Design Issues in Matrix package Development

Martin Maechler and Douglas Bates
R Core Development Team
maechler@stat.math.ethz.ch, bates@r-project.org

Spring 2008 (typeset on March 13, 2014)

Abstract

This is a (**currently very incomplete**) write-up of the many smaller and larger design decisions we have made in organizing functionalities in the Matrix package.

Classes: There's a rich hierarchy of matrix classes, which you can visualize as a set of trees whose inner (and "upper") nodes are *virtual* classes and only the leaves are non-virtual "actual" classes.

Functions and Methods:
- setAs()
- others

1 The Matrix class structures

Take Martin's DSC 2007 talk to depict class hierarchy.
— — —

1.1 Diagonal Matrices

The class of diagonal matrices is worth mentioning for several reasons. First, we have wanted such a class, because *multiplication* methods are particularly simple with diagonal matrices. The typical constructor is `Diagonal()` whereas the accessor (as for traditional matrices), `diag()` simply returns the *vector* of diagonal entries:

```
> library(Matrix)
> (D4 <- Diagonal(4, 10*(1:4)))

4 x 4 diagonal matrix of class "ddiMatrix"
     [,1] [,2] [,3] [,4]
[1,]   10    .    .    .
[2,]    .   20    .    .
[3,]    .    .   30    .
[4,]    .    .    .   40
```

```
> str(D4)

Formal class 'ddiMatrix' [package "Matrix"] with 4 slots
  ..@ diag    : chr "N"
  ..@ Dim     : int [1:2] 4 4
  ..@ Dimnames:List of 2
  .. ..$ : NULL
  .. ..$ : NULL
  ..@ x       : num [1:4] 10 20 30 40

> diag(D4)

[1] 10 20 30 40
```

We can *modify* the diagonal in the traditional way (via method definition for `diag<-()`):

```
> diag(D4) <- diag(D4) + 1:4
> D4

4 x 4 diagonal matrix of class "ddiMatrix"
     [,1] [,2] [,3] [,4]
[1,]   11    .    .    .
[2,]    .   22    .    .
[3,]    .    .   33    .
[4,]    .    .    .   44
```

Note that **unit-diagonal** matrices (the identity matrices of linear algebra) with slot `diag = "U"` can have an empty `x` slot, very analogously to the unit-diagonal triangular matrices:

```
> str(I3 <- Diagonal(3)) ## empty 'x' slot

Formal class 'ddiMatrix' [package "Matrix"] with 4 slots
  ..@ diag    : chr "U"
  ..@ Dim     : int [1:2] 3 3
  ..@ Dimnames:List of 2
  .. ..$ : NULL
  .. ..$ : NULL
  ..@ x       : num(0)

> getClass("diagonalMatrix") ## extending "denseMatrix"

Virtual Class "diagonalMatrix" [package "Matrix"]

Slots:

Name:       diag       Dim  Dimnames
```

```
Class: character   integer      list

Extends:
Class "sparseMatrix", directly
Class "Matrix", by class "sparseMatrix", distance 2
Class "mMatrix", by class "Matrix", distance 3

Known Subclasses: "ddiMatrix", "ldiMatrix"
```

We have implemented diagonal matrices as *dense* rather than sparse matrices, for the following reasons:

1. The `diag()`onal (vector) is the basic part of such a matrix, and this is simply the `x` slot unless the `diag` slot is `"U"`, the unit-diagonal case, which is the identity matrix.

2. given '1)', it does not make much sense to store *where* the matrix is non-zero. This contrasts with all sparse matrix implementations, and hence we did not want to call a diagonal matrix "sparse".

2 Matrix Transformations

2.1 Coercions between Matrix classes

You may need to transform Matrix objects into specific shape (triangular, symmetric), content type (double, logical, ...) or storage structure (dense or sparse). Every useR should use `as(x, <superclass>)` to this end, where `<superclass>` is a *virtual* Matrix super class, such as `"triangularMatrix"` `"dMatrix"`, or `"sparseMatrix"`.

In other words, the user should *not* coerce directly to a specific desired class such as `"dtCMatrix"`, even though that may occasionally work as well.

Here is a set of rules to which the Matrix developers and the users should typically adhere:

Rule 1 : `as(M, "matrix")` should work for **all** Matrix objects `M`.

Rule 2 : `Matrix(x)` should also work for matrix like objects `x` and always return a "classed" Matrix.

Applied to a `"matrix"` object `m`, `M. <- Matrix(m)` can be considered a kind of inverse of `m <- as(M, "matrix")`. For sparse matrices however, `M.` well be a `CsparseMatrix`, and it is often "more structured" than `M`, e.g.,

```
> (M <- spMatrix(4,4, i=1:4, j=c(3:1,4), x=c(4,1,4,8))) # dgTMatrix

4 x 4 sparse Matrix of class "dgTMatrix"

[1,] . . 4 .
```

```
[2,] . 1 . .
[3,] 4 . . .
[4,] . . . 8

> m <- as(M, "matrix")
> (M. <- Matrix(m)) # dsCMatrix (i.e. *symmetric*)

4 x 4 sparse Matrix of class "dsCMatrix"

[1,] . . 4 .
[2,] . 1 . .
[3,] 4 . . .
[4,] . . . 8
```

Rule 3 : All the following coercions to *virtual* matrix classes should work:

1. `as(m, "dMatrix")`
2. `as(m, "lMatrix")`
3. `as(m, "nMatrix")`
4. `as(m, "denseMatrix")`
5. `as(m, "sparseMatrix")`
6. `as(m, "generalMatrix")`

whereas the next ones should work under some assumptions:

1. `as(m1, "triangularMatrix")`
   should work when `m1` is a triangular matrix, i.e. the upper or lower triangle of `m1` contains only zeros.
2. `as(m2, "symmetricMatrix")` should work when `m2` is a symmetric matrix in the sense of `isSymmetric(m2)` returning `TRUE`. Note that this is typically equivalent to something like `isTRUE(all.equal(m2, t(m2)))`, i.e., the lower and upper triangle of the matrix have to be equal *up to small numeric fuzz*.

3 Session Info

```
> toLatex(sessionInfo())
```

- R version 2.15.3 (2013-03-01), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=de_CH.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=de_CH.UTF-8`, `LC_PAPER=C`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=de_CH.UTF-8`, `LC_IDENTIFICATION=C`

- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils
- Other packages: Matrix 1.1-3
- Loaded via a namespace (and not attached): grid 2.15.3, lattice 0.20-27