Basic Arrays Operations in R

Fan Wang

2023-03-08

Contents

1 Array Basics

Go to the **RMD**, **R**, **PDF**, or **HTML** version of this file. Go back to fan's REconTools research support package, R4Econ examples page, PkgTestR packaging guide, or Stat4Econ course page.

1.1 Sum and Product of Elements in Array

Product of Elements in Array.

```r
ar_a <- c(1,2,3)
ar_b <- c(1,2,3,4)
prod(ar_a)
```

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

```r
prod(ar_b)
```

```
## [1] 24
```

1.2 Multidimesional Arrays

1.2.1 Repeat one Number by the Size of an Array

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

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

1.2.2 Generate 2 Dimensional Array

First, we will generate an NaN matrix with 3 rows and 3 columnes.

```
mt_x <- array(NA, dim=c(3, 3))
dim(mt_x)
```

```
## [1] 3 3
```

```
print(mt_x)
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
## [3,]   NA   NA   NA
```

Second, we will generate a matrix with 2 rows and four columns.

```
mt_x <- array(c(1, 1.5, 0, 2, 0, 4, 0, 3), dim=c(2, 4))
dim(mt_x)
```

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

```
print(mt_x)
```

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

1.2.3 Generate 3 Dimensional Array

First, we will create a three dimensional array with the same data as what was used to create the 2-dimensional array on top.

```
# 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
```

Second, in the example below, we will generate a 3-dimensional array. The first dimension corresponds to different income levels, the second marital status, and the third the number of kids.We compute in the example below taxable income in 2008 given income levels given IRS rules.

```
# A, Income Array
ar_income <- seq(0,200000,length.out=3)

# B. Exemptions and Deductions
fl_exemption <- 3500# exemption amount per household member
mt_deduction <- matrix(data=NA, nrow=2, ncol=5)# Marital-status and number of children-specific deducti
mt_deduction[1,1] <- 5450# Single filers
mt_deduction[1,2:5] <- 8000# Single filer with children
mt_deduction[2,] <- 10900# Married couples filing jointly

# C. Taxable Income
mn_taxable_income <- array(NA, dim=c(length(ar_income), 2, 5))
for (y in 1:length(ar_income)){
    for (m in 1:2){
        for (k in 0:4){
            mn_taxable_income[y,m,k+1] <- ar_income[y]-fl_exemption*m-fl_exemption*k-mt_deduction[m,k+1
        }
    }
}

# D. Name dimensions
dimnames(mn_taxable_income)[[1]] = paste0('income=', round(ar_income, 0))
dimnames(mn_taxable_income)[[2]] = paste0('married=', 0:1)
dimnames(mn_taxable_income)[[3]] = paste0('kids=', 0:4)

# E. Print
dim(mn_taxable_income)
```

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

```
print(mn_taxable_income)
```

```
## , , kids=0
##
##               married=0 married=1
## income=0          -8950    -17900
## income=1e+05      91050     82100
## income=2e+05     191050    182100
##
## , , kids=1
##
##               married=0 married=1
## income=0         -15000    -21400
## income=1e+05      85000     78600
## income=2e+05     185000    178600
##
```

```
## , , kids=2
##
##               married=0 married=1
## income=0         -18500    -24900
## income=1e+05      81500     75100
## income=2e+05     181500    175100
##
## , , kids=3
##
##               married=0 married=1
## income=0         -22000    -28400
## income=1e+05      78000     71600
## income=2e+05     178000    171600
##
## , , kids=4
##
##               married=0 married=1
## income=0         -25500    -31900
## income=1e+05      74500     68100
## income=2e+05     174500    168100
```

1.3 Array Slicing

1.3.1 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)
```

1.3.2 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
```

1.4 NA in Array

1.4.1 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
```

1.5 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
```

1.6 Number Formatting

1.6.1 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

1.7 String Conversions

1.7.1 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"
```

1.8 Basic array calculations

First, we demonstrate how purrr::reduce() works with a simple summation example. We use the addition operator.

```
# Using R pipe operator
# 1 + 2 + 3 = 6
fl_sum <- 1:3 |> purrr::reduce(`+`)
print(fl_sum)
```

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

Second, what if there is an NA value? NA will be ignored, we will write a custom function. The custom function, to work with reduce, should be such that it is “a binary function that takes two values and returns a single value”.

```r
# define sum function that ignores NA
sum_ignore_na <- function(x,y) {
  if (!is.na(x) && !is.na(y)) {
    x + y
  } else if (is.na(x)) {
    y
  } else if (is.na(y)) {
    x
  } else {
    NA
  }
}

# Using R pipe operator
# 1 + 10 + 1 = 12
fl_sum <- c(1, 10, NA, 1) |> purrr::reduce(sum_ignore_na)
print(fl_sum)
```

```
## [1] 12
```