Coursera Notes for Bayesian Statistics: Techniques and Models

Jordan Katz

June 1, 2020

Week 2

Metropolis-Hastings

Allows us to sample from generic distribution (whose normalizing constant may not be known). To accomplish this, we effectively construct a Markov Chain whose stationary distribution is the target distribution.

Say we want to know $p(\theta)$ but we only know $g(\theta)$ where $p(\theta) \propto q(\theta)$.

Algorithm:

1. Select initial value θ_0
2. for $i = 1, \ldots, m$ repeat:
   a. Draw candidate $\theta^* \sim q(\theta^*|\theta_{i-1})$
   b. Define $\alpha = \frac{g(\theta^*)/q(\theta^*|\theta_{i-1})}{g(\theta_{i-1})/q(\theta_{i-1}|\theta^*)} = \frac{g(\theta^*)}{g(\theta_{i-1})} \frac{q(\theta_{i-1}|\theta^*)}{q(\theta^*|\theta_{i-1}))}$
      i. if $\alpha \geq 1$:
         accept θ^* and set $\theta_i \leftarrow \theta^*$
      ii. $0 < \alpha < 1$:
         with prob α: accept θ^* and set $\theta_i \leftarrow \theta^*$
         with prob $1-\alpha$: reject θ^* and set $\theta_i \leftarrow \theta_{i-1}$

Where q here is the candidate generating distribution which may or may not depend on θ_{i-1}.

One choice is to make q the same distribution regardless of the value θ_{i-1}. If we take this option, we want $q(\theta)$ to be similar to $p(\theta)$ to best approximate it. A high acceptance rate is a good sign here but still may want q to have a larger variance then p to assure we are exploring the space well.

Another choice – one which *does* depend on θ_{i-1} – is to choose a distribution q that is centered on θ_{i-1}. In any symmetric case, we have the property $q(a|b) = q(b|a)$, so step 2 in the algorithm above reduces to

$$\alpha = \frac{g(\theta^*)}{g(\theta_{i-1})}$$

A common choice for such a distribution is $N(\theta_{i-1}, 1)$, or in order words, a Gaussian random walk: $\theta^* = \theta_{i-1} + N(0,1)$ In this particular case, we have

$$q(\theta^*|\theta_{i-1}) = \frac{1}{\sqrt{2\pi}} \exp\left[-0.5(\theta^* - \theta_{i-1})^2\right] = q(\theta_{i-1}|\theta^*)$$

The "size" of the random walk step can affect acceptance (and thus convergence) rate. A high acceptance rate is not a good sign here. If random walk is taking too small of steps, it will accept candidate more often but will take a long time to fully explore the space. If it is taking too large of steps, many proposals will have low probabilities which leads to a low acceptance rate. This amounts to "wasted" samples. Ideally, a random walk sampler should have an acceptance rate between 23% and 50%.

Example: Suppose $y_i|\mu \overset{iid}{\sim} N(\mu, 1)$ for $i = 1, \ldots, n$ and $\mu \sim t(0, 1, 1)$. We want to sample from the posterior distribution $p(\mu|y_1, \ldots, y_n)$, which we can analytically show is proportional to

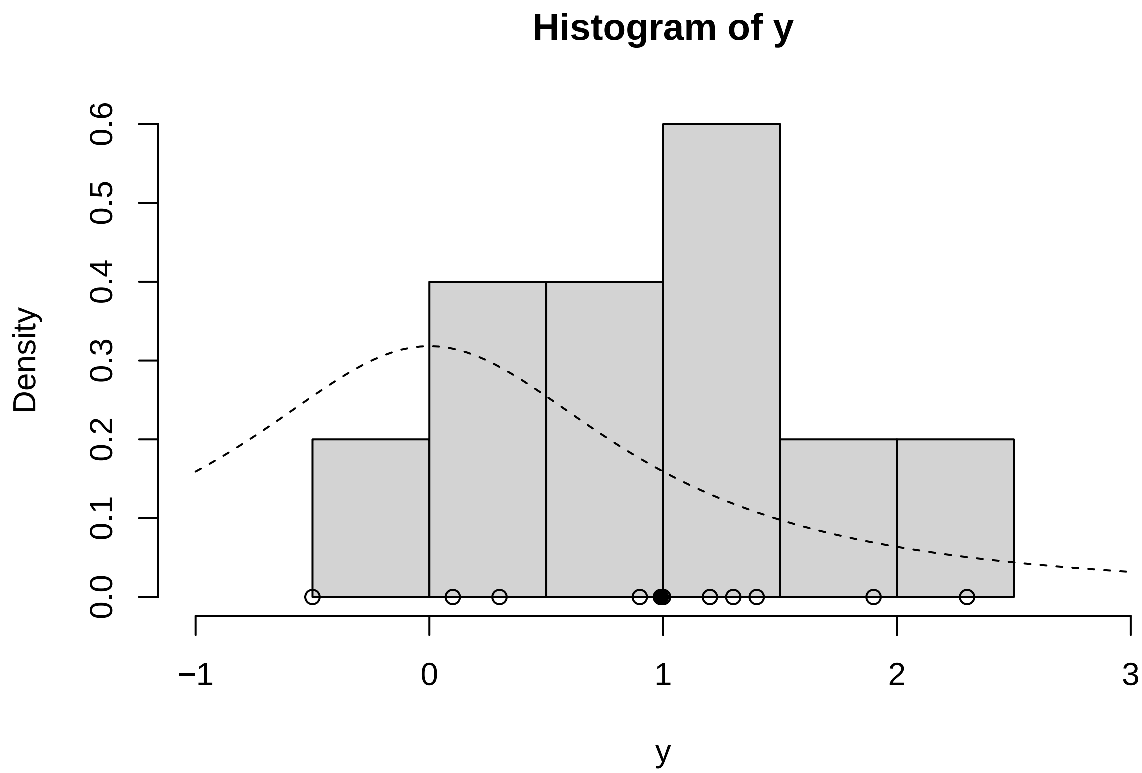
$$g(\mu) := \frac{\exp[n(\bar{y}\mu - \mu^2/2)]}{1 + \mu^2}$$

```r
# using log(g(x)) instead of g(x) for numerical stability
LOGg = function(mu, n, ybar) {
  n * (ybar * mu - mu^2 / 2) - log(1 + mu^2)
}
```

```r
metropolis_hastings = function(n, ybar, n_iter, mu_init, cand_sd) {
  # Random-Walk Metropolis-Hastings algorithm

  # initializations
  mu_out = numeric(n_iter)
  n_accept = 0

  # step 1
  mu_now = mu_init
  LOGg_now = LOGg(mu=mu_now, n=n, ybar=ybar)

  for (i in 1:n_iter) {
    # step 2a
    mu_cand = rnorm(1, mean=mu_now, sd=cand_sd) # draw candidate

    # step 2b
    LOGg_cand = LOGg(mu=mu_cand, n=n, ybar=ybar)
    LOGalpha = LOGg_cand - LOGg_now
    alpha = exp(LOGalpha)

    u = runif(1) # less than alpha with prob min(1, alpha)
    if (u < alpha) { # accept candidate
      n_accept = n_accept + 1
      mu_now = mu_cand
      LOGg_now = LOGg_cand
    }

    mu_out[i] = mu_now
  }

  list(mu=mu_out, accept_rate=n_accept/n_iter)
}
```

Problem set up:

```r
y = c(1.2, 1.4, -0.5, 0.3, 0.9, 2.3, 1.0, 0.1, 1.3, 1.9) # data
ybar = mean(y) # sample mean
n = length(y)

hist(y, freq=FALSE, xlim=c(-1, 3)) # histogram of data
curve(dt(x=x, df=1), lty=2, add=TRUE) # prior for mu
points(y, rep(0,n), pch=1) # individual data points
points(ybar, 0, pch=19) # sample mean
```



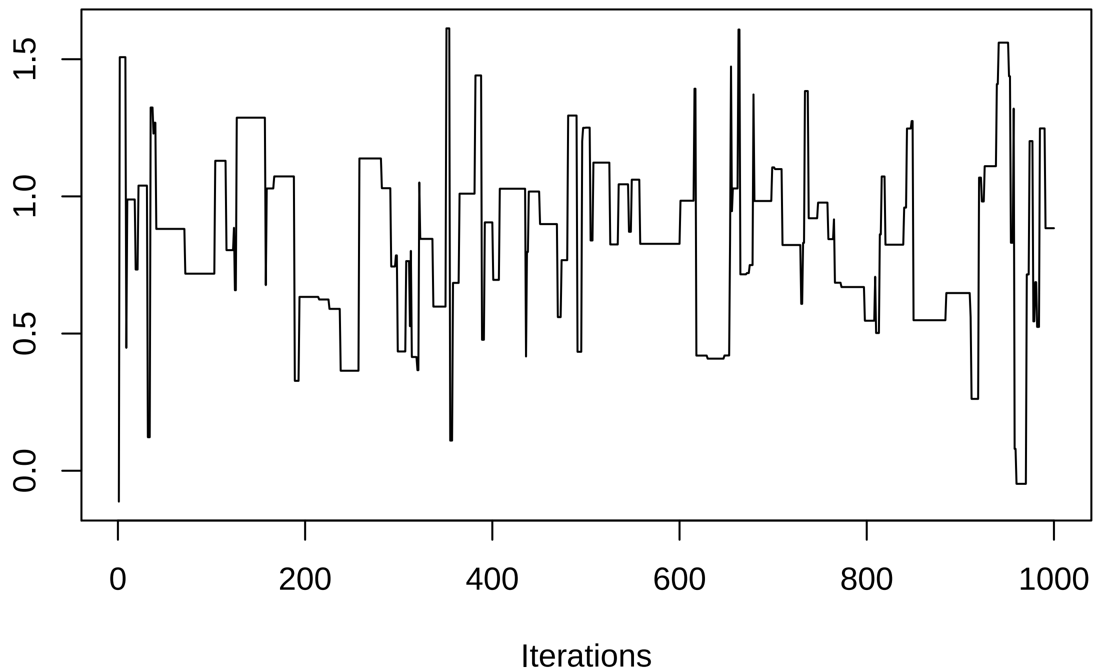

```r
set.seed(43) # for reproducibility
library("coda") # traceplot --> helpful to determine convergence
```

Posterior sampling:

```r
post = metropolis_hastings(n=n, ybar=ybar, n_iter=1e3, mu_init=0, cand_sd=3)
str(post)
```

```
## List of 2
##  $ mu         : num [1:1000] -0.113 1.507 1.507 1.507 1.507 ...
##  $ accept_rate: num 0.122
```

```r
traceplot(as.mcmc(post$mu))
```

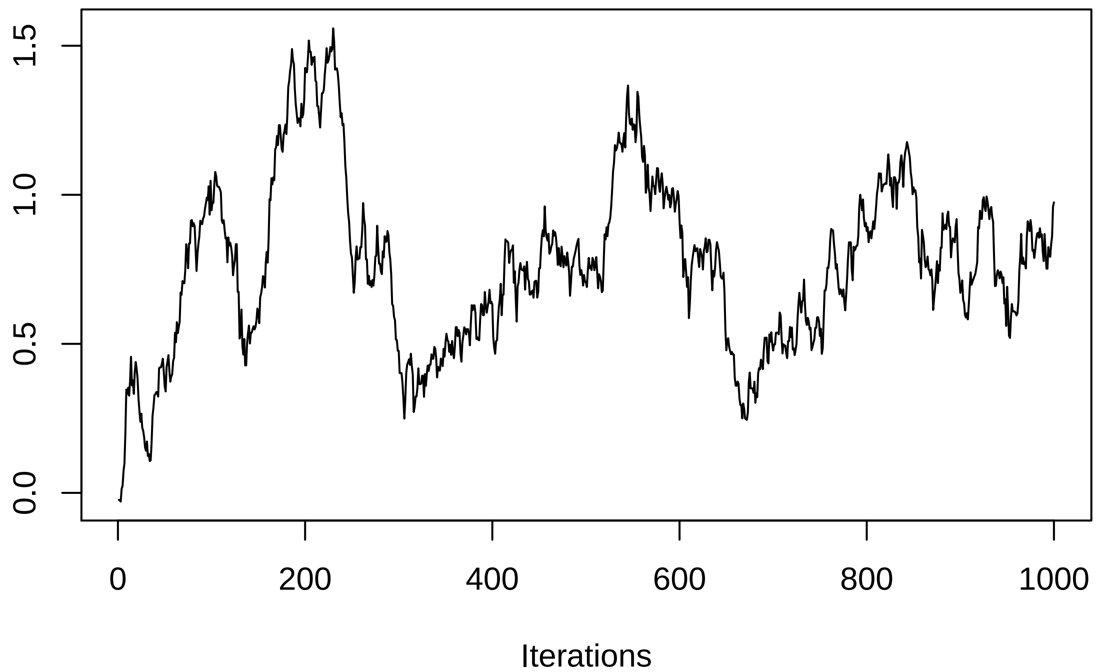


Step size too large (low acceptance rate). Let's try another.

```
post = metropolis_hastings(n=n, ybar=ybar, n_iter=1e3, mu_init=0, cand_sd=0.05)
str(post)
```

```
## List of 2
##  $ mu         : num [1:1000] -0.0236 -0.0247 -0.0293 0.0142 0.0235 ...
##  $ accept_rate: num 0.946
```

```
traceplot(as.mcmc(post$mu))
```

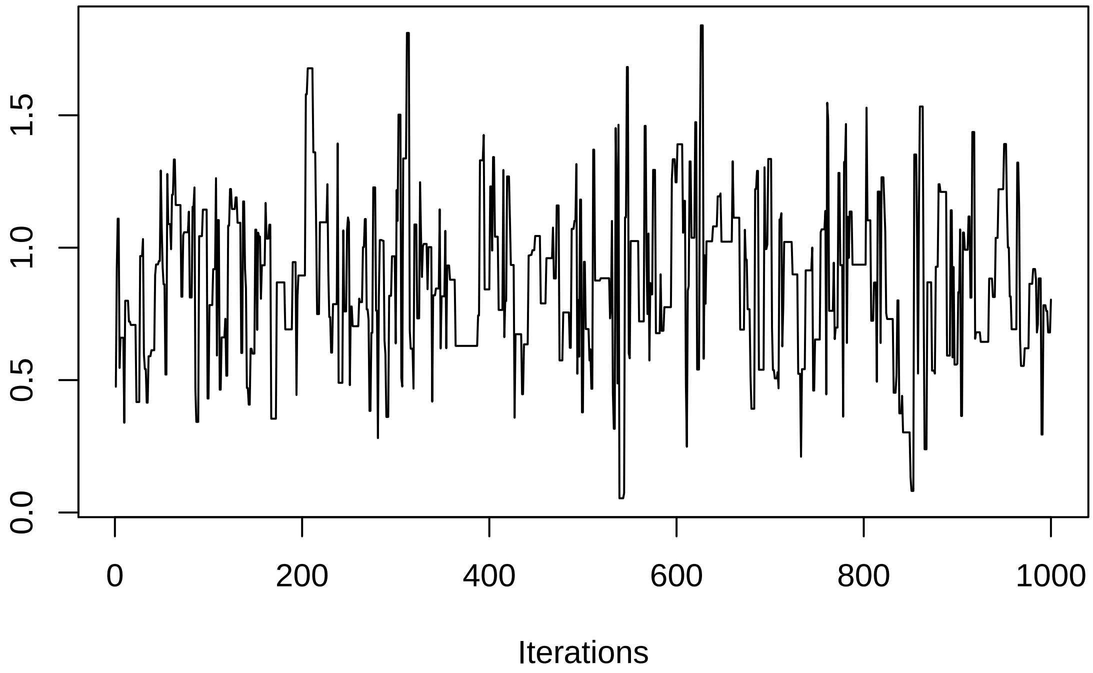


Step size too small (high acceptance rate). Let's try another.

```
post = metropolis_hastings(n=n, ybar=ybar, n_iter=1e3, mu_init=0, cand_sd=0.9)
str(post)
```

```
## List of 2
##  $ mu         : num [1:1000] 0.475 0.92 1.109 1.109 0.546 ...
##  $ accept_rate: num 0.38
```

```
traceplot(as.mcmc(post$mu))
```

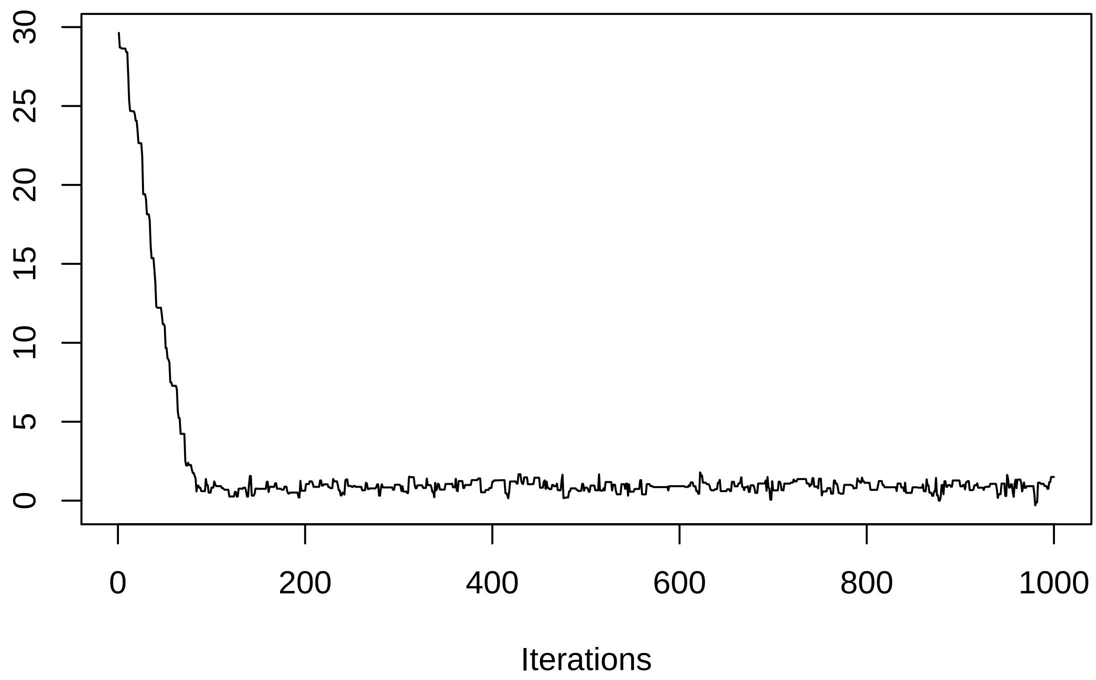


Looks good. Experimenting with different initial value:

```
post = metropolis_hastings(n=n, ybar=ybar, n_iter=1e3, mu_init=30, cand_sd=0.9)
str(post)
```

```
## List of 2
##  $ mu         : num [1:1000] 29.6 28.7 28.7 28.6 28.6 ...
##  $ accept_rate: num 0.387
```

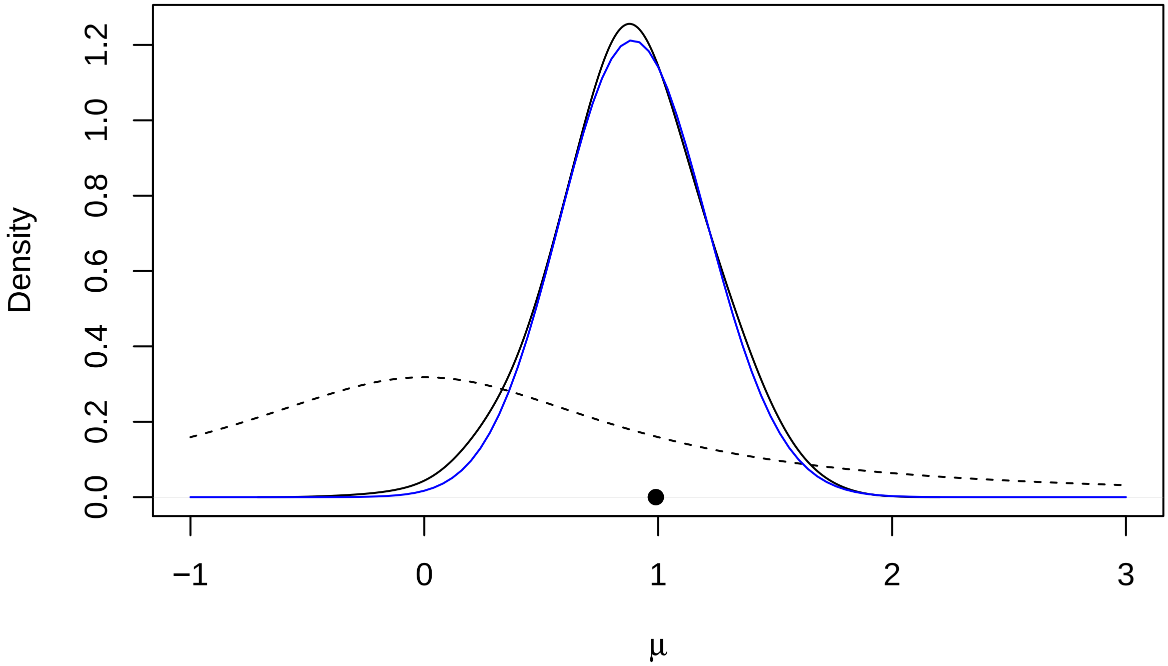
```
traceplot(as.mcmc(post$mu))
```




```
post$mu_keep = post$mu[-c(1:100)] # discard the first 100 samples
str(post)
```

```
## List of 3
##  $ mu         : num [1:1000] 29.6 28.7 28.7 28.6 28.6 ...
##  $ accept_rate: num 0.387
##  $ mu_keep    : num [1:900] 0.826 0.826 1.215 1.026 0.918 ...
```

```
plot(density(post$mu_keep, adjust=2), main="", xlim=c(-1, 3), xlab=expression(mu))
# plot estimated posterior distribution

curve(dt(x=x, df=1), lty=2, add=TRUE) # prior for mu
points(ybar, 0, pch=19) # sample mean

curve(0.017*exp(LOGg(mu=x, n=n, ybar=ybar)), from=-1, to=3, add=TRUE, col="blue")
```




```
# approximation to the true posterior in blue
```

JAGS