Lecture 9

Metropolis-Hastings

Contents

1 Introduction

Hastings generalized the original Metropolis algorithm to allow for **non-symmetric proposals**.

Detailed balance requires that the "net traffic" between any two points x_c and x_n in the state space be zero.

$$\begin{aligned} W(x_c \to x_n)\pi(x_c) &= W(x_n \to x_c)\pi(x_n), \\ W_{nc}\pi_c &= W_{cn}\pi_n, \\ p_{nc}a_{nc}\pi_c &= p_{cn}a_{cn}\pi_n \end{aligned} \tag{1}$$

Where transition probability $W = pa$ has been decomposed into the proposal p, and acceptance a probabilities.

In Metropolis MCMC, proposals are symmetric,

$$p_{nc} = p_{cn}. \tag{2}$$

From detailed balance eqn **??**, this implies,

$$\begin{aligned} p_{nc}a_{nc}\pi_c &= p_{cn}a_{cn}\pi_n \\ a_{nc}\pi_c &= a_{cn}\pi_n. \\ \frac{a_{nc}}{a_{cn}} &= \frac{\pi_n}{\pi_c} \end{aligned} \tag{3}$$

The Metropolis criterion satisfies this constraint with,

$$a_{nc} = \min\left\{1, \frac{\pi_n}{\pi_c}\right\}. \tag{4}$$

In Metropolis-Hastings, the proposal moves can be asymmetric,

$$p_{nc} \neq p_{cn}.$$

This leads to a slight alteration in the acceptance criterion. Since the alteration is minor, it is useful to simply present the result up front.

We define Hastings' ratio r_H as,

$$r_H = \frac{a_{nc}}{a_{cn}} = \frac{p_{cn}\pi_n}{p_{nc}\pi_c} \tag{5}$$

Then the Metropolis-Hastings acceptance criterion is given by,

$$a_{nc} = \min\left\{1, \frac{p_{cn}\pi_n}{p_{nc}\pi_c}\right\}. \tag{6}$$

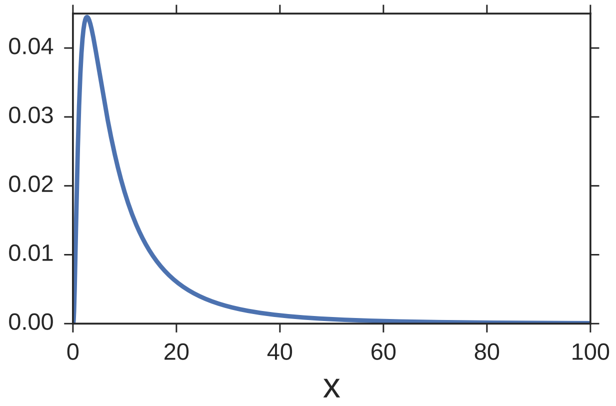
2 Metropolis-Hastings

For symmetric moves, note that the Hastings criterion reduces to the Metropolis acceptance criterion. The difference and utility of using non-symmetric proposals is best illustrated with an example.

Consider a *wide and asymmetric* probability distribution like a log-normal distribution:[1]

$$\pi(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}}\, e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}, \quad x>0 \tag{7}$$

In this example, we set $\mu = 2.0$, and $\sigma = 1.0$.



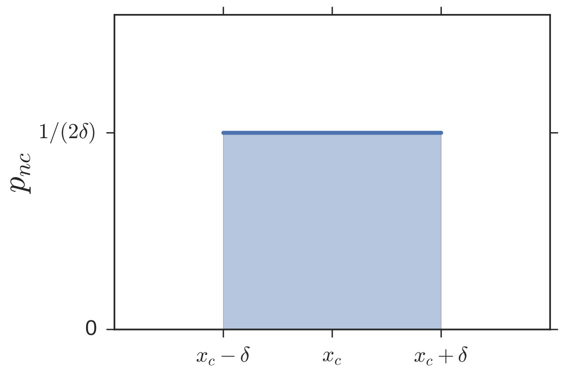
For this lognormal distribution the mean and variance are,

$$\langle x\rangle = e^{\mu+(\sigma^2/2)} = e^{2.5} = 12.18,$$
$$\langle x^2\rangle - \langle x\rangle^2 = (e^{\sigma^2}-1)e^{2\mu+\sigma^2} = (e-1)e^5 \approx 255.$$

2.1 Symmetric Proposals

Let's consider a simple symmetric proposal of the form,

$$x_n \sim p_{nc} = U[x_c-\delta, x_c+\delta].$$



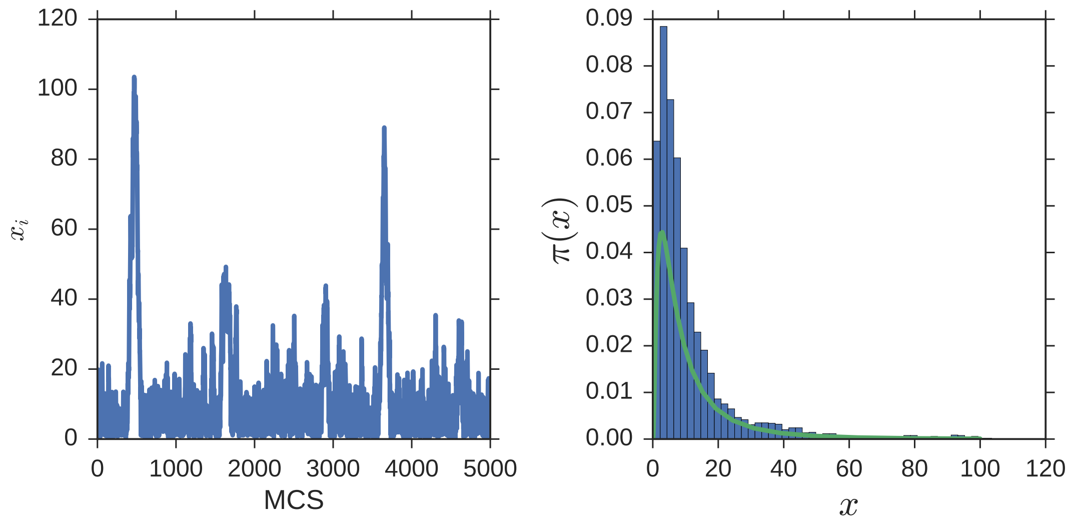
```
def metro_proposal(x, delta):
    return np.random.uniform(x-delta, x+delta)
```

We can implement a simple Metropolis MCMC code around this proposal (see sec. **??**). We set the initial state $x_0 = 5$, and generate 5,000 samples, after recording every tenth sample (`nsteps = 50000, thinning = 10`).[2]

[1] the lognormal distribution is used as a minimal example that exhibits features which make it more amenable for asymmetric moves.

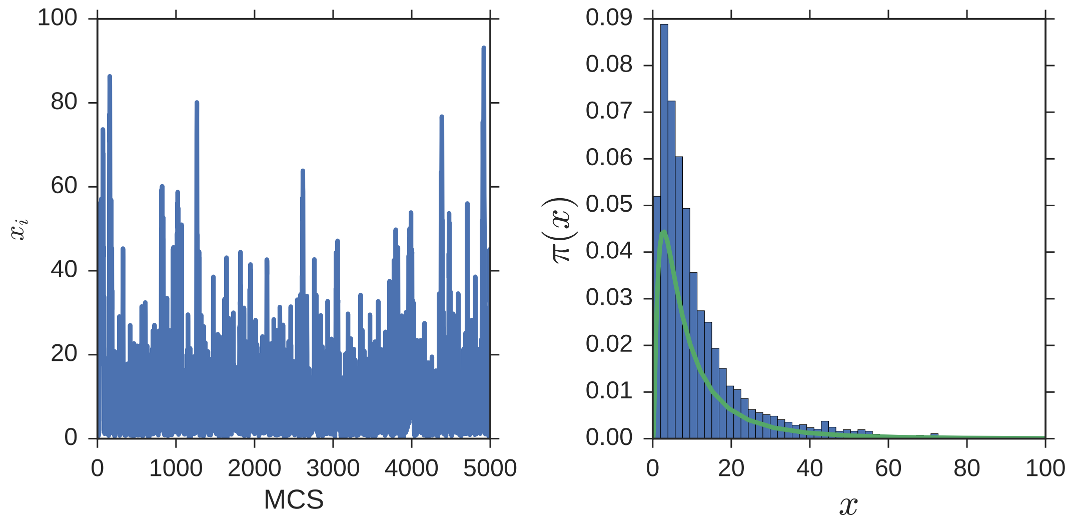
[2] I use a good initial state in this example, so burn-in has only a marginal effect.

For maximum step-size $\delta = 2$, we obtain the following traceplot.



The subfigure on the right compares the histogram with the target distribution. The acceptance ratio is quite high at 0.91. The mean and variance of the samples are $\bar{x}$ = 10.59, and s_x^2 = 82.73, respectively.

Note that both the mean and the standard deviation are under-estimated.[3] Let us repeat the calculation with a larger $\delta = 5$ before summarizing our observations.



The acceptance ratio now drops from 0.91 to 0.78. The mean and variance of the samples are $\bar{x}$ = 10.30, and s_x^2 = 96.8, respectively.

We can compare the $\delta = 2$ and $\delta = 5$ simulations. As expected, when δ increases, the average acceptance rate goes down. Furthermore, as δ increases, the long tail is sampled better; this is evident in the larger variance.

A key problem with both these simulations is the oversampling the small x regime, while undersampling the large x regime. This is reflected in the histogram, and also in the direction of the deviation of the sample means and variances from their true values.

[3]True mean and variance are 12.18 and 255, respectively.

Would you agree or disagree with the statement, “Metropolis MCMC seems to do a reasonable job, although the error bars on the mean are somewhat large.”

2.1.1 Diagnosis

Let us now play doctor, and figure out limitations of Metropolis MCMC for this problem. We observe that the large x regime is sometimes not well-explored leading to an potential underestimation of $\langle x \rangle$.

In principle, a long enough Metropolis MCMC simulation will overcome this deficiency.

However, this seems to be a structural problem with the proposed moves. To jump from $x = 5$ to $x = 100$, we have to take a lot of small hops of size δ. The shape of $\pi(x)$ makes this even harder, since moves that veer too far away from the peak are less likely to be accepted.

One possibility to propose trials that can propose larger moves by increasing δ. Another is to propose asymmetric moves.

3 Asymmetric Proposals

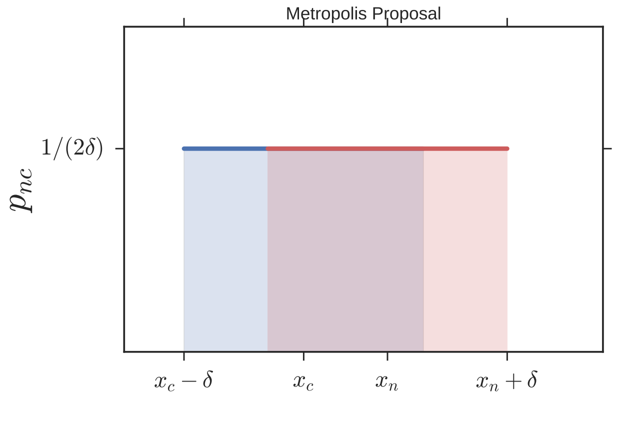
In Metropolis MCMC we use *additive* random steps,

$$x_n \sim p_{nc} = U[x_c - \delta, x_c + \delta],$$

with a proposal PDF given by,

$$p_{nc} = p_{cn} = \frac{1}{2\delta}. \tag{8}$$

The symmetry is evident when we plot the two PDFs together.



Instead, we can propose multiplicative random steps,

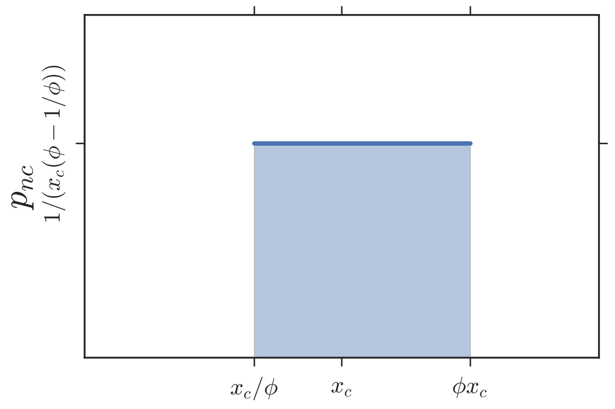
$$x_n = \beta x_c, \tag{9}$$

where $\beta \sim U[1/\phi, \phi]$, with say $\phi = 1.5$. That is, at each step we draw β from a uniform distribution $U[2/3, 3/2]$, and propose x_n by multiplying it with β.

The PDF of the proposal move p_{nc} is,

$$p_{nc} = \frac{1}{x_c\,(\phi - 1/\phi)}. \tag{10}$$

It is a uniform distribution whose height depends on x_c!



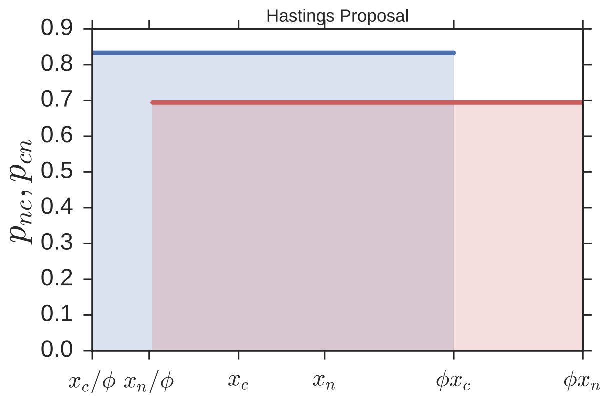
Note that

$$p_{cn} = \frac{1}{x_n\,(\phi - 1/\phi)}, \tag{11}$$

which implies,

$$\frac{p_{nc}}{p_{cn}} = \frac{x_n}{x_c} \neq 1. \tag{12}$$

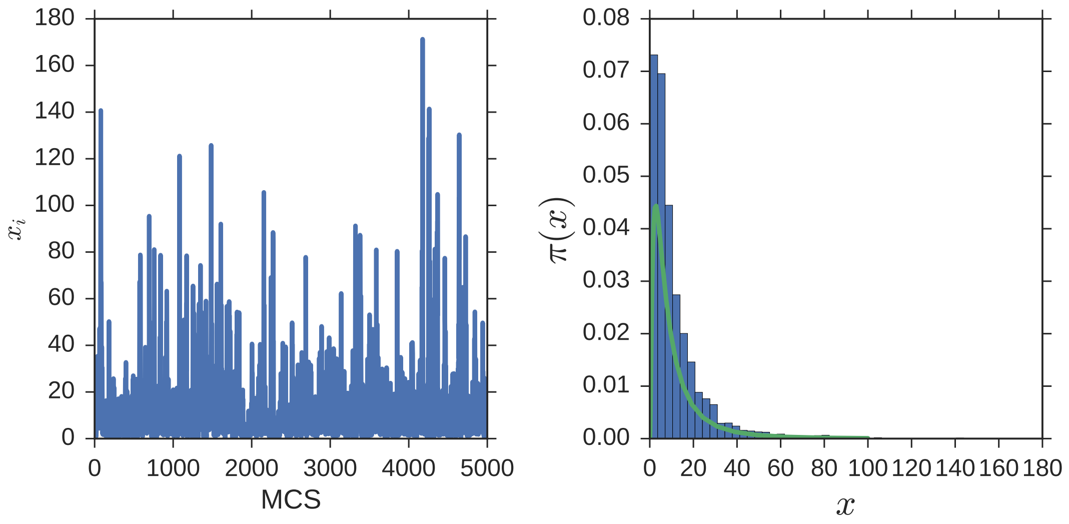
In this asymmetric proposal, the distribution becomes wider as x_c (or x_n) increases.



In the figure above p_{nc} is represented in blue, while p_{cn}, which is centered around x_n is shown in red.

We can incorporate this asymmetric proposal move, and modify the acceptance ratio to reflect the new criterion. The python programs are presented in the appendix (sec. **??**).

When we run the code with $\rho = 1.5$, we get an acceptance ratio ≈ 0.80. The mean $\bar{x}$ = 12.52, and the variance s_x^2 = 350.0, which are much better and conservative estimates.



4 Summary

Hastings' 1970 paper generalized the Metropolis paper. Interestingly, he wrote only three peer-reviewed papers in his career, and supervised a single PhD student.

So when should you prefer Metropolis-Hastings over plain Metropolis? A typical use case is when the distribution $\pi(x)$ is truncated (e.g. $\pi(x < 0) = 0$) or skewed (asymmetric).

Convergence diagnostics and block averaging can be applied to analyze the output of a Metropolis-Hastings sampler, without any changes.

5 Problems

5.1 Thought Questions

(i) What happens if you use the Metropolis acceptance rule for asymmetric proposals?

5.2 Numerical Problems

(i) Sample the lognormal distribution (eqn **??**) with symmetric proposal moves.

Using the Gelman-Rubin convergence criterion, for $\delta = 2.0$, and 5.0 check if $\hat{R} \leq 1.1$ for $M = 5000$ with thinning = 10. Select x_0 uniformly on [0, 100] to create initially overdispersed samples.

Use it, or block averaging to find bound the error on the estimated mean $\bar{x}$.

(ii) Consider the asymmetric triangular proposal distribution.

$$x_n \sim p_{nc} = p(x|x_c) = \begin{cases} 2x/3 & \text{for } 0 \leq x < x_c \\ \dfrac{3x_c - x}{3x_c^2} & \text{for } x_c \leq x < 3x_c \end{cases} \tag{13}$$

(a) Write a program to efficiently sample from this distribution

(b) If this proposal distribution were used to sample a target distribution $\pi(x)$, develop an expression for the Hastings ratio.

(iii) Consider the "fat-tailed" distribution:

$$\pi(x) = \begin{cases} \dfrac{2}{x^3} & x \geq 1 \\ 0 & x < 1. \end{cases}$$

Use the Metropolis-Hastings proposal $p(x|x_c) = \beta x_c$, where $\beta \sim U[1/\phi, \phi]$, with $\phi = 2.0$ to sample the distribution.

(a) Find the expected mean $\langle x \rangle$ analytically?

(b) Use Gelman-Rubin criterion to find a simulation run long enough to convince you that the MCMC chains have converged

(c) Use block averaging to determine an estimate of the error in the computed mean.

(d) What happens to the computed $\langle x \rangle$ if you accidentally use the Metropolis acceptance rule for the Metropolis-Hastings proposal above?

A Appendix

A.1 Python Code

A.1.1 Metropolis MCMC for Lognormal Distribution

```python
def pdist(x, mu=2.0, sigma=1.0):
    """the log normal distribution"""
    if(x > 0.):
        f = 1/(np.sqrt(2*np.pi)*x*sigma) *
                np.exp(-0.5*((np.log(x)-mu)/sigma)**2)
    else:
        f = 0.
    return f

def metro_proposal(x, delta):
    """symmetric uniform proposal"""
    newx = np.random.uniform(x-delta, x+delta)
    return newx

def metro_accept(xn, fc, f):
    """metropolis acceptance"""
    acc = False
    fn  = f(xn)

    ratio = f(xn)/fc

    if ratio > 1.:
        acc = True
    elif np.random.rand() < ratio:
        acc = True
    else:
        fn = fc

    return acc, fn
```

The driver routine to tie these subroutines together is next.

```
def metropolis(delta, nsteps=50000, thin=10):
    """delta = proposal window size,
       nsteps = #MCS (default is 50000)
       thinning = (default = 10)"""

    # initial state
    x = 5.
    f = pdist(x)

    AccRatio = 0.0
    NumSucc = 0

    # store samples in recz
    recz = np.zeros(int(nsteps/thin))

    # main loop
    for iMCS in range(nsteps):

        newx = metro_proposal(x, delta)
        accept, newf = metro_accept(newx, f, pdist)

        if accept:
            NumSucc += 1
            x = newx
            f = newf

        if (iMCS % thin) == 0:
            recz[int(iMCS/thin)] = x

    AccRatio = float(NumSucc)/float(nsteps)
    return recz, AccRatio
```

A.1.2 Metropolis-Hastings MCMC for Lognormal Distribution

```
def hastings_proposal(x, rho=1.5):
    beta = np.random.uniform(1./rho, rho)
    return beta * x

def hastings_accept(xn, xc, fc, f):
    acc = False
    fn  = f(xn)
    ratio = f(xn)/fc * xc/xn
    if ratio > 1.:
        acc = True
    elif np.random.rand() < ratio:
        acc = True
    else:
        fn = fc
    return acc, fn
```

Finally, the driver.

```
def hastings(rho, x0=5.0, nsteps=50000, thin=10):
    x = x0
    f = pdist(x)

    AccRatio = 0.0
    NumSucc = 0

    recz = np.zeros((nsteps/thin))

    # main loop
    for iMCS in range(nsteps):
        # proposal and accept calls different
        newx = hastings_proposal(x, rho)
        accept, newf = hastings_accept(newx, x, f, pdist)

        if accept:
            NumSucc += 1
            x = newx
            f = newf

        if (iMCS % thin) == 0:
            recz[iMCS/thin] = x

    AccRatio = float(NumSucc)/float(nsteps)
    return recz, AccRatio
```