CSC 121: Computer Science for Statistics

Radford M. Neal, University of Toronto, 2017

http://www.cs.utoronto.ca/∼radford/csc121/

Week 8

Using Numeric Vectors as Subscripts

A subscript used with `[ ]` can be a vector of indexes, rather than just one index, yielding a subset of elements having those indexes, not just one element:

```
> v <- c(9,10,3)
> names(v) <- c("abc","def","xyz")
> v
abc def xyz
  9  10   3
> v[c(1,3)]     # Notice that names of elements are carried along
abc xyz
  9   3
```

You can also index with a vector of *negative* numbers. This gets you all elements *except* those whose indexes are in the index vector (negated):

```
> v[-2]
abc xyz
  9   3
> v[c(-1,-length(v))]
def
 10
```

Difference Between [ ] and [[ ]]

We can now see better what the difference is between subscripting with [ ] and with [[ ]] — [ ] extracts a subset of elements (which might be just one), whereas [[ ]] extracts a single element.

```
> v
abc def xyz
  9  10   3
> v[2]
def
 10
> v[[2]]          # Notice there's no name here, just the element
[1] 10
> L <- list (a="xy", b=9, c=TRUE)
> L[2]            # Notice that the result is still a list
$b
[1] 9


> L[[2]]          # ... but this is an element of the list
[1] 9
```

Using Logical Vectors as Subscripts

A subscript can also be a logical vector, which selects elements in positions where this subscript is `TRUE`:

```
> v
abc def xyz
  9  10   3
> v[c(TRUE,FALSE,TRUE)]
abc xyz
  9   3
> v[v>5]
abc def
  9  10
```

R's "and" (`&`) and "or" (`|`) operators can be useful for this:

```
> v[v>5 & v<10]
abc
  9
> v[v>9 | v<7]
def xyz
 10   3
```

Vectors as Matrix Subscripts

Vector subscripts can also be used to select rows or columns of a matrix:

```
> M <- matrix (1:12, nrow=3, ncol=4)
> rownames(M) <- c("ab","cd","ef")
> colnames(M) <- c("w","x","y","z")
> M
   w x y  z
ab 1 4 7 10
cd 2 5 8 11
ef 3 6 9 12
> M[c(3,1),c(2,4,4)] # Indexes needn't be in order, can be duplicates
   x  z  z
ef 6 12 12
ab 4 10 10
> M[c(TRUE,FALSE,TRUE),]
   w x y  z
ab 1 4 7 10
ef 3 6 9 12
```

Using Vector Indexes to Replace Elements in a Vector

Numeric and logical vectors can be used as indexes when we replace elements in a vector rather than get them out.

For example:

```
> v <- c(66,33,99,10,12)
> v[c(2,4,1)] <- c(100,200,300)
> v
[1] 300 100  99 200  12
> v[c(TRUE,FALSE,FALSE,FALSE,TRUE)] <- c(800,900)
> v
[1] 800 100  99 200 900
```

Here's how we can use this to make a modified version of the `airquality` data frame (see last week's slides) with missing values for `Solar.R` filled in:

```
mod_airquality <- airquality
mod_airquality$Solar.R [is.na(airquality$Solar.R)] <-
  mean (airquality$Solar.R, na.rm=TRUE)
```

Re-Ordering a Vector, Matrix, or Data Frame

We can change the order of elements in a matrix, or of rows in a matrix or data frame, using an index that is a *permutation* of the possible indexes.

One use is to change the order to be increasing in some variable. The `order` function produces the permutation needed to do this. For example:

```
> heights_and_weights
  name height weight
1 Fred     62    144
2 Mary     60    131
3  Joe     71    182
> by_weight <- order (heights_and_weights$weight)
> by_weight
[1] 2 1 3
> new <- heights_and_weights [by_weight, ]
> new
  name height weight
2 Mary     60    131
1 Fred     62    144
3  Joe     71    182
```

Selecting a Subset of Rows in a Data Frame

Another use of logical indexes is in selecting a subset of rows in a data frame for which the variables have certain values.

For example, here we select only people with weight greater than 140:

```
> heights_and_weights
  name height weight
1 Fred     62    144
2 Mary     60    131
3  Joe     71    182
> heights_and_weights [heights_and_weights$weight > 140, ]
  name height weight
1 Fred     62    144
3  Joe     71    182
```

And here we get only people with weight greater than 140 and height less than 70:

```
> heights_and_weights [heights_and_weights$weight > 140
+                       & heights_and_weights$height < 70, ]
  name height weight
1 Fred     62    144
```

Some Design Flaws in R

R is a very useful language, but like all programming languages, it's not perfect. Indeed, some of R's features are poorly designed, making it too easy to write code that doesn't always work.

I'll talk about two of these here:

- You can't get an empty vector when making a sequence with an expression like `i:j`.
- R will sometimes convert matrices to plain vectors when you don't want it to.

The Problem of Reversing Sequences

The `:` operator will produce either an increasing sequence or a decreasing sequence, depending on whether the first operand is less or greater than the second:

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1
```

This may seem convenient — and it is for Small Assignment 3 — but it's a bad idea. When you use `:` in a program, you need to be sure which sort of sequence you're going to get!

An Illustration of Why Reversing Sequences are Bad

Here's a function that is supposed to return a modified square matrix in which all the elements above the diagonal have been set to one:

```
ones_above_diagonal <- function (M) {
    n <- nrow(M)
    for (i in 1:n)
        for (j in (i+1):n)
            M[i,j] <- 1
    M
}
```

Here's what happens when we try to use it:

```
> ones_above_diagonal(matrix(0,nrow=4,ncol=4))
Error in M[i, j] <- 1 : subscript out of bounds
```

(The exact error message depends on the version of R used.)

Why the error? We need to get a zero-length sequence from `(i+1):n` when `i` equals `n`. But instead we get a sequence of length two, containing `n+1` and `n`.

How could we fix it?

The Problem of Dropped Dimensions

When you index a matrix with a single row or column index, R converts the result to a vector, rather than keep it as a matrix.

Sometimes this is what you want:

```
> M <- matrix(1:6,nrow=2,ncol=3)
> M[1,2]
[1] 3
> M[1,2:ncol(M)]
[1] 3 5
```

But sometimes not:

```
> A <- M[,2:ncol(M)]
> A[1,1]
[1] 3
> B <- M[2:nrow(M),]
> B[1,1]
Error in B[1, 1] : incorrect number of dimensions
```

Stopping R From Dropping Dimensions

You can tell R to not drop dimensions from a matrix with the `drop=FALSE` option:

```
> M <- matrix(1:6,nrow=2,ncol=3)
> M[,2:ncol(M)]
     [,1] [,2]
[1,]    3    5
[2,]    4    6
> M[2:nrow(M),]
[1] 2 4 6
> M[,2:ncol(M),drop=FALSE]
     [,1] [,2]
[1,]    3    5
[2,]    4    6
> M[2:nrow(M),,drop=FALSE]
     [,1] [,2] [,3]
[1,]    2    4    6
```

But adding `drop=FALSE` all the time makes everything longer and messier. So it's tempting not to. But then you may get unexpected bugs once in a while...