Advanced array operations in the gRbase package – NEW Rcpp-based version

Søren Højsgaard

gRbase version 1.7-0 as of 2014-03-22

1 Arrays / tables

This note describes various functions in the gRbase package for operations on arrays in R. We shall use the words "array" and "table" interchangably here. By an array we here mean a vector with dim and a dimnames attribute. (Arrays do not need a dimnames attribute, but dimnames are essential in most of what follows here). Consider the lizard data in gRbase:

```
R> data(lizard, package="gRbase")
R> lizard
```

```
, , species = anoli

     height
diam  >4.75 <=4.75
  <=4    32     86
  >4     11     35

, , species = dist

     height
diam  >4.75 <=4.75
  <=4    61     73
  >4     41     70
```

Data is of class table and has dim and dimnames attributes. Data is internally just a vector,

```
R> class( lizard )
```

```
[1] "table"
```

```
R> str( lizard )
```

```
 table [1:2, 1:2, 1:2] 32 11 86 35 61 41 73 70
 - attr(*, "dimnames")=List of 3
```

```
  ..$ diam   : chr [1:2] "<=4" ">4"
  ..$ height : chr [1:2] ">4.75" "<=4.75"
  ..$ species: chr [1:2] "anoli" "dist"
```

```
R> str( dimnames( lizard ) )
```

```
List of 3
 $ diam   : chr [1:2] "<=4" ">4"
 $ height : chr [1:2] ">4.75" "<=4.75"
 $ species: chr [1:2] "anoli" "dist"
```

The variables [`diam`, `height`, `species`] and their levels defines a universe. A table is a mapping from the universe to the value of a particular entry in a vector. Note that the ordering of the variables in the universe is important and that the first variable (here `diam`) varies fastest.

2 Operations on tables

Let $[A, B, C]$ be a universe where (a, b, c) denotes a specific configuration of $[A, B, C]$. Hence a table $P_{[ABC]}$ is indexed by (a, b, c).

The $[A, C]$ marginal of $P_{[ABC]}$ is the table $P_{[AC]}$ defined by

$$P_{[AC]}(a, c) \leftarrow \sum_b P_{[ABC]}(a, b, c)$$

The $C = c'$ slice of $P_{[ABC]}$ is

$$P_{[AB]}(a, b) \leftarrow P_{[ABC]}(a, b, c')$$

A permutation of $P_{[ABC]}$ into, say $P_{[CBA]}$ is a reordering of the values in the table so that

$$P_{[ABC]}(a, b, c) = P_{[CBA]}(c, b, a)$$

The domain extension of two tables $P_{[ABC]}$ and $Q_{[CDB]}$ is two new tables defined on the same set of variables (but possibly in different orders). For example $P_{[DABC]}$ and $Q_{[ACDB]}$ where

$$P_{[DABC]}(d, a, b, c) \leftarrow P_{[ABC]}(a, b, c) \text{ for all values of } d.$$

An alignment of two tables $P_{[DABC]}$ and $Q_{[ACDB]}$ over the same variables refers to a permutation such that the ordering of the variables become the same, e.g. $P_{[ABCD]}$ and $Q_{[ABCD]}$.

The product of tables $P_{[ABC]}$ and $Q_{[CDB]}$ is defined as the table $R_{(ABCD)}$ given by

$$R_{[ABCD]}(a, b, c, d) \leftarrow P_{[ABC]}(a, b, c)Q_{[CDB]}(c, d, b)$$

The quotient, sum and difference is defined similarly. By definition $0/0 = 0$. The extension and alignment defined above is used for computing the product etc.

3 Operations on tables – R

To illustrate we find two marginal tables

```
R> P <- arrayMargin(lizard, c("species","height")); P
```

```
       height
species >4.75 <=4.75
  anoli    43    121
  dist    102    143
```

```
R> Q <- arrayMargin(lizard, c("diam","species")); Q
```

```
     species
diam  anoli dist
  <=4   118  134
  >4     46  111
```

A slice of a table is obtained with `tableSlice`:

```
R> tableSlice(lizard, "species", "anoli")
```

```
     height
diam  >4.75 <=4.75
  <=4    32     86
  >4     11     35
```

A permutation is:

```
R> R  <- lizard
R> Rp <- arrayPerm(R, c("species","height","diam")); ftable( Rp )
```

```
               diam <=4 >4
species height
anoli   >4.75        32 11
        <=4.75       86 35
dist    >4.75        61 41
        <=4.75       73 70
```

From the users perspective, multiplication etc. of these is done with

```
R> R <- arrayOp( P, Q, op = "*" ); ftable( R )
```

```
             height >4.75 <=4.75
diam species
<=4  anoli           5074  14278
     dist           13668  19162
>4   anoli           1978   5566
     dist           11322  15873
```

Extending P and Q to have the variables (but possibly in different order):

```
R> PeQe <- extendArrays( P, Q ); lapply( PeQe, ftable )
```

```
[[1]]
             height >4.75 <=4.75
diam species
```

```
<=4  anoli            43    121
     dist            102    143
>4   anoli            43    121
     dist            102    143

[[2]]
            species anoli dist
height diam
>4.75  <=4            118  134
       >4              46  111
<=4.75 <=4            118  134
       >4              46  111

R> #str( lapply( PeQe, dimnames ), max.level=2)
```

Aligning tables: If two tables do not have the same domain, they are aligned before the extension is made:

```
R> PeQe2 <- alignArrays( P, Q )
R> lapply( PeQe2, ftable )
[[1]]
             height >4.75 <=4.75
diam species
<=4  anoli            43    121
     dist            102    143
>4   anoli            43    121
     dist            102    143

[[2]]
             height >4.75 <=4.75
diam species
<=4  anoli           118    118
     dist            134    134
>4   anoli            46     46
     dist            111    111

R> #str( lapply( PeQe, dimnames ), max.level=2)
```

4 Defining tables / arrays

The examples here relate to the chest clinic example of Lauritzen and Spiegelhalter. The following two specifications are equivalent:

```
R> yn <- c('y','n')
R> parray(c("tub","asia"), levels=list(yn, yn), values=c(5,95,1,99))

   asia
tub  y  n
```

```
  y  5  1
  n 95 99
```

```
R> array(c(5,95,1,99), dim=c(2,2), dimnames=list("tub"=yn, "asia"=yn))
```

```
   asia
tub  y  n
  y  5  1
  n 95 99
```

Using `parray()`, arrays can be normalized in two ways: Normalization can be over the first variable for *each* configuration of all other variables or over all configurations. We illustrate this by defining the probability of tuberculosis given a recent visit to Asia and by defining the marginal probability of a recent visit to Asia:

```
R> p.t.a <- parray(c("tub","asia"), levels=list(yn, yn),
+                 values=c(5,95,1,99), normalize="first")
```

```
   asia
tub    y    n
  y 0.05 0.01
  n 0.95 0.99
```

```
R> p.a <- parray("asia", list(yn), values=c(1,99),
+                normalize="all")
```

```
asia
   y    n
0.01 0.99
```

5 Calculations with probability tables

The joint distributions is

```
R> p.ta <- arrayOp(p.t.a, p.a, op="*")
```

```
   asia
tub      y      n
  y 0.0005 0.0099
  n 0.0095 0.9801
```

The marginal distribution of `"tub"` is

```
R> p.t <- arrayMargin(p.t.a, "tub")
```

```
tub
   y    n
0.06 1.94
```

The conditional distribution of `"asia"` given `"tub"` is

```
R> arrayOp(p.ta, p.t, op="/")
```

```
   asia
tub           y         n
  y 0.008333333 0.1650000
  n 0.004896907 0.5052062
```

6 IPS

A rudimentary implementation of iterative proportional scaling for log–linear models is straight forward:

```
R> myips <- function(indata, glist){
+     fit   <- indata
+     fit[] <-  1
+     ## List of sufficient marginal tables
+     md    <- lapply(glist, function(g) arrayMargin(indata, g))
+
+     for (i in 1:4){
+         for (j in seq_along(glist)){
+             mf  <- arrayMargin(fit, glist[[j]])
+             adj <- arrayOp( md[[j]], mf, op="/" )
+             fit <- arrayOp( fit, adj, op="*" )
+         }
+     }
+     pearson=sum( (fit-indata)^2 / fit)
+     pearson
+ }
R> glist<-list(c("species","diam"),c("species","height"),c("diam","height"))
R> str( myips(lizard, glist), max.level=2)
```

```
 num 0.151
```

```
R> str( loglin(lizard, glist), max.level = 2)
```

```
4 iterations: deviation 0.009618708
List of 4
 $ lrt    : num 0.149
 $ pearson: num 0.151
 $ df     : num 1
 $ margin :List of 3
  ..$ : chr [1:2] "species" "diam"
  ..$ : chr [1:2] "species" "height"
  ..$ : chr [1:2] "diam" "height"
```