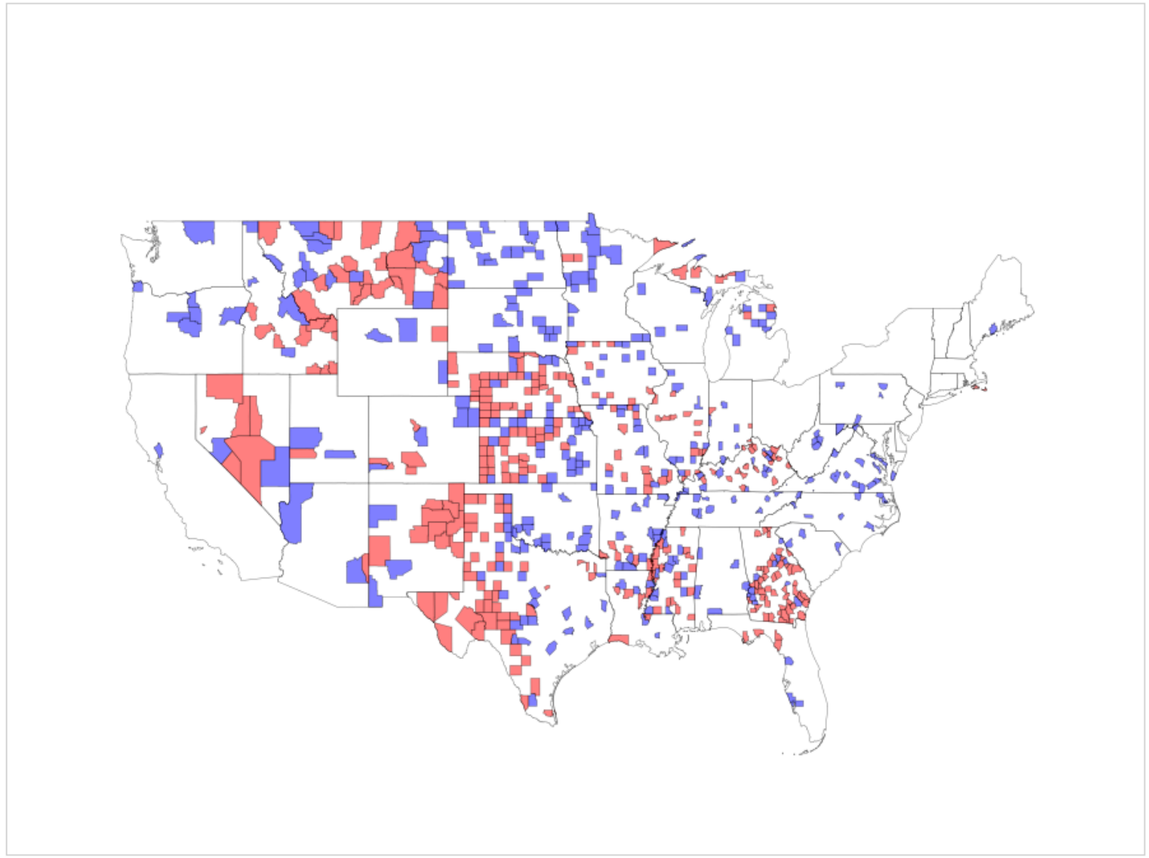
Howto Hierarchical models

- a DAG, with observations at the bottom of a tree, next layer intermediate parameters, upper layers hyper-parameters
- sample conditionals from parents up the tree.
- the y_j were exchangeable since we had no additional information about experimental conditions.
- if specific groups of experiments came from specific laboratories, assume experiments interchangeable if from the same lab.

Tumors in pymc3

```
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:
    #obtain starting values via MAP
    start = find_MAP(model=tumor_model)
    # instantiate sampler
    step = pm.Metropolis()
    # draw 2000 posterior samples
    tumor_trace = pm.sample(500000, step=step, start=start)
```

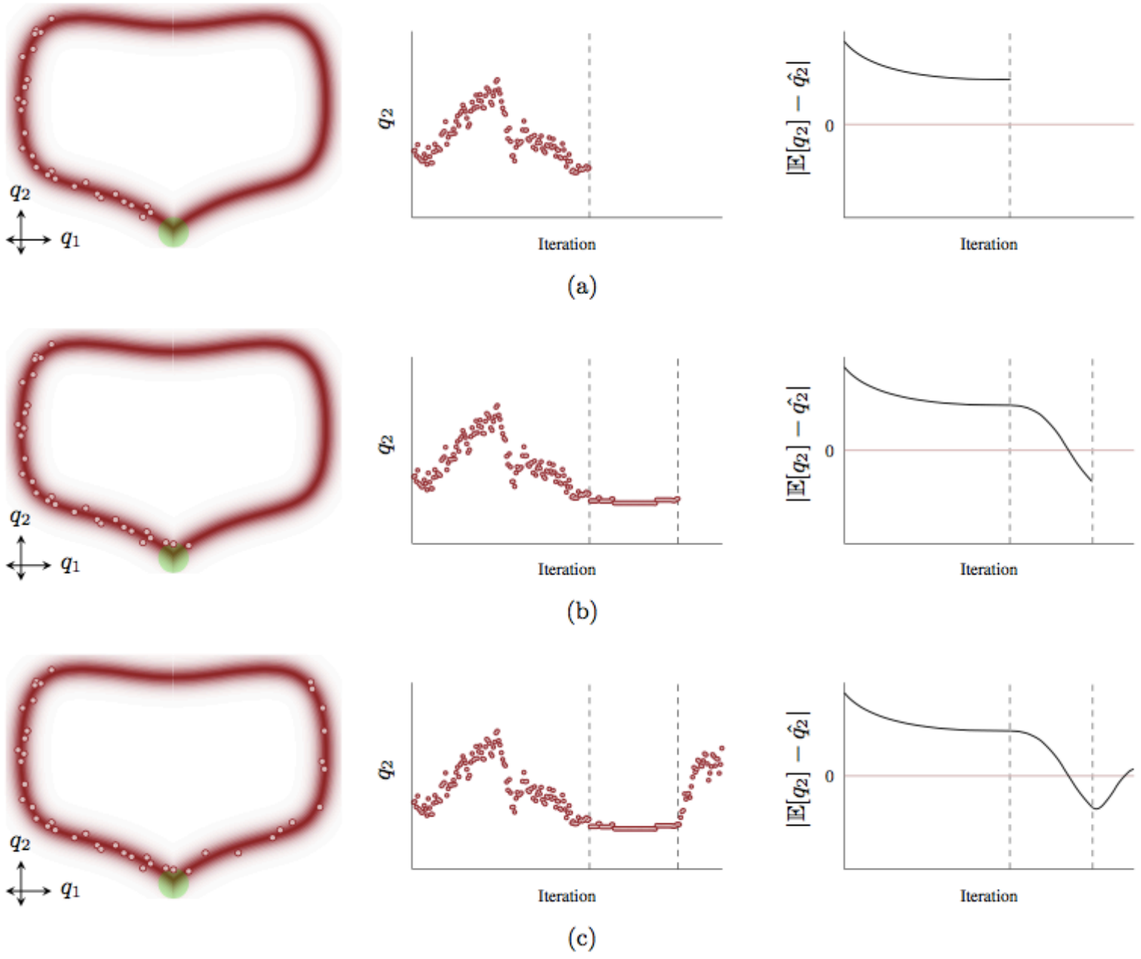
Homework



Levels of Bayes

Method	Definition
Maximum Likelihood	$\hat{\theta} = argmax_\theta p(D\|\theta)$
MAP estimation	$\hat{\theta} = argmax_\theta p(D\|\theta)p(\theta\|\eta)$
ML-2 (Empirical Bayes)	$\hat{\eta} = argmax_\eta \int d\theta\, p(D\|\theta)p(\theta\|\eta) = argmax_\eta p(D\|\eta)$
MAP-2	$\hat{\eta} = argmax_\eta \int d\theta\, p(D\|\theta)p(\theta\|\eta)p(\eta) = argmax_\eta p(D\|\eta)p(\eta)$
Full Bayes	$p(\theta, \eta\|D) \propto p(D\|\theta)p(\theta\|\eta)p(\eta)$

Problems with MCMC

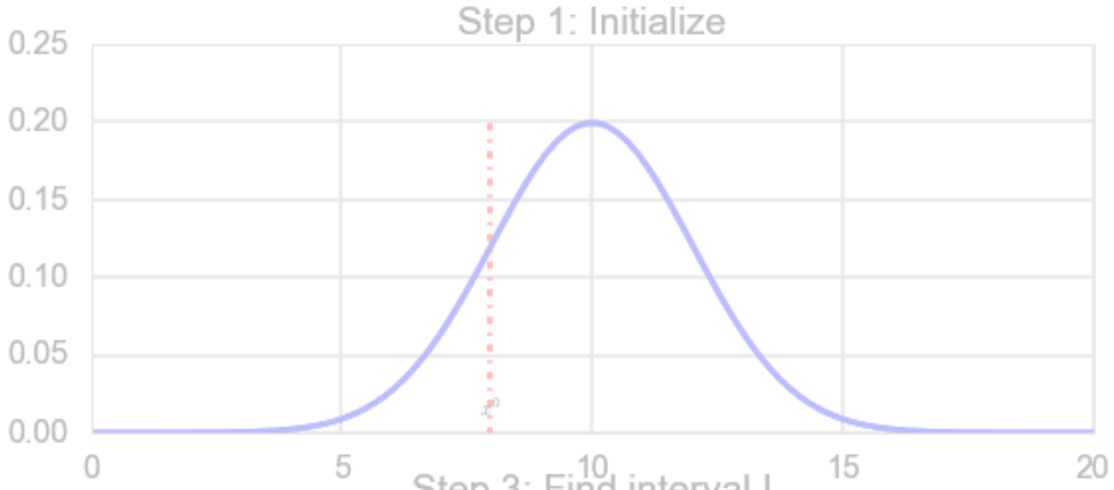


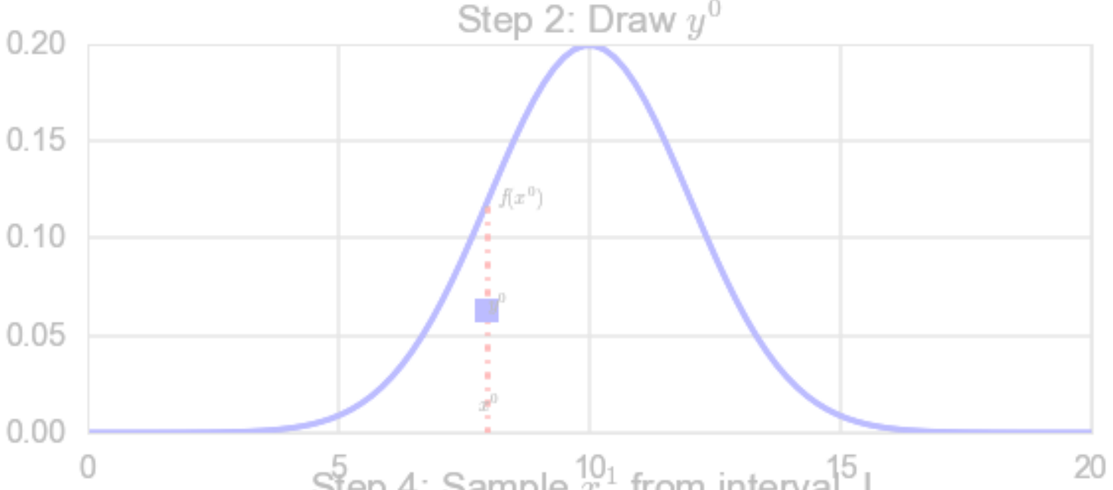
- overshoot and oscillate at pinches
- need to specify step step sizes
- large steps go outside typical set and are not accepted
- small steps accepted but go nowhere
- large correrlations

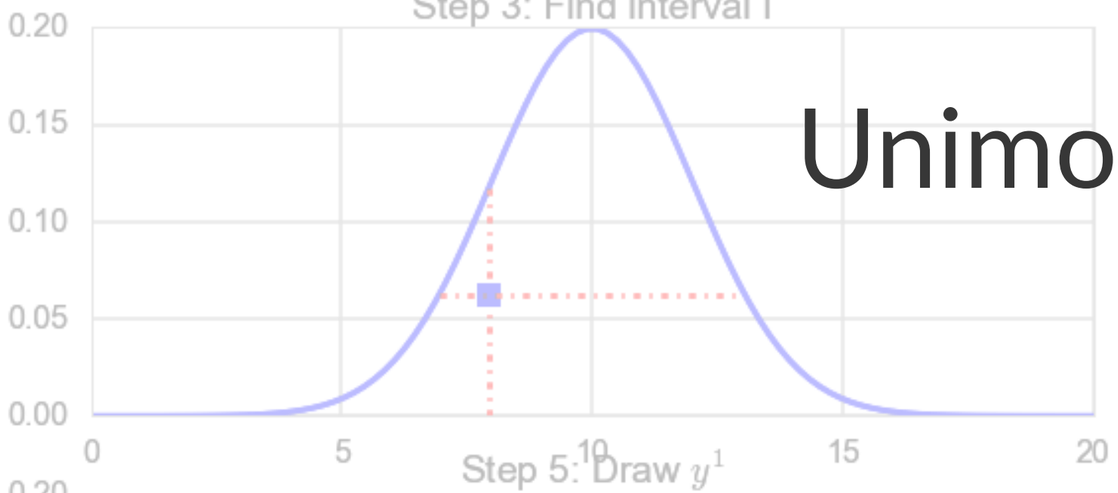
SLICE

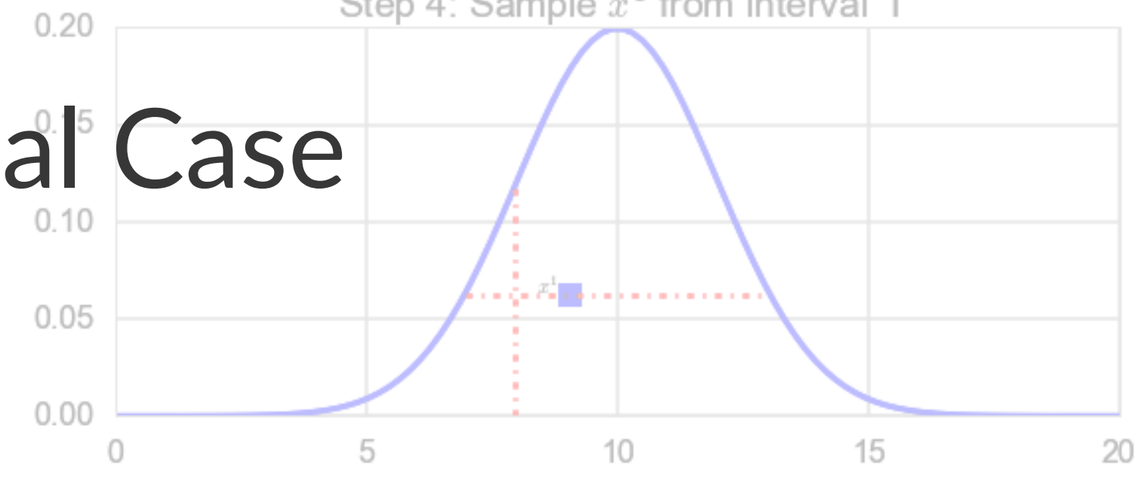
to the rescue

- Pick an initial point x_0 from our posterior
- Draw y_0 from U(0, f(x_0))
- Repeat for N samples
  - Select the interval (e.g. stepping out, etc)
  - Sample x_i from that interval (e.g. shrinkage)
  - Draw y_i from U(0, f(x_i))

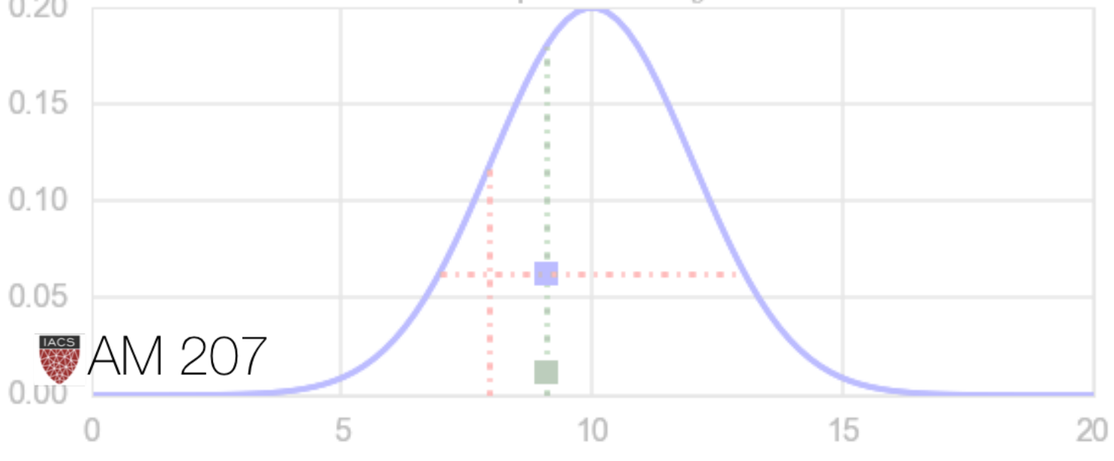




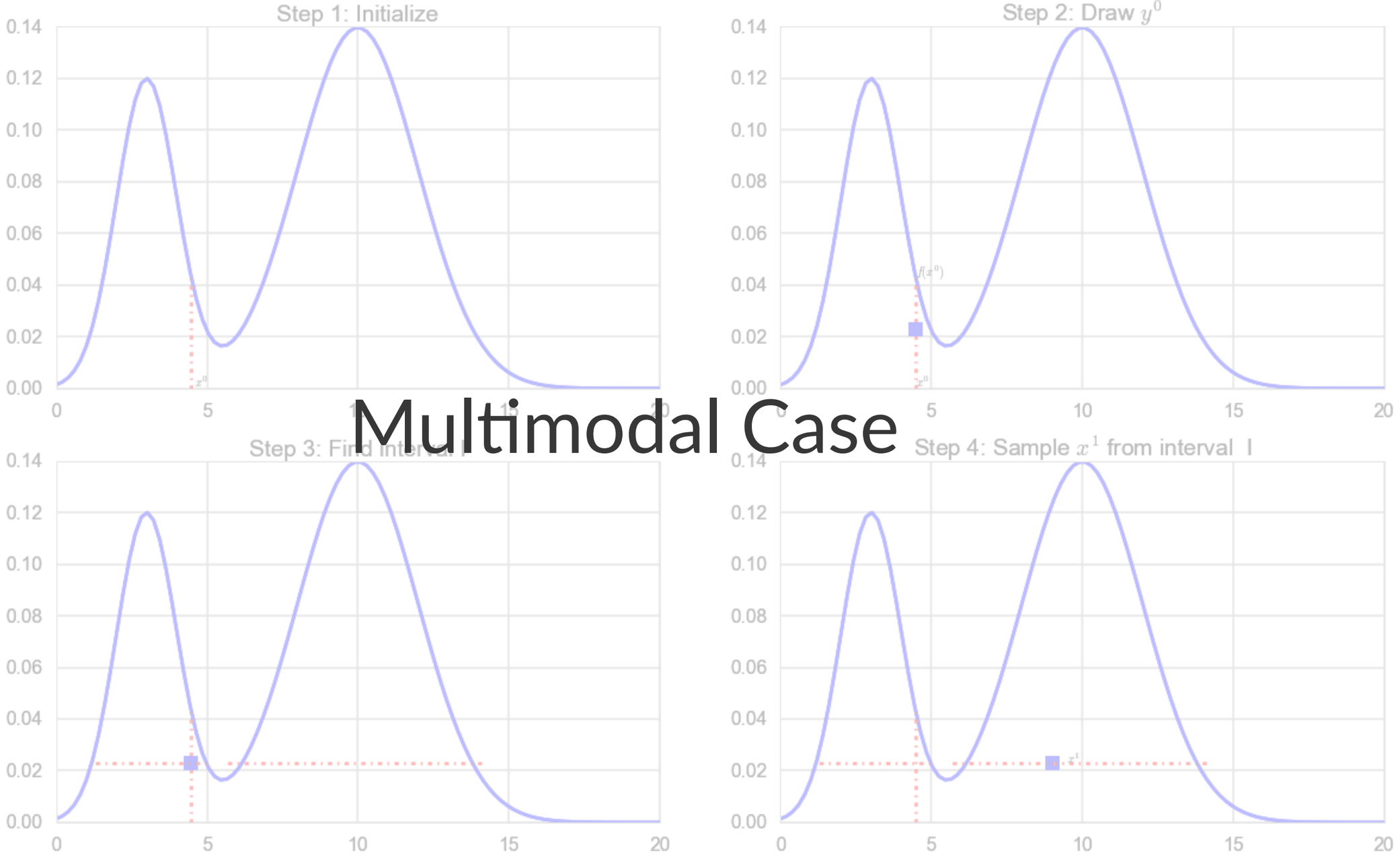




Unimodal Case



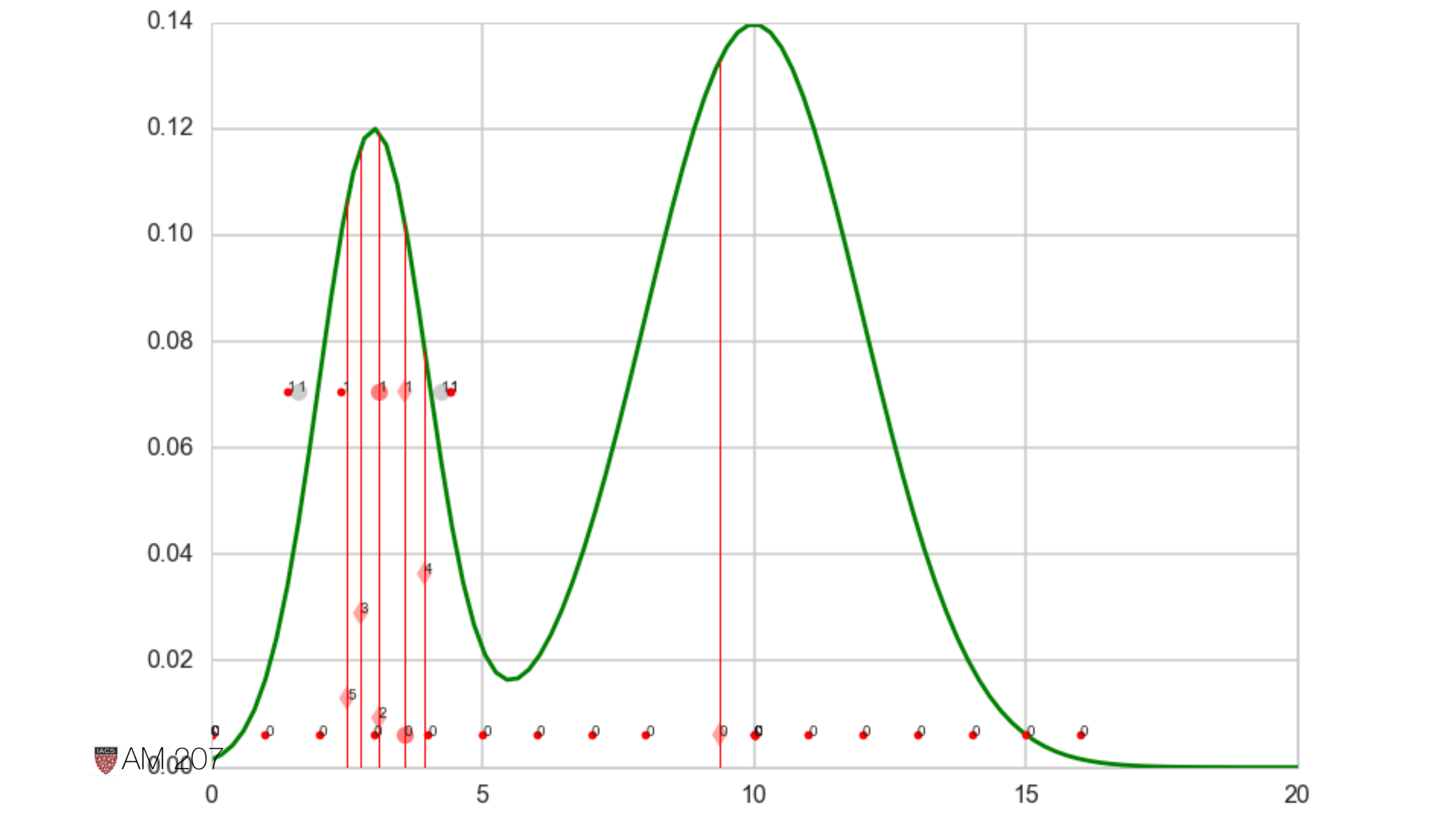
Multimodal Case



Stepping Out

- set w width and draw u ~ Unif(0,1)
- set L = $x^{(0)} - wu, R = L + w$ (so $x^{(0)}$ lies in [L, R])
- while y < f(L) (here's where we extend left interval) L = L - w
- while y < f(R) (here's where we extend the right interval) R = R + w

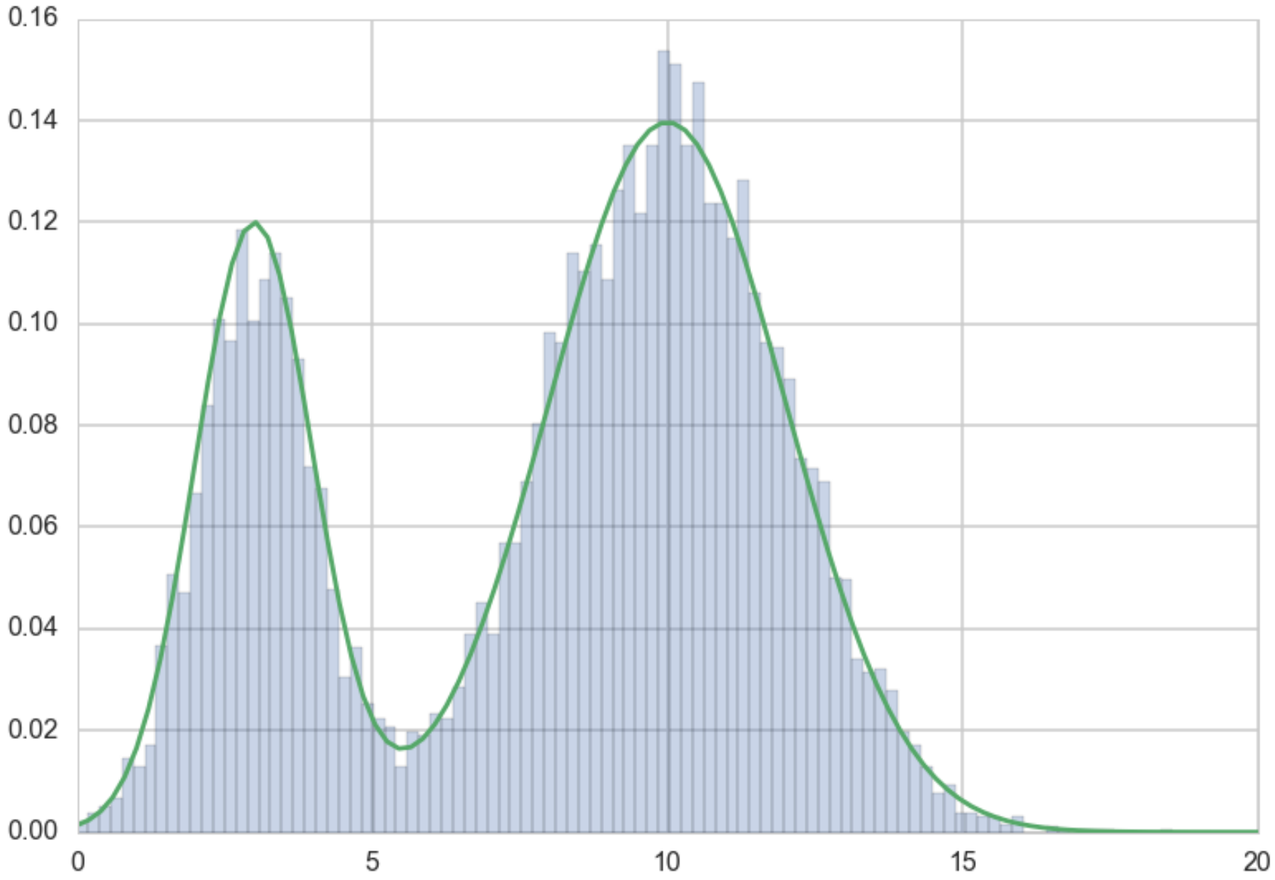
The final interval will be larger than S.


0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20

Shrinkage

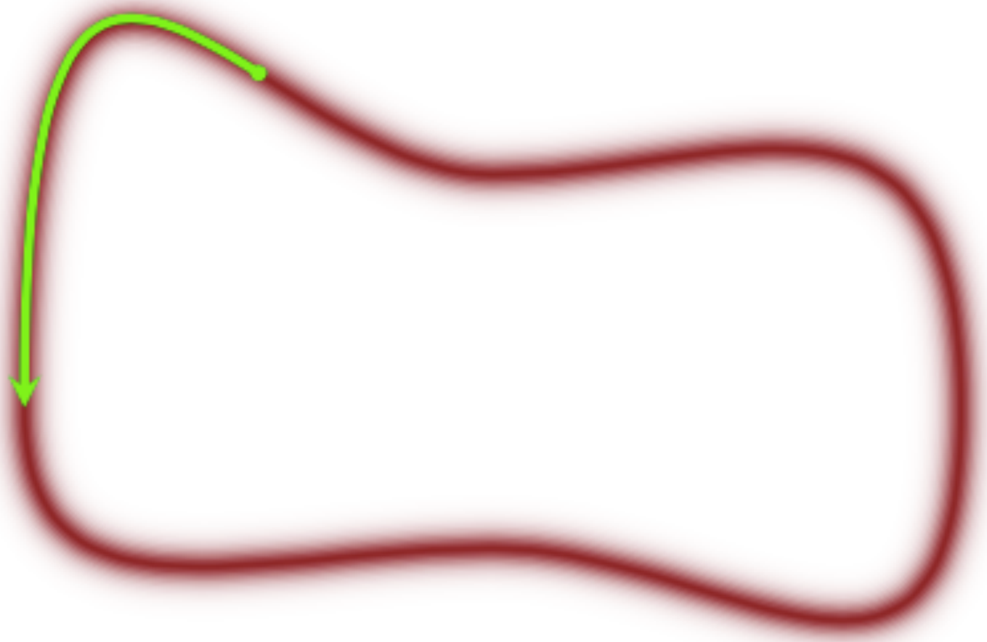
- start with interval $I = (L, R)$
- current sample is $x^{(k)}$ and $y^{(k)}$
- repeat until loop exits
  - sample x^* uniformly from (L, R)
  - if $y^{(k)} < f(x^*)$
    - accept x* and end loop
  - else
    - if $x^* < x^{(k)} \rightarrow L = x^*$
    - if $x^* > x^{(k)} \rightarrow R = x^*$

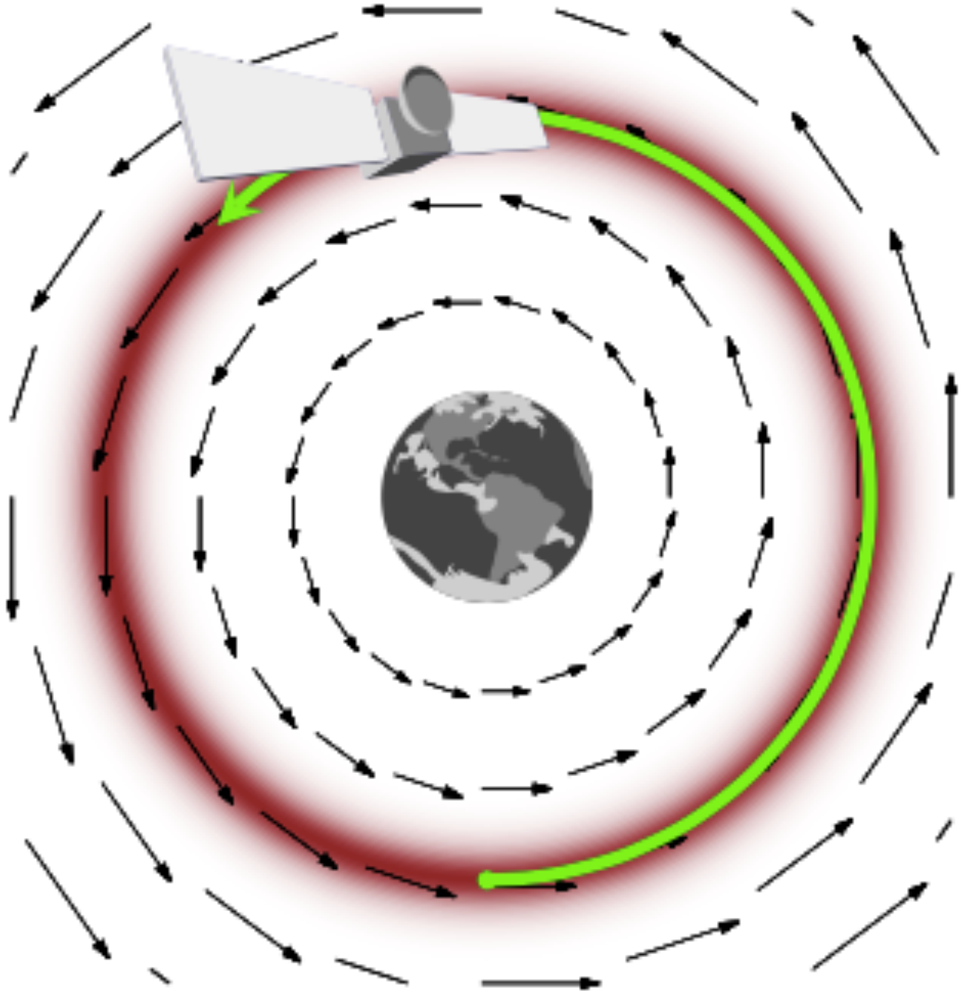
```
w=1.0
L=0; R=0;
x_prev = np.random.uniform(low=0, high=17)
iters=10000
trace=[]
kmax=1
for k in range(iters):
    y_samp = np.random.uniform(low=0, high=fun(x_prev))
    # widen left
    U = np.random.rand()
    L=x_prev-U*w
    R=x_prev+w*(1.0-U)
    while fun(L)>y_samp:
        L = L-w
    while fun(R)>y_samp:
        R = R+w
    #now propose new x on L,R

    while 1:
        x_prop= np.random.uniform(low=L, high=R)
        if k <= kmax:
            print("L,R, xprop", L, R, x_prop)
        if y_samp < fun(x_prop):
            x_prev = x_prop
            trace.append(x_prop)
            break
        elif x_prop > x_prev:
                R = x_prop
        elif x_prop < x_prev:
                L = x_prop
```



Hamiltonian Monte Carlo

Need a Coherent Glide





Balance between gravity and momentum in a rocket provides it

Now, like in annealing, let
$p(p,q) = e^{-Energy}$

Carry out an augmentation with an additional momentum with the energy **Hamiltonian**

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

Canonical distribution

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

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

The choice of a kinetic energy term = choice of a conditional probability distribution over the "augmented" momentum such that:

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

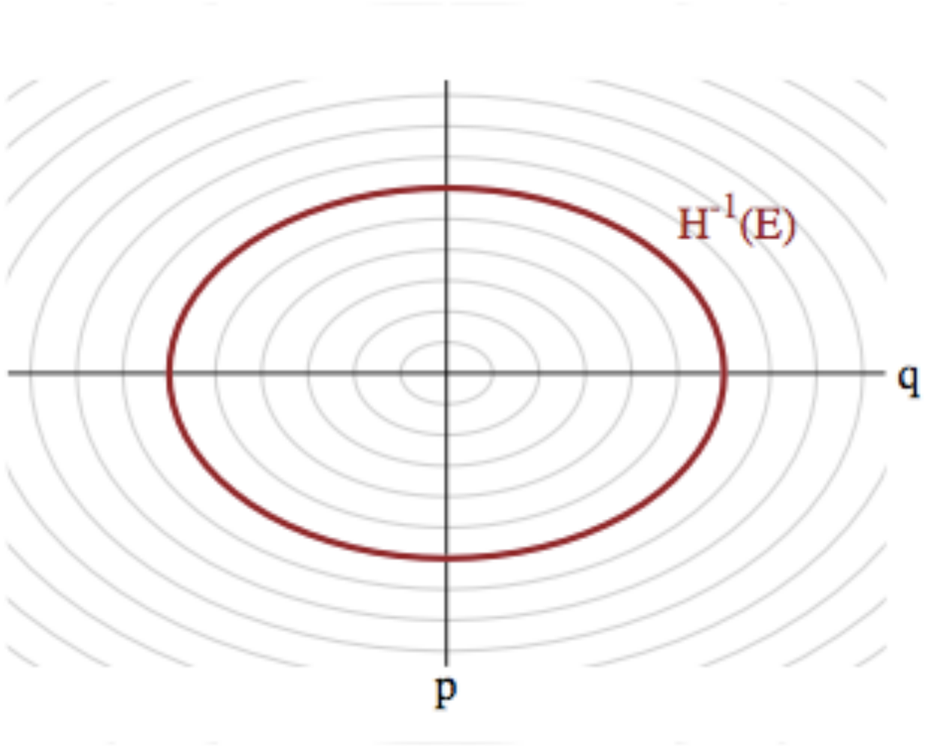
Phase Space level sets

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.



We are looking at level sets of the

Joint distribution

Why do it this way?

Because Hamiltonian flow conserves energy, leading naturally to using level sets and the

Microcanonical distribution

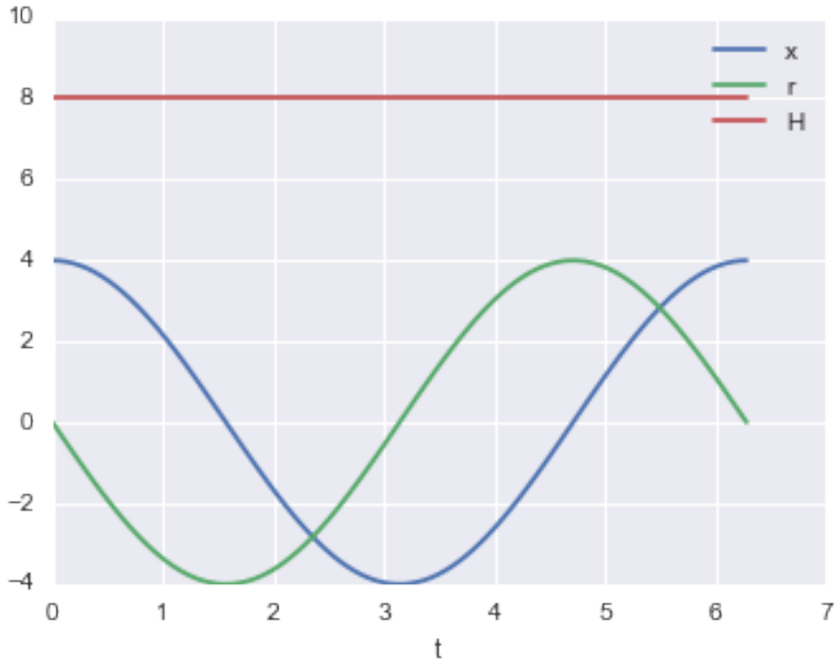
Hamiltonian Mechanics

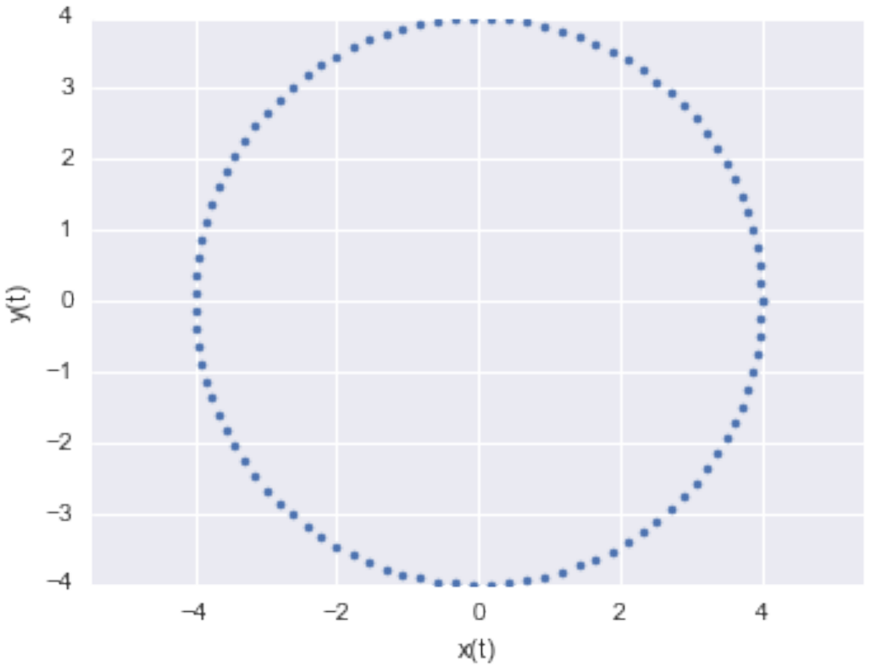
Physics equations of motion in the **Hamiltonian Formalism** set up the "glide" (over a level set).

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

$H = p^2/2m + V(q)$, $\frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial V}{\partial q} = Force$: Newton's law.

Oscillator






```
q_t = lambda t: 4. * np.cos(t)
p_t = lambda t: -4. * np.sin(t)
```

Explicitly time-independent Hamiltonian is conserved

If the Hamiltonian H doesn't have a functional dependence on time we see that

$$\frac{dH}{dt} = \sum_i \left[\frac{\partial H}{\partial q_i} \frac{dq_i}{dt} + \frac{\partial H}{\partial p_i} \frac{dp_i}{dt} \right] + \frac{\partial H}{\partial t}$$

$$\frac{dH}{dt} = \sum_i \left[\frac{\partial H}{\partial q_i} \frac{\partial H}{\partial p_i} + (\frac{\partial H}{\partial p_i})(-\frac{\partial H}{\partial q_i}) \right] + \frac{\partial H}{\partial t}$$

if

$$\frac{\partial H}{\partial t} = 0, \frac{dH}{dt} = 0$$

Then

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

This time independence is crucial to reversibility: cannot figure which direction equations are being run

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:

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

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 backwords in time until you get back to the original position and momentum in phase space at time t, then flip the momentum again so it is pointing in the right direction.

This is like in the superman movie!

Volume in phase space is conserved

T_s for small change $s = \delta$ can be written as:

$$T_\delta = \begin{pmatrix} q \\ p \end{pmatrix} + \delta \begin{pmatrix} \frac{dq}{dt} \\ \frac{dp}{dt} \end{pmatrix} + O(\delta^2)$$

Jacobian:

$$\begin{bmatrix} 1 + \delta \frac{\partial^2 H}{\partial q \partial p} & \delta \frac{\partial^2 H}{\partial p^2} \\ \delta \frac{\partial^2 H}{\partial q^2} & 1 - \delta \frac{\partial^2 H}{\partial p \partial q} \end{bmatrix}$$ and thus the determinant is $1 + O(\delta^2)$.

Thus as our system evolves, any contraction or expansion in position space must be compensated by a respective expansion or compression in momentum space.

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.

Microcanonical distribution: states for given energy.

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

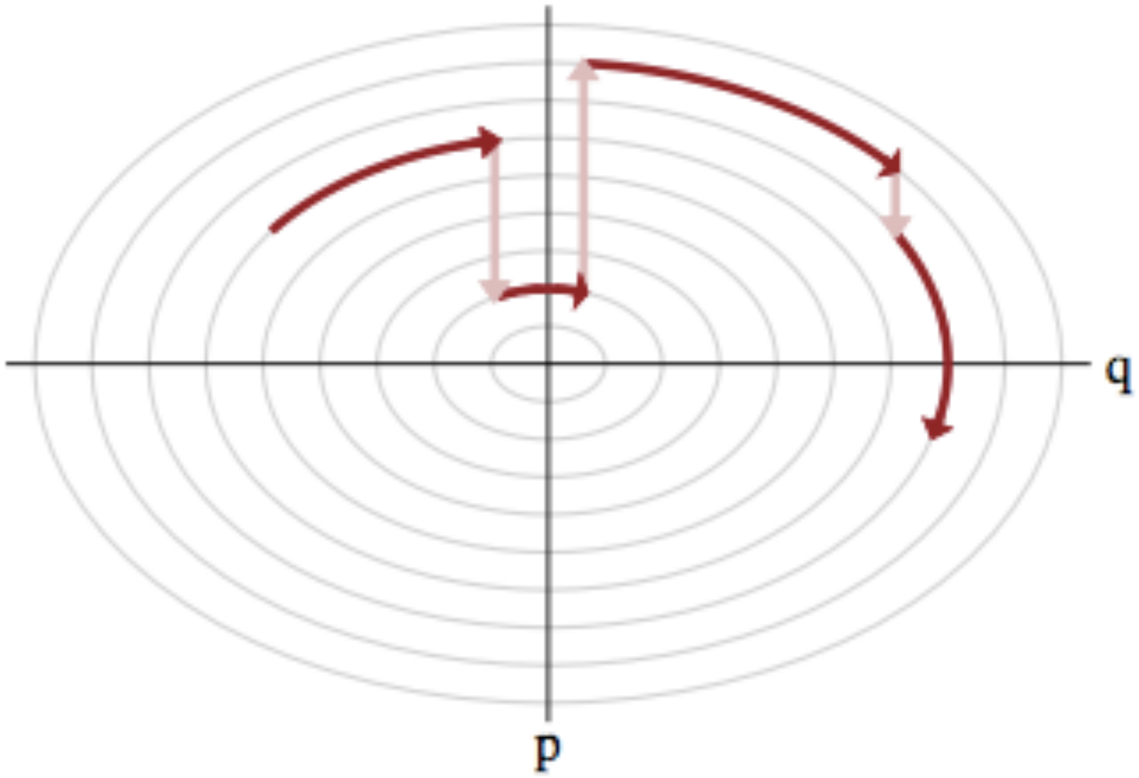
The canonical distribution can be written as a product of this microcanonical distribution and a **marginal energy distribution**:

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

where θ_E indexes the position on the level set.

Marginal Energy Distrib: probability of level set in the typical set.

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.

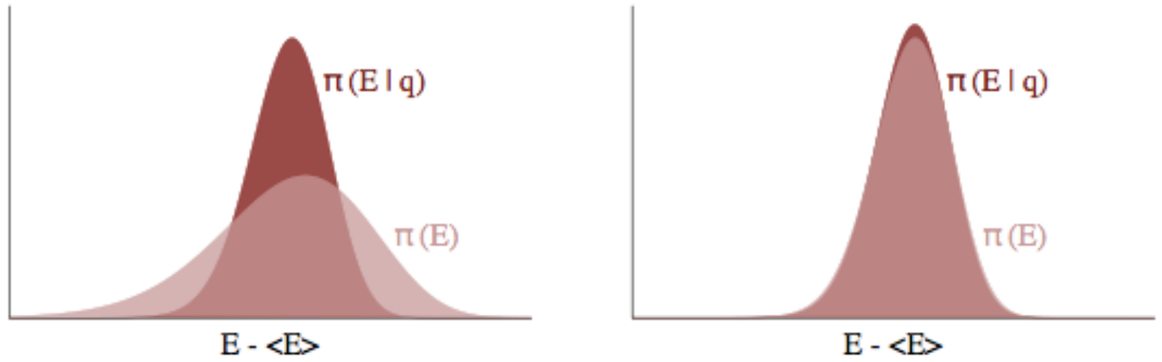
Firing the thruster moves us between level sets!

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.



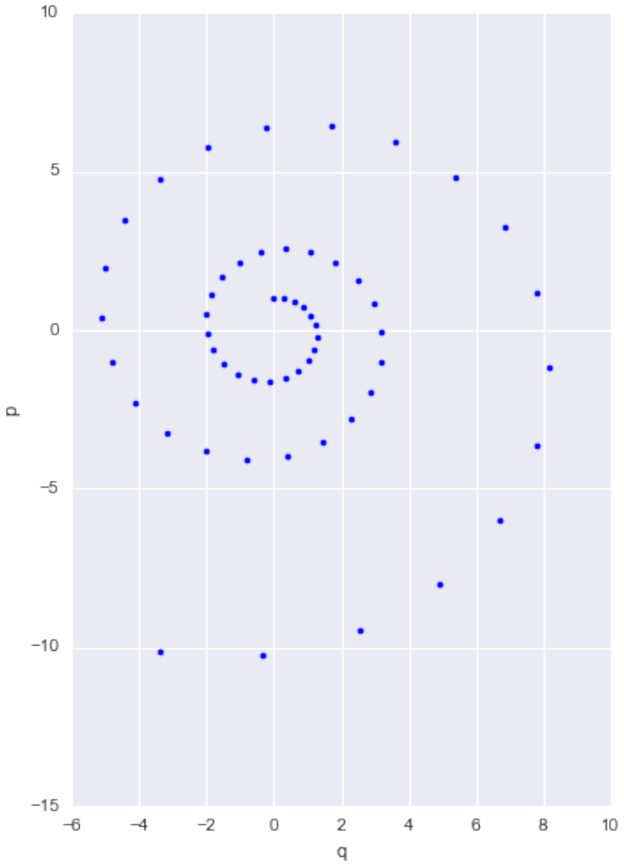
Tuning: choice of Kinetic energy

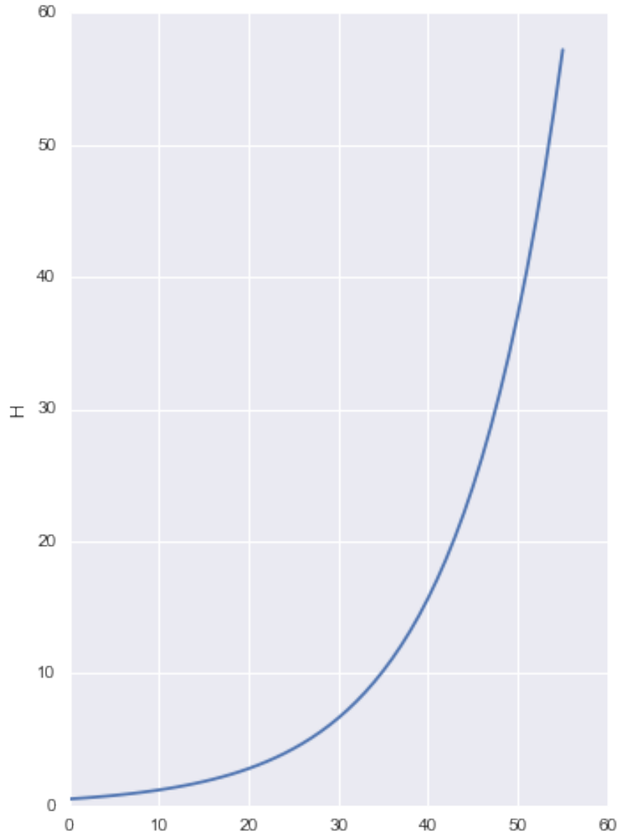
- The ideal kinetic energy interacts with target to make microcanonical exploration easy and uniform and marginal exploration well matched by the transition distribution.
- In practice we often use $K(p) = p' M^{-1} p$
- Set inverse mass matrix to the covariance of the target distribution: maximally decorrelate the target. Do in warmup phase.

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, undersamples tails (high-energy microcanonicals). Estimate dynamically: NUTS (pymc3 and Stan)

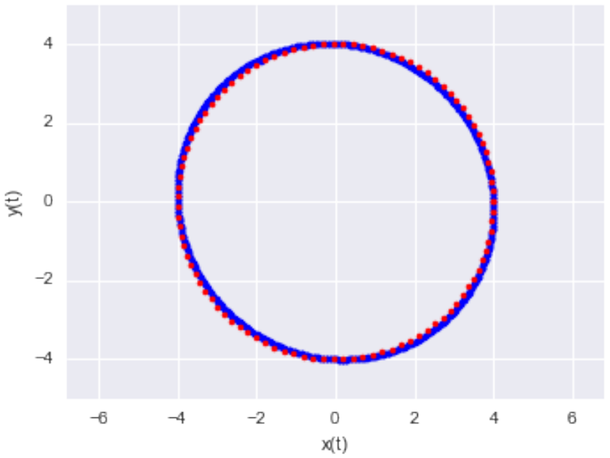
Discretization: Non symplectic integration

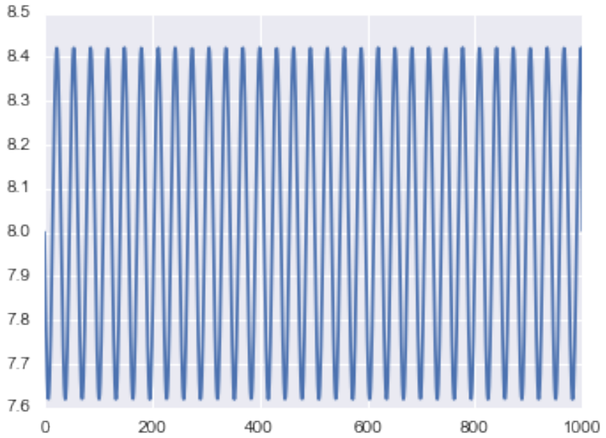




- $p_i(t+\epsilon) = p_i(t) - \epsilon \frac{\partial U}{\partial q_i}|_{q(t)}$
- $q_i(t+\epsilon) = q_i(t) + \epsilon \frac{p_i(t)}{m_i}$
- off-diagonal terms of size ϵ makes volume not preserved
- leads to drift over time

Sympletic Leapfrog



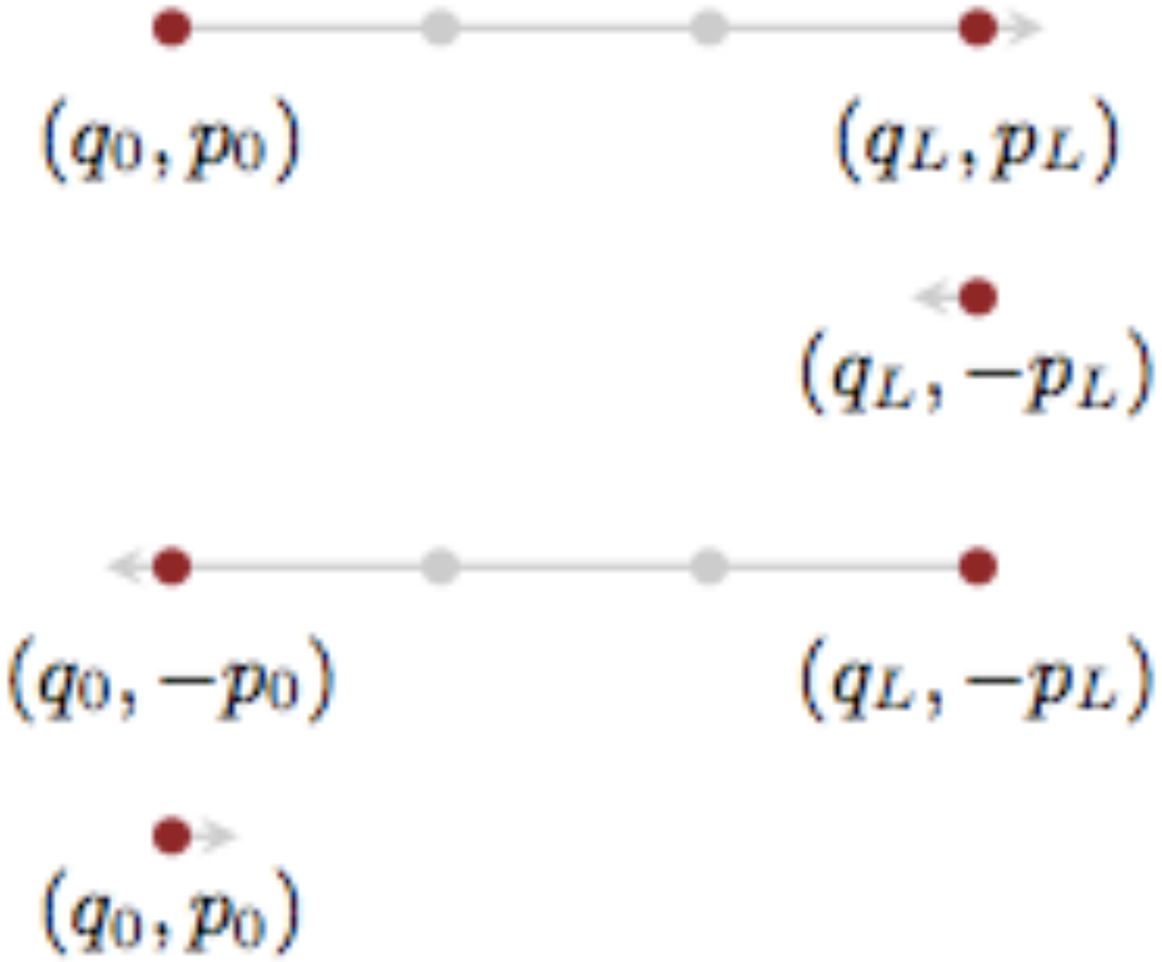


- Only *shear* transforms allowed, will preserve volume.
- $p_i(t+\frac{\epsilon}{2}) = p_i(t) - \frac{\epsilon}{2}\frac{\partial U}{\partial q_i}|_{q(t)}$
- $q_i(t+\epsilon) = q_i(t) + \epsilon\frac{p_i(t+\frac{\epsilon}{2})}{m_i}$
- $p_i(t+\epsilon) = p_i(t+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{\partial U}{\partial q_i}|_{q(t+\epsilon)}$
- 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 $(q, p) \rightarrow (q_L, -p_L)$. Superman to the rescue!
- proposal now: $Q(q', p'|q, p) = \delta(q' - q_L)\delta(p' + p_L)$.
- Acceptance: $A = \mathbf{min}[1, \frac{p(q_L, -p_L)\delta(q_L - q_L)\delta(-p_L + p_L)}{p(q, p)\delta(q - q)\delta(p - p)}]$

- 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 close to 1, and we have a very efficient sampler.
- In general we'll want to sum over all such points in the orbit
- momentum reversal could be left out if not within a more complex sampling scheme



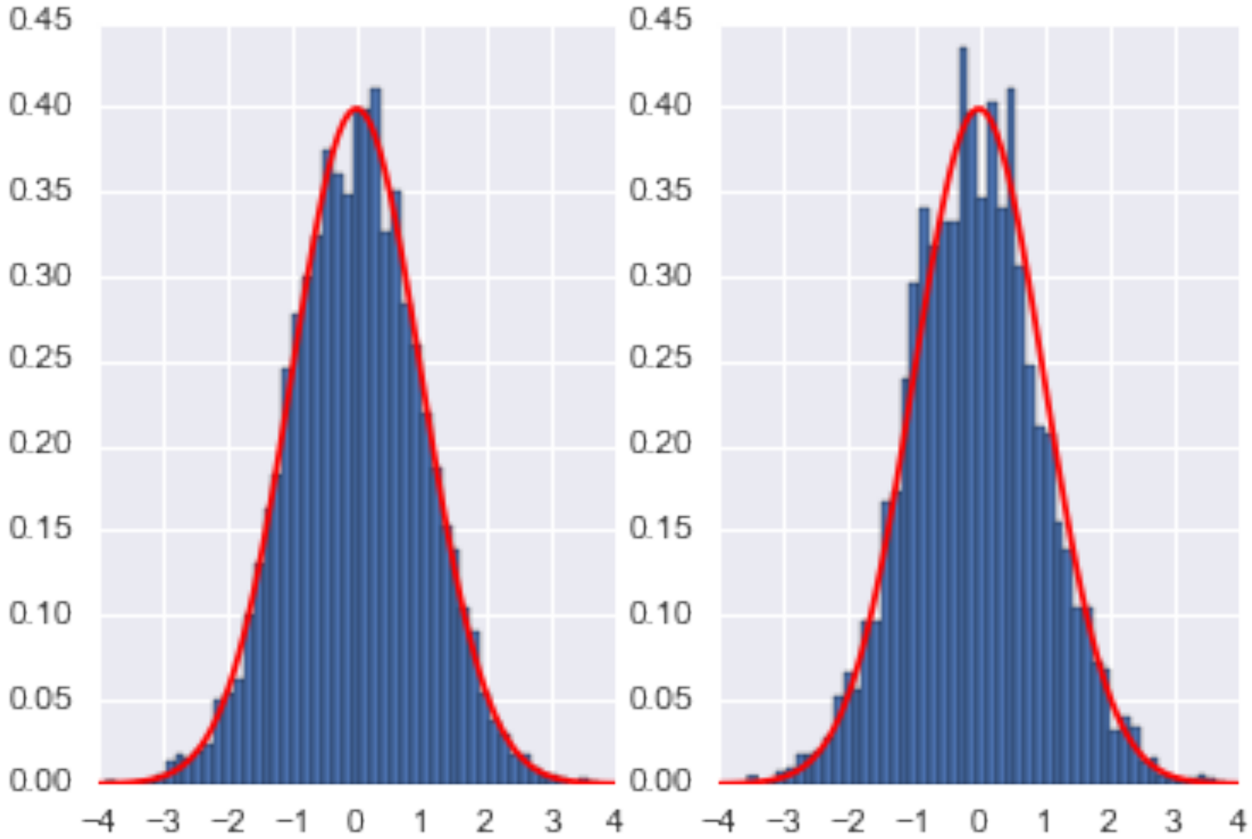
HMC Algorithm

- 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
    - $q^* = q^* + \epsilon p$
    - if not the last step, $p = p - \epsilon \nabla U(q)$
  - 6. Complete leapfrog: $p = p - \frac{\epsilon \nabla U(q)}{2}$

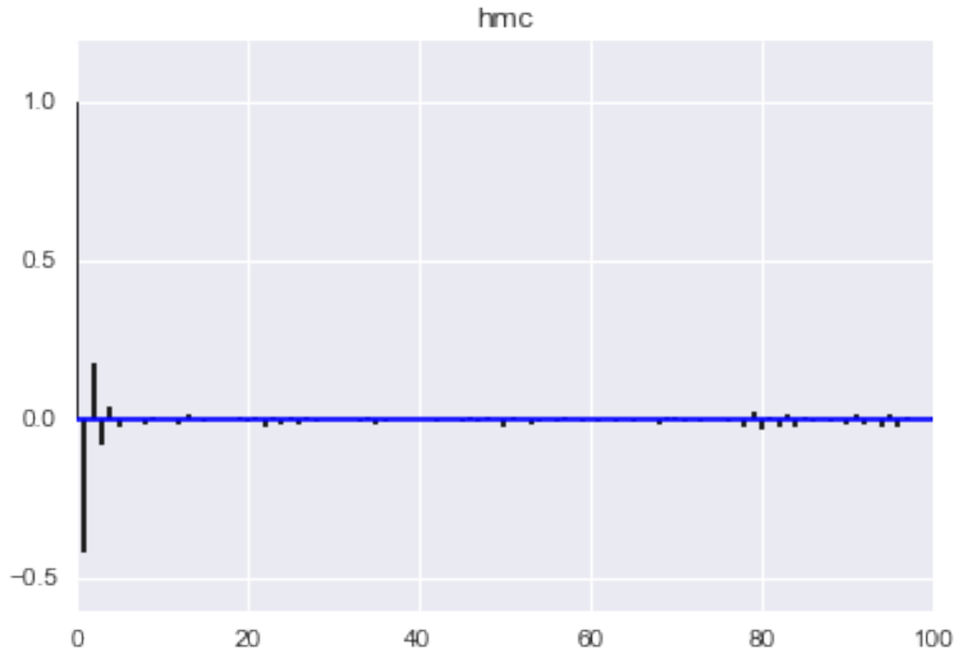
HMC (contd)

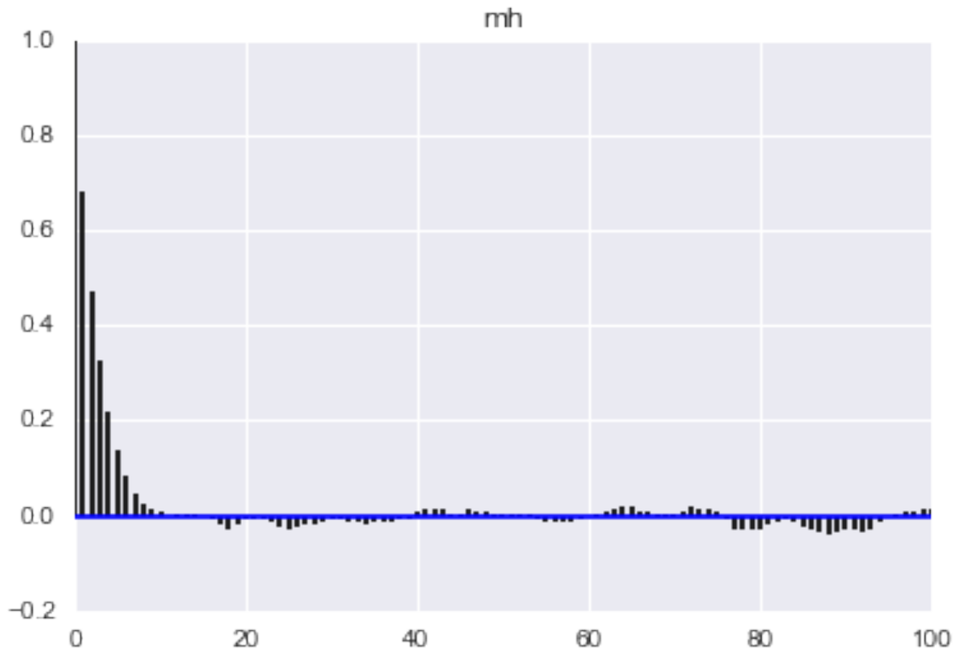
- for i=1:N_samples
  - 7. $p^* = -p$
  - 8. $U_c = U(q_c), \;\; K_c = \frac{p_c^\top M^{-1} p_c}{2}$
  - 9. $U^* = U(q^*), \;\; K^* = \frac{p^{\top *} M^{-1} p^*}{2}$
  - 10. $r \sim \mathbf{Unif}(0,1)$
  - 11. if $r < e^{(U_c - U^* + K_c - K^*)}$
    - accept $q_i = q^*$
    - otherwise reject

```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)
```

Tumors in pymc3

```
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)
```