STAT 37810 Final Project: Metropolis-Hastings in R

Eileen Li

October 27, 2016

Introduction

To generate a sample from Beta(6,4), we'll use the Metropolis-Hastings scheme, which proceeds from ϕ_{old} to ϕ_{new} as follows.

1. Given $c > 0$, generate ϕ_{prop} from the distribution
$$\phi_{\text{prop}} \mid \phi_{\text{old}} \sim \text{Beta}\left(c\phi_{\text{old}}, c(1-\phi_{\text{old}})\right).$$
2. With probability
$$\min\left(1, \frac{\pi(\phi_{\text{prop}})\tilde{\pi}(\phi_{\text{old}} \mid \phi_{\text{prop}})}{\pi(\phi_{\text{old}})\tilde{\pi}(\phi_{\text{prop}} \mid \phi_{\text{old}})}\right),$$
where π is the target Beta(6,4) distribution and $\tilde{\pi}$ is the proposal distribution from step 1, let $\phi_{\text{new}} = \phi_{\text{prop}}$. Otherwise, let $\phi_{\text{new}} = \phi_{\text{old}}$.

First, we specify the necessary parameters.

```
set.seed(1993)
c = 1
alpha = 6
beta = 4
niter = 10000 # Number of iterations
```

The following function returns the acceptance threshold from step 2.

```
acc = function(phi_prop, phi_old){ # Acceptance threshold
  dbeta(phi_prop, alpha, beta) * dbeta(phi_old, c * phi_prop, c * (1 - phi_prop)) /
    dbeta(phi_old, alpha, beta) / dbeta(phi_prop, c * phi_old, c * (1-phi_old))
}
```

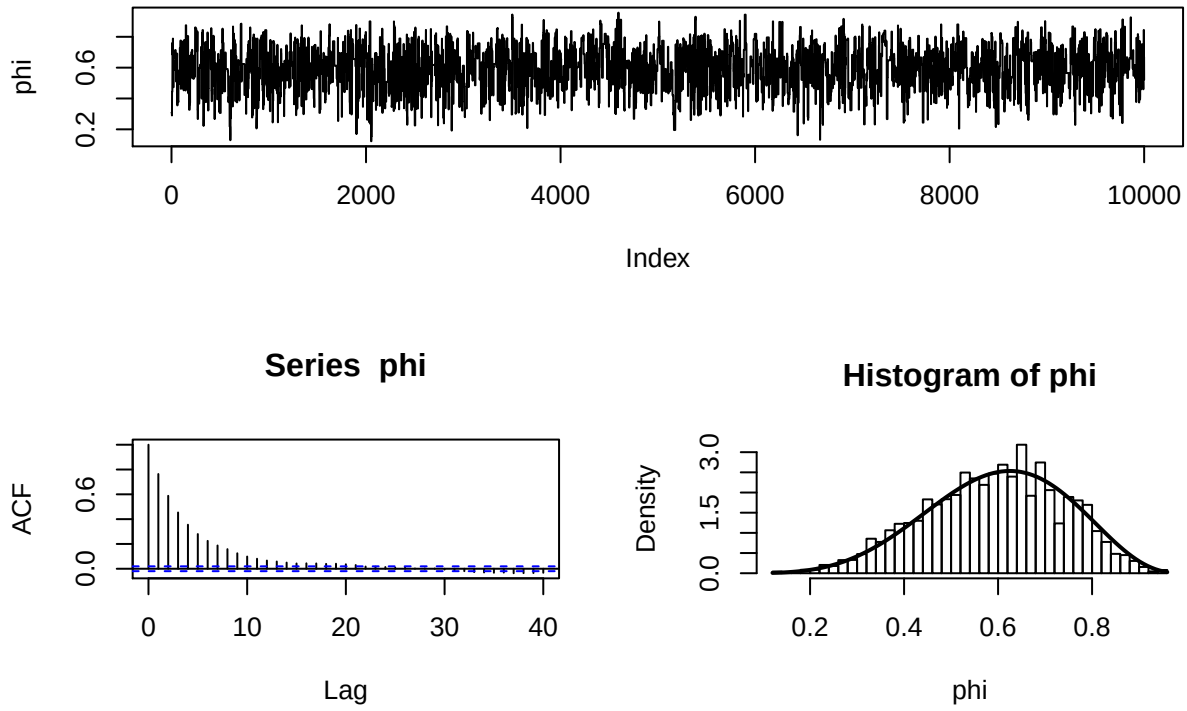
The first sample will be generated from Unif(0,1).

```
phi = rep(0, niter) # Allocate a vector
phi[1] = runif(1) # Initialize
```

Running the algorithm

```
for(n in 2:niter){
  phi_prop = rbeta(1, c * phi[n-1], c * (1 - phi[n-1]))
  if(runif(1) < acc(phi_prop, phi[n-1])){
    phi[n] = phi_prop # Accept
  } else phi[n] = phi[n-1] # Reject
}
```

Evaluating performance



We've plotted the trace (top), autocorrelation (bottom left), and histogram (bottom right) of the sample with the Beta(6,4) density overlaid. It appears that the chain has settled almost immediately and its autocorrelations decay pretty quickly to 0 – roughly speaking, 25 or so samples from this chain has the same statistical value as a single independent sample from Beta(6,4). Visually, the empirical density seems to fit the theoretical one quite well, but we can further check this using the Kolmogorov-Smirnov test.

```
ks.test(jitter(phi[5000:10000]), "pbeta", alpha, beta) # Jitter is to break ties
```

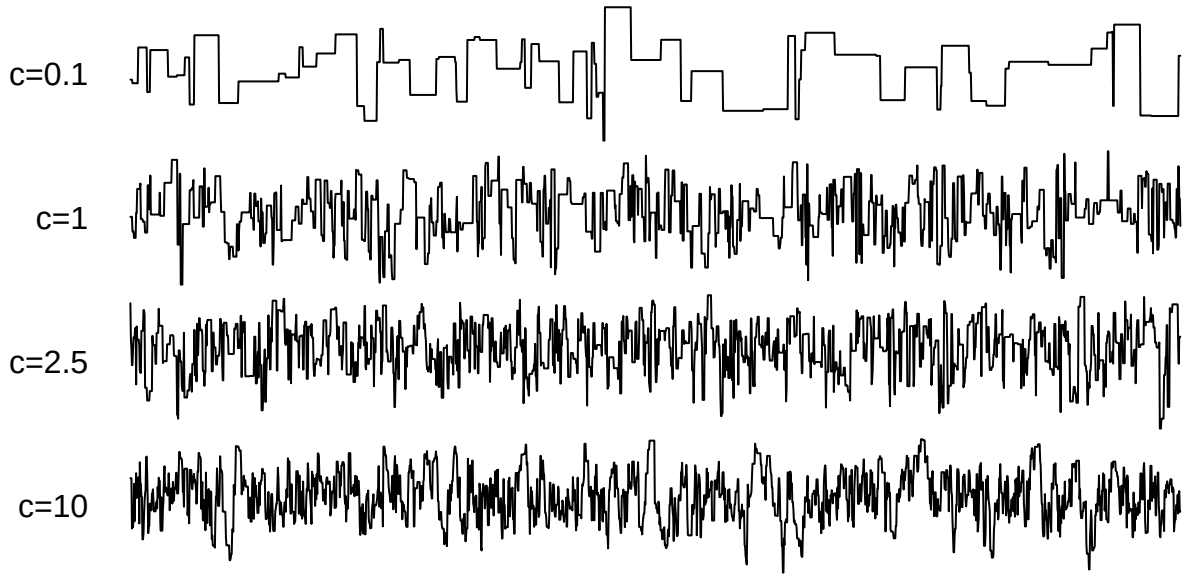
```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  jitter(phi[5000:10000])
## D = 0.026109, p-value = 0.002187
## alternative hypothesis: two-sided
```

The p-value is significant, which means we reject the null hypothesis that the sample is drawn from Beta(6,4).

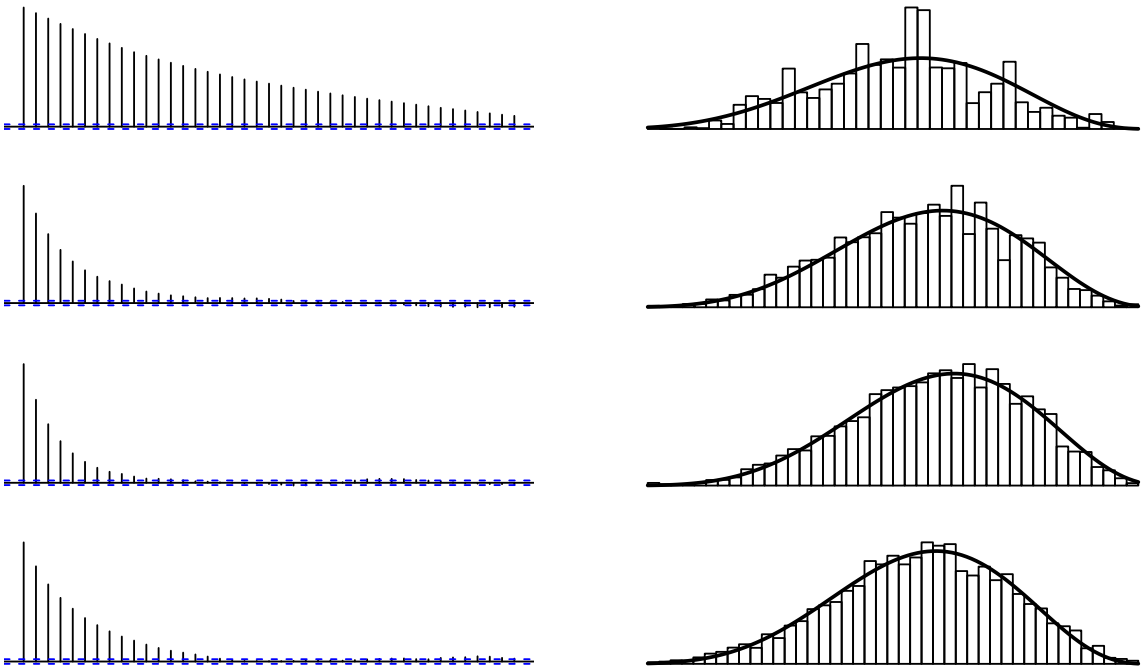
Modifying the proposal distribution

Now we'll modify step 1 of the scheme and instead try setting $c = 0.1$, $c = 2.5$, and $c = 10$ in the proposal distribution. We'll call the resulting samples respectively `phi_0.5`, `phi_2.5`, and `phi_10`.

Trace plots for the last 2000 steps of each chain are shown below. Notice that $c = 0.1$ probably isn't optimal because the acceptance rate is too low – most of the time ϕ_{prop} is rejected and the chain stays at its previous value.



Autocorrelation plots and histograms are shown below. The autocorrelation of $c = 2.5$ decays the quickest, and while $c = 2.5$ and $c = 10$ appear to match the theoretical density better than $c = 0.1$ and $c = 1$ do, it's difficult to tell which one fits better.



Performing the Kolmogorov-Smirnov test quantitatively confirms our visual inspections. The only p-value that isn't significant is the one for $c = 2.5$ ($p = 0.1125$), so the test cannot distinguish the Metropolis-Hastings sample from a sample actually drawn from Beta(6,4). With all the above considerations in mind, we hope the reader agrees that $c = 2.5$ performs the best among these samples.