Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

diagonals: Fat Diagonals in R*

Bastiaan Quast
The Graduate Institute, Geneva

Abstract

The diagonals package implements functions for handling fat diagonal matrices, such as those that occur when multiple dimensions are mapped along one edge of a matrix.

Keywords: matrix, diagonal, block, R.

1. Introduction

Diagonals are an important matrix manipulation. We present the `diagonals` R package, which implements functions for dealing with **fat diagonals**. Fat diagonals are block matrix-diagonals that occur when two or more dimensions are mapped along a single edge of a matrix. For an asymmetric network graph (e.g. a dyadic social network) to be mapped to a matrix, we would need each node along each edge of matrix, however we would also need to map the direction of the tie, which is an additional dimension. Typically these would be represented as higher-order arrays (i.e. `length(dim(m))>=3`). In order to effectively visualise such arrays, it can be helpful to do so in a matrix (i.e. `length(dim(m))==2`). This could for instance be represented as in the following matrix (where the `i` and `o` suffices represent incoming and outgoing respectively).

```
  Ai Ao Bi Bo Ci Co Di Do
A  1  1  0  0  1  0  1  0
B  0  1  1  1  0  1  0  1
C  1  0  0  0  1  1  0  0
D  1  0  1  0  1  1  1  1
```

*This research was financed by the Swiss National Science Foundation (SNSF) under the grant 'Development Aid and Social Dynamics' (100018-140745) administered by the Graduate Institute's Centre on Conflict, Development and Peacebuilding (CCDP) and led by Jean-Louis Arcand. We thank Sandra Reimann and Oliver Jutersonke at the CCDP for their generous support.

Sometimes the ties of a node to itself are not particularly meaningful (e.g. feeling of amiability towards oneself) and can be removed. For a symmetric network this can simply be done using the function `diag()` in R's `base` package, e.g.

```
sm <- matrix(1, nrow=4, ncol=4,
             dimnames = list(c("A","B","C","D"),c("A","B","C","D")))
diag(sm) <- NA
sm
```

```
   A  B  C  D
A NA  1  1  1
B  1 NA  1  1
C  1  1 NA  1
D  1  1  1 NA
```

However, for higher-order matrices this does not work well, since the diagonal follows the shortest dimension.

```
diag(m) <- NA
m
```

```
  Ai Ao Bi Bo Ci Co Di Do
A NA  1  0  0  1  0  1  0
B  0 NA  1  1  0  1  0  1
C  1  0 NA  0  1  1  0  0
D  1  0  1 NA  1  1  1  1
```

In comes the `diagonals` package and its workhorse `fatdiag()` function. The function is designed to mimic the behaviour of the `diag()` as closely as possible, but with then for **fat diagonals**.

```
library(diagonals)
```

```
# the matrix m was restored to its original state
fatdiag(m, steps=4) <- NA
m
```

```
  Ai Ao Bi Bo Ci Co Di Do
A NA NA  0  0  1  0  1  0
B  0  1 NA NA  0  1  0  1
C  1  0  0  0 NA NA  0  0
D  1  0  1  0  1  1 NA NA
```

Note that the `steps` argument defines the number of steps on the diagonal ladder. Alternatively we could set the `size` of the step, more on this later.

The functions is this package where originally written in order to support the `gvc` package, which implements a collection of trade-flow indicators.

In trade flows use Inter-Country Input Output tables (ICIOs), which map every `country * industry` combination along both edges of the matrix. These tables are used to compute the Leontief inverse (Leontief 1936). In order to make the value of the Leontief internally comparable we can normalise the inverse using post multiplication. Generally, this post multiplication is done using a country's own exports, final demand, etc. and since the `country * industry` dimensions are on each edge, this gives us a higher-order matrix for which we want to extrac the diagonal.

2. Data

As mentioned in the introduction, the typical use case is the mapping of a 3- or 4-dimensional array to a matrix with two edges, this can be done by combining two dimensions along one edge. This form of representation occupies a middle group between the mathematically efficient `array`, and the intuitive tidy data (see Wickham 2014), being the most efficient way that is directly representable on paper.

The matrix used in the introduction is an example of a mapping of a 3-dimensional array.

```
  Ai Ao Bi Bo Ci Co Di Do
A  1  1  0  0  1  0  1  0
B  0  1  1  1  0  1  0  1
C  1  0  0  0  1  1  0  0
D  1  0  1  0  1  1  1  1
```

The **fat diagonals** occur when we have such a matrix and want to select the diagonal along two of these dimensions, and all elements along the third dimension. In the above matrix for instance, we would select the diagonal along the dimensions `A:D` and `A:D`, and all elements along the dimension `i:o`. The `size` of the the block on the diagonal here take the values `c(1,n)` with `n` being the `length` of the third dimension.

```
  Ai Ao Bi Bo Ci Co Di Do
A NA NA  0  0  1  0  1  0
B  0  1 NA NA  0  1  0  1
C  1  0  0  0 NA NA  0  0
D  1  0  1  0  1  1 NA NA
```

In the case of a 4-dimensional array we would combine 2 dimensions along each of the edges of the new matrix. An oft encountered example is trade flow register called the Inter Country Input-Output table (ICIO). ICIOs map all trade flows from countries and industries to each other. These tables are generally represented as matrices where each industry for each country is present along each edge, whereby rows represent outputs and columns represent inputs. In order to compute certain trade indicators for countries, it is often necessary to either extract the fat diagonal, or zero it out. Implementing these indicators for the `gvc` R package was the original impertus for implementing the procedures which became the `diagonals` package.

An example of a minitiature ICIO is this, which `CH` represent Switzerland and `RoW` represents the rest of the world, each have three industries named `1`, `2`, and `3`.

```
     CH1 CH2 CH3 RoW1 RoW2 RoW3
CH1    1   0   0    1    1    0
CH2    0   0   1    1    0    1
CH3    1   0   0    0    0    0
RoW1   1   1   0    1    1    1
RoW2   1   0   1    1    0    0
RoW3   1   1   0    1    1    1
```

These fat diagonal matrices can be thought of as a generalised version of a block diagonal matrix (see Rowland and Weisstein 2007). However, although it is common for the combinations of dimensions along the edges to be identicatal, and hence the `steps` square, this is not necessary.

3. Design

The implementation of fat diagonals in the `diagonals` package is intended to be as close as possible to the functions dealing with diagonals included in the `base` package. As such, the package includes two functions.

- `fatdiag()`
- `fatdiag()<-`

These functions offer a very similar syntax to the base functions:

- `diag()`
- `diag()<-`

With the exception that the fat diagonal functions generally need more information, in terms of the number of `steps` on the diagonal ladder, or the `size` of these steps.

The function `fatdiag<-` like its base package equivalent, replaces the (fat) diagonal of its first argument `x` with the right side argument `value`. The `value` argument can either be a scalar, in which case it is recycled for the length of the diagonal, or it can be vector. For the base package function `diag()<-` this vector has be of the same length as the diagonal (here, the shortest dimension of the matrix), however, the `fatdiag()<-` function will accept any vector that is of a length that is an integer divisor of the length of the diagonal. For example, if the length of the diagonal is `12`, then the follow lengths for the replacement vector are accepted: `1`, `2`, `3`, `4`, and `6`.

The `fatdiag` function act similar to the `diag()` function. Both these functions have two main applications . The first application is (fat) diagonal extraction, is the first argument `x` is a matrix, i.e. `length(dim(x)) == 2`, then the function extracts the diagonal matrix and returns it as a vector.

The second application is (fat) diagonal matrix creation. This can be done in two ways, using a scalar, or using a vector. If a scalar is used for `x`, the `diag()` function returns an identity matrix `Ix`, i.e. a matrix of dimensions `x` times `x` is returned, with `1` on the diagonal positions and `0` elsewhere. The `fatdiag()` function supports the creation of non-square matrices (e.g. using `size = c(3,2)`) and therefore uses `x` as the longest dimension of the matrix, where the other dimension is determined automatically using the `size` argument.

4. Usage

In the introduction we briefly demonstrate the usage of the `fatdiag()` function for assigning a new `value` to the fat diagonal. Here we take a closer look at some of the additional functionality that is implemented.

```
fatdiag(m, size=c(1,2) ) <- 881:888
m
```

```
   Ai  Ao  Bi  Bo  Ci  Co  Di  Do
A 881 882   0   0   1   0   1   0
B   0   1 883 884   0   1   0   1
C   1   0   0   0 885 886   0   0
D   1   0   1   0   1   1 887 888
```

So far we have been using the set `fatdiag()`, i.e. `fatdiag()<-`. However, we can also use the `fatdiag()` function either for diagonal extraction, or diagonal matrix creation.

```
fatdiag(m, steps = 4)
```

```
[1] 881 882 883 884 885 886 887 888
```

Fat diagonal matrices can be created using a scalar:

```
fatdiag(6, steps=3)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    0    0    0    0
[2,]    1    1    0    0    0    0
[3,]    0    0    1    1    0    0
[4,]    0    0    1    1    0    0
[5,]    0    0    0    0    1    1
[6,]    0    0    0    0    1    1
```

or using a vector:

```
fatdiag(1:12, steps=3)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    0    0    0    0
```

```
[2,]    2    4    0    0    0    0
[3,]    0    0    5    7    0    0
[4,]    0    0    6    8    0    0
[5,]    0    0    0    0    9   11
[6,]    0    0    0    0   10   12
```

We can extract a fat diagonal and diagonalise it again.

```
m <- matrix(631:666, nrow=6, ncol=6)
(extr_fat_diag <- fatdiag(m, steps=3))
```

```
 [1] 631 632 637 638 645 646 651 652 659 660 665 666
```

```
fatdiag(extr_fat_diag , steps=3)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  631  637    0    0    0    0
[2,]  632  638    0    0    0    0
[3,]    0    0  645  651    0    0
[4,]    0    0  646  652    0    0
[5,]    0    0    0    0  659  665
[6,]    0    0    0    0  660  666
```

Note that the above code combines the two different ways in which the `fatdiag()` function can be used, the first iteration extracts the fat diagonal from the matrix `m` and returns it as a vector `extr_fat_diag`, the second iteration takes the vector returned by the first iteration and diagonalises it in a matrix, which is returned.

5. Conclusion

Higher-order arrays can sometimes be mapped to a matrix, which enables us to visualise these arrays in a intuitive manner. However, the standard matrix manipulations relating to diagonals become more complex when we do so. The **diagonals** package provides the `fatdiag()` function family, which enables the manipulation of fat diagonals in R, using a syntax that is very close to the `diag()` function family from Rs **base** package.

References

Leontief, Wassily W. 1936. "Quantitative Input and Output Relations in the Economic Systems of the United States." *The Review of Economic Statistics*. JSTOR, 105–25.

Rowland, Todd, and Eric Weisstein. 2007. "Block Diagonal Matrix." MathWorld. http://mathworld.wolfram.com/BlockDiagonalMatrix.html.

Wickham, Hadley. 2014. "Tidy Data." *The Journal of Statistical Software* 59 (10). http://www.jstatsoft.org/v59/i10/.

Affiliation:

Bastiaan Quast
The Graduate Institute, Geneva
Maison de la paix Geneva, Switzerland
E-mail: bquast@gmail.com
URL: http://qua.st/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd