Arrays Operations in R

Fan Wang

2021-11-02

Contents

1 Array Basics

Go to the **RMD**, **R**, **PDF**, or **HTML** version of this file. Go back to fan's REconTools Package, R Code Examples Repository (bookdown site), or Intro Stats with R Repository (bookdown site).

1.1 Multidimesional Arrays

1.1.1 Repeat one Number by the Size of an Array

```r
ar_a <- c(1,2,3)
ar_b <- c(1,2,3/1,2,3)
rep(0, length(ar_a))
```

```
## [1] 0 0 0
```

1.1.2 Generate 2 Dimensional Array

```r
# Multidimensional Array
# 1 is r1c1t1, 1.5 in r2c1t1, 0 in r1c2t1, etc.
# Three dimensions, row first, column second, and tensor third
x <- array(c(1, 1.5, 0, 2, 0, 4, 0, 3), dim=c(2, 2, 2))
dim(x)
```

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

```
print(x)
```

```
## , , 1
##
##      [,1] [,2]
## [1,]  1.0    0
## [2,]  1.5    2
##
## , , 2
##
##      [,1] [,2]
## [1,]    0    0
## [2,]    4    3
```

1.2 Array Slicing

1.2.1 Get a Subset of Array Elements, N Cuts from M Points

There is an array with M elements, get N elements from the M elements.

First cut including the starting and ending points.

```
it_M <- 5
it_N <- 4
ar_all_elements = seq(1,10,10)
```

1.2.2 Remove Elements of Array

Select elements with direct indexing, or with head and tail functions. Get the first two elements of three elements array.

```
# Remove last element of array
vars.group.bydf <- c('23','dfa', 'wer')
vars.group.bydf[-length(vars.group.bydf)]
```

```
## [1] "23"  "dfa"
```

```
# Use the head function to remove last element
head(vars.group.bydf, -1)
```

```
## [1] "23"  "dfa"
```

```
head(vars.group.bydf, 2)
```

```
## [1] "23"  "dfa"
```

Get last two elements of array.

```
# Remove first element of array
vars.group.bydf <- c('23','dfa', 'wer')
vars.group.bydf[2:length(vars.group.bydf)]
```

```
## [1] "dfa" "wer"
```

```
# Use Tail function
tail(vars.group.bydf, -1)
```

```
## [1] "dfa" "wer"
```

```
tail(vars.group.bydf, 2)
```

```
## [1] "dfa" "wer"
```

Select all except for the first and the last element of an array.

```
# define array
ar_amin <- c(0, 0.25, 0.50, 0.75, 1)
# select without head and tail
tail(head(ar_amin, -1), -1)
```

```
## [1] 0.25 0.50 0.75
```

Select the first and the last element of an array. The extreme values.

```
# define array
ar_amin <- c(0, 0.25, 0.50, 0.75, 1)
# select head and tail
c(head(ar_amin, 1), tail(ar_amin, 1))
```

```
## [1] 0 1
```

1.3 NA in Array

1.3.1 Check if NA is in Array

```
# Convert Inf and -Inf to NA
x <- c(1, -1, Inf, 10, -Inf)
na_if(na_if(x, -Inf), Inf)
```

```
## [1]  1 -1 NA 10 NA
```

1.4 Complex Number

Handling numbers with real and imaginary components. Two separate issues, given an array of numbers that includes real as well as imaginary numbers, keep subset that only has real components. Additionally, for the same array, generate an equal length version of the array that includes the real components of all numbers.

Define complex numbers.

```
# Define a complex number
cx_number_a <- 0+0.0460246857561777i
# Define another complex number
cx_number_b <- complex(real = 0.02560982, imaginary = 0.0460246857561777)
# An array of numbers some of which are complex
ar_cx_number <- c(0.02560982+0.000000000i, 0.00000000+0.044895305i,
                  0.00000000+0.009153429i, 0.05462045+0.000000000i,
                  0.00000000+0.001198538i, 0.00000000+0.019267050i)
```

Extract real components from a complex array.

```
# equi-length real component
ar_fl_number_re <- Re(ar_cx_number)
print(ar_fl_number_re)
```

```
## [1] 0.02560982 0.00000000 0.00000000 0.05462045 0.00000000 0.00000000
```

```
# equi-length img component
ar_fl_number_im <- Im(ar_cx_number)
print(ar_fl_number_im)
```

```
## [1] 0.000000000 0.044895305 0.009153429 0.000000000 0.001198538 0.019267050
```

Keep only real elements of array.

```
# subset of array that is real
ar_fl_number_re_subset <- Re(ar_cx_number[Re(ar_cx_number)!=0])
print(ar_fl_number_re_subset)
```

```
## [1] 0.02560982 0.05462045
```

1.5 Number Formatting

1.5.1 e notation

1. Case one: *1.149946e+00*
   - this is approximately: 1.14995
2. Case two: *9.048038e-01*
   - this is approximately: 0.90480
3. Case three: *9.048038e-01*
   - this is approximately: 0.90480

1.6 String Conversions

1.6.1 Add Positive and Negative Sign in Front of Values

We have a sequence of integers, some positive and some negative. We convert this into a string array, and append positive sign in front of positive values.

```
# An array of integers
ar_it_vals <- seq(-5, 5, by = 1)
# Add positive sign in front of positive and zero elements
st_it_vals <- paste0(ar_it_vals)
st_it_vals[ar_it_vals>0] <- paste0("+", st_it_vals[ar_it_vals>0])
st_it_vals[ar_it_vals==0] <- paste0("±", st_it_vals[ar_it_vals==0])
# Display
print(st_it_vals)
```

```
##  [1] "-5" "-4" "-3" "-2" "-1" "±0" "+1" "+2" "+3" "+4" "+5"
```