Vectors

Derek Sonderegger

December 29, 2015

Vector Creation

R operates on vectors where we think of a vector as a collection of objects, usually numbers. The first thing we need to be able to do is define an arbitrary collection using the `c()` function[1].

```
# Define the vector of numbers 1, ..., 4
c(1,2,3,4)
```

```
## [1] 1 2 3 4
```

There are many other ways to define vectors. The function `rep(x, times)` just repeats `x` a the number times specified by `times`.

```
rep(2, 5)              # repeat 2 five times... 2 2 2 2 2
```

```
## [1] 2 2 2 2 2
```

```
rep( c('A','B'), 3 )   # repeat A B three times  A B A B A B
```

```
## [1] "A" "B" "A" "B" "A" "B"
```

Finally, we can also define a sequence of numbers using the `seq(from, to, by, length.out)` function which expects the user to supply 3 out of 4 possible arguments. The possible arguments are `from`, `to`, `by`, and `length.out`. From is the starting point of the sequence, to is the ending point, by is the difference between any two successive elements, and `length.out` is the total number of elements in the vector.

```
seq(from=1, to=4, by=1)
```

```
## [1] 1 2 3 4
```

```
seq(1,4)        # 'by' has a default of 1
```

```
## [1] 1 2 3 4
```

```
1:4             # a shortcut for seq(1,4)
```

```
## [1] 1 2 3 4
```

[1] The "c" stands for collection.

```r
seq(1,5, by=.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```r
seq(1,5, length.out=11)
```

```
##  [1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0
```

If we have two vectors and we wish to combine them, we can again use the `c()` function.

```r
vec1 <- c(1,2,3)
vec2 <- c(4,5,6)
vec3 <- c(vec1, vec2)
vec3
```

```
## [1] 1 2 3 4 5 6
```

Accessing Vector Elements

Suppose I have defined a vector

```r
foo <- c('A', 'B', 'C', 'D', 'F')
```

and I am interested in accessing whatever is in the first spot of the vector. Or perhaps the 3rd or 5th element. To do that we use the `[]` notation, where the square bracket represents a subscript.

```r
foo[1]  # First element in vector foo
```

```
## [1] "A"
```

```r
foo[4]  # Fourth element in vector foo
```

```
## [1] "D"
```

This subscripting notation can get more complicated. For example I might want the 2nd and 3rd element or the 3rd through 5th elements.

```r
foo[c(2,3)]  # elements 2 and 3
```

```
## [1] "B" "C"
```

```r
foo[ 3:5 ]   # elements 3 to 5
```

```
## [1] "C" "D" "F"
```

Finally, I might be interested in getting the entire vector except for a certain element. To do this, R allows us to use the square bracket notation with a negative index number.

```
foo[-1]          # everything but the first element
```

```
## [1] "B" "C" "D" "F"
```

```
foo[ -1*c(1,2) ] # everything but the first two elements
```

```
## [1] "C" "D" "F"
```

Now is a good time to address what is the `[1]` doing in our output? Because vectors are often very long and might span multiple lines, R is trying to help us by telling us the index number of the left most value. If we have a very long vector, the second line of values will start with the index of the first value on the second line.

```
# The letters vector is a vector of all 26 lower-case letters
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

Here the `[1]` is telling me that `a` is the first element of the vector and the `[18]` is telling me that `r` is the 18th element of the vector.

Scalar Functions Applied to Vectors

It is very common to want to perform some operation on all the elements of a vector simultaneously. For example, I might want take the absolute value of every element. Functions that are inherently defined on single values will almost always apply the function to each element of the vector if given a vector.

```
x <- -5:5
x
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

```
abs(x)
```

```
##  [1] 5 4 3 2 1 0 1 2 3 4 5
```

```
exp(x)
```

```
##  [1] 6.737947e-03 1.831564e-02 4.978707e-02 1.353353e-01 3.678794e-01
##  [6] 1.000000e+00 2.718282e+00 7.389056e+00 2.008554e+01 5.459815e+01
## [11] 1.484132e+02
```

Vector Algebra

All algebra done with vectors will be done element-wise by default.For matrix and vector multiplication as usually defined by mathematicians, use `%*%` instead of `*`. So two vectors added together result in their individual elements being summed.

```
x <- 1:4
y <- 5:8
x + y
```

```
## [1]  6  8 10 12
```

```
x * y
```

```
## [1]  5 12 21 32
```

R does another trick when doing vector algebra. If the lengths of the two vectors don't match, R will recycle the elements of the shorter vector to come up with vector the same length as the longer. This is potentially confusing, but is most often used when adding a long vector to a vector of length 1.

```
x <- 1:4
x + 1
```

```
## [1] 2 3 4 5
```

Commonly Used Vector Functions

Function	Result
`min(x)`	Minimum value in vector x
`max(x)`	Maximum value in vector x
`length(x)`	Number of elements in vector x
`sum(x)`	Sum of all the elements in vector x
`mean(x)`	Mean of the elements in vector x
`median(x)`	Median of the elements in vector x
`var(x)`	Variance of the elements in vector x
`sd(x)`	Standard deviation of the elements in x

Putting this all together, we can easily perform tedious calculations with ease. To demonstrate how scalars, vectors, and functions of them work together, we will calculate the variance of 5 numbers. Recall that variance is defined as

$$Var\left(x\right) = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{n-1}$$

```
x <- c(2,4,6,8,10)
xbar <- mean(x)         # calculate the mean
xbar
```

```
## [1] 6
```

```
x - xbar                # calculate the errors
```

```
## [1] -4 -2  0  2  4
```

```
(x-xbar)^2
```

```
## [1] 16  4  0  4 16
```

```
sum( (x-xbar)^2 )
```

```
## [1] 40
```

```
n <- length(x)          # how many data points do we have
n
```

```
## [1] 5
```

```
sum((x-xbar)^2)/(n-1)   # calculating the variance by hand
```

```
## [1] 10
```

```
var(x)                  # Same thing using the built-in variance function
```

```
## [1] 10
```

Exercises

1. Create a vector of three elements (2,4,6) and name that vector `vec_a`. Create a second vector, `vec_b`, that contains (8,10,12). Add these two vectors together and name the result `vec_c`.
2. Create a vector, named `vec_d`, that contains only two elements (14,20). Add this vector to `vec_a`. What is the result and what do you think R did (look up the recycling rule using Google)? What is the warning message that R gives you?
3. Next add 5 to the vector vec_a. What is the result and what did R do? Why doesn't in give you a warning message similar to what you saw in the previous problem?
4. Generate the vector of integers $\{1, 2, \ldots 5\}$ in two different ways.
    a) First using the `seq()` function
    b) Using the `a:b` shortcut.
5. Generate the vector of even numbers $\{2, 4, 6, \ldots, 20\}$
    a) Using the seq() function and
    b) Using the a:b shortcut and some subsequent algebra. *Hint: Generate the vector 1-10 and then multiple it by 2.*
6. Generate a vector of 1001 elements that are evenly placed between 0 and 1 using the `seq()` command and name this vector `x`.
7. Generate the vector $\{2, 4, 8, 2, 4, 8, 2, 4, 8\}$ using the `rep()` command to replicate the vector c(2,4,8).
8. Generate the vector $\{2, 2, 2, 2, 4, 4, 4, 4, 8, 8, 8, 8\}$ using the `rep()` command. You might need to check the help file for rep() to see all of the options that rep() will accept. In particular, look at the optional argument `each=`.

9. The vector `letters` is a built-in vector to R and contains the lower case English alphabet.

   a) Extract the 9th element of the letters vector.
   b) Extract the sub-vector that contains the 9th, 11th, and 19th elements.
   c) Extract the sub-vector that contains everything except the last two elements.