

Primitive array operations in the **gRbase** package

Søren Højsgaard

gRbase version 1.6-11 as of 2013-07-01

Contents

1	Introduction	1
1.1	Arrays in R	1
1.2	Terminology	2
2	cell2entry() and entry2cell()	2
3	nextCell() and nextCellSlice()	3
4	slice2entry()	3
5	permuteCellEntries()	3
6	factGrid() – Factorial grid	4

1 Introduction

This note describes some operations on arrays in R. These operations have been implemented to facilitate implementation of graphical models and Bayesian networks in R.

1.1 Arrays in R

The documentation of R states the following about arrays:

An array in R can have one, two or more dimensions. It is simply a vector which is stored with additional attributes giving the dimensions (attribute "dim") and optionally names for those dimensions (attribute "dimnames").

A two-dimensional array is the same thing as a matrix.

One-dimensional arrays often look like vectors, but may be handled differently by some functions.

Hence the defining characteristic of an array is that it is a vector with a dim attribute. For example

```
R> ## 1-dimensional array
R> ##
R> x1 <- 1:8
R> dim(x1) <- 8
R> x1
[1] 1 2 3 4 5 6 7 8
R> c(is.array(x1), is.matrix(x1))
```

```

[1] TRUE FALSE
R> ## 2-dimensional array (matrix)
R> ##
R> x2 <- 1:8
R> dim(x2) <- c(2,4)
R> x2
      [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
R> c(is.array(x2), is.matrix(x2))
[1] TRUE TRUE
R> ## 3-dimensional array
R> ##
R> x3 <- array(1:8, dim=c(2,2,2))
R> x3
, , 1
      [,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
      [,1] [,2]
[1,]    5    7
[2,]    6    8
R> c(is.array(x3), is.matrix(x3))
[1] TRUE FALSE

```

1.2 Terminology

Consider a set $\Delta = \{\delta_1, \dots, \delta_K\}$ of $|\Delta| = K$ factors where the factor δ_k has levels $I_k = \{1, \dots, L_k\}$. The cross product $I = I_1 \times \dots \times I_K$ defines an array where $i = (i_1, \dots, i_K) \in I$ is a cell. It is the convention here that the first factor varies fastest. To each cell $i \in I$ there is often a value $f(i)$.

As shown above, an array is implemented as a vector x of length $L = |I|$, that is $x \equiv (f(i), i \in I)$. In practice x is indexed by an entry e as $x[e]$ for $e = 1, \dots, L$.

The factor levels (I_1, \dots, I_K) are denoted `adim` in the code below. As an example we take the following:

```

R> adim2222 <- c(2,2,2,2)
R> adim2323 <- c(2,3,2,3)

```

2 cell2entry() and entry2cell()

The map from a cell to the corresponding entry is provided by `cell2entry()`. The reverse operation, going from an entry to a cell (which is much less needed) is provided by `entry2cell()`.

```

R> cell2entry(c(1,1,1,1), adim2222)
[1] 1
R> entry2cell(1, adim2222)
[1] 1 1 1 1

```

```
R> cell2entry(c(2,1,2,1), adim2222)
[1] 6
R> entry2cell(6, adim2222)
[1] 2 1 2 1
```

3 nextCell() and nextCellSlice()

Given a cell, say $i = (1,1,2,1)$ we often want to find the next cell in the table following the convention that the first factor varies fastest, that is $(2,1,2,1)$. This is provided by `nextCell()`.

```
R> nextCell(c(1,1,2,1), adim2222)
[1] 2 1 2 1
R> nextCell(c(2,2,2,1), adim2222)
[1] 1 1 1 2
```

Given $A \subset \Delta$ and a cell $i_A \in I_A$ consider the cells $I(i_A) = \{j \in I | j_A = i_A\}$. For example, the cells satisfying that factor 2 is at level 1. Given such a cell, say $(2,1,1,2)$ we often want to find the next cell also satisfying this constraint following the convention that the first factor varies fastest, that is $(1,1,2,2)$. This is provided by `nextCellSlice()`.

```
R> nextCellSlice(c(2,1,1,2), sliceset=c(2), adim2323)
[1] 1 1 2 2
R> nextCellSlice(c(1,3,2,1), sliceset=c(2,3), adim2323)
[1] 2 3 2 1
```

4 slice2entry()

Given $A \subset \Delta$ and a cell $i_A \in I_A$. This cell defines a slice of the original array, namely the cells $I(i_A) = \{j \in I | j_A = i_A\}$. We often want to find the entries in x for the cells $I(i_A)$. This is provided by `slice2entry()`. For example, we may want the entries for the cells $(*,1,2,*)$ or $(2,2,*,*)$:

```
R> (r1<-slice2entry(slicecell=c(1,2), sliceset=c(2,3), adim2222))
[1] 5 6 13 14
```

To verify that we indeed get the right cells:

```
R> do.call(rbind, lapply(r1, entry2cell, adim2222))
      [,1] [,2] [,3] [,4]
[1,]    1    1    2    1
[2,]    2    1    2    1
[3,]    1    1    2    2
[4,]    2    1    2    2
```

5 permuteCellEntries()

In a 2×3 table, entries $1, \dots, 6$ correspond to combinations $(1,1), (2,1), (1,2), (2,2), (1,3), (2,3)$. If we permute the table to a 3×2 table the entries become as follows:

```
R> (p<-permuteCellEntries(perm=c(2,1), adim=c(2,3)))
[1] 1 3 5 2 4 6
```

So for example,

```
R> (A <- array(11:16, dim=c(2,3)))
```

```
      [,1] [,2] [,3]
[1,]   11   13   15
[2,]   12   14   16
```

```
R> Ap <- A[p]
```

```
R> dim(Ap) <- c(3,2)
```

```
R> Ap
```

```
      [,1] [,2]
[1,]   11   12
[2,]   13   14
[3,]   15   16
```

This corresponds to

```
R> aperm(A, c(2,1))
```

```
      [,1] [,2]
[1,]   11   12
[2,]   13   14
[3,]   15   16
```

6 factGrid() – Factorial grid

Using the operations above we can obtain the combinations of the factors as a matrix:

```
R> ff <- factGrid(adim2222)
```

```
R> head(ff)
```

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

```
R> tail(ff)
```

```
      [,1] [,2] [,3] [,4]
[11,]    1    2    1    2
[12,]    2    2    1    2
[13,]    1    1    2    2
[14,]    2    1    2    2
[15,]    1    2    2    2
[16,]    2    2    2    2
```

This is the same as (but faster)

```
R> aa <- expand.grid(list(1:2,1:2,1:2,1:2))
```

```
R> head(aa)
```

```
      Var1 Var2 Var3 Var4
1         1    1    1    1
2         2    1    1    1
3         1    2    1    1
4         2    2    1    1
5         1    1    2    1
6         2    1    2    1
```

There is a slice version as well:

```
R> factGrid(adim2222, slicecell=c(1,2), sliceset=c(2,3))
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	1	2	1
[2,]	2	1	2	1
[3,]	1	1	2	2
[4,]	2	1	2	2