R Generate Arrays

Fan Wang

2021-04-03

Contents

1 Generate Arrays

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 Generate Often Used Arrays

1.1.1 Equi-distance Array with Bound

Consider multiple income groups in income bins that are equal-width, for the final income group, consider all individuals above some final bin minimum bound. Below the code generates this array of numbers: $0, 20000, 40000, 60000, 80000, 100000, 100000000$.

```
# generate income cut-offs
fl_bin_start <- 0
# width equal to 20,000
fl_bin_width <- 2e4
# final point is 100 million
fl_bin_final_end <- 1e8
# final segment starting point is 100,000 dollars
fl_bin_final_start <- 1e5
# generate tincome bins
ar_income_bins <- c(seq(fl_bin_start, fl_bin_final_start, by=fl_bin_width),
                    fl_bin_final_end)
# Display
print(ar_income_bins)
```

```
## [1]         0     20000     40000     60000     80000    100000 100000000
```

Generate finer bins, at 5000 USD intervals, and stopping at 200 thousand dollars.

```
fl_bin_start <- 0
fl_bin_width <- 5e3
fl_bin_final_end <- 1e8
fl_bin_final_start <- 2e5
```

```
ar_income_bins <- c(seq(fl_bin_start, fl_bin_final_start, by=fl_bin_width),
                    fl_bin_final_end)
print(ar_income_bins)
```

```
##  [1]         0      5000     10000     15000     20000     25000     30000     35000     40000     45
## [17]     80000     85000     90000     95000    100000    105000    110000    115000    120000    125
## [33]    160000    165000    170000    175000    180000    185000    190000    195000    200000 100000
```

1.1.2 Log Space Arrays

Often need to generate arrays on log rather than linear scale, below is log 10 scaled grid.

```
# Parameters
it.lower.bd.inc.cnt <- 3
fl.log.lower <- -10
fl.log.higher <- -9
fl.min.rescale <- 0.01
it.log.count <- 4
# Generate
ar.fl.log.rescaled <- exp(log(10)*seq(log10(fl.min.rescale),
                                      log10(fl.min.rescale +
                                              (fl.log.higher-fl.log.lower)),
                                      length.out=it.log.count))
ar.fl.log <- ar.fl.log.rescaled + fl.log.lower - fl.min.rescale
# Print
ar.fl.log
```

```
## [1] -10.000  -9.963  -9.793  -9.000
```

1.2 Generate Arrays Based on Existing Arrays

1.2.1 Probability Mass Array and Discrete Value Array

There are two arrays, an array of values, and an array of probabilities. The probability array sums to 1. The array of values, however, might not be unique.

First, generate some array of numbers not sorted and some proability mass for each non-sorted, non-unique element of the array.

```
set.seed(123)
it_len <- 10
ar_x <- ceiling(runif(it_len)*5+10)
ar_prob <- dbinom(seq(0,it_len-1,length.out = it_len), it_len-1, prob=0.5)
print(cbind(ar_x,ar_prob))
```

```
##       ar_x  ar_prob
##  [1,]   12 0.001953
##  [2,]   14 0.017578
##  [3,]   13 0.070312
##  [4,]   15 0.164063
##  [5,]   15 0.246094
##  [6,]   11 0.246094
##  [7,]   13 0.164063
##  [8,]   15 0.070312
##  [9,]   13 0.017578
## [10,]   13 0.001953
```

```
print(paste0('sum(ar_prob)=',sum(ar_prob)))
```

```
## [1] "sum(ar_prob)=1"
```

Second, sorting index for ar_x, and resort ar_prob with the same index:

```
ls_sorted_res <- sort(ar_x, decreasing = FALSE, index.return=TRUE)
ar_idx_increasing_x <- ls_sorted_res$ix
ar_x_sorted <- ls_sorted_res$x
ar_prob_sorted <- ar_prob[ar_idx_increasing_x]
print(cbind(ar_x_sorted,ar_prob_sorted))
```

```
##       ar_x_sorted ar_prob_sorted
##  [1,]          11       0.246094
##  [2,]          12       0.001953
##  [3,]          13       0.070312
##  [4,]          13       0.164063
##  [5,]          13       0.017578
##  [6,]          13       0.001953
##  [7,]          14       0.017578
##  [8,]          15       0.164063
##  [9,]          15       0.246094
## [10,]          15       0.070312
```

Third, sum within group and generate unique, using the aggregate function. Then we have a column of unique values and associated probabilities.

```
ar_x_unique <- unique(ar_x_sorted)
mt_prob_unique <- aggregate(ar_prob_sorted, by=list(ar_x_sorted), FUN=sum)
ar_x_unique_prob <- mt_prob_unique$x
print(cbind(ar_x_unique, ar_x_unique_prob))
```

```
##      ar_x_unique ar_x_unique_prob
## [1,]          11         0.246094
## [2,]          12         0.001953
## [3,]          13         0.253906
## [4,]          14         0.017578
## [5,]          15         0.480469
```

Finally, the several steps together.

```
# data
set.seed(123)
it_len <- 30
ar_x <- ceiling(runif(it_len)*20+10)
ar_prob <- runif(it_len)
ar_prob <- ar_prob/sum(ar_prob)
# step 1, sort
ls_sorted_res <- sort(ar_x, decreasing = FALSE, index.return=TRUE)
# step 2, unique sorted
ar_x_unique <- unique(ls_sorted_res$x)
# step 3, mass for each unique
mt_prob_unique <- aggregate(ar_prob[ls_sorted_res$ix], by=list(ls_sorted_res$x), FUN=sum)
ar_x_unique_prob <- mt_prob_unique$x
# results
print(cbind(ar_x_unique, ar_x_unique_prob))
```

```
##       ar_x_unique ar_x_unique_prob
##  [1,]          11          0.07172
##  [2,]          13          0.04004
##  [3,]          15          0.01771
##  [4,]          16          0.14120
##  [5,]          17          0.02021
##  [6,]          19          0.05249
##  [7,]          20          0.04910
##  [8,]          21          0.06733
##  [9,]          22          0.10945
## [10,]          23          0.06071
## [11,]          24          0.10767
## [12,]          25          0.01569
## [13,]          26          0.06857
## [14,]          28          0.09093
## [15,]          29          0.00187
## [16,]          30          0.08530
```