Lecture 10
Metropolis-Hastings Sampler and Bayesian Stats

Last time: Metropolis, Markov, and MCMC

- Simulated Annealing samples from a ever 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

Markov Chain

$$T(x_n|x_{n-1}, x_{n-1} \ldots, x_1) = T(x_n|x_{n-1})$$

- non IID, stochastic process
- but one step memory only
- widely applicable, first order equations

Stationarity

$$sT = s \text{ or } \sum_i s_i T_{ij} = s_j \text{ or}$$

Continuous case: define T so that:

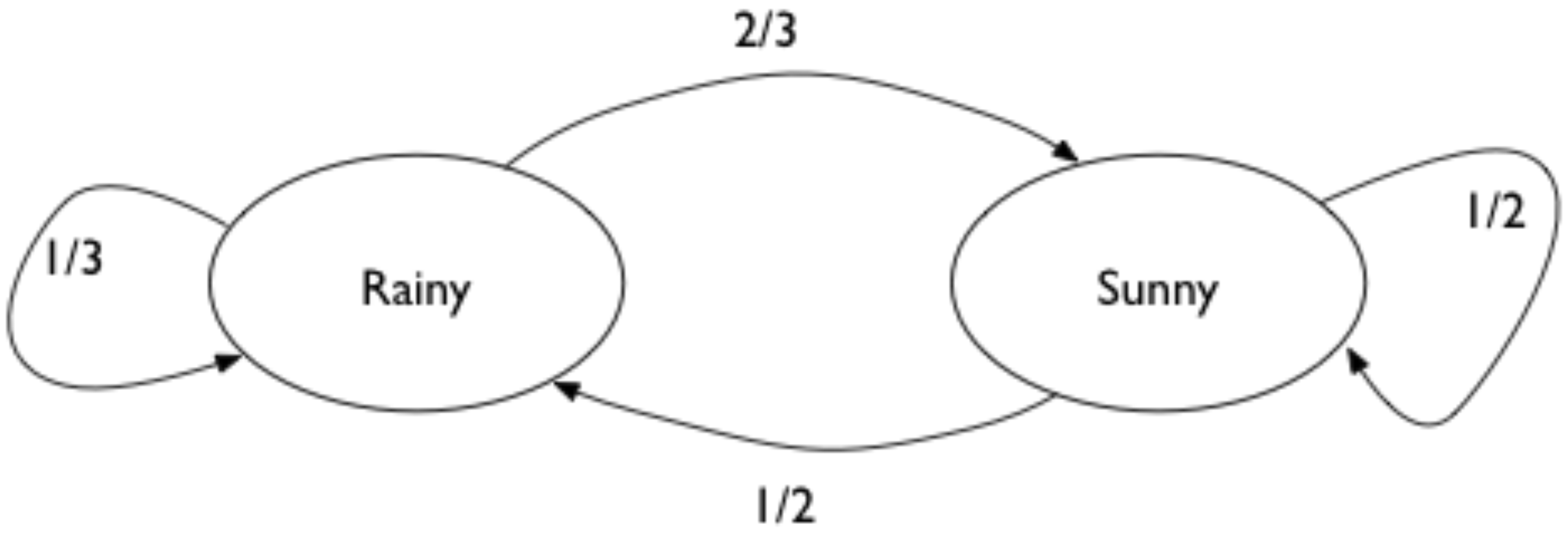
$$\int dx_i s(x_i) T(x_{i+1} | x_i) = s(x_{i+1}) \text{ then}$$

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

Jargon

- **Irreducible**: can go from anywhere to everywhere
- **Aperiodic**: no finite loops
- **Recurrent**: visited repeatedly. Harris recurrent if all states are visited infinitely as $t \to \infty$.

Rainy Sunny Markov chain



aperiodic and irreducible

Transition matrix, applied again and again

```
array([[ 0.33333333,  0.66666667],
       [ 0.5       ,  0.5       ]])
```

```
[[ 0.44444444  0.55555556]
 [ 0.41666667  0.58333333]]
-----------------
[[ 0.42592593  0.57407407]
 [ 0.43055556  0.56944444]]
-----------------
[[ 0.42901235  0.57098765]
 [ 0.42824074  0.57175926]]
-----------------
[[ 0.42849794  0.57150206]
 [ 0.42862654  0.57137346]]
-----------------
[[ 0.42858368  0.57141632]
 [ 0.42856224  0.57143776]]
```

Stationary distribution can be solved for:

Assume that it is $s = [p, 1-p]$

Then: $sT = s$

gives us

$$p \times (1/3) + (1-p) \times 1/2 = p$$

and thus $p = 3/7$

```
np.dot([0.9,0.1], tm_before): array([ 0.42858153,  0.57141847])
```

Stationarity, again

A irreducible (goes everywhere) and aperiodic (no cycles) markov chain will eventually converge to a stationary markov chain. It is the marginal distribution of this chain that we want to sample from, and which we do in metropolis (and for that matter, in simulated annealing).

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

Detailed balance is enough for stationarity

$$s(x)T(y|x) = s(y)T(x|y)$$

If one sums both sides over x

$$\int dx s(x)t(y|x) = s(y) \int dx T(x|y)$$ which gives us back the stationarity condition from above.

Proposal, redux

- all the positions x in the domain we wish to minimize a function f over ought to be able to communicate: IRREDUCIBLE
- detailed balance: proposal is symmetric
- ensures $\{x_t\}$ generated by simulated annealing is a stationary markov chain with target boltzmann distribution: equilibrium
- ensures $\{x_t\}$ generated by metropolis is a stationary markov chain with appropriate target.

Today

- conditions for a MCMC algorithm
- metropolis-hastings sampler
- sampling from discrete distributions
- introduction to bayesian statistics
- normal-normal model

Ergodicity and Stationarity

- These are not the same concept
- detailed balance implies stationarity. Needs irreducibility.
- aperiodic, irreducible, harris recurrent markov chain $\implies$ ergodic
- what is ergodic?

Ergodicity

- Aperiodic, irreducible, positive Harris recurrent markov chains are ergodic
- i.e., in the limit of infinite (many) steps, the marginal distribution of the chain is the same. This means that if we take largely spaced about (some thinning T) samples from a stationary markov chain (after burnin B), we can draw independent samples.

- "Ergodic" law of large numbers:

$$\int g(x)f(x)dx = \frac{1}{N} \sum_{j=B+1:B+N:T} g(x_j)$$

- equivalent, for very large N:

$$\int g(x)f(x)dx = \frac{1}{N} \sum_{j=B+1}^{B+N} g(x_j)$$

- the jury is out on thinning. Most dont think one needs it
- you can get a similar central limit theorem as well

Sketch of proof (here and here for details)

- by Perron-Frobenius theorem, irreducible, aperiodic stochastic matrices (rows sum to 1 with non-negative elements) have one eigenvalue $\lambda_0 = 1$ and positive eigenvector $e_0 > 0$. All other eigenvalues have absolute value less than 1.

- $p^{(t)} = T^n p^{(0)}$ where $p^{(0)} = \sum_i \alpha_i e_i$

- Then $p^(t) = \sum_i \alpha_i \lambda_i^n e_i = \alpha_0 e_0 = e_0$

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:

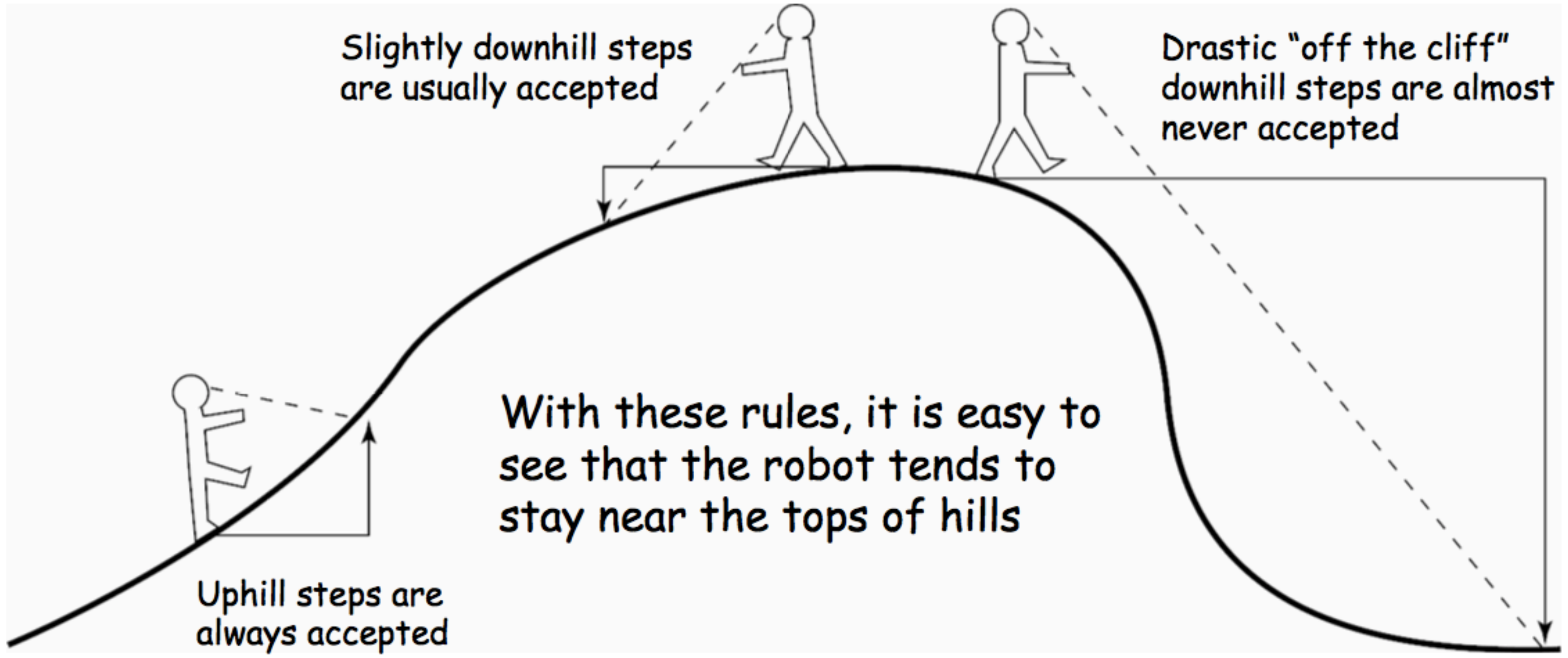
$$T(x_i|x_{i-1}) = q(x_i|x_{i-1})\,A(x_i, x_{i-1}) + \delta(x_{i-1} - x_i)r(x_{i-1})$$ where

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

is the Metropolis acceptance probability and

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

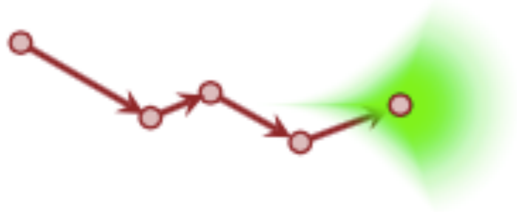
MCMC robot's rules

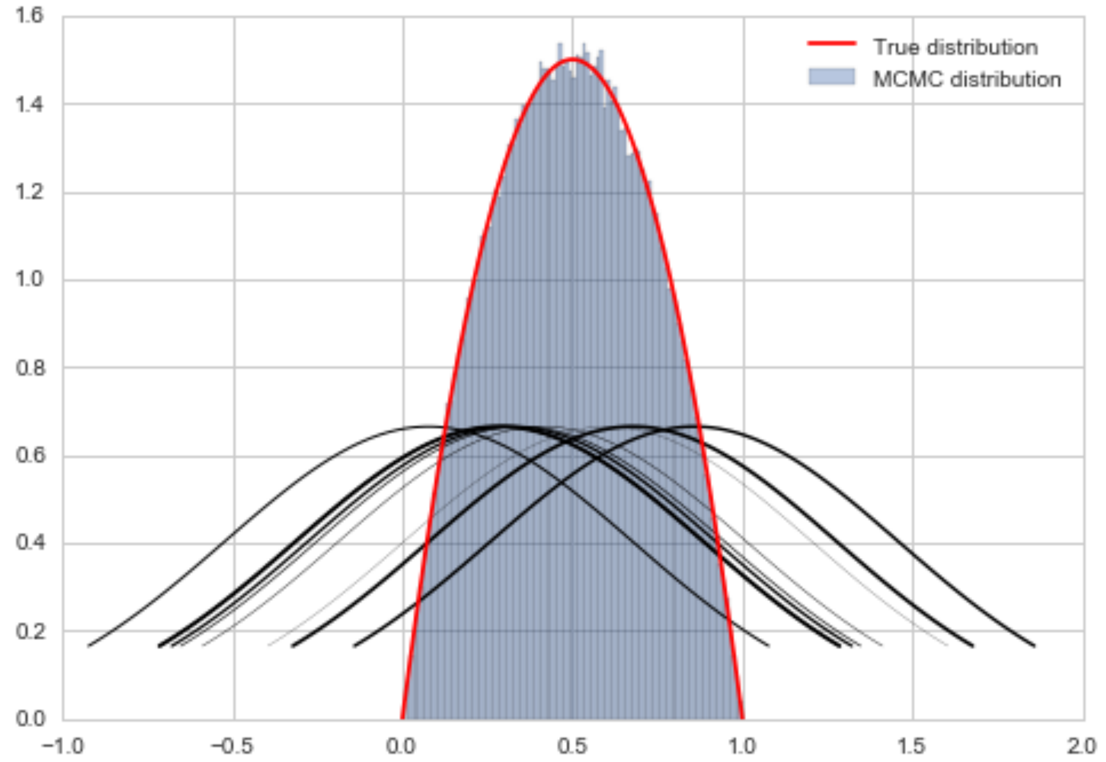


(from Paul Lewis)

```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





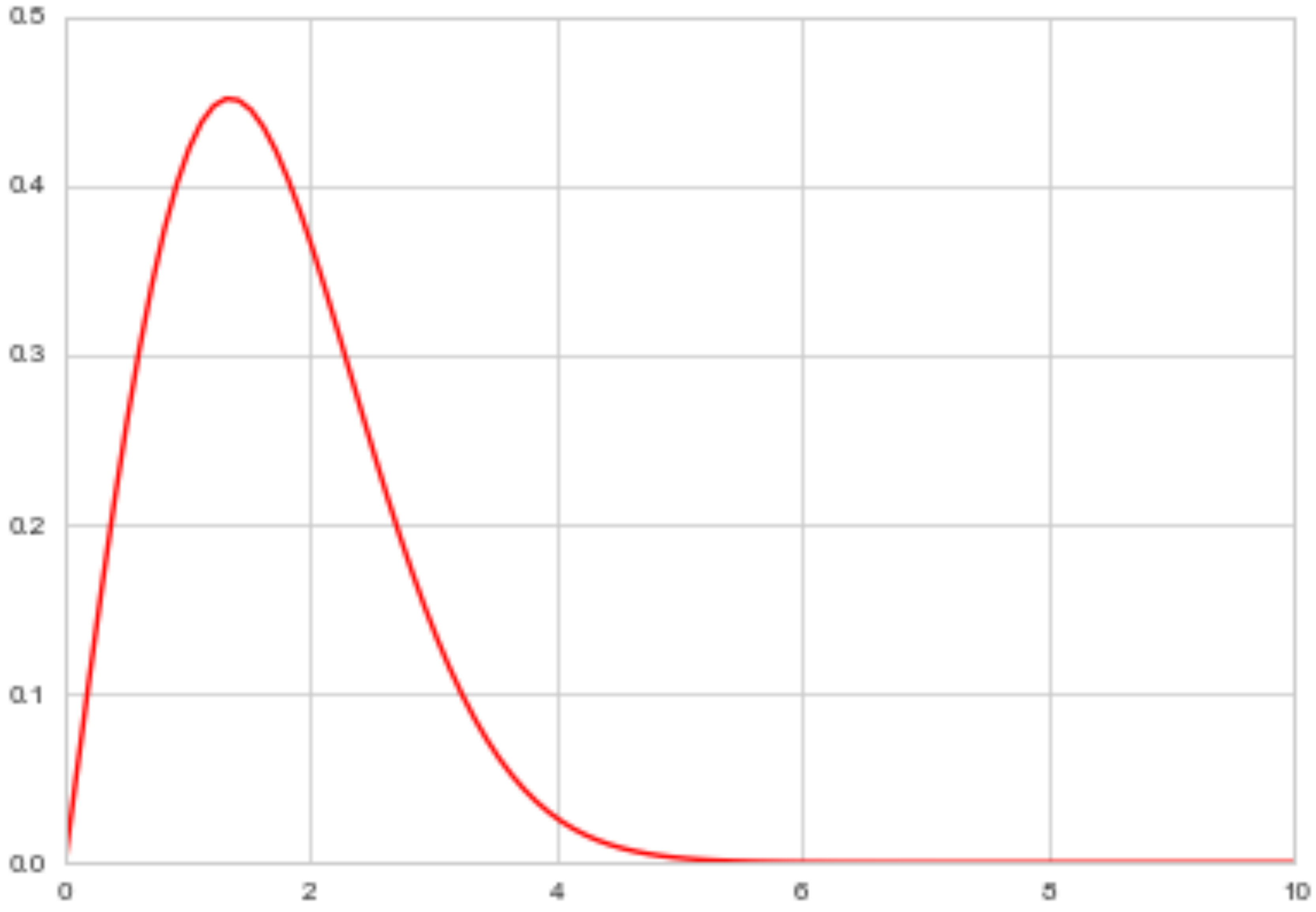
Instead of sampling p we sample q, yielding a new state, and a new proposal distribution from which to sample.

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

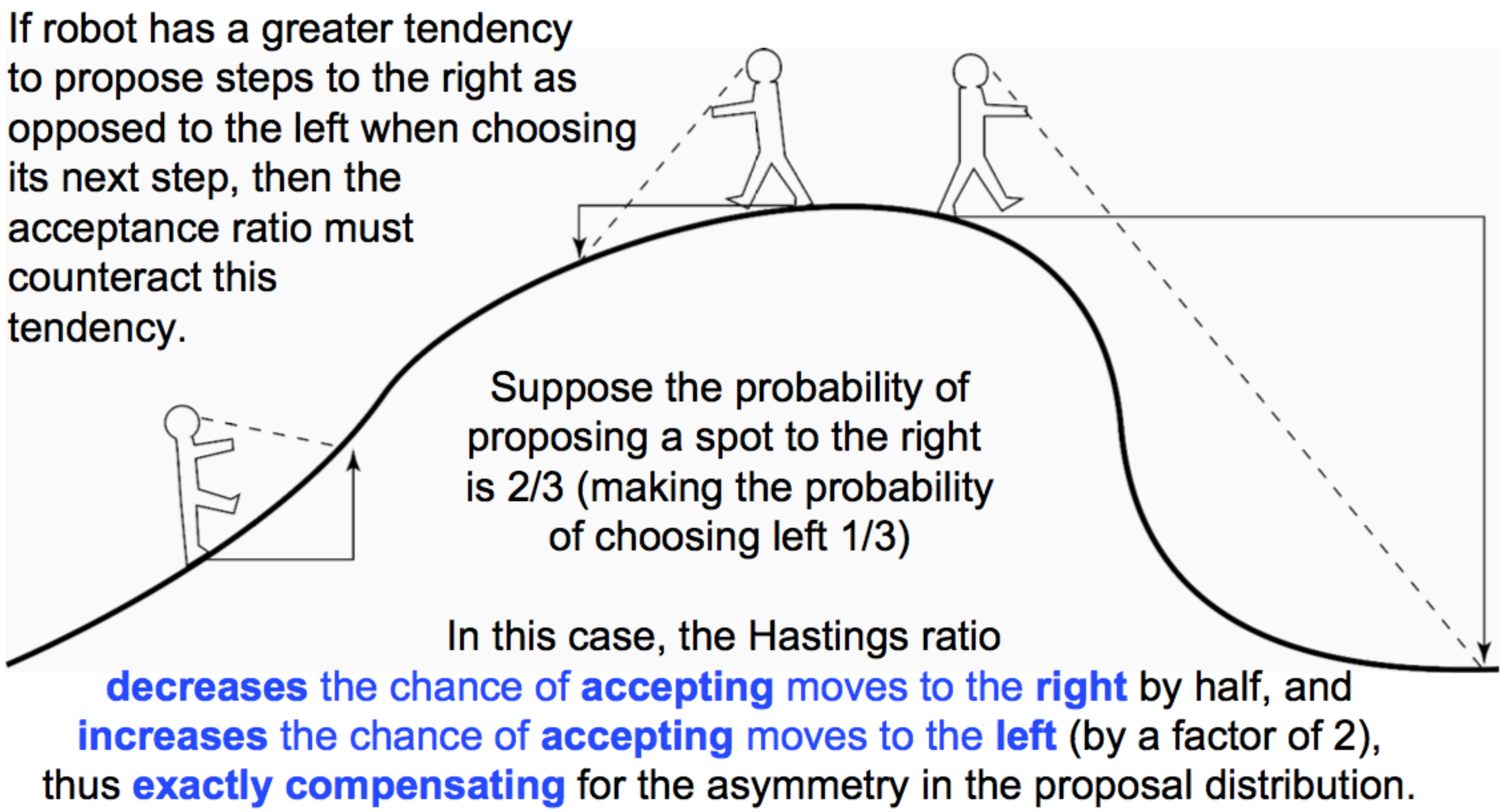
```python
def metropolis_hastings(p,q, qdraw, nsamp, xinit):
    samples=np.empty(nsamp)
    x_prev = xinit
    for i in range(nsamp):
        x_star = qdraw(x_prev)
        p_star = p(x_star)
        p_prev = p(x_prev)
        pdfratio = p_star/p_prev
        proposalratio = q(x_prev, x_star)/q(x_star, 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.

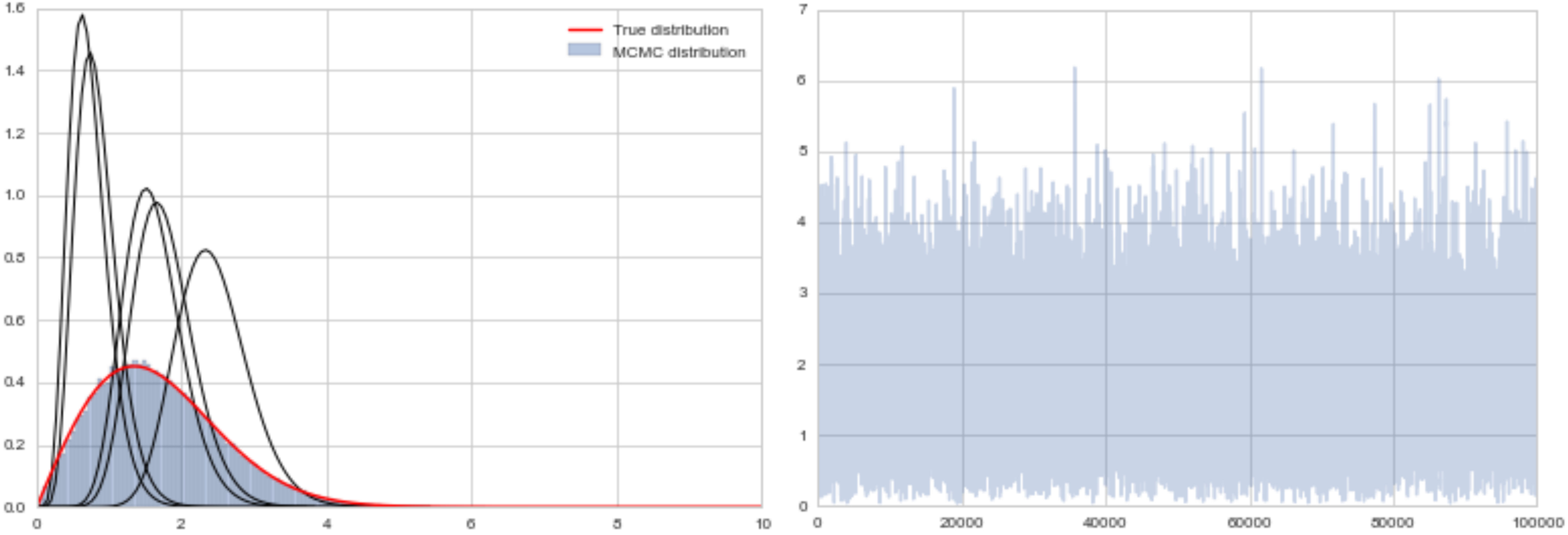


(from Paul Lewis)

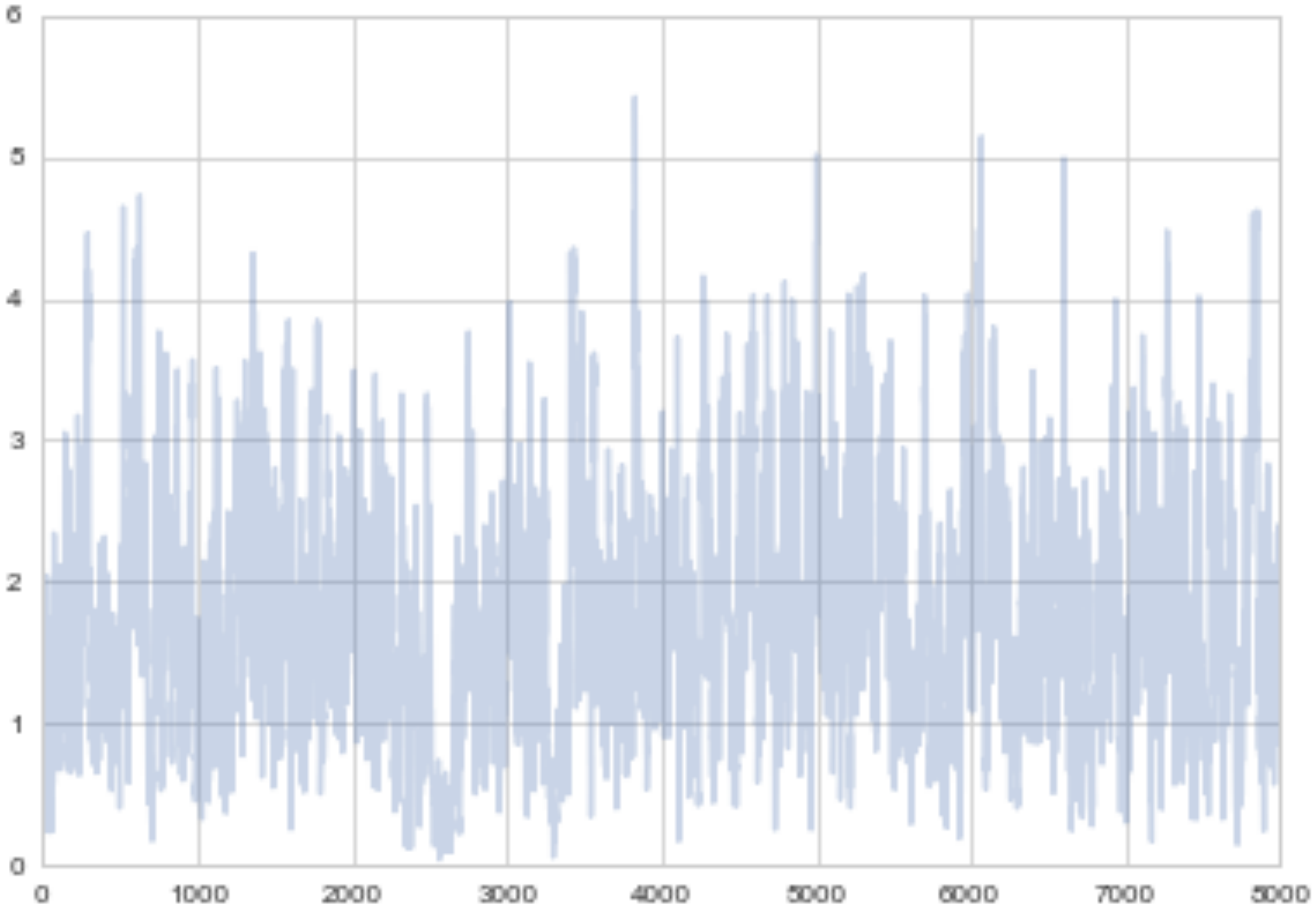
Choice of Proposal

- 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.

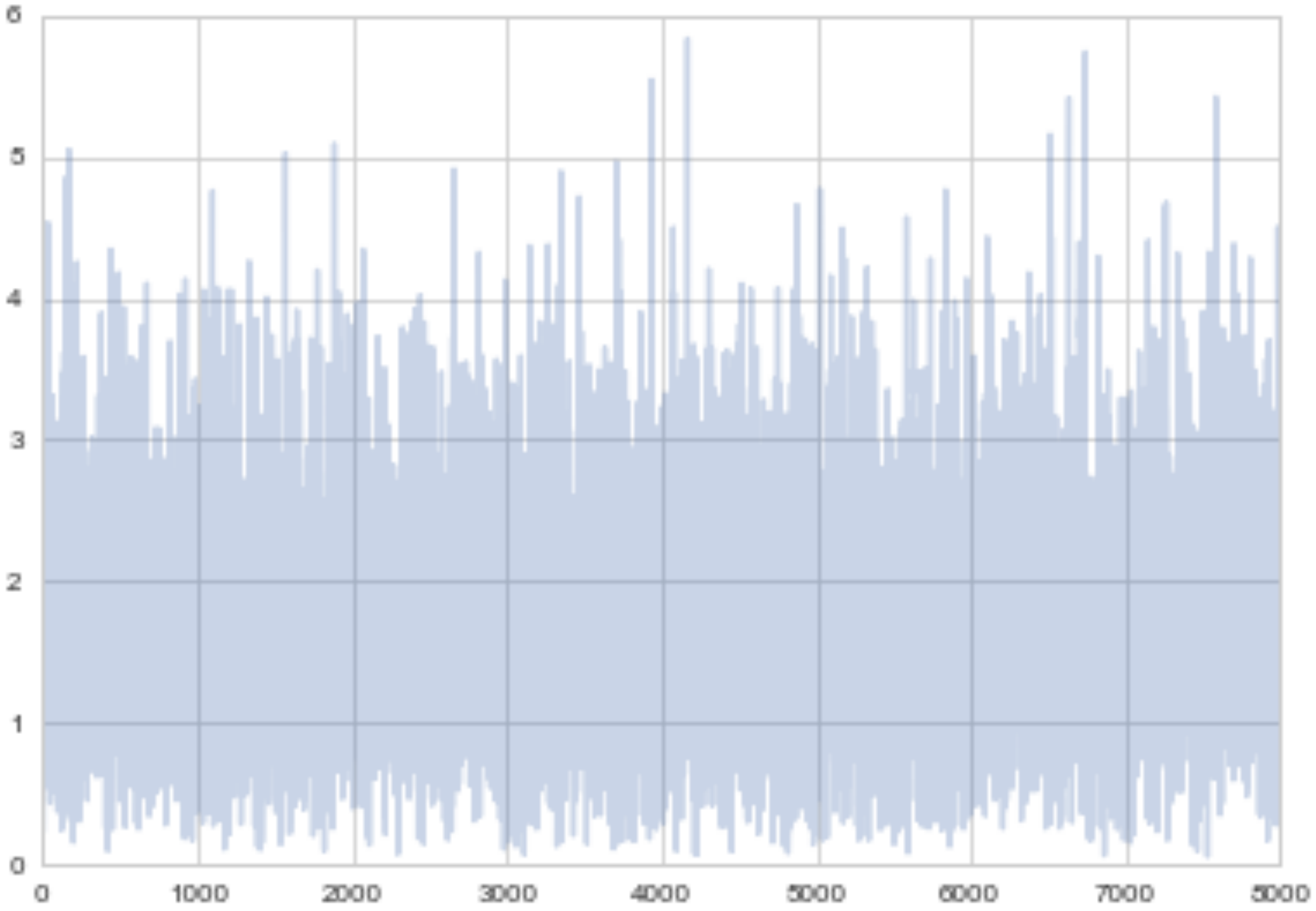
Gamma-Weibull with traceplot



Traceplot after burnin but without thinning



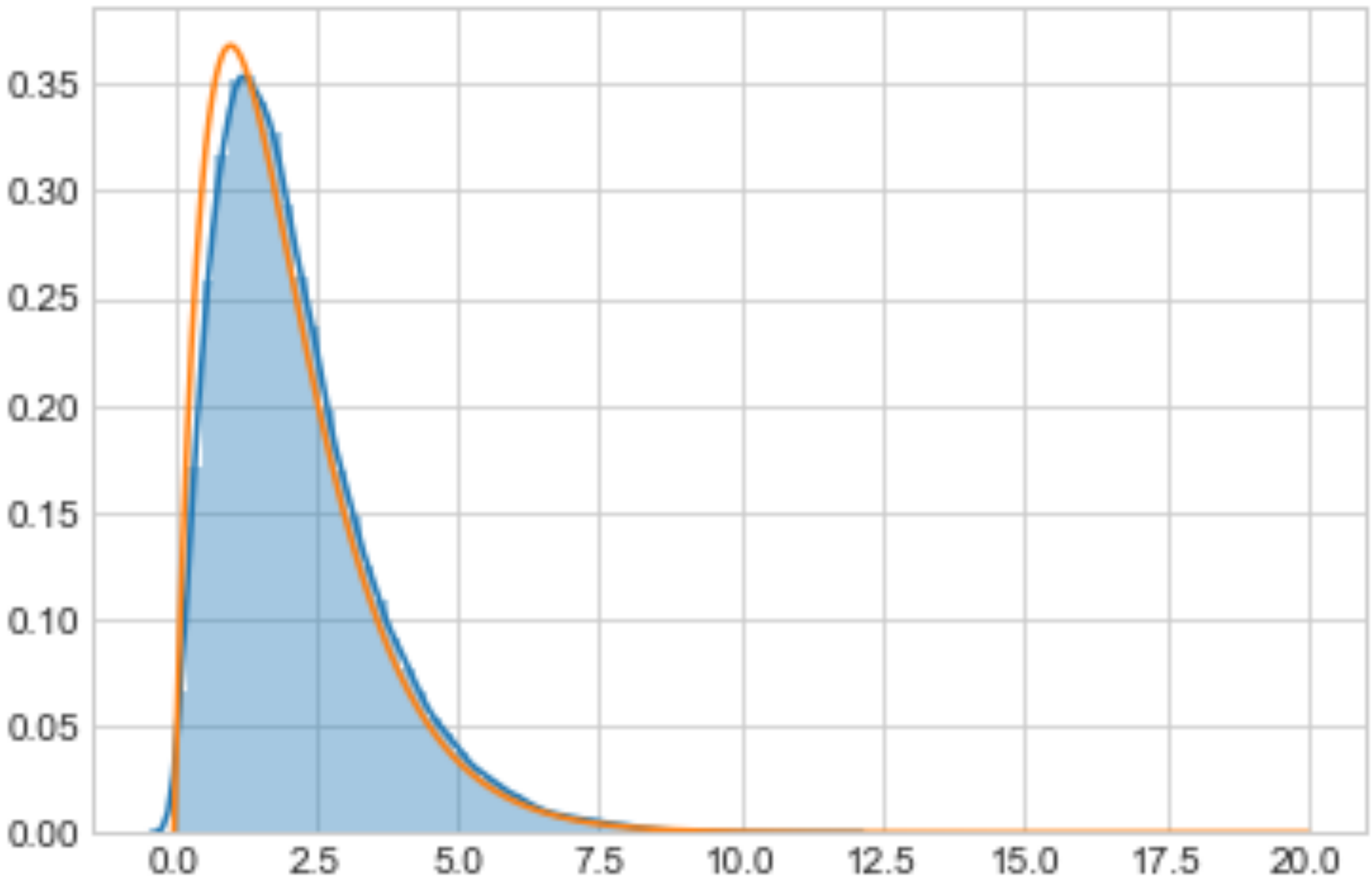
Traceplot after burning and thinning



Is thinning needed?

- jury is out but current thought is no
- does reduce space requirements and remove autocorrelation
- but removing autocorrelation is strictly not needed by ergodicity
- but how much burnin do we need? And how many effective samples?
- soon…

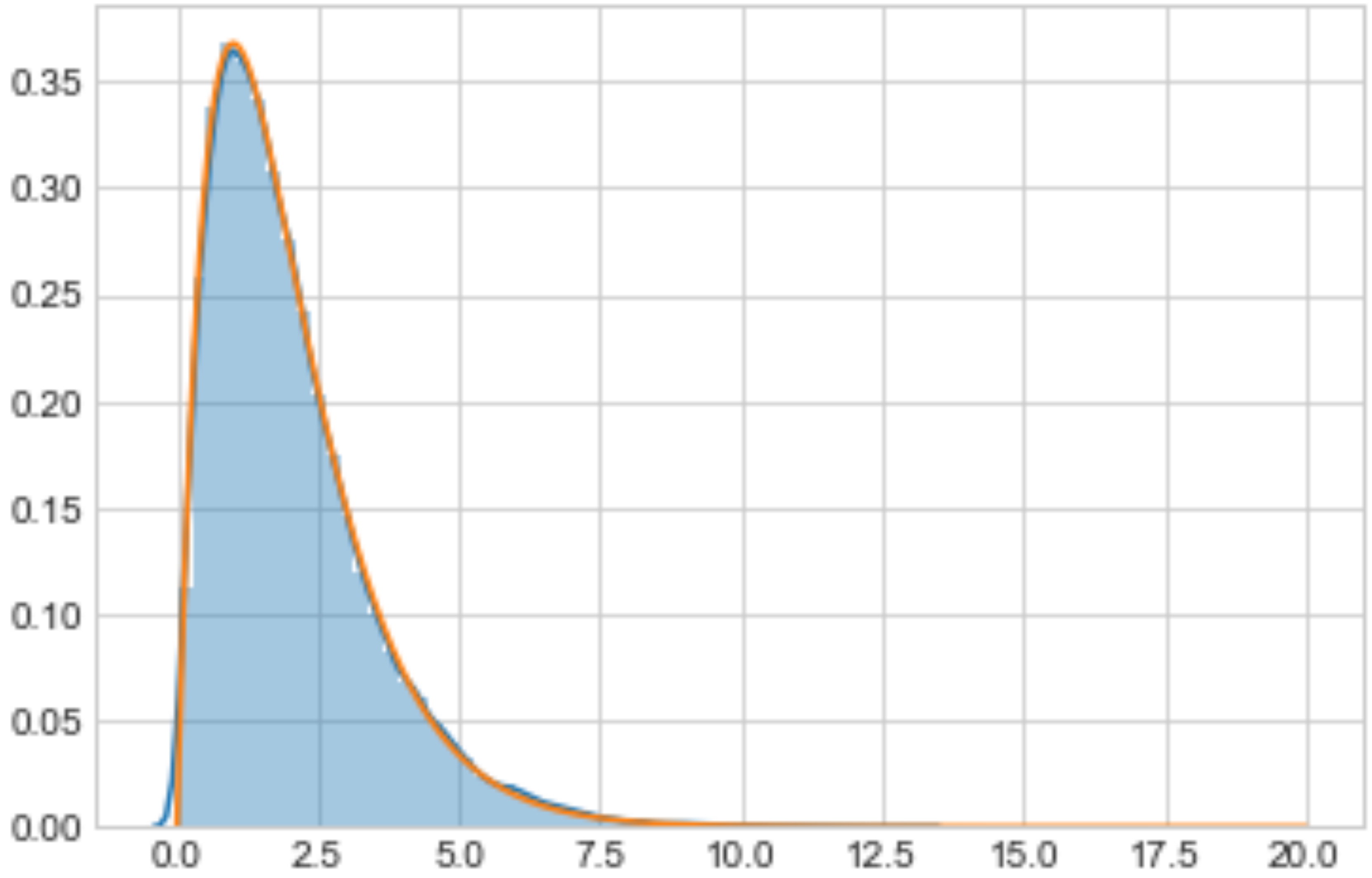
Why not reject? $xe^{-x}, x > 0$




```
target = lambda x: x*np.exp(-x)
proposal = lambda x: np.random.normal(x, 1.0)
def metropolis_broken(p, qdraw, nsamp, xinit):
    samples=np.empty(nsamp)
    x_prev = xinit
    for i in range(nsamp):
        while 1:
            x_star = qdraw(x_prev)
            if x_star > 0:
                break
        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
```

Do it right

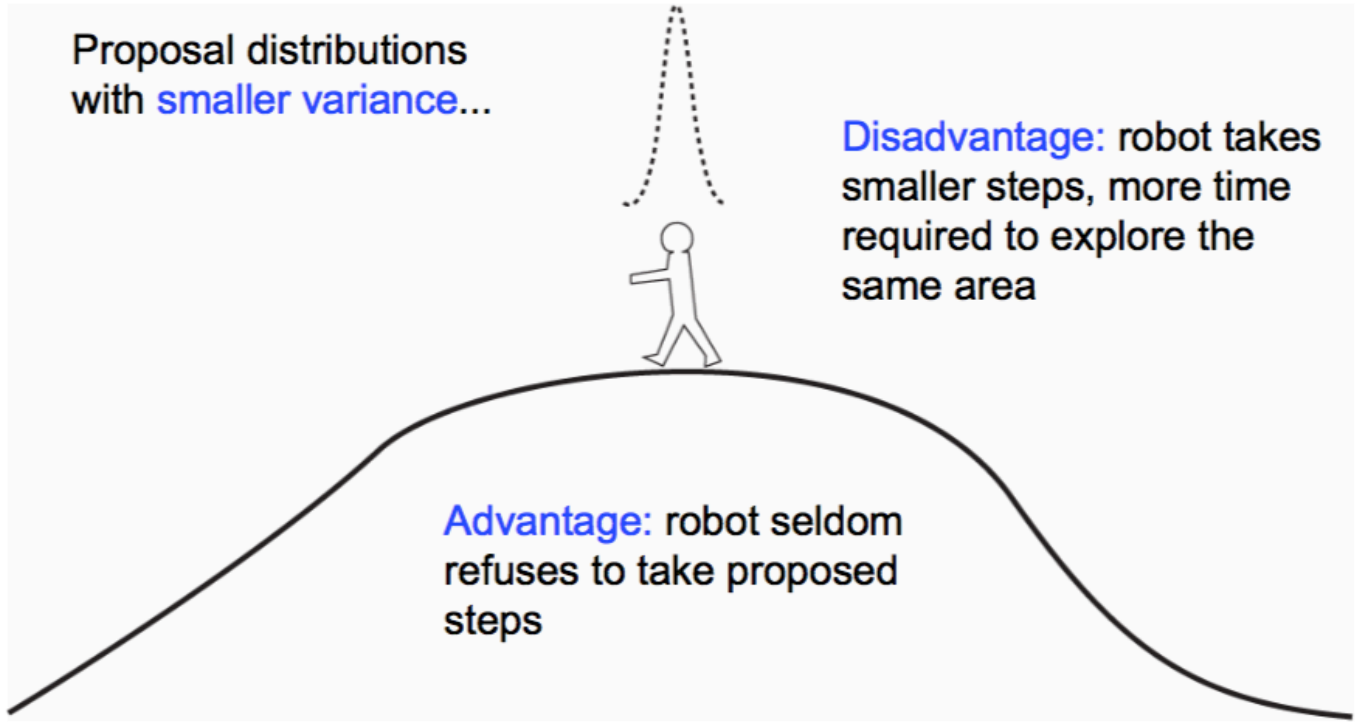
```
prop2 = lambda x: x + np.random.normal()
q = lambda x_prev, x_star: norm.cdf(x_prev)
def metropolis_hastings(p,q, qdraw, nsamp, xinit):
    samples=np.empty(nsamp)
    x_prev = xinit
    accepted=0
    for i in range(nsamp):
        while 1:
            x_star = qdraw(x_prev)
            if x_star > 0:
                break
        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
            accepted +=1
        else:#we always get a sample
            samples[i]= x_prev
    return samples, accepted
```

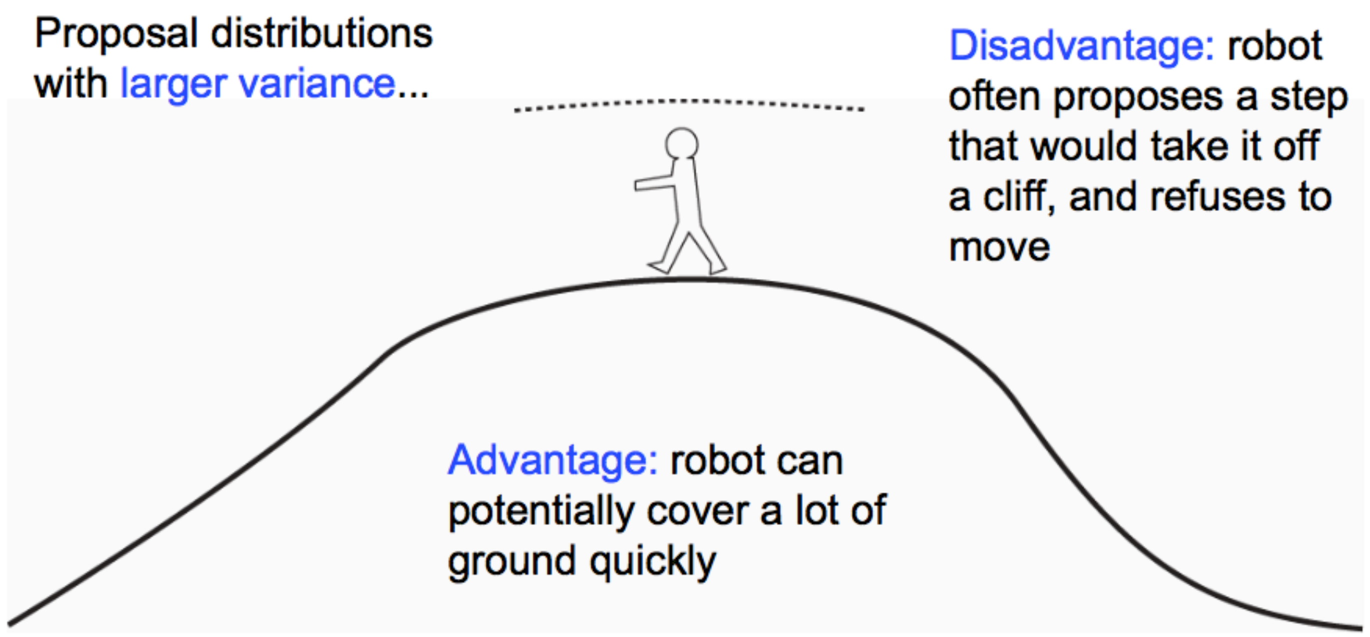


Normalization of distributions

- we dont need to normalize target once we have samples
- unless we need to calculate the "evidence" to compare models
- we do need to make sure we have a normalized proposal

Tuning the width or precision



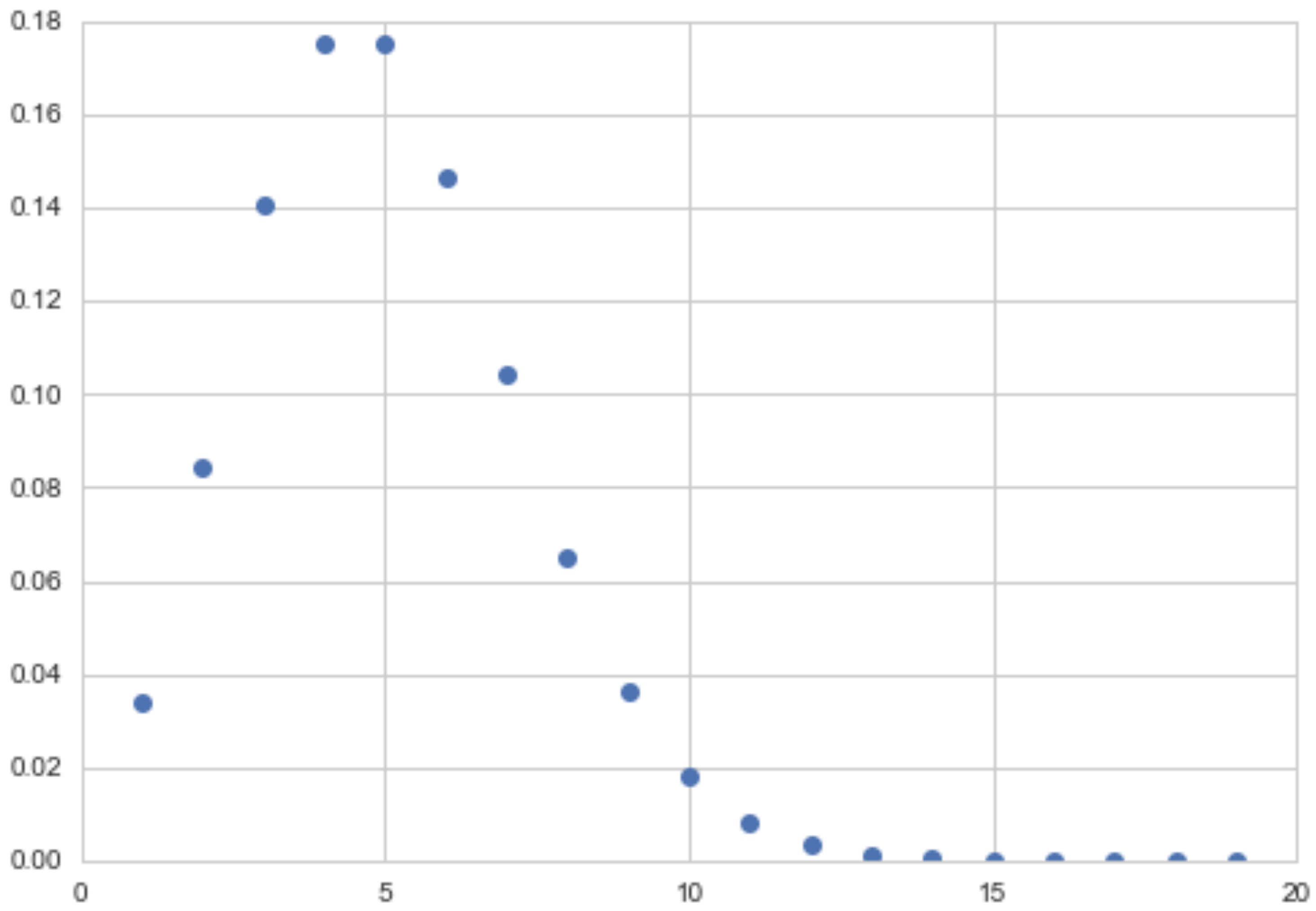


(from Paul Lewis)

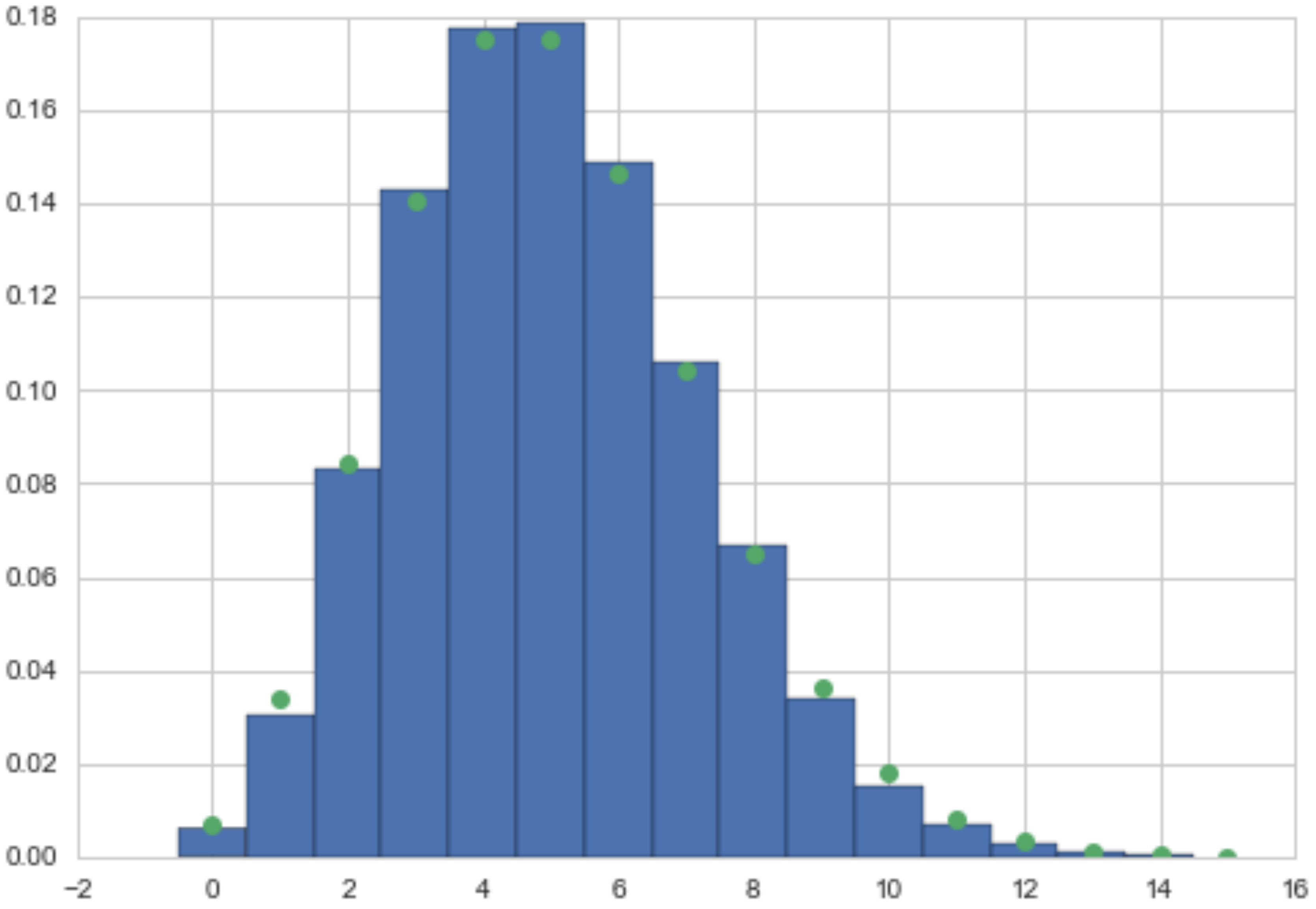
Discrete distribution MCMC

- 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



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

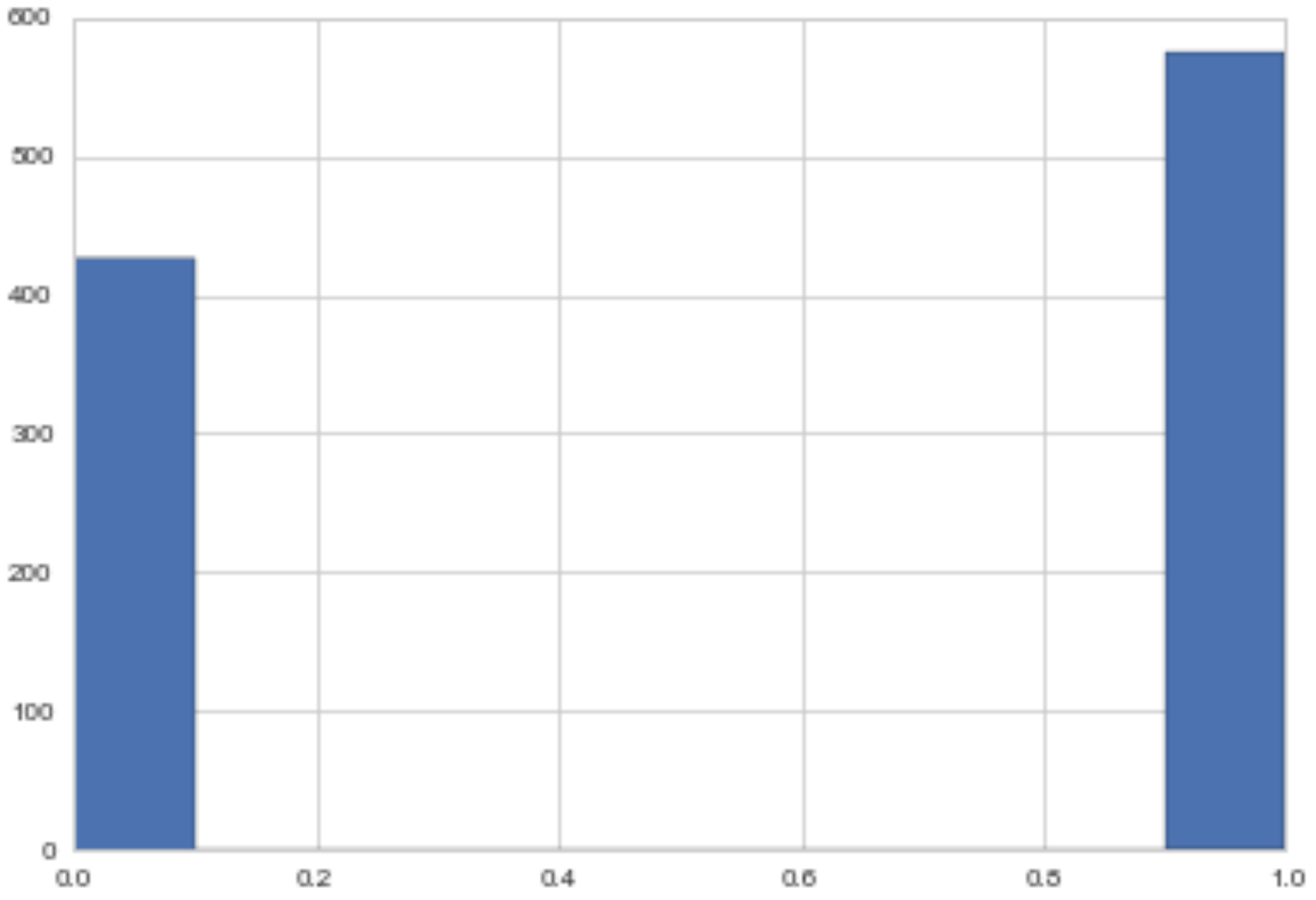


$$p(k) = e^{-\mu} \frac{\mu^k}{k!}.$$

```python
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
```

Summary: 3 Concepts

- proposal
- pdf/pmf
- transition


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 $\theta_{ML} = \underset{\theta}{\operatorname{argmax}} \mathcal{L}$
- 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

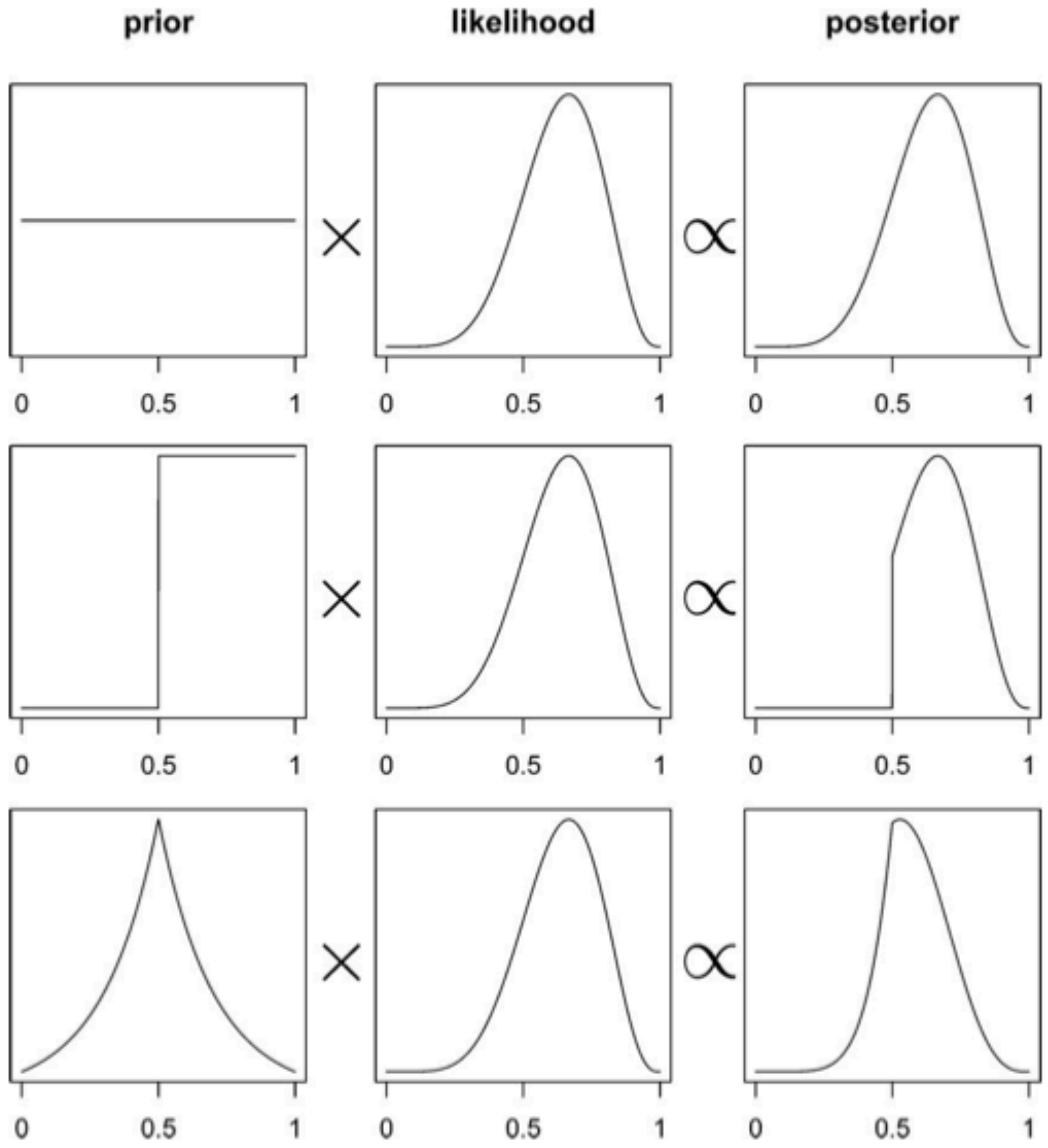
$$p(\theta|y) = \frac{p(y|\theta)\, p(\theta)}{p(y)}$$

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)

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

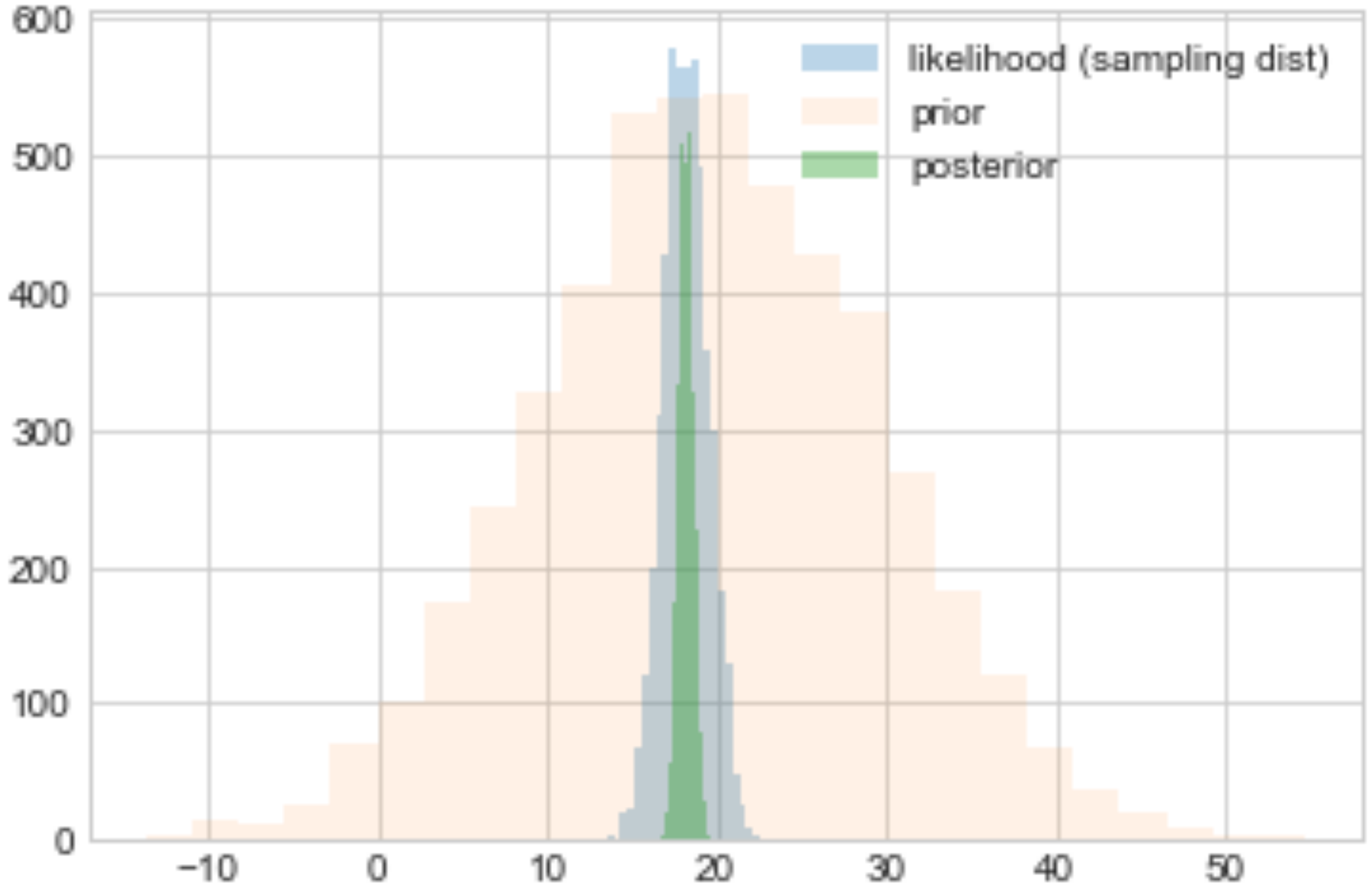
Plug-in Approximation: $p(\theta|y) = \delta(\theta - \theta_{MAP})$

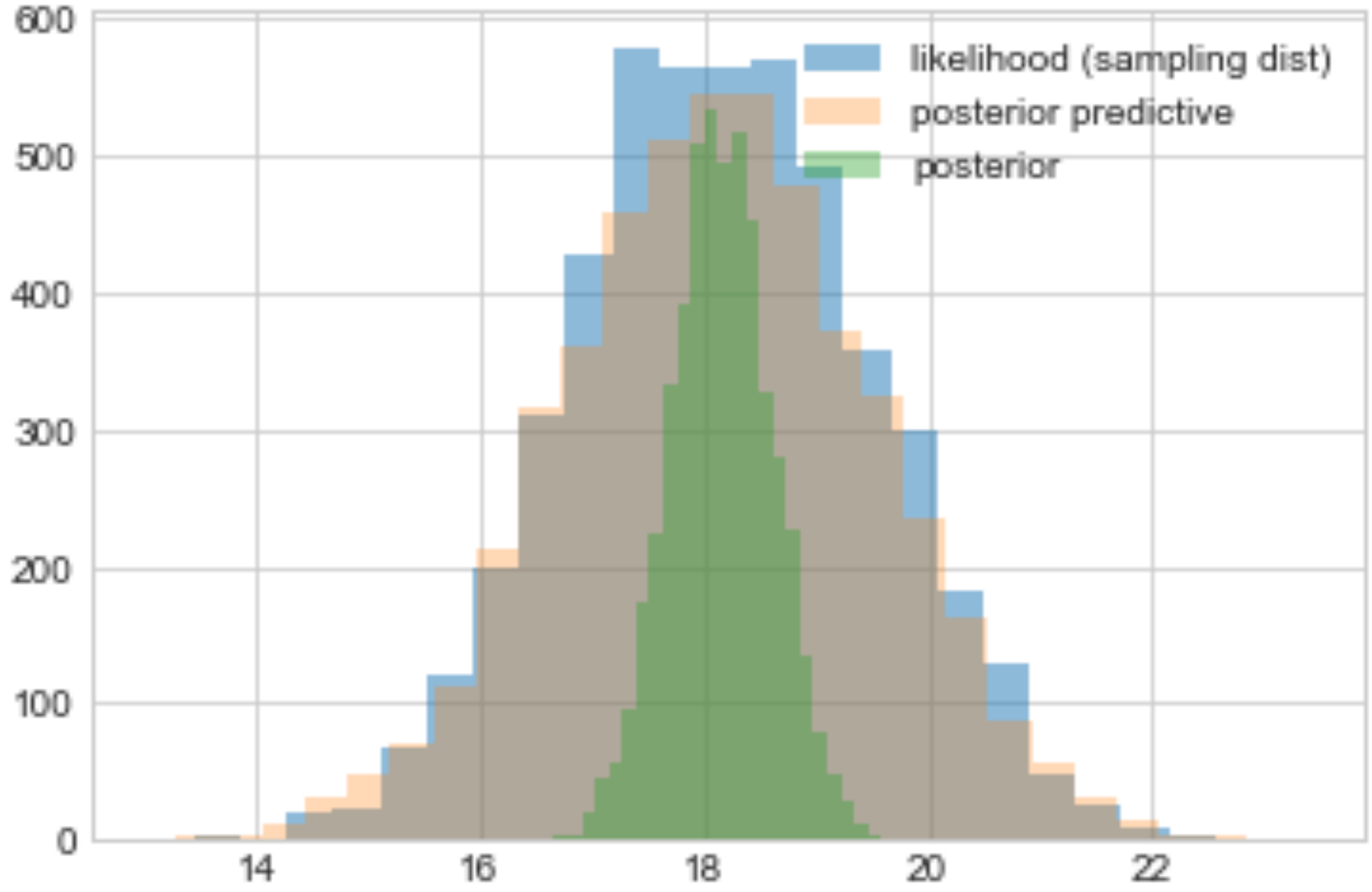
and then draw

$$p(y^*|y) = p(y^*|\theta_{MAP})$$ a sampling distribution.

Normal-Normal Model

```
logprior = lambda mu: norm.logpdf(mu, loc=mu_prior, scale=std_prior)
loglike = lambda mu: np.sum(norm.logpdf(Y, loc=mu, scale=np.std(Y)))
logpost = lambda mu: loglike(mu) + logprior(mu)
def metropolis(logp, qdraw, stepsize, nsamp, xinit):
    samples=np.empty(nsamp)
    x_prev = xinit
    accepted = 0
    for i in range(nsamp):
        x_star = qdraw(x_prev, stepsize)
        logp_star = logp(x_star)
        logp_prev = logp(x_prev)
        logpdfratio = logp_star -logp_prev
        u = np.random.uniform()
        if np.log(u) <= logpdfratio:
            samples[i] = x_star
            x_prev = x_star
            accepted += 1
        else:#we always get a sample
            samples[i]= x_prev

    return samples, accepted
```

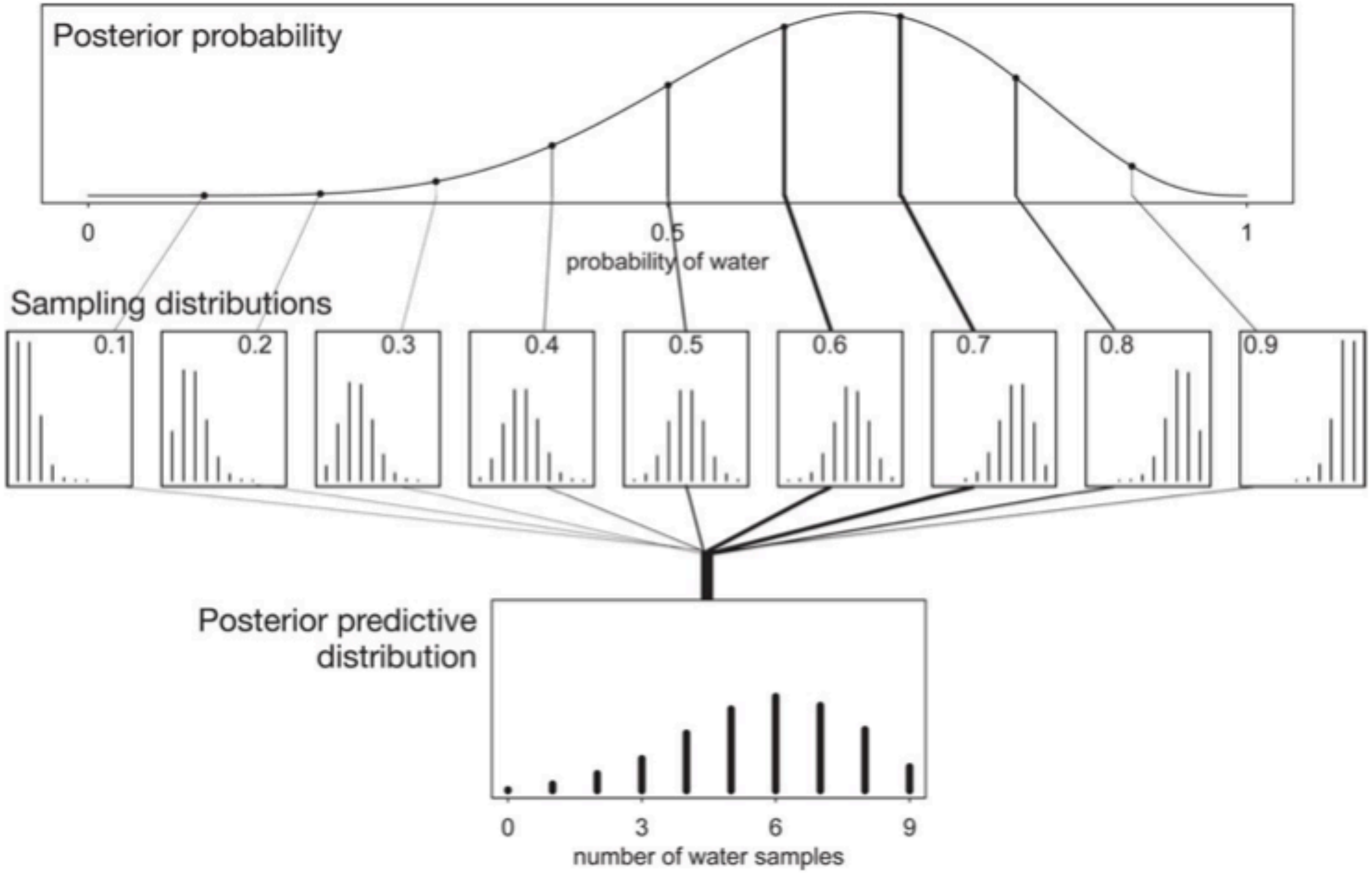

likelihood (sampling dist)
prior
posterior
600
500
400
300
200
100
0
−10
0
10
20
30
40
50


likelihood (sampling dist)
posterior predictive
posterior
600
500
400
300
200
100
0
14
16
18
20
22

Posterior predictive from sampling

- first draw the thetas from the posterior
- then draw y's from the likelihood
- and histogram the likelihood
- these are draws from joint y, θ

Posterior predictive Idea


Posterior probability
0
0.5
1
probability of water
Sampling distributions
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Posterior predictive
distribution
0
3
6
9
number of water samples