Lecture 10
Metropolis-Hastings Sampler and Bayesian Stats

Last time: Metropolis, Markov, and MCMC

- Simulated Annealing samples from a evert tighter boltzmann distribution
- thus making its acceptance probability $A = \exp\left(-\Delta f / kT\right)$
- generally sample from any distribution by making its transition kernel/matrix satisfy detailed balance (reversibility)
- this ensures ergodicity and sample averages are time averages
- a symmetric proposal (like in SA) leads to a metropolis sampler

Today

- metropolis-hastings sampler
- sampling from discrete distributions
- introduction to bayesian statistics
- beta binomial updating
- normal-normal model

Metropolis

- probability increases, accept. decreases, accept some of the time.
- get aperiodic, irreducible, harris recurrent markov chain $\implies$ ergodic but takes a while to reach the **stationary distribution**

$$\int dx s(x) T(y|x) = \int p(y,x) dx = s(y)$$

- arrange transition matrix(kernel) to get desired stationary distribution

Transition matrix for Metropolis:

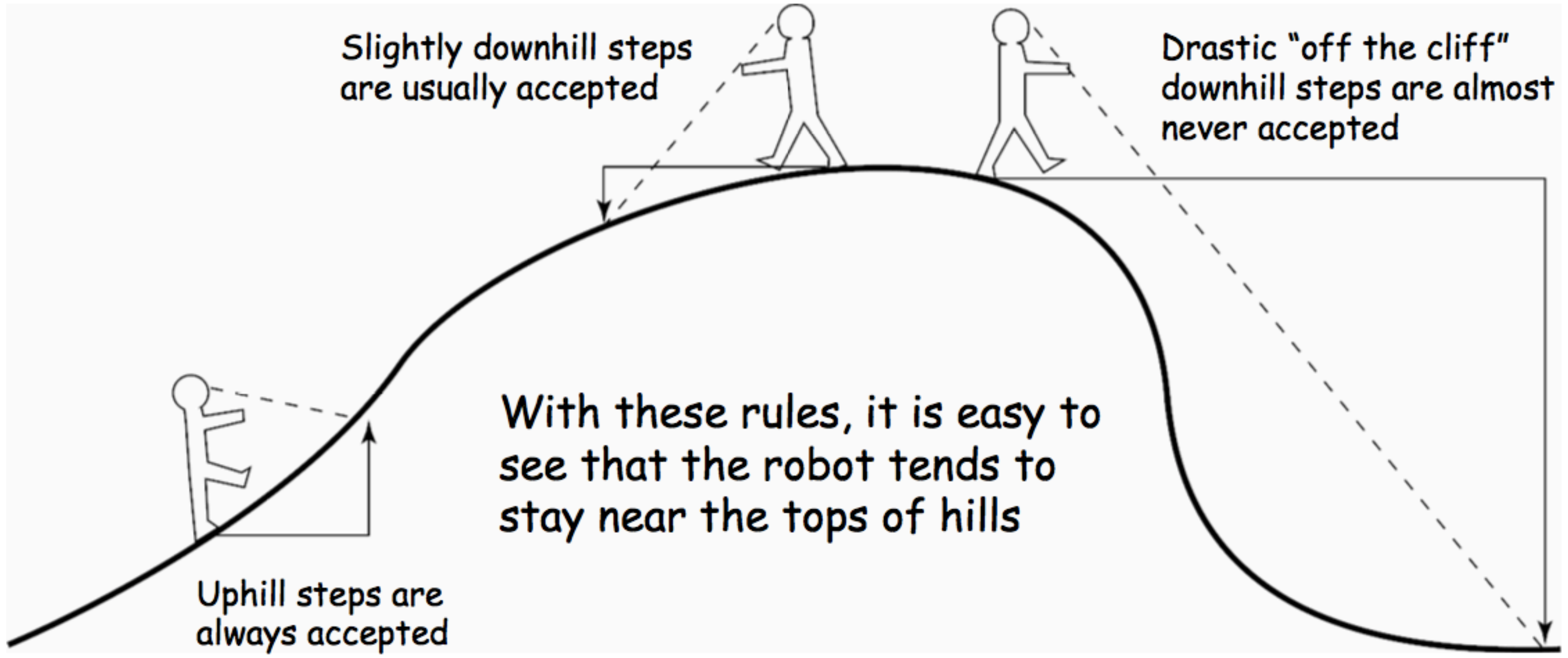
$T(x_i|x_{i-1}) = q(x_i|x_{i-1})\, A(x_i, x_{i-1}) + \delta(x_{i-1} - x_i) r(x_{i-1})$ where

$$A(x_i, x_{i-1}) = min(1, \frac{s(x_i)}{s(x_{i-1})})$$

is the Metropolis acceptance probability and

$$r(x_i) = \int dy q(y|x_i)(1 - A(y, x_i))$$ is the rejection term.

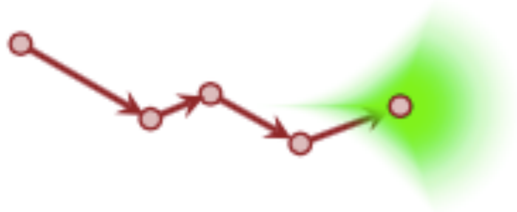
MCMC robot's rules

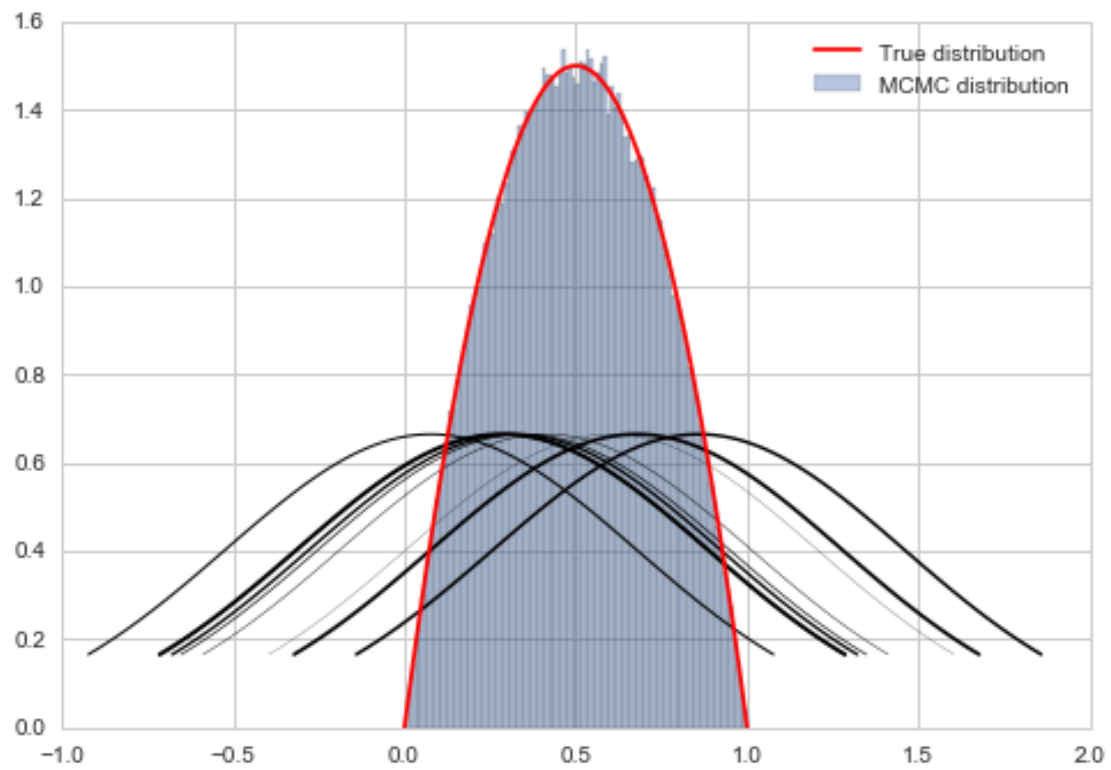


(from Paul Lewis)

```python
def metropolis(p, qdraw, nsamp, xinit):
    samples=np.empty(nsamp)
    x_prev = xinit
    for i in range(nsamp):
        x_star = qdraw(x_prev)
        p_star = p(x_star)
        p_prev = p(x_prev)
        pdfratio = p_star/p_prev
        if np.random.uniform() < min(1, pdfratio):
            samples[i] = x_star
            x_prev = x_star
        else:#we always get a sample
            samples[i]= x_prev

    return samples
```

Intuition: approaches typical set





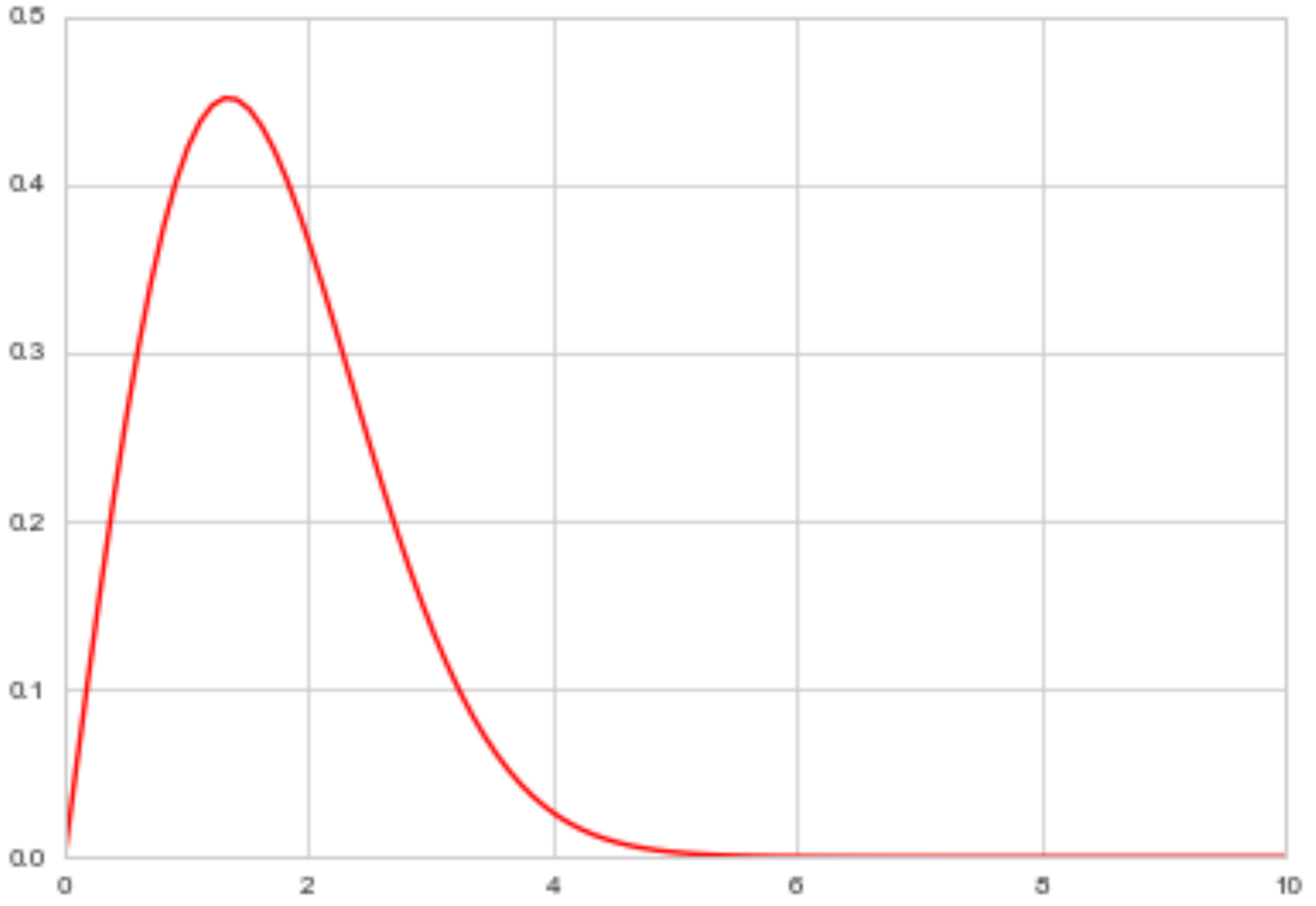
Instead of sampling p we sample q, yielding a new state, and a new proposal distribution from which to sample.

Metropolis-Hastings

- want to handle distributions with limited support
- proposal like normal leads to a lot of wasteful comparisons
- building in rejection breaks symmetry or proposal, the distribution needs to be normalized by some part of cdf.
- you might want to sample from a asymmetric distribution which matches targets support

Metropolis-Hastings

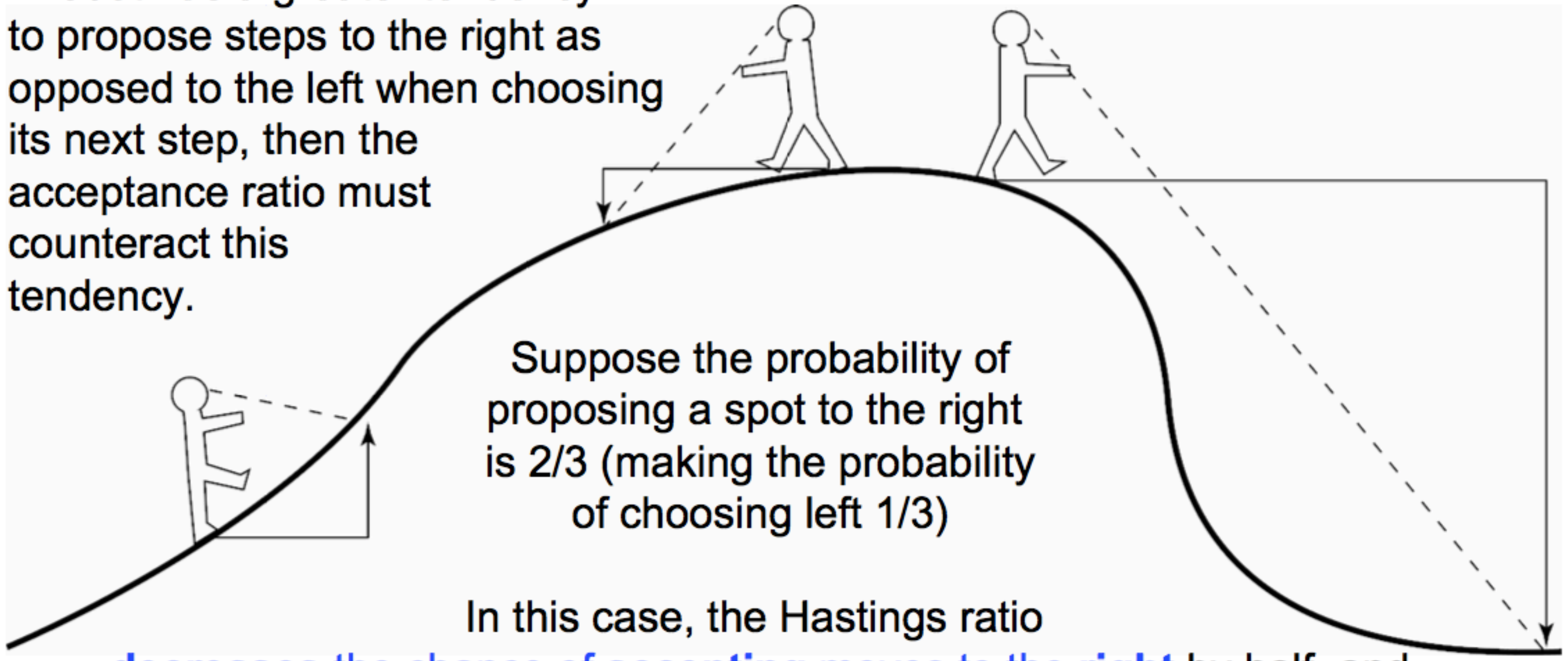
```
def metropolis_hastings(p,q, qdraw, nsamp, xinit):
    samples=np.empty(nsamp)
    x_prev = xinit
    for i in range(nsamp):
        x_star = qdraw(x_prev)
        p_star = p(x_star)
        p_prev = p(x_prev)
        pdfratio = p_star/p_prev
        proposalratio = q(x_prev, x_star)/q(x_star, x_prev)
        if np.random.uniform() < min(1, pdfratio*proposalratio):
            samples[i] = x_star
            x_prev = x_star
        else:#we always get a sample
            samples[i]= x_prev

    return samples
```



Acceptance is now

$$A(x_i, x_{i-1}) = min(1, \frac{s(x_i) \times q(x_{i-1}|x_i)}{s(x_{i-1}) \times q(x_i|x_{i-1})}).$$

- correct the sampling of q to match p, corrects for any asymmetries in the proposal distribution.
- A good rule of thumb is that the proposal has the same or larger support then the target, with the same support being the best.



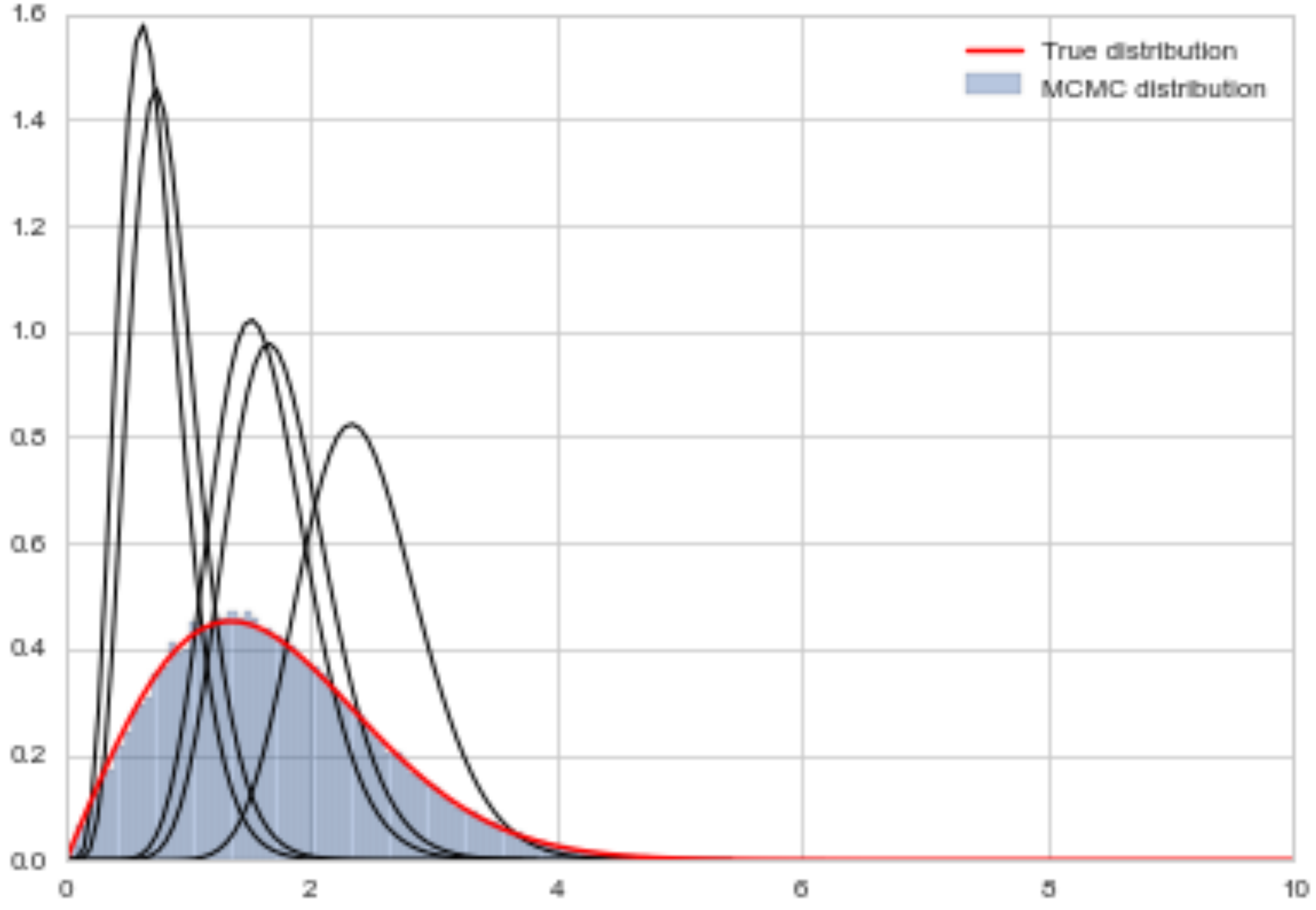
In this case, the Hastings ratio **decreases** the chance of **accepting** moves to the **right** by half, and **increases** the chance of **accepting** moves to the **left** (by a factor of 2), thus **exactly compensating** for the asymmetry in the proposal distribution.

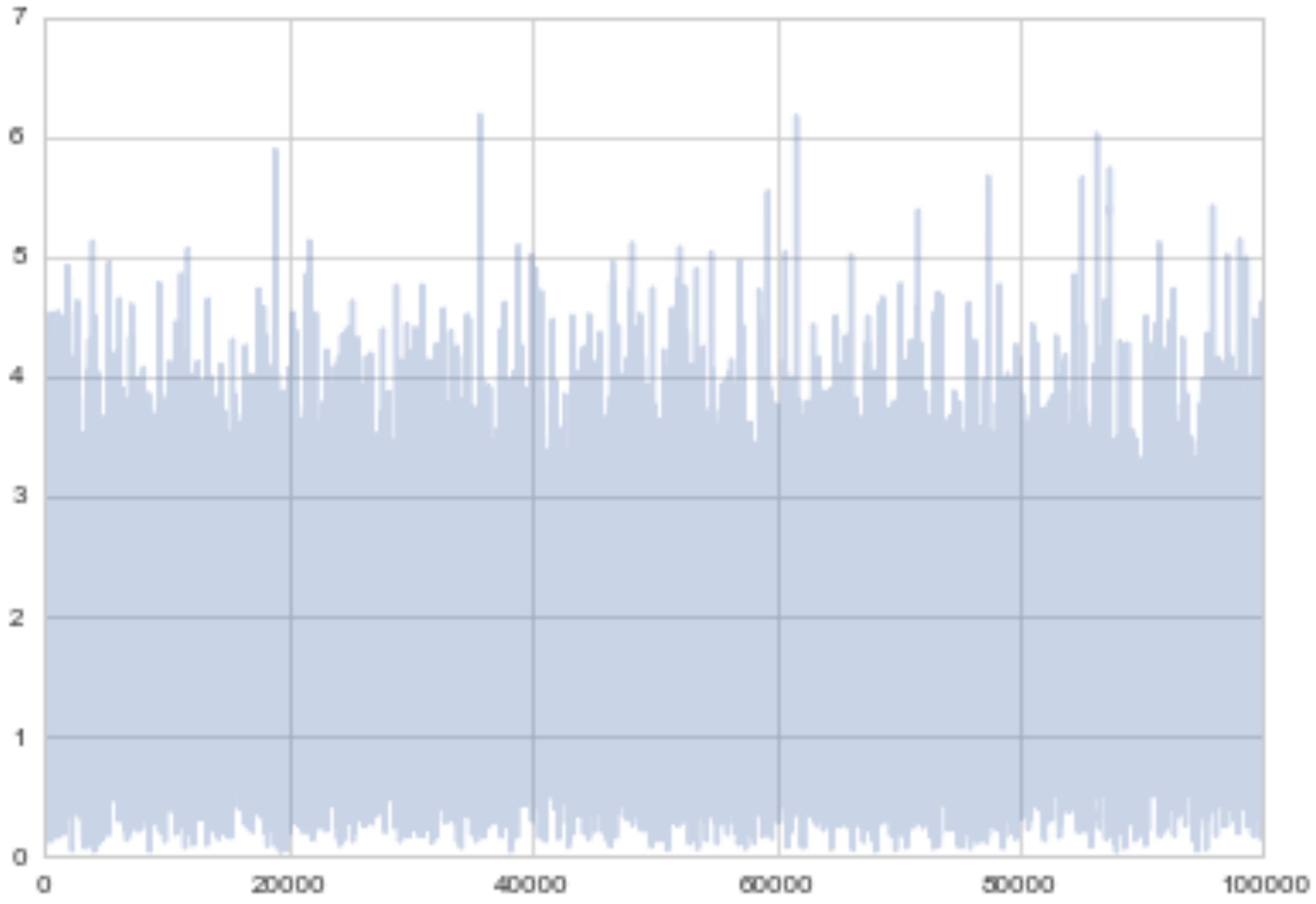
(from Paul Lewis)

Choice of Proposal

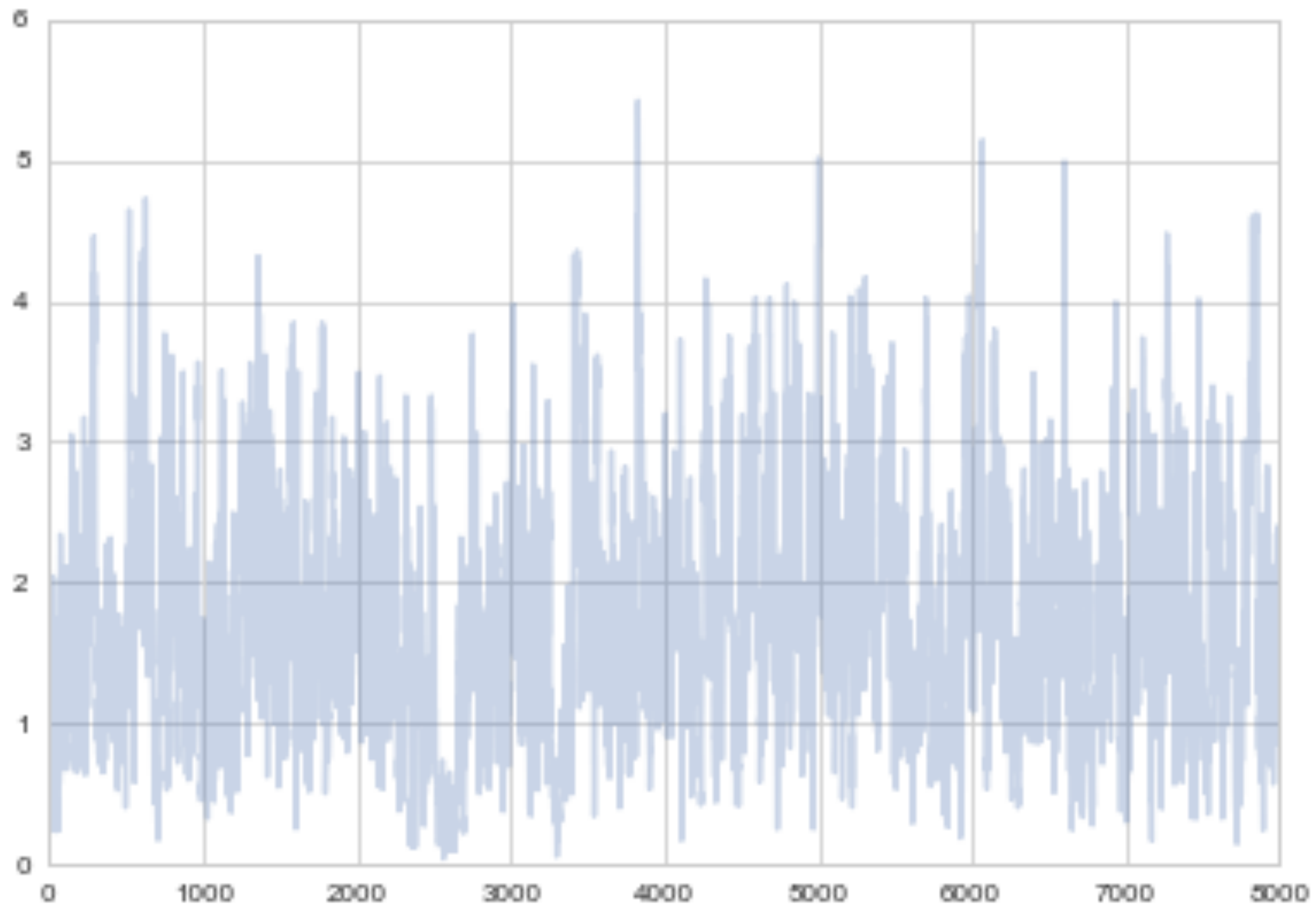
- Our Weibull is: $0.554xe^{-(x/1.9)^2}$
- A rule of thumb for choosing proposal distributions is to parametrize them in terms of their mean and variance/precision since that provides a notion of "centeredness" which we can use for our proposals
- Use a Gamma Distribution with parametrization $Gamma(x\tau, 1/\tau)$ in the shape-scale argument setup.

Gamma-Weibull with traceplot

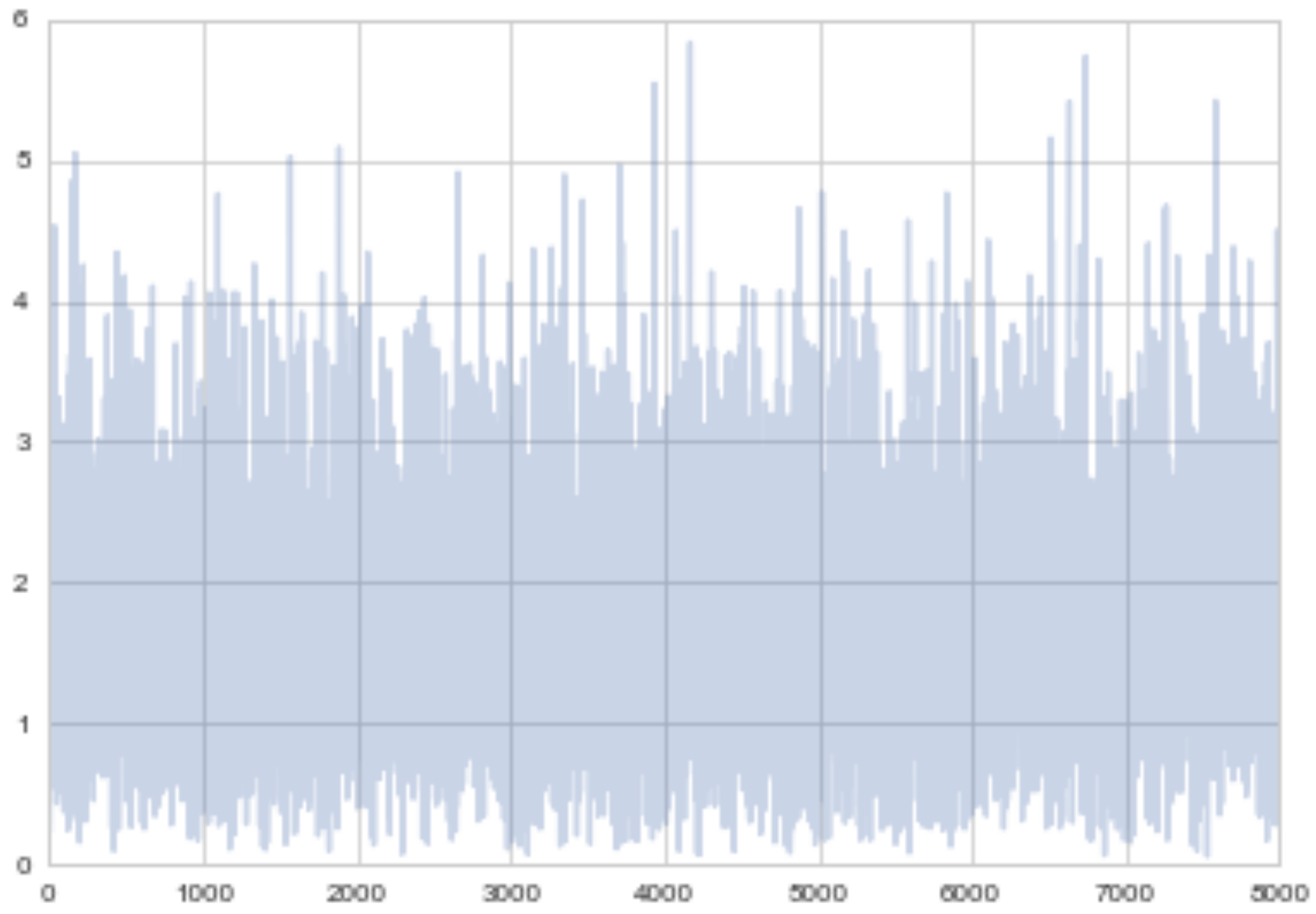




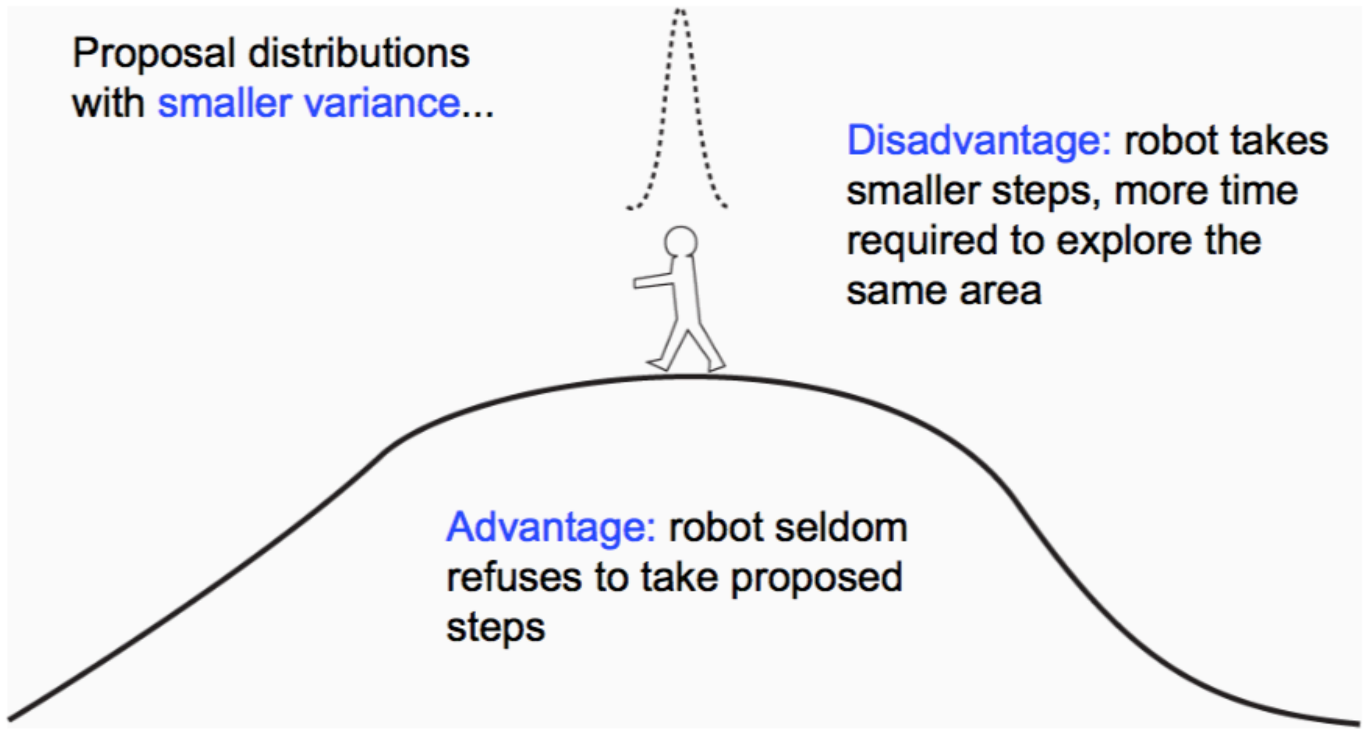
Traceplot after burnin but without thinning

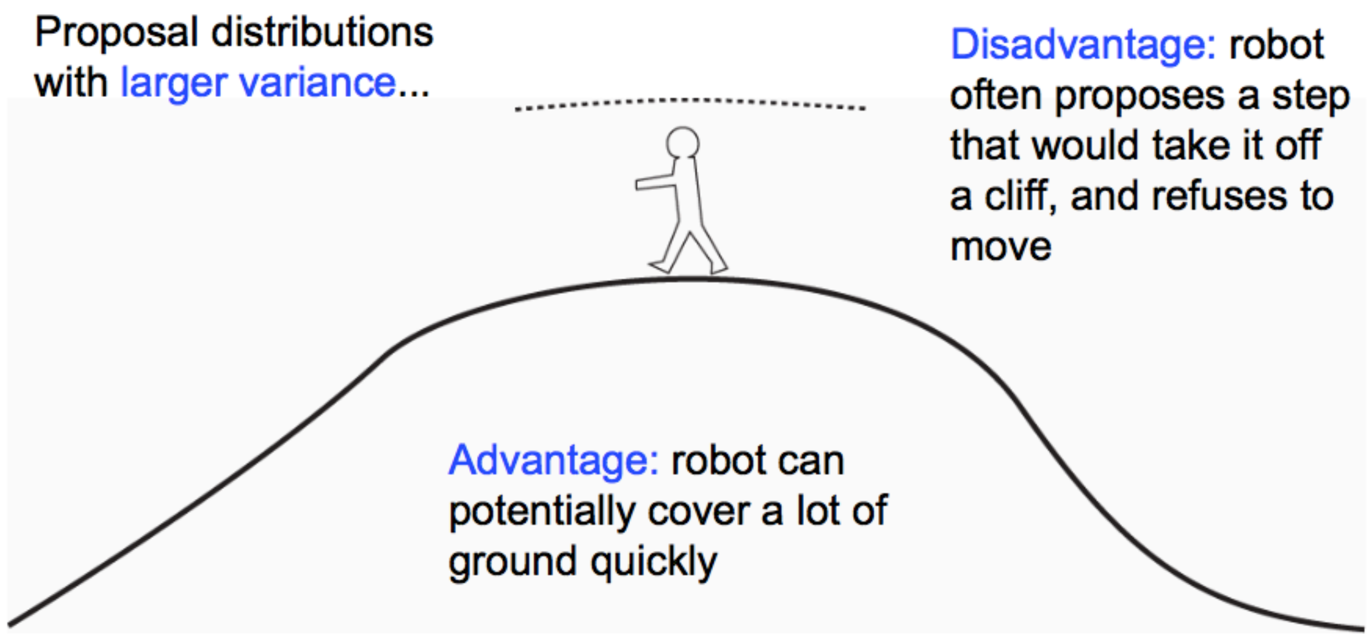


Traceplot after burning and thinning



Tuning the width or precision



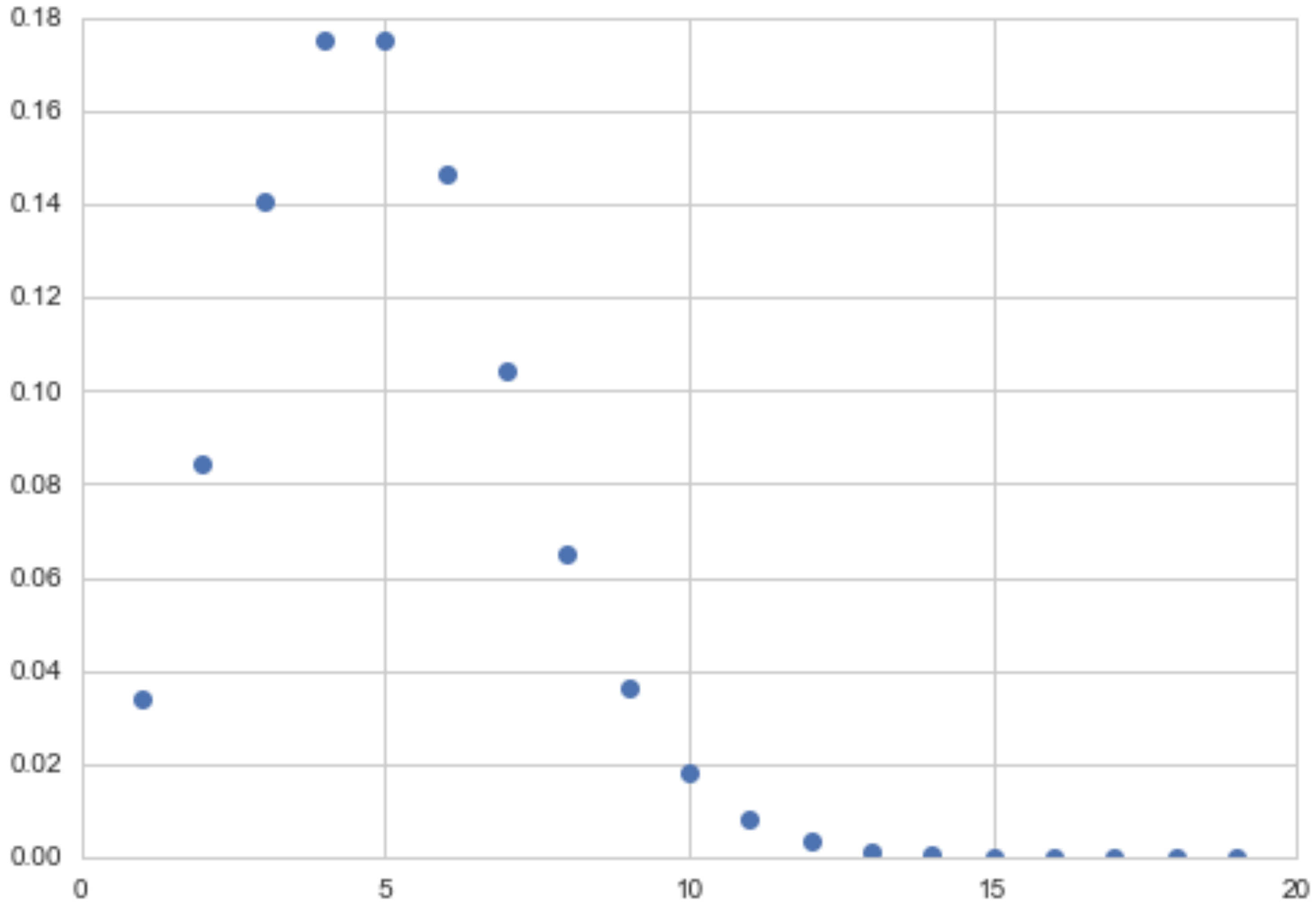


(from Paul Lewis)

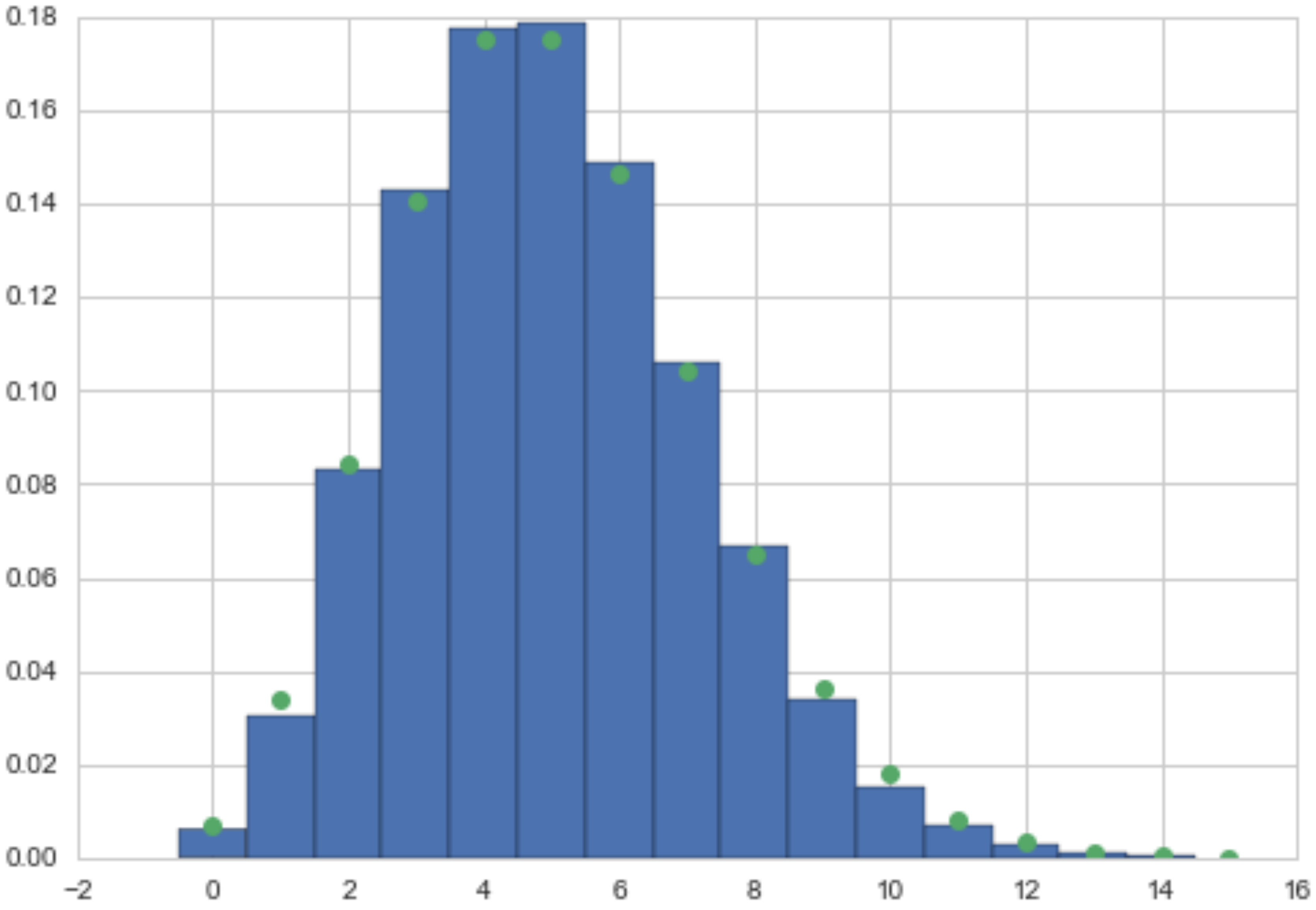
Discrete distribution MCMC

- proposal distribution becomes proposal matrix
- index the discrete outcomes
- can use symmetric or asymmetric proposal as long as rows sum to 1
- make sure matrix is irreducible: ie you can get from any index to any other one.

Example: generate poisson



$$Q = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & & \cdots \\ 1/2 & 0 & 1/2 & 0 & 0 & \cdots \\ 0 & 1/2 & 0 & 1/2 & 0 & \cdots \\ 0 & 0 & 1/2 & 0 & 1/2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$



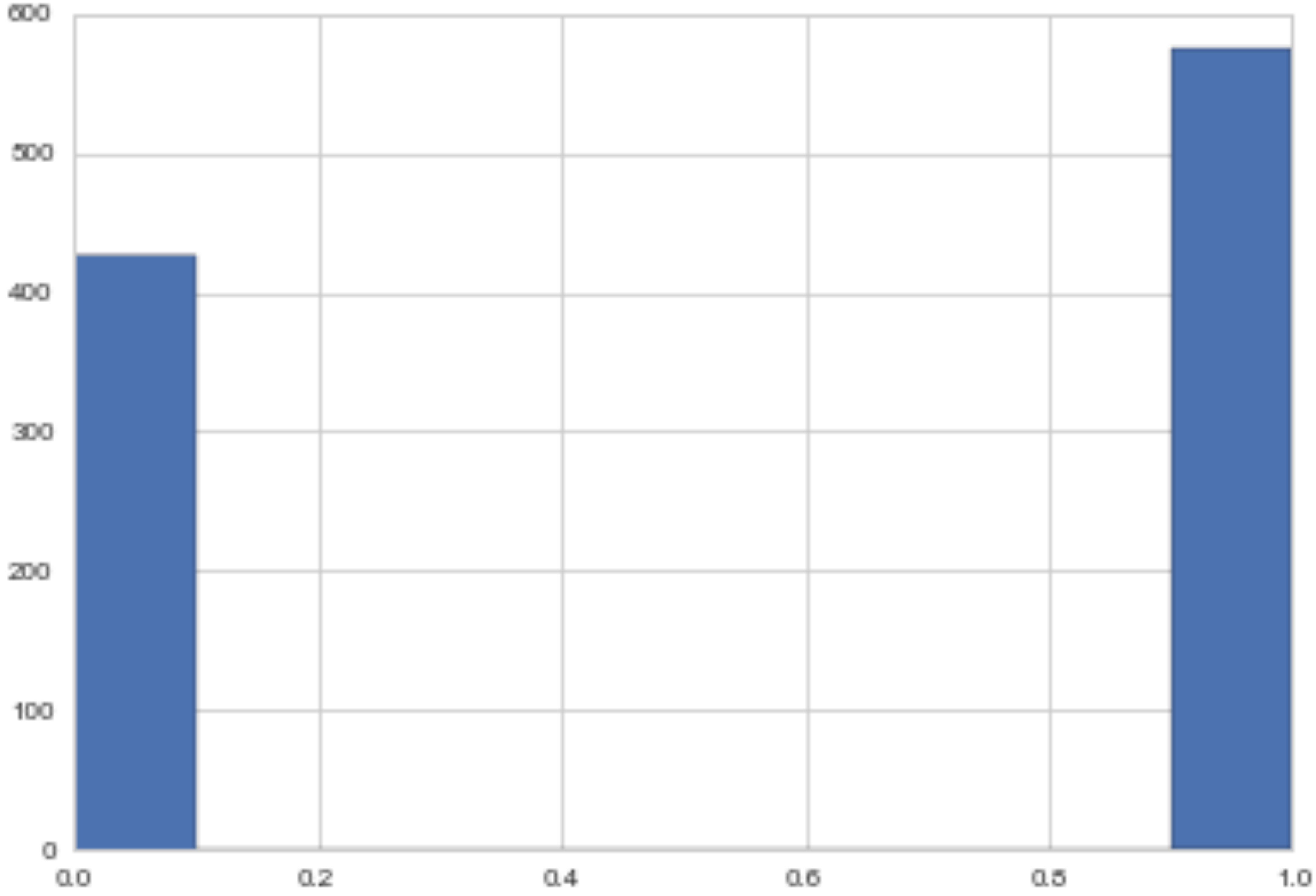
$$p(k) = e^{-\mu} \frac{\mu^k}{k!}.$$

```
def prop_draw(ifrom):
    u = np.random.uniform()
    if ifrom !=0:
        if u < 1/2:
            ito = ifrom -1
        else:
            ito = ifrom + 1
    else:
        if u < 1/2:
            ito=0
        else:
            ito=1
    return ito

def prop_pdf(ito, ifrom):
    if ito == ifrom - 1:
        return 0.5
    elif ito == ifrom + 1:
        return 0.5
    elif ito == ifrom and ito == 0:#needed to make first row sum to 1
        return 0.5
    else:
        return 0
```

Rain-Sun with MH

```
def rainsunpmf(state_int):
    p = 0.416667
    if state_int==0:
        return p
    else:#anything else is treated as a 1
        return 1 - p
def rainsunprop2(sint_old):
    return np.random.choice(2,p=t_asym[sint_old])
def rainsunpropfunc(sint_new, sint_old):
    return t_asym[sint_old][sint_new]
```

```
[[ 0.1,  0.9],
 [ 0.3,  0.7]]
```



600
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

Bayesian statistics

Frequentist Stats

- parameters are fixed, data is stochastic
- true parameter θ^* characterizes population
- we estimate $\hat{theta}$ on sample
- we can use MLE $\theta_{ML} = \underset{\theta}{\mathbf{argmax}}\, \mathcal{L}$
- we obtain sampling distributions (using bootstrap)

Bayesian Stats

- assume sample IS the data, no stochasticity
- parameters θ are stochastic random variables
- associate the parameter θ with a prior distribution $p(\theta)$
- The prior distribution generally represents our belief on the parameter values when we have not observed any data yet (to be qualified later)

Posterior distribution

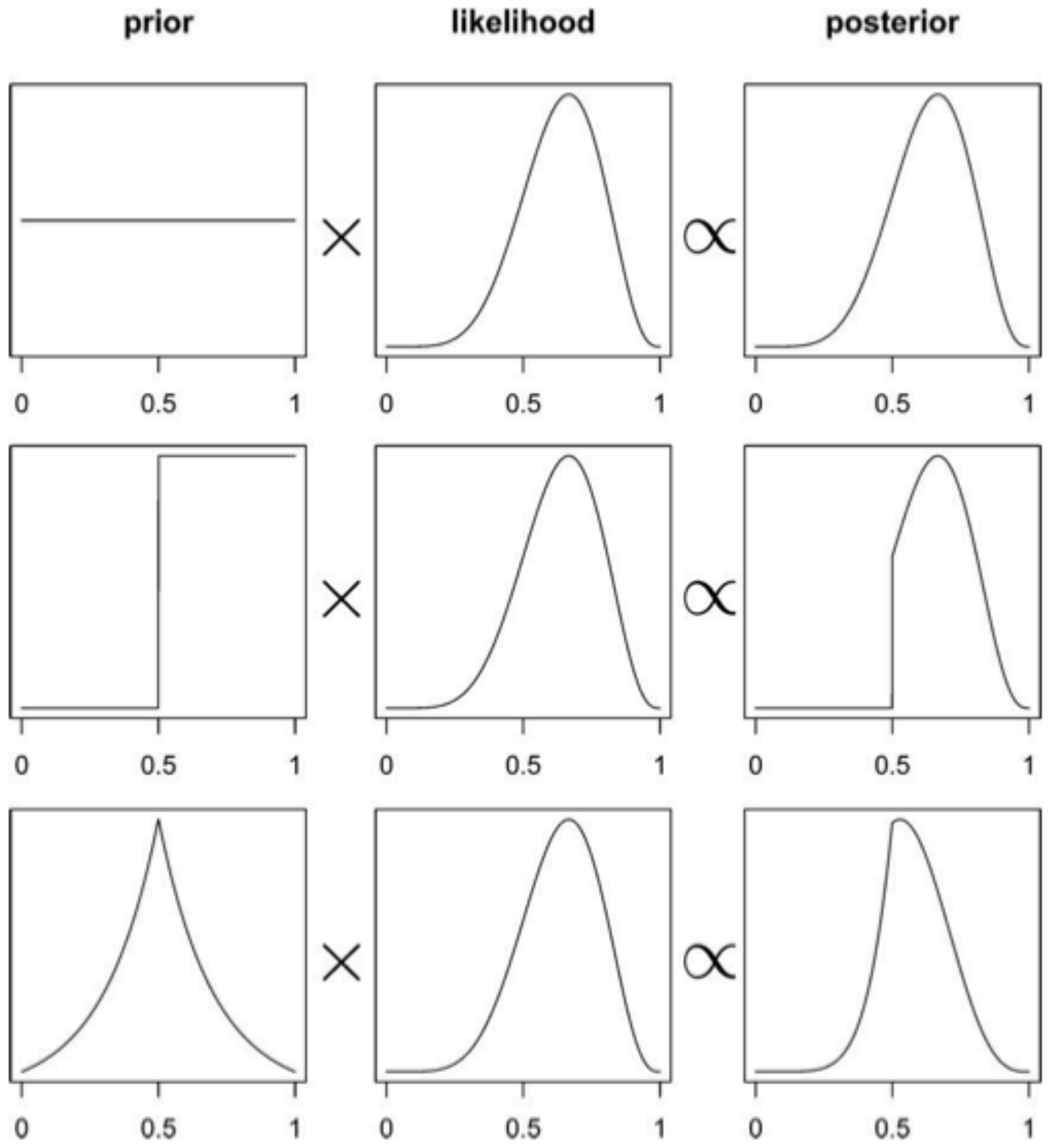
$$p(\theta|y) = \frac{p(y|\theta)\,p(\theta)}{p(y)}$$

with the evidence $p(D)$ or $p(y)$ the expected likelihood (on existing data points) over the prior $E_{p(\theta)}[\mathcal{L}]$:

$$p(y) = \int d\theta p(y|\theta)p(\theta).$$

- $posterior = \frac{likelihood \times prior}{evidence}$
- evidence is just the normalization
- usually dont care about normalization (until model comparison), just samples
- What if θ is multidimensional? Marginal posterior:

$$p(\theta_1 | D) = \int d\theta_{-1} p(\theta | D).$$



Posterior Predictive for predictions

The distribution of a future data point y^*:

$$p(y^*|D = \{y\}) = \int d\theta p(y^*|\theta)p(\theta|\{y\}).$$

Expectation of the likelihood at a new point(s) over the posterior $E_{p(\theta|D)}[p(y|\theta)]$.

Summary via MAP (a point estimate)

$$
\begin{aligned}
\theta_{\text{MAP}} &= \arg\max_{\theta} \; p(\theta|D) \\
&= \arg\max_{\theta} \frac{\mathcal{L}\, p(\theta)}{p(D)} \\
&= \arg\max_{\theta} \; \mathcal{L}\, p(\theta)
\end{aligned}
$$

corresponds to Bayes risk for 1-0 loss.

mean corresponds to Bayes risk for squared error loss

Conjugate Prior

- A **conjugate prior** is one which, when multiplied with an appropriate likelihood, gives a posterior with the same functional form as the prior.
- Likelihoods in the exponential family have conjugate priors in the same family
- analytical tractability AND interpretability

Coin Toss Model

- Coin tosses are modelled using the Binomial Distribution, which is the distribution of a set of Bernoulli random variables.
- The Beta distribution is conjugate to the Binomial distribution

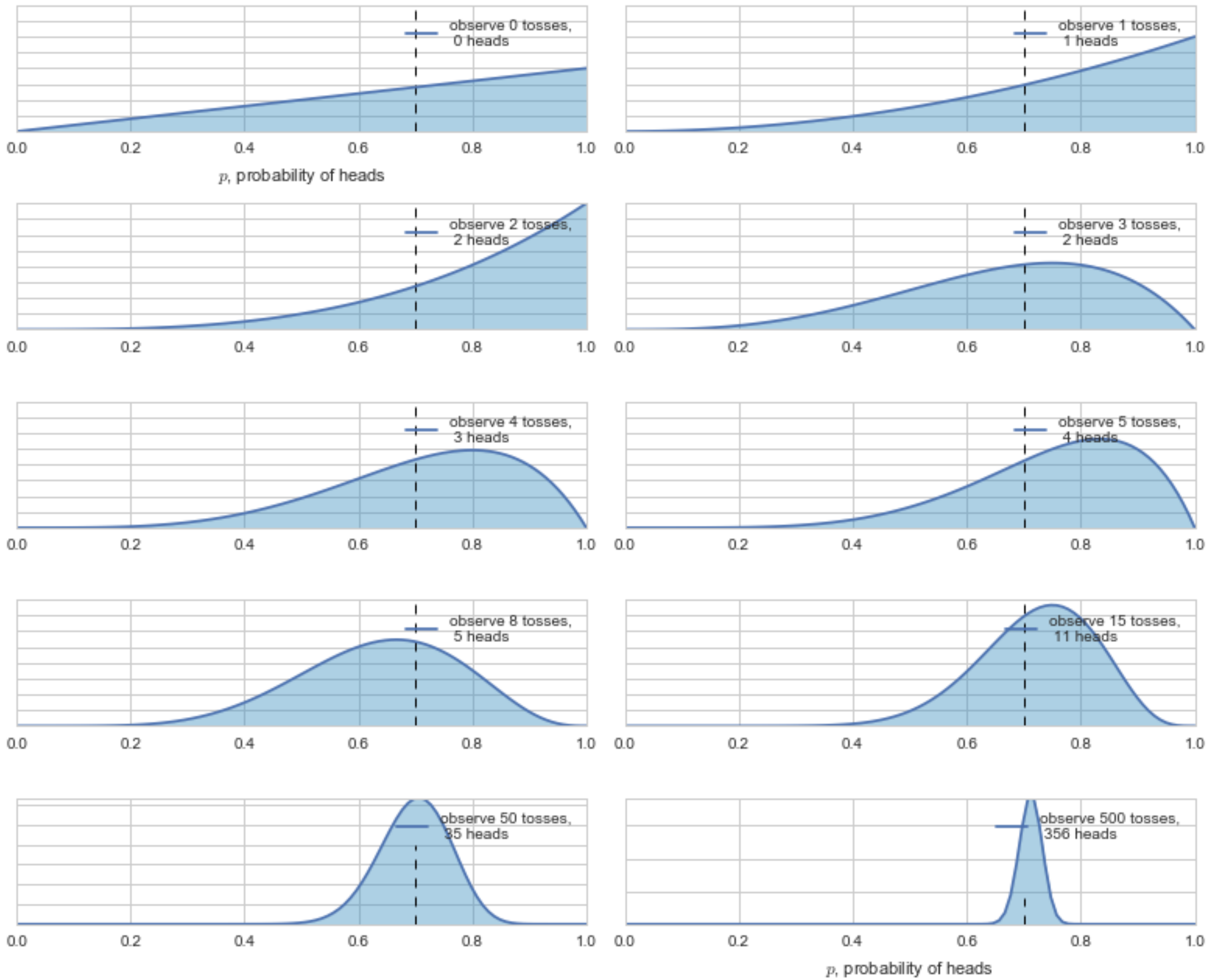
$$p(p|y) \propto p(y|p)P(p) = Binom(n, y, p) \times Beta(\alpha, \beta)$$

Because of the conjugacy, this turns out to be:

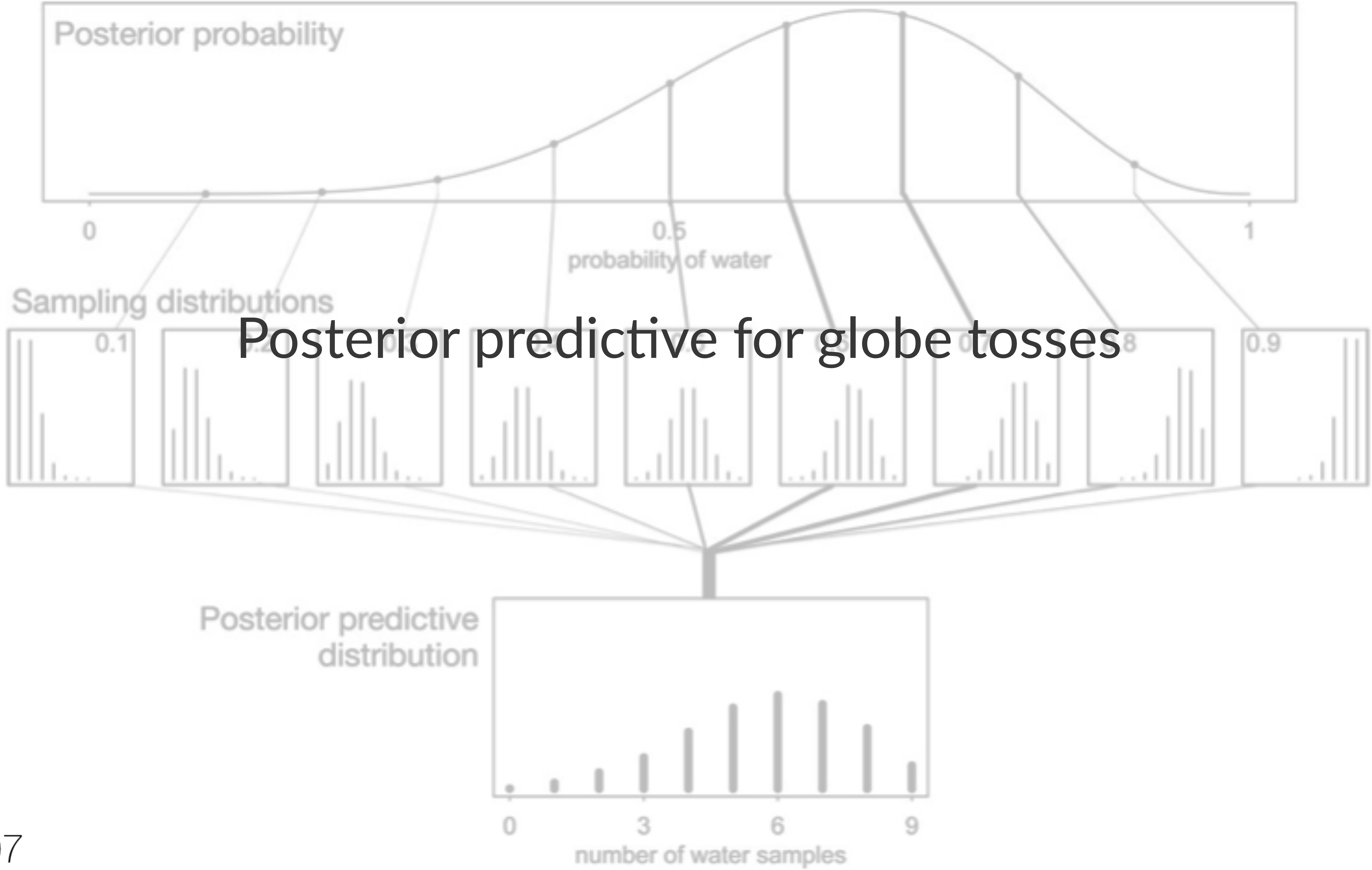
$$Beta(y + \alpha, n - y + \beta)$$

- think of a prior as a regularizer.
- a $Beta(1, 1)$ prior is equivalent to a uniform distribution.
- This is an **uninformative prior**. Here the prior adds one heads and one tails to the actual data, providing some "towards-center" regularization
- especially useful where in a few tosses you got all heads, clearly at odds with your beliefs.
- a $Beta(2, 1)$ prior would bias you to more heads (water in globe toss).

Bayesian updating of posterior probabilities



Posterior predictive for globe tosses



Normal-Normal Model

Posterior for a gaussian likelihood:

$$p(\mu, \sigma^2 | y_1, \ldots, y_n, \sigma^2) \propto \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2} \sum (y_i - \mu)^2} \, p(\mu, \sigma^2)$$

What is the posterior of μ assuming we know σ^2?

Prior for σ^2 is $p(\sigma^2) = \delta(\sigma^2 - \sigma_0^2)$

$$p(\mu|y_1,\ldots,y_n,\sigma^2=\sigma_0^2) \propto p(\mu|\sigma^2=\sigma_0^2)\, e^{-\frac{1}{2\sigma_0^2}\sum(y_i-\mu)^2}$$

The conjugate of the normal is the normal itself.

Say we have the prior

$$p(\mu|\sigma^2) = \exp\left\{-\frac{1}{2\tau^2}(\hat{\mu}-\mu)^2\right\}$$

posterior: $p(\mu|y_1,\ldots,y_n,\sigma^2) \propto \exp\left\{-\frac{a}{2}(\mu-b/a)^2\right\}$

Here

$$a = \frac{1}{\tau^2} + \frac{n}{\sigma_0^2}, \qquad b = \frac{\hat{\mu}}{\tau^2} + \frac{\sum y_i}{\sigma_0^2}$$

Define $\kappa = \sigma^2/\tau^2$

$$\mu_p = \frac{b}{a} = \frac{\kappa}{\kappa + n}\hat{\mu} + \frac{n}{\kappa + n}\bar{y}$$

which is a weighted average of prior mean and sampling mean.

The variance is

$$\tau_p^2 = \frac{1}{1/\tau^2 + n/\sigma^2}$$

or better

$$\frac{1}{\tau_p^2} = \frac{1}{\tau^2} + \frac{n}{\sigma^2}.$$

as n increases, the data dominates the prior and the posterior mean approaches the data mean, with the posterior distribution narrowing...

```python
Y = [16.4, 17.0, 17.2, 17.4, 18.2, 18.2, 18.2, 19.9, 20.8]
#Data Quantities
sig = np.std(Y) # assume that is the value of KNOWN sigma (in the likelihood)
mu_data = np.mean(Y)
n = len(Y)
# Prior mean
mu_prior = 19.5
# prior std
tau = 10
# plug in formulas
kappa = sig**2 / tau**2
sig_post =np.sqrt(1./( 1./tau**2 + n/sig**2));
# posterior mean
mu_post = kappa / (kappa + n) *mu_prior + n/(kappa+n)* mu_data
#samples
N = 15000
theta_prior = np.random.normal(loc=mu_prior, scale=tau, size=N);
theta_post = np.random.normal(loc=mu_post, scale=sig_post, size=N);
```