Markov Chain Monte Carlo based on the Metropolis-Hastings Algorithm

Spencer Woolf

In this exercise we will be using a Markov Chain Monte Carlo simulation to estimate a mass probability distribution of a discrete random variable. We will also be taking advantage of the Metropolis-Hastings Algorithm for acceptance-rejection.

Lets define how we will be accepting/rejecting generated samples:

Let $\pi_i = b_i/C$ for $i = 1, ..., m$, where $b_i > 0$ and $C = \sum b_i$
Let X_n be a Markov chain and let $Q = (q_{ij})$ be a transition matrix:

- When $X_n = i$ generate a random variable Y satisfying $P(Y = j) = q_{ij}, j = 1, ..., m$
- If $Y = J$, let

$$\begin{aligned} X_{n+1} = j \text{ with probability } \alpha_{i,j} = min(\frac{\pi_j q_{ji}}{\pi_i q_{ij}}, 1), \\ = i \text{ with probability } 1 - \alpha_{i,j} \end{aligned}$$

- X_n has its transition matrix $P = (p_{ij})$ as

$$\begin{aligned} p_{ij} = q_{ij}\alpha_{ij} \text{ if } i \neq j \\ = 1 - \sum_{k \neq i} q_{ik}\alpha_{ik} \text{ if } i = j \end{aligned}$$

First we'll make a function to compute the alpha matrix with inputs π and Q. π will be a made up mass probability for a discrete random variable (does not need to sum to 1 in this case) and Q will be a probability transition matrix.

```
##  [1] 15.00  5.00  1.00  3.00  6.00  0.05 18.00  9.00  1.00  2.00
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [2,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [3,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [4,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [5,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [6,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [7,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [8,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
##  [9,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
## [10,]  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1   0.1
```

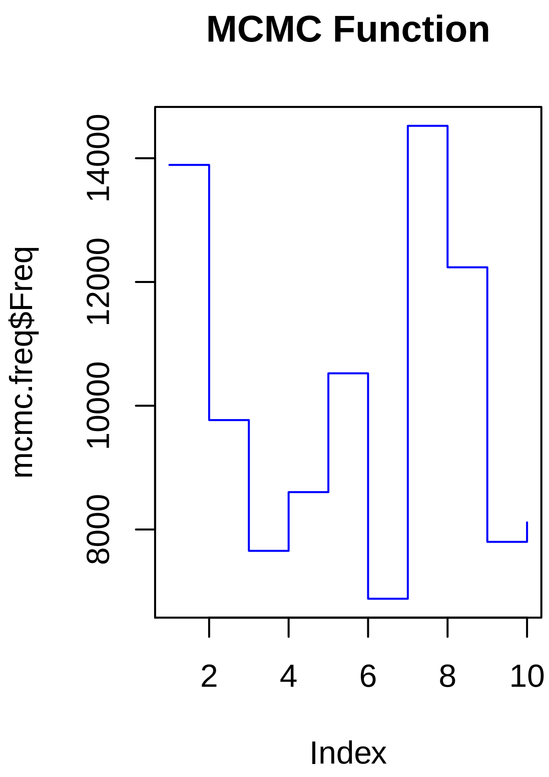
```
alpha = function(my.pi, Q){
  if (length(my.pi[my.pi<0]) > 0) stop("Must contain only positive values")
  if (nrow(Q) != ncol(Q) || nrow(Q) != length(my.pi)) stop("Must be a square matrix with the same dimens
  if (length(which(Q < 0)) > 0) stop("Matrix must contain only positive values")
  if (FALSE %in% apply(Q, 2, function(x) sum(x)==1)) stop("Must be a transitional matrix")

  mat = t(my.pi*Q)/(my.pi*Q)
  mat[which(mat > 1)] = 1
  return(mat)
}
a = alpha(my.pi, Q)
```

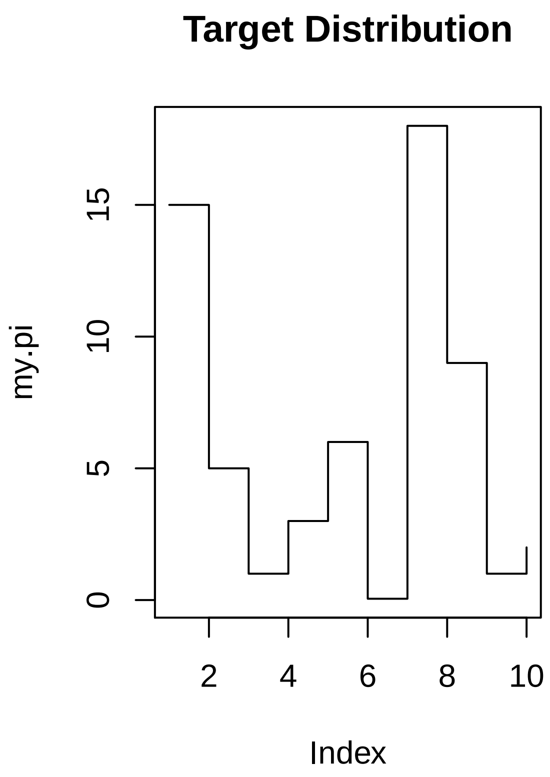
Next is the MCMC function. It will take the inputs n, π, and K, where n is the total number of samples to be simulated and K is the number of Markov Chain iterations. Vectorized code was used to perform n Markov Chains simultaneously instead of running each of n Markov Chains K times.

```
mcmc = function(n, my.pi, K = 20){
  m = length(my.pi)
  Q = matrix(rep(1, m*m), nrow = m)/m
  x = matrix(nrow = K, ncol = n) # Output matrix, n numbers generated for each of K iterations

  for (k in 1:K) {
    if (k == 1) {
      i = sample(1:m, n, replace = T) # Starts with n samples of 1 to m
    }
    else{
      i = x[k-1,] # Uses previous row to compare to generated y values
    }

    y = sample(rep(1:m, n), n, prob = Q[i,], replace = T)
    bernoulli = rbinom(n,1,prob = a[i,y])
    x[k,] = c(y[which(bernoulli==1)], i[which(bernoulli==0)])

  }

  return(x)

}
```

The function produces a bernoulli outcome for each i, y pair based on the alpha matrix. Then it adds the values that passed the bernoulli test from y (accepted) and the previous values from i which were false (rejected).

Lets plot the stationary distribution next to our MCMC results:





We can see that the simulated values from the MCMC closely resemble the target distribution.