Stats 102C - Lecture 6-1

Miles Chen

Week 6 Monday

Section 1

Island Hopping

The politician and the island nation

The island chain has 7 islands. The islands are numbered by θ From $\theta = 1$ in the west to $\theta = 7$ in the east. The relative populations of each island are proportional their island numbers. So island $\theta = 1$ has a population that is $1/7$th the size of island $\theta = 7$'s population.

The politician wants to visits the islands proportional to their population. The target distribution is:

$$\boldsymbol{\pi} = \left\{ \frac{1}{28}, \frac{2}{28}, \frac{3}{28}, \frac{4}{28}, \frac{5}{28}, \frac{6}{28}, \frac{7}{28} \right\}$$

The politician's algorithm

- Each morning the politician flips a coin:
  - if it lands heads he proposes the island on the west.
  - if it lands tails he proposes the island on the east.
- He checks the population of the proposed island and the population of the current island.
  - if the proposed island has a larger population than the current island, he definitely goes to the proposed island
  - if the proposed island has a smaller population than the current island, he goes to the island probabilistically. That is his probability of moving that that island is:
    - $p_{move} = \frac{p_{proposed}}{p_{current}}$
    - To do this, he can select a random uniform value. If $u < p_{move}$ he moves to the proposed island. If not, he stays at the current island.

The system works! In the long run, the probability the politician is on any one of the islands exactly matches the relative population of the island!

Last Time

In Lecture 5-3, we found the transition probability matrix, $\mathbb{P}$, of the Markov chain.

$$\mathbb{P} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{6} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{3}{8} & \frac{1}{8} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{2}{5} & \frac{1}{10} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{5}{12} & \frac{1}{12} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & \frac{3}{7} & \frac{4}{7} \end{bmatrix}$$

We found the left eigenvector of $\mathbb{P}$ and showed that it is equal to the target distribution.

Running the actual Markov Chain

What we didn't do last time was actually run the Markov Chain.

The politician's algorithm is:

1. Flip a coin to propose the next island.
   - $X_{proposed} = X_t + 1$ with probability 0.5.
   - $X_{proposed} = X_t - 1$ with probability 0.5.
2. Calculate probability p_{move}
   - $p_{move} = \min\left\{1, \frac{p_{proposed}}{p_{current}}\right\}$ (If $p_{proposed} > p_{current}$ we move with probability 1.)
3. Decide to move by sampling $U \sim \text{Unif}(0, 1)$.
   - If $U \leq p_{move}$ then move: $X_{t+1} = X_{proposed}$
   - If $U > p_{move}$ then do not move: $X_{t+1} = X_t$

Code for Markov Chain

```r
# proposed is a function that returns current value plus or minus one
propose <- function(current) {  current + sample(c(1, -1), size = 1) }

# p is function that returns target probability, if x is 1 through 7, the target prob is x/28
p <- function(x) { ifelse(x %in% 1:7, x/28, 0) }

n <- 10^3
x <- c(4, rep(NA, n-1)) # start at 4
set.seed(1)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- p(proposed) / p(current)
  # pmove is min(1, p(propose)/p(current)), but if pmove > 1, random U will be less than it anyway
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
```

Results

```
counts <- table(x)
results <- rbind( counts/n , (1:7)/28 ); rownames(results) <- c("empirical", "target")
round(results, 4)
```

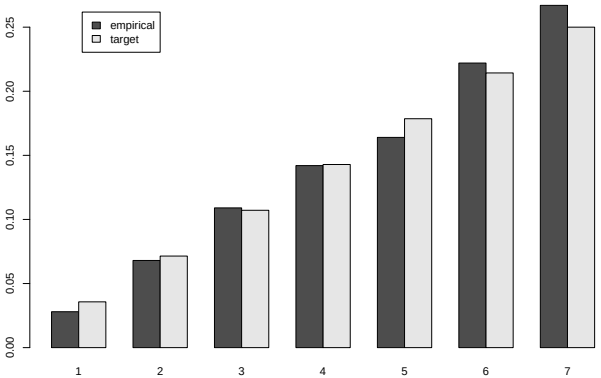
```
##                1      2      3      4      5      6     7
## empirical 0.0280 0.0680 0.1090 0.1420 0.1640 0.2220 0.267
## target    0.0357 0.0714 0.1071 0.1429 0.1786 0.2143 0.250
```

```
chisq.test(counts, p = (1:7)/28 )
```

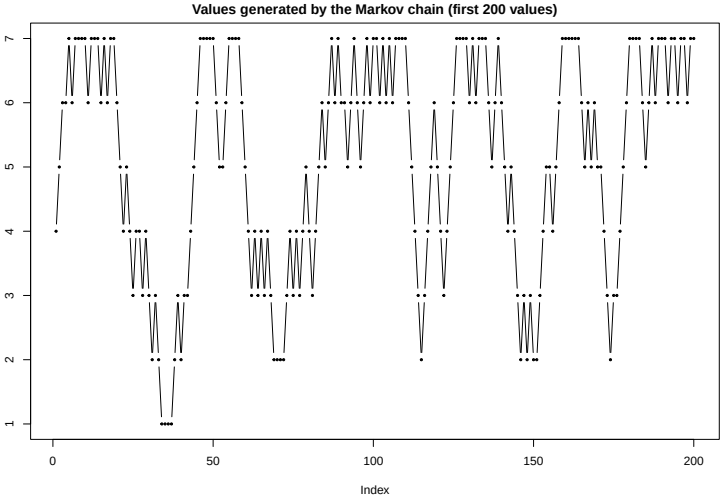
```
##
##  Chi-squared test for given probabilities
##
## data:  counts
## X-squared = 4.4909, df = 6, p-value = 0.6105
```

We see close alignment between the empirical probabilities and target distribution. The chi-squared goodness of fit test produces a very large p-value indicating that we do not have enough evidence to say that the values generated by the Markov chain do not come from the target distribution.

```
barplot(results, beside = TRUE, legend.text = row.names(results), args.legend = list(x = 5))
```



```
par(mar = c(4, 2, 2, 0))
plot(x[1:200], type = "b", pch = 19, cex = 0.4,
     main = "Values generated by the Markov chain (first 200 values)")
```



Running the chain longer

```
n <- 10^5
x <- c(7, rep(NA, n-1)) # start at 7
set.seed(5)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- p(proposed) / p(current)
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
```

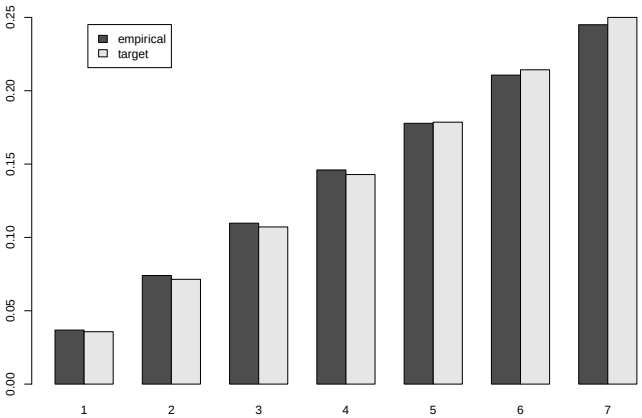
Results

```
counts <- table(x)
results <- rbind( counts/n , (1:7)/28 )
rownames(results) <- c("empirical", "target")
round(results, 4)
```

```
##                1      2      3      4      5      6     7
## empirical 0.0368 0.0740 0.1097 0.1460 0.1778 0.2107 0.245
## target    0.0357 0.0714 0.1071 0.1429 0.1786 0.2143 0.250
```

Comparing the empirical and theoretic probabilities, we see the values align quite closely.

```
barplot(results, beside = TRUE, legend.text = row.names(results), args.legend = list(x = 5))
```



Comparing the empirical and theoretic probabilities, we see the values align quite closely.

```r
chisq.test(counts, p = (1:7)/28 )
```

```
##
##  Chi-squared test for given probabilities
##
## data:  counts
## X-squared = 42.452, df = 6, p-value = 1.497e-07
```

The chi-squared goodness-of-fit test for this sample, however, gives a very small p-value. The problem is that the chi-squared test is very sensitive to sample size. With large samples ($n > 2000$), the chi-squared test will detect even the smallest of differences.

In these situations, we might consider using a p-value much smaller than 5%. The discussion is a bit beyond the scope of the course, but is worth reading.

https://stats.stackexchange.com/questions/74780/goodness-of-fit-for-very-large-sample-sizes

In short, don't worry if your p-value ends up very small if you have a very large sample size.

You might be better served simply comparing the empirical and theoretic probabilities using your best judgement.

Section 2

The Metropolis Algorithm

The Metropolis Algorithm

Let $f(x)$ be function proportional to the desired probability distribution $P(x)$.

Start with an arbitrary point x_t to be the first sample in the Markov chain.

Choose an arbitrary proposal distribution $g(x|x_t)$. For the Metropolis algorithm, the proposal distribution must be symmetric. $g(x_a|x_b) = g(x_b|x_a)$

For each iteration:

1. Generate a proposed value $x_{proposed}$ based on your current value from the distribution $g(x|x_t)$
2. Use the function $f(x)$ which is proportional to the target distribution to calculate the probability of moving.

$$p_{move} = \min\left\{1, \frac{f(x_{proposed})}{f(x_t)}\right\}$$

3. Accept or reject
   - Generate a random number $U \sim \text{Unif}(0, 1)$
   - If $U \leq p_{move}$, accept the candidate. Set $x_{t+1} = x_{proposed}$
   - If $U > p_{move}$, reject the candidate. Resuse current value. Set $x_{t+1} = x_t$

$f(x)$ only needs to be proportional to $P(x)$

The function $f(x)$ only needs to be proportional to $P(x)$. The exact form of $P(x)$ is not needed.

$$f(x) = c \cdot P(x)$$

where c is some unknown positive constant.

$$p_{move} = \min\left\{1, \frac{f(x_{proposed})}{f(x_{current})}\right\} = \min\left\{1, \frac{\cancel{c} \cdot P(x_{proposed})}{\cancel{c} \cdot P(x_{current})}\right\} = \min\left\{1, \frac{P(x_{proposed})}{P(x_{current})}\right\}$$

When calculating p_{move} we can use $f(x_{proposed})/f(x_{current})$ and we would get the same result as if we had used the actual probability distribution $P(x_{proposed})/P(x_{current})$

$f(x)$ only needs to be proportional to $P(x)$

Being able to use a function $f(x)$ that is proportional to $P(x)$ is especially useful in the context of Bayesian statistics. In Bayesian statistics, we will often want to generate samples from the posterior distribution of the parameter θ given our data $\mathbf{x}$: $P(\theta|\mathbf{x})$.

$$P(\theta|\mathbf{x}) = \frac{P(\mathbf{x}|\theta)P(\theta)}{P(\mathbf{x})}$$

The marginal probability $P(\mathbf{x})$ can be difficult to calculate sometimes. It is a normalizing constant and does not depend on θ. The marginal probability can be found by summing or integrating the numerator across all possible values of θ. In some cases, this calculation can be incredibly difficult.

Instead of trying to find the exact form of $P(\theta|\mathbf{x})$, we can use a function that is proportional to it.

$$f(\theta) = \Pr(\mathbf{x}|\theta)\Pr(\theta)$$

$f(\theta)$ is proportional to $P(x)$

The proposal distribution

$g(x|x_t)$ is the proposal distribution.

For the Metropolis algorithm, it must be a symmetric distribution: $g(x_a|x_b) = g(x_b|x_a)$

Common choices that work for the Metropolis algorithm are:

- Normal distribution centered at x_t with some arbitrary σ
  - $x_{proposed} \sim \mathcal{N}(\mu = x_t, \sigma)$
  - `proposed <- rnorm(1, mean = x_t, sd = sigma)`
- Uniform distribution centered at x_t with some arbitrary width $2c$
  - $x_{proposed} \sim \text{Unif}(x_t - c, x_t + c)$
  - `proposed <- runif(1, min = x_t - c, max = x_t + c)`

For both distributions, we can see that the probability of proposing $x_{proposed}$ if it is centered at x_t is the same as the probability of proposing x_t if it is centered at $x_{proposed}$.

Example

Let's say our target distribution is the normal distribution centered at 15 with a sd of 3.

The PDF of the normal is:

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

We can ignore the normalizing constant in the front and simply use:

$$f(x) = \exp\left(-\frac{1}{2}\left(\frac{x-15}{3}\right)^2\right)$$

For our proposal distribution, we'll use a uniform proposal centered at x_t with a width of 6.

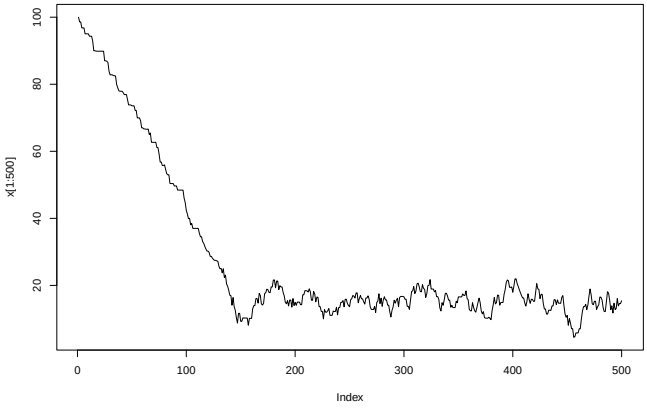
$$x_{proposed} \sim \text{Unif}(x_t - 3, x_t + 3)$$

To illustrate the algorithm's power, I'll start with an **absolutely terrible** starting location of $x_t = 100$.

```r
# f is proportional to target PDF
f <- function(x) { exp( -.5 * ((x - 15) / 3)^2 ) }
# our proposal function centered at the current value
propose <- function(current){
  runif(1, min = current - 3, current + 3)
}

n <- 5000
x <- c(100, rep(NA, n-1)) # start at 100
set.seed(1)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- f(proposed) / f(current)
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
```

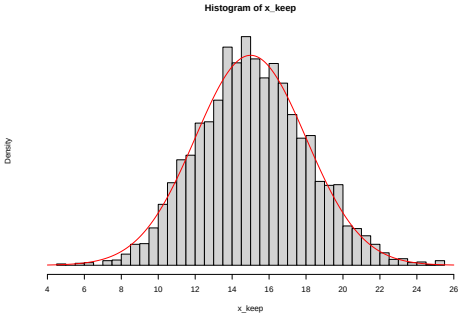
```r
plot(x[1:500], type = "l")
```



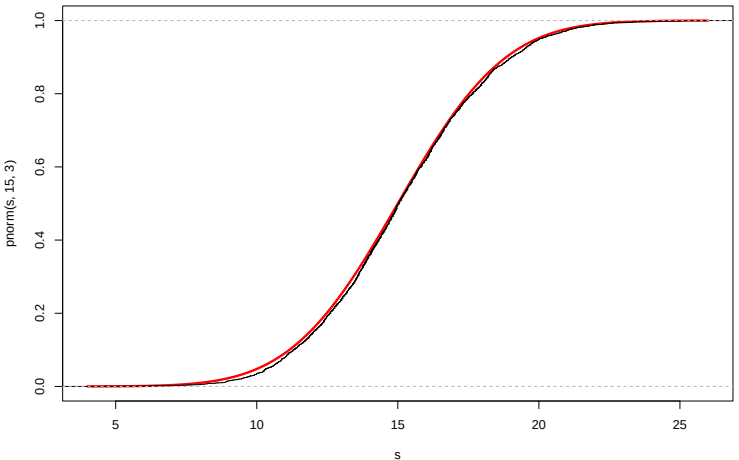
Despite starting at the terrible location of $x_1 = 100$, the chain manages to find the region where $f(x)$ is high within 180 iterations or so. Once it reaches that region, it appears to generate values according to the normal distribution.

We discard the first 180 values. We call this the "burn-in" period. When we use the Metropolis algorithm, we can give the algorithm some poor starting conditions (if we don't know much about the target distribution) and allow it to find the region where samples should be abundant.

```
x_keep <- x[-(1:180)]
hist(x_keep, breaks = 30, freq = FALSE, axes = FALSE)
axis(1, at = seq(4, 26, by = 2))
s <- seq(4, 26, by = .01)
lines(s, dnorm(s, 15, 3), col = "red")
```



```r
plot(s, pnorm(s, 15, 3), col = "red", lwd = 3, type = "l")
plot(ecdf(x_keep), add = TRUE)
```

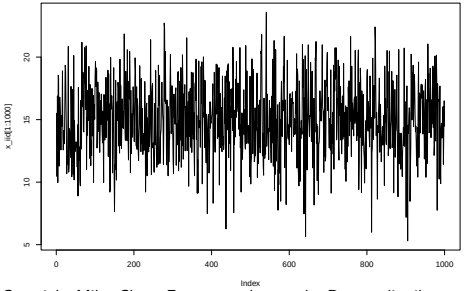


Diagnosing efficiency

In the absolute ideal case, the value generated by the chain would resemble random independent draws from the target distribution.

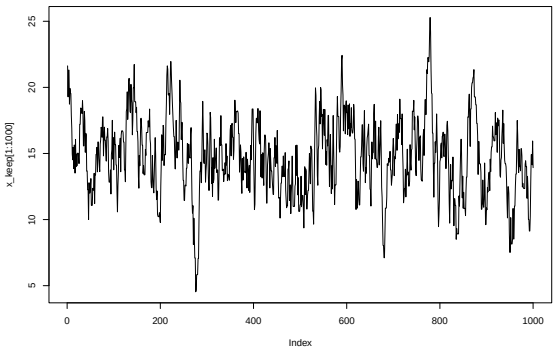
Here is a plot of 1000 iid values drawn from the target distribution.

```
x_iid <- rnorm(1000, mean = 15, sd = 3)
plot(x_iid[1:1000], type = "l")
```



Let's take a look at the Markov chain after the burn-in period. We see the values bounce around, but not showing the same amount of variance as the iid samples. In this case, we might want to experiment with a proposal distribution with greater variance.

```
plot(x_keep[1:1000], type = "l")
```



Different proposal distributions

The choice of a proposal distribution is arbitrary.

For our generated sample, I used a uniform distribution centered at x_t with a width of 6.

We can have used other proposal distributions.

I'll illustrate a few other choices:

- uniform distribution centered at x_t with a width of 100.
- uniform distribution centered at x_t with a width of 1.
- uniform distribution centered at x_t with a width of 12.

For each chain, I will start it at 15, so we don't have to worry about getting rid of "burn-in" values.

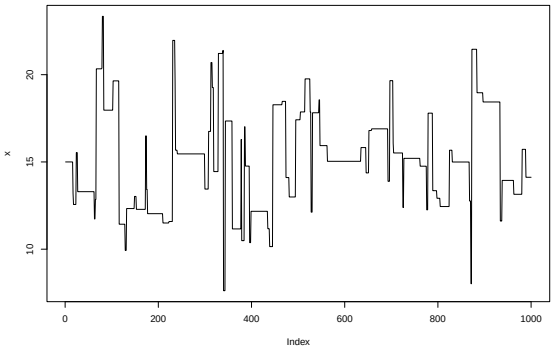
Uniform proposal distribution with width = 100

```
propose <- function(current){
  runif(1, min = current - 50, current + 50)
}

n <- 1000
x <- c(15, rep(NA, n-1)) # start at 15
set.seed(1)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- f(proposed) / f(current)
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
```

With a very wide proposal distribution, many of the proposed values of X will fall in a region with very low probability and these values will be rejected. When a proposed x is rejected, the current value of x is entered in the chain again. This results in a bunch of "flat" regions where the same value appears multiple times in the chain.

```
plot(x, type = "l")
```



Uniform proposal distribution with width = 1

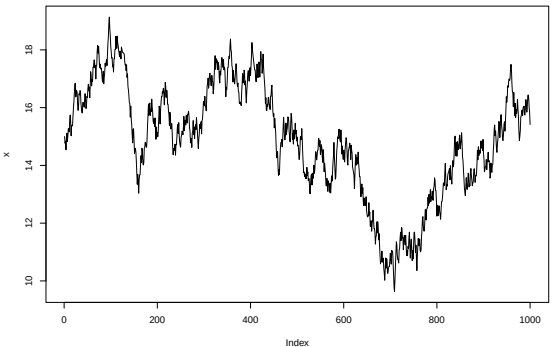
```
propose <- function(current){
  runif(1, min = current - 0.5, current + 0.5)
}

n <- 1000
x <- c(15, rep(NA, n-1)) # start at 15
set.seed(1)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- f(proposed) / f(current)
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
```

On the other extreme, with a very narrow proposal distribution, the proposed value of X will be almost approximately equal to the current value of x. If $x_{proposed} \approx x_{current}$, then $f(x_{proposed}) \approx f(x_{current})$, and $p_move \approx 1$.

Thus nearly every proposed value will be accepted. The chain will move only small increments. This results in a chain that resembles a random walk and the chain can "drift" into regions of low probability.

```
plot(x, type = "l")
```



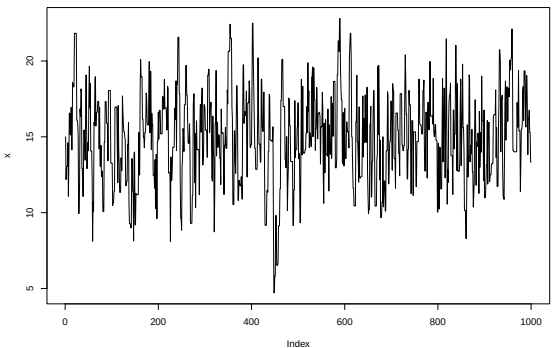
Uniform proposal distribution with width = 12

```
propose <- function(current){
  runif(1, min = current - 6, current + 6)
}

n <- 1000
x <- c(15, rep(NA, n-1)) # start at 15
set.seed(1)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- f(proposed) / f(current)
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
```

Using a width of 12 appears to give moderately good results.

```
plot(x, type = "l")
```



Proper implementation of MCMC often requires experimenting with proposal distributions.