Benford's Law worksheet

Jonathan Rougier
School of Mathematics
University of Bristol UK

File this experiment under 'Fun'! We are going to simulate from a Markov Chain on $\mathcal{X} = \{1, \ldots, 9\}$ with unique stationary distribution

$$\pi_i = \log_{10}\left(\frac{i+1}{i}\right) \quad i = 1, \ldots, 9.$$

which is Benford's Law.

```
#### Benford's Law is the target distribution

curlyX <- seq.int(1, 9)
pi <- log10((curlyX + 1) / curlyX)
print(round(pi, 3))
```

```
## [1] 0.301 0.176 0.125 0.097 0.079 0.067 0.058 0.051 0.046
```

We will use a random proposal for $h(i \to j)$:

```
## random proposal for h

r <- length(curlyX)
h <- matrix(runif(r * r), nrow = r)
h <- h / rowSums(h)
print(round(h, 2))
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,] 0.09 0.13 0.10 0.09 0.05 0.06 0.11 0.24 0.14
##  [2,] 0.01 0.19 0.01 0.09 0.17 0.02 0.17 0.18 0.18
##  [3,] 0.12 0.12 0.12 0.11 0.00 0.15 0.11 0.12 0.14
##  [4,] 0.12 0.14 0.18 0.08 0.18 0.15 0.10 0.00 0.05
##  [5,] 0.08 0.28 0.06 0.10 0.13 0.03 0.21 0.09 0.01
##  [6,] 0.08 0.12 0.17 0.05 0.12 0.02 0.15 0.23 0.06
##  [7,] 0.10 0.10 0.16 0.03 0.13 0.07 0.14 0.16 0.11
##  [8,] 0.06 0.15 0.14 0.10 0.06 0.12 0.15 0.04 0.17
##  [9,] 0.12 0.04 0.01 0.18 0.08 0.14 0.12 0.18 0.13
```

Your `h` will be different from mine, because of different random number seeds. My choice of `h` gives the following acceptance ratio:

```
## compute a

foo <- pi * h # each element is pi_i * h(i -> j)
```

```r
foo <- t(foo) / foo # p_j h(j -> i) / p_i h(i -> j)
a <- ifelse(foo > 1, 1, foo) # min{1, foo}
print(round(a, 2)) # shows a(i -> j)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    1 0.04 0.50 0.43 0.40 0.31 0.18 0.04 0.13
##  [2,]    1 1.00 1.00 0.89 0.74 1.00 0.20 0.24 0.06
##  [3,]    1 0.10 1.00 1.00 1.00 0.60 0.63 0.49 0.04
##  [4,]    1 1.00 0.79 1.00 0.45 0.24 0.17 1.00 1.00
##  [5,]    1 1.00 0.08 1.00 1.00 1.00 0.45 0.42 1.00
##  [6,]    1 0.34 1.00 1.00 0.33 1.00 0.42 0.41 1.00
##  [7,]    1 1.00 1.00 1.00 1.00 1.00 1.00 0.81 0.81
##  [8,]    1 1.00 1.00 0.07 1.00 1.00 1.00 1.00 0.92
##  [9,]    1 1.00 1.00 0.64 0.33 0.61 1.00 1.00 1.00
```

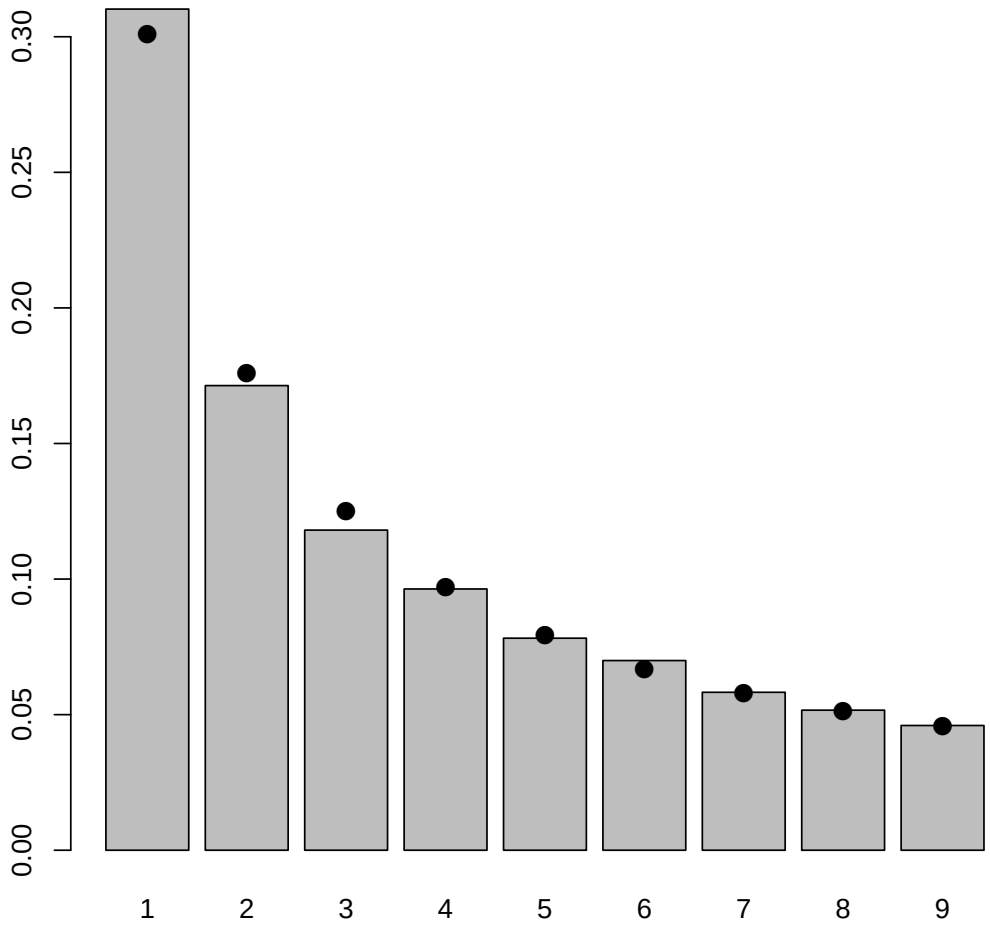
What a mess! Still, we must have faith in the mathematics.

It should be obvious that if $h(i \to j) = \pi_j$ then $a(i \to j) = 1$ and $X_1, X_2, \ldots$ is a random sample from $\boldsymbol{\pi}$. This would be the best choice for h. What this experiment will show is that the MH algorithm works even for arbitrary choices for h.

```r
#### run the Metropolis-Hastings sampler

nsim <- 20000 # always need a big number here
X <- rep(NA, nsim)
X[1] <- curlyX[1] # arbitrary initialisation

for (i in 2:nsim) {
  Xtilde <- sample(curlyX, 1, prob = h[X[i-1], ])
  move <- runif(1) <= a[X[i-1], Xtilde]
  if (move)
    X[i] <- Xtilde
  else
    X[i] <- X[i-1]
}
```

Are the estimated probabilities in the sample like those in the target?

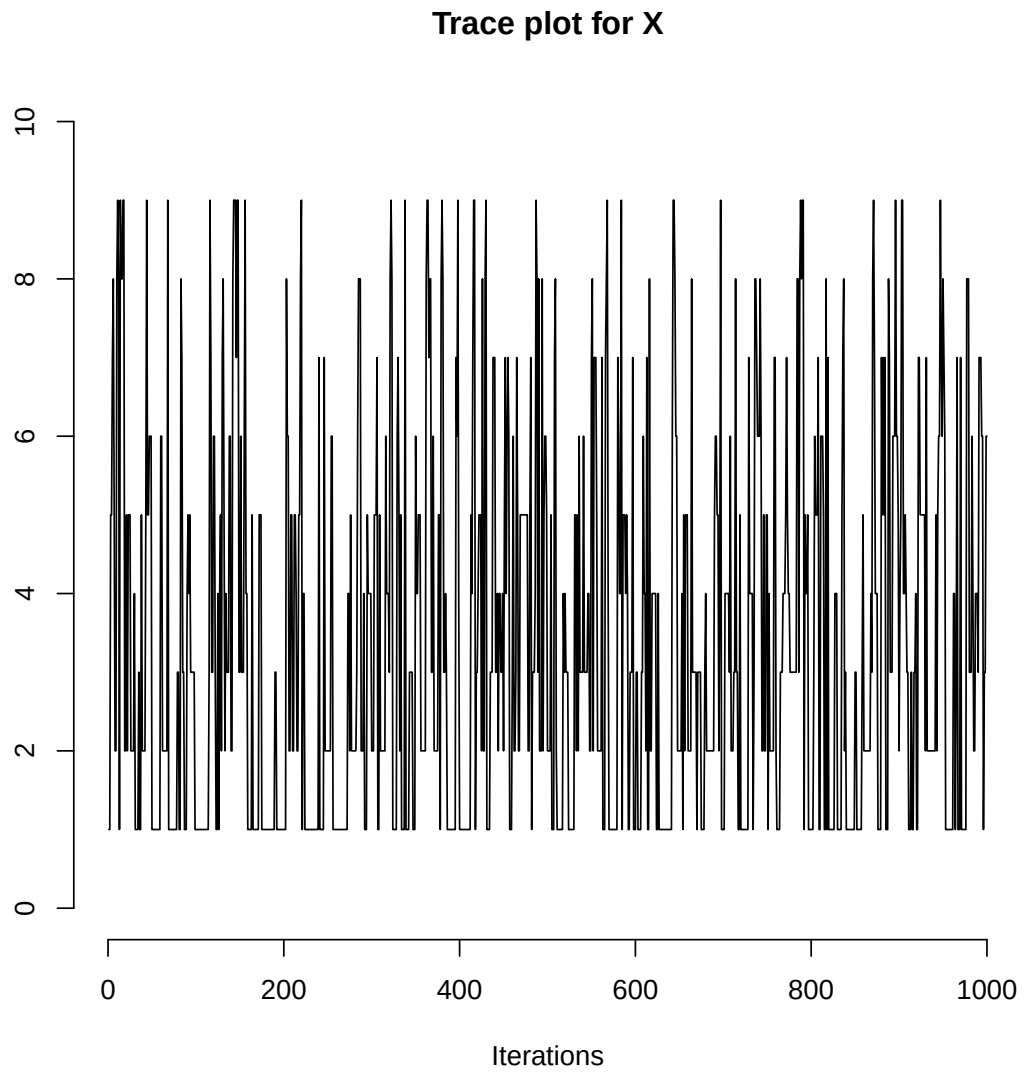
```r
## plot the estimated probabilities

phat <- sapply(curlyX, function(x) mean(X == x))
midp <- barplot(phat, names.arg = curlyX)
points(midp, pi, pch = 16, cex = 1.5, xpd = NA)
```



It's a mathematical miracle! Remember your results will be different from mine, because your `h` is different, and because the random choices in the Metropolis-Hastings (MH) algorithm are also different. But your picture should look similar to mine.

Note that I am skipping a formal assessment of convergence which requires multiple chains with over-dispersed initial values, because this is just a bit of fun. *Always check convergence if your results matter.* Instead, I'll just do a trace plot to check the rate of mixing:

```
## trace plot to check mixing

plot(X[1:1000], type = "l", ylim = c(0, 10),
  xlab = "Iterations", ylab = "", main = "Trace plot for X", bty = "n")
```



This looks as though it is mixing well (fat hairy caterpillar).

Computing the Monte Carlo Standard Errors. We will use the batch means approach, following the handout.

```
## batch means to compute MCSEs, need to estimate sigma^2_g

a <- floor(sqrt(nsim))
q <- floor(nsim / a)
Y <- matrix(tail(X, a * q), nrow = a) # a by q matrix

sig2g <- sapply(curlyX, function(x) {
  Z <- Y == x # computing the probability that X = x
  gbar <- colMeans(Z)
  gbarbar <- mean(Z)
  (a / (q - 1)) * sum((gbar - gbarbar)^2)
})

MCSE <- sqrt(sig2g) / sqrt(nsim)
```

```r
print(data.frame(curlyX, phat, MCSE = round(MCSE, 3)),
  row.names = FALSE)
```

```
##  curlyX    phat  MCSE
##       1 0.31020 0.009
##       2 0.17135 0.006
##       3 0.11805 0.004
##       4 0.09635 0.003
##       5 0.07820 0.003
##       6 0.06995 0.003
##       7 0.05825 0.002
##       8 0.05165 0.002
##       9 0.04600 0.002
```