

R Programming: Worksheet 6

Today we'll study a few useful functions we haven't come across yet:

`all()`, `any()`, ``%in%``, `match()`, `pmax()`, `pmin()`, `unique()`

We'll also apply our knowledge to the bootstrap.

1. Some Useful Functions

The `any()` and `all()` functions are useful generalizations of the 'and' and 'or' operators. They determine whether (respectively) any or all of the elements of a logical vector are `TRUE`.

```
> any(c(TRUE, FALSE, FALSE))  
  
## [1] TRUE  
  
> all(c(TRUE, FALSE, FALSE))  
  
## [1] FALSE
```

- (a) Write a function with argument `n` that randomly permutes the numbers $1, \dots, n$, and checks whether any of them are in their original correct position. For example, in the permutation 4, 1, 3, 2, the number 3 is still in the 3rd entry, so the function would return `TRUE`.

```
> permFix = function(n) {  
+   out = sample(n)  
+   any(out == seq(n))  
+ }
```

- (b) Use `replicate` to estimate the probability of getting `TRUE` for a few different values of `n`. *The answer doesn't change very much for different n , being approximately $1 - e^{-1} = 0.6321$.*

```
> mean(replicate(10000, permFix(100)))  
  
## [1] 0.6415
```

We have seen the `max()` and `min()` functions, which determine the largest and smallest elements of a vector. There are vectorized versions of these functions available for comparing entries pointwise in a vector: `pmax()` and `pmin()`.

```
> x = c(1, -4, 9)  
> y = c(1, 3, 3)  
> pmin(x, y)  
  
## [1] 1 -4 3
```

Of course, vector recycling can be used here:

```
> pmax(x, 0)
## [1] 1 0 9
```

- (c) Write a function which truncates the numbers in a vector above 1 or below 0. For example:

```
> x <- c(0.2, 0.9, -0.3, 1.1, 0.5)
> trunc01(x)
## [1] 0.2 0.9 0.0 1.0 0.5
```

```
> trunc01 = function(x) pmax(pmin(x, 1), 0)
```

The function `match()` and the binary operator `%in%` are useful for finding items within vectors or lists. Given two vectors, `%in%` returns a logical vector telling you whether or not each entry in the first vector is contained somewhere in the second.

```
> c(3, 2, 5) %in% c(5, 4, 5, 6, 2)
## [1] FALSE TRUE TRUE

> "U" %in% LETTERS[1:10]
## [1] FALSE
```

`match()` is similar, but also tells you *where* in the second vector the item is found.

```
> match(c("C", "B", "E"), c("E", "D", "E", "F", "B"))
## [1] NA 5 1
```

Note that it only gives the first position if the element is repeated, and (by default) it returns `NA` if there is no match.

- (d) How would you check whether every element of a vector `x` is contained within a second vector `y`?

```
> all(x %in% y)
```

- (e) Write a function `rmv()` of two vector arguments `x` and `y`. The function should remove any element of `y` which is also in `x` and then return what remains. It should make use of `match()` and/or `%in%`. For example:

```

> rmv(c(1, 2), c(0, 1, 2, 1, 3, 1, 4))

## [1] 0 3 4

> rmv(c("A", "E", "I", "O", "U"), LETTERS)

## [1] "B" "C" "D" "F" "G" "H" "J" "K" "L" "M" "N" "P" "Q" "R" "S" "T" "V"
## [18] "W" "X" "Y" "Z"

```

You might find the function `na.omit()` useful. Note that the function `setdiff()` does exactly this, but using it wouldn't be as fun as making our own routine, would it? *No it wouldn't. One possibility is:*

```

> rmv = function(x, y) {
+   y[!(y %in% x)]
+ }

```

2. Counting

Look at the data set `faithful` in the `MASS` package.

- (a) Plot the data as a scatter plot, and comment.

```

> plot(faithful)

```

There appear to be two distinct 'groups' in the data, suggesting two types of eruption.

- (b) Dichotomize (i.e. split into two groups) each of the two series using the `cut()` command. Choose a sensible point to split in each case, and label your groups 'short' and 'long'.

```

> Erup = cut(faithful$eruptions, breaks = c(0, 3, Inf), labels = c("short",
+   "long"))
> Wait = cut(faithful$waiting, breaks = c(0, 69, Inf), labels = c("short",
+   "long"))

```

- (c) Produce a two-way contingency table of these discretized data.

```

> table(Erup, Wait)

##           Wait
## Erup      short long
## short      96    1
## long        7   168

```

3. Bootstrap

Suppose we have X_1, \dots, X_n i.i.d. random variables from some unknown distribution P , and we have a function $\hat{\theta} = f(X_1, \dots, X_n)$ used to estimate some parameter $\theta(P)$. For example, if $\theta(P)$ is the mean of the distribution P , we might use the function

$$f(X_1, \dots, X_n) = \frac{1}{n} \sum_{i=1}^n X_i.$$

Now, suppose we wish to estimate the amount of uncertainty associated with using the estimator f . Ideally, we would draw lots of independent samples of size n from P , and see how much the value of our estimator changes.

Unfortunately P is unknown, so instead we can draw a sample from P^* , the empirical distribution of the data (X_1, \dots, X_n) . In other words, we draw a sample of size n with replacement from the set $\{X_1, \dots, X_n\}$. If we repeat this a large number of times it mimics the properties of samples from the original distribution. This is called the **bootstrap** method.

- (a) Write a function `bootsamp(x)` which, given a vector `x` of length `n`, returns a single bootstrap sample of size `n`. *Either of the following will work: (why?)*

```
> bootsamp = function(x) sample(x, length(x), replace = TRUE)
> bootsamp = function(x) x[sample(length(x), replace = TRUE)]
```

- (b) Let $N = 100$. Draw a sample of N independent gamma variables with shape 2 and rate 3 (use `rgamma()`); then take a bootstrap sample and see how many **unique** values it contains. The function `unique()` may be useful here.

```
> x = rgamma(100, 2, 3)
> b = bootsamp(x)
> length(unique(b))

## [1] 63
```

- (c) Try this a few times, and for various N (e.g. 1,000, 10,000, 100,000). Any comments?

It always seems to be about 63% of the sample size. [A fairly hard mathematical exercise will show that the number of unique samples is approximately $N(1 - e^{-1}) \approx 0.632N$.]

Now suppose we wish to obtain a bootstrap estimate of the uncertainty in the standard deviation function. To compute the sample standard deviation we can just use the `sd()` function, so this will be our f .

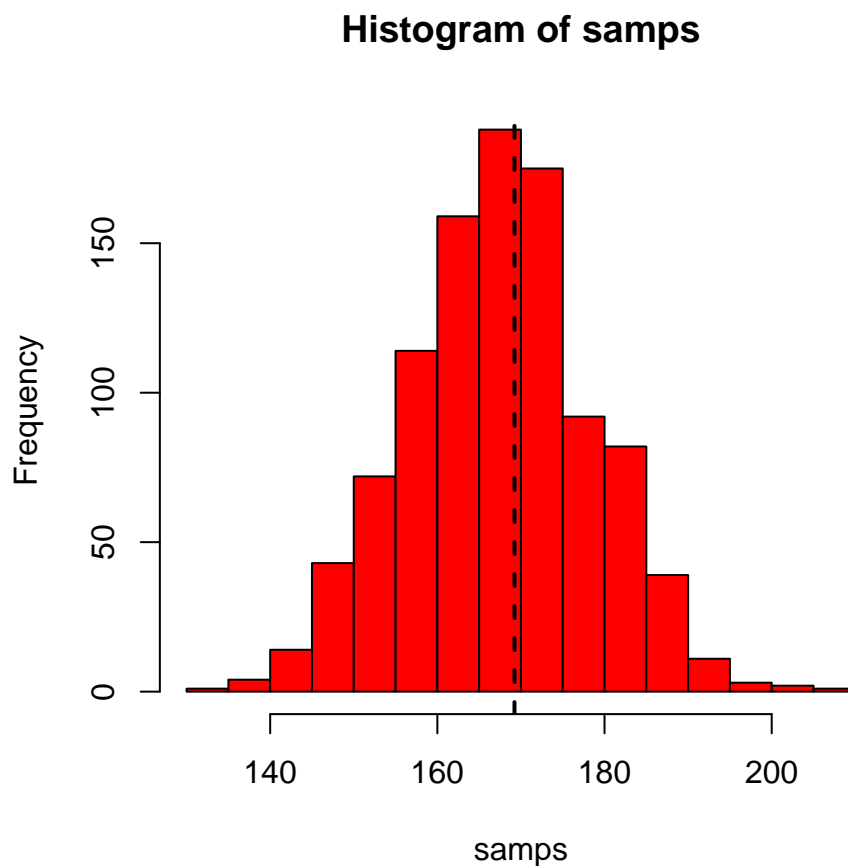
- (f) Write a function `bootsd(x,B)` with arguments `x`, a vector, and `B` an integer. The function should draw a bootstrap sample of `x`, and find the sample standard deviation of that sample. It should repeat this a total of B times, and return the results as a vector of length B . Set B to default to 1,000.

Try to do this without using a loop.

```
> bootsd = function(x, B = 1000) {
+   replicate(B, sd(bootsamp(x)))
+ }
```

- (g) Apply your function to the Nile data, and plot the results as a histogram. Add the actual sample standard deviation as a vertical line on your histogram.

```
> samps = bootsd(Nile)
> hist(samps, col = 2, breaks = 20)
> abline(v = sd(Nile), lty = 2, lwd = 2)
```



- (h) The **kurtosis** of a distribution with mean μ and standard deviation σ is defined as $\beta_2 \equiv \sigma^{-4} \mathbb{E}(X - \mu)^4$, and is typically estimated in a sample X_1, \dots, X_n by

$$\hat{\beta}_2 \equiv \frac{\frac{1}{n-1} \sum_i (X_i - \bar{X}_n)^4}{s^4},$$

where \bar{X}_n is the sample mean and s is the sample standard deviation. Obtain the sample kurtosis of the Nile data.

```
> sampkurt = function(x) (length(x) - 1) * sum((x - mean(x))^4) / sum((x -
+   mean(x))^2)^2
> sampkurt(Nile)
## [1] 2.668
```

- (i) Generate 10,000 bootstrap samples for the Nile data, and use them to obtain a 95% confidence interval for the sample kurtosis. *Once we have the sample, we just need to find the 2.5% and 97.5% quantiles, which can be done manually or using `quantile()`.*

```
> out = replicate(10000, sampkurt(bootsamp(Nile)))  
> quantile(out, c(0.025, 0.975))  
  
## 2.5% 97.5%  
## 1.940 3.458
```

4. *Gibbs Sampler

Let

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim N\left(\mathbf{0}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right),$$

so that $X|Y \sim N(\rho Y, 1 - \rho^2)$ and vice-versa.

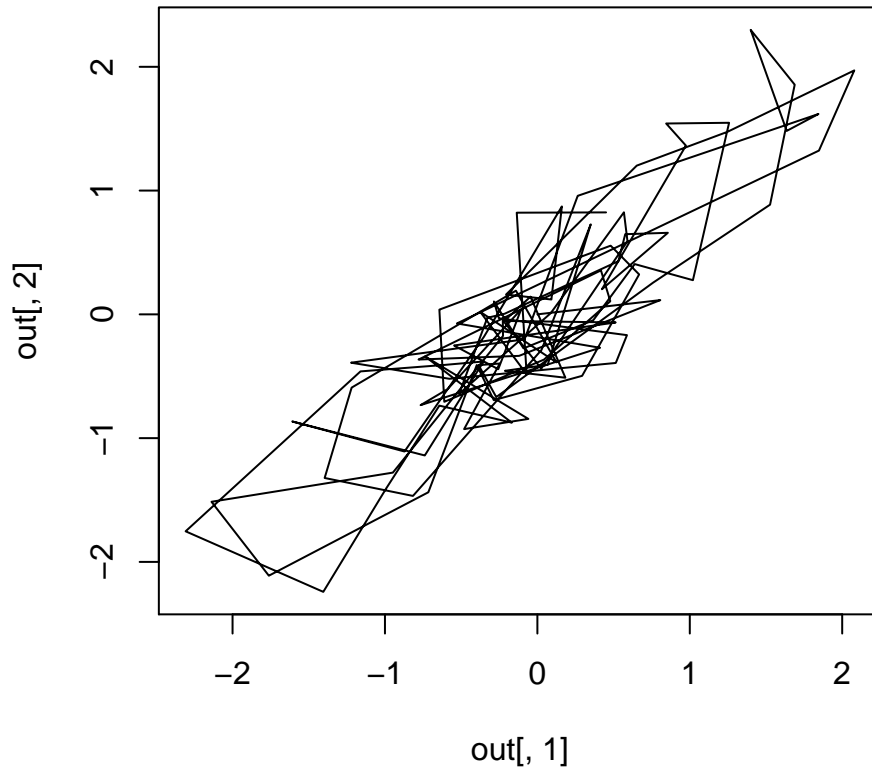
A **Gibbs sampler** explores a distribution by repeatedly drawing samples from the univariate conditional distributions. In other words, choose some starting values (X_0, Y_0) , and then draw

$$\begin{aligned} X_{i+1} &\sim N(\rho Y_i, 1 - \rho^2) \\ Y_{i+1} &\sim N(\rho X_{i+1}, 1 - \rho^2) \end{aligned}$$

for $i = 0, 1, 2, \dots$

- (a) Write a function which implements a Gibbs sampler to explore the joint distribution of $(X, Y)^T$. It should take arguments `n` giving the number of steps to take, and `rho` which defaults to 0. It should return a $(n + 1) \times 2$ -matrix with a row for each observation, starting with $(X_0, Y_0) = (0, 0)$.

```
> gibbs = function(n, rho = 0) {  
+   out = matrix(0, n + 1, 2)  
+   for (i in 1:n) {  
+     out[i + 1, 1] = rnorm(1, rho * out[i, 2], sqrt(1 -  
+       rho^2))  
+     out[i + 1, 2] = rnorm(1, rho * out[i + 1, 1], sqrt(1 -  
+       rho^2))  
+   }  
+   out  
+ }  
> out = gibbs(100, 0.9)  
> plot(out[, 1], out[, 2], type = "l")
```



- (b) Write a function which generates n samples using the Gibbs sampler, and then estimates the mean of X . Do this $N = 1000$ different times with $\rho = 0.5$ and $n = 100$ (this might take a few seconds). What would you expect the distribution of the mean to be if we had n independent samples from the distribution?

```
> gmean = function(n, rho = 0.5) {
+   mean(gibbs(n, rho)[, 1])
+ }
> means = replicate(1000, gmean(100, rho = 0.9))
> sd(means)

## [1] 0.3115
```

We would expect that the mean of 100 independent standard normals would have a standard deviation of about $1/\sqrt{100} = 0.1$, what we see is slightly higher.

- (c) Try repeating the previous function with $\rho = 0, 0.8, 0.99$ and comparing the variance of your estimates with your answer above. What do you find? Why? (Try plotting your samples as a line.)

For $\rho = 0$ the standard deviation is about what we expect, but it increases as ρ increases. This is because the samples become more and more dependent for $|\rho|$ close to 1.

The difference in efficiency between independent samples and dependent ones is related to the *effective sample size*.

By simulating estimates of the mean of X_i a large number of times, estimate the effective sample size for $n = 1,000$ and $\rho = 0.9$.