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; Aug 2022 (typeset on August 11, 2023)

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 the Matrix class hierarchy; available from `https://stat.ethz.ch/~maechler/R/DSC-2007_MatrixClassHierarchies.pdf`
.

— — —

From far, there are **three** separate class hierarchies, and every **Matrix** package matrix has an actual (or "factual") class inside these three hierarchies: More formally, we have three (3) main "class classifications" for our Matrices, i.e., three "orthogonal" partitions of "Matrix space", and every Matrix object's class corresponds to an *intersection* of these three partitions; i.e., in R's S4 class system: We have three independent inheritance schemes for every Matrix, and each such Matrix class is simply defined to `contain` three *virtual* classes (one from each partitioning scheme), e.g,

The three partioning schemes are

1. Content `type`: Classes `dMatrix`, `lMatrix`, `nMatrix`, (`iMatrix`, `zMatrix`) for entries of type **d**ouble, **l**ogical, patter**n** (and not yet **i**nteger and complex) Matrices.

   `nMatrix` only stores the *location* of non-zero matrix entries (where as logical Matrices can also have `NA` entries!)

2. structure: general, triangular, symmetric, diagonal Matrices
3. sparsity: `denseMatrix`, `sparseMatrix`

For example in the most used sparseMatrix class, `"dgCMatrix"`, the three initial letters `dgC` each codes for one of the three hierarchies:

d: **d**ouble

g: **g**eneral

C: **C**sparseMatrix, where **C** is for **C**olumn-compressed.

Part of this is visible from printing `getClass("<classname>")`:

```
> getClass("dgCMatrix")

Class "dgCMatrix" [package "Matrix"]

Slots:

Name:         i        p      Dim Dimnames        x  factors
Class:  integer  integer  integer     list  numeric     list

Extends:
Class "CsparseMatrix", directly
Class "dsparseMatrix", directly
Class "generalMatrix", directly
Class "dCsparseMatrix", directly
Class "dMatrix", by class "dsparseMatrix", distance 2
Class "sparseMatrix", by class "dsparseMatrix", distance 2
Class "compMatrix", by class "generalMatrix", distance 2
Class "Matrix", by class "CsparseMatrix", distance 3
Class "xMatrix", by class "dMatrix", distance 3
Class "mMatrix", by class "Matrix", distance 4
Class "replValueSp", by class "Matrix", distance 4
```

Another example is the `"nsTMatrix"` class, where `nsT` stands for

n: **n** is for "patter**n**", boolean content where only the *locations* of the non-zeros need to be stored.

t: **t**riangular matrix; either **U**pper, or **L**ower.

T: **T**sparseMatrix, where **T** is for **T**riplet, the simplest but least efficient way to store a sparse matrix.

From R itself, via `getClass(.)`:

```
> getClass("ntTMatrix")
```

```
Class "ntTMatrix" [package "Matrix"]

Slots:

Name:           i         j       Dim  Dimnames      uplo      diag
Class:    integer   integer   integer      list character character

Extends:
Class "TsparseMatrix", directly
Class "nsparseMatrix", directly
Class "triangularMatrix", directly
Class "nMatrix", by class "nsparseMatrix", distance 2
Class "sparseMatrix", by class "nsparseMatrix", distance 2
Class "Matrix", by class "triangularMatrix", distance 2
Class "mMatrix", by class "Matrix", distance 4
Class "replValueSp", by class "Matrix", distance 4
```

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:

```
> (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 "sparseMatrix"

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
Class "replValueSp", by class "Matrix", distance 3

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

Originally, we had implemented diagonal matrices as *dense* rather than sparse matrices. After several years it became clear that this had not been helpful really both from a user and programmer point of view. So now, indeed the `"diagonalMatrix"` class does also extend `"sparseMatrix"`, i.e., is a subclass of it. However, we do *not* store explicitly where the non-zero entries are, and the class does *not* extend any of the typical sparse matrix classes, `"CsparseMatrix"`, `"TsparseMatrix"`, or `"RsparseMatrix"`. Rather, 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.

Further note, e.g., from the `?Diagonal` help page, that we provide (low level) utility function `.sparseDiagonal()` with wrappers `.symDiagonal()` and `.trDiagonal()` which will provide diagonal matrices inheriting from `"CsparseMatrix"` which may be advantageous in *some cases*, but less efficient in others, see the help page.

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 4.3.1 Patched (2023-08-09 r84931), `x86_64-pc-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=de_CH.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=de_CH.UTF-8`, `LC_IDENTIFICATION=C`
- Time zone: `Europe/Zurich`
- TZcode source: `system (glibc)`
- Running under: `Fedora Linux 36 (Thirty Six)`
- Matrix products: default
- BLAS: `/u/maechler/R/D/r-patched/F36-64-inst/lib/libRblas.so`

- LAPACK: `/usr/lib64/liblapack.so.3.10.1`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: Matrix 1.6-1
- Loaded via a namespace (and not attached): compiler 4.3.1, grid 4.3.1, lattice 0.21-8, tools 4.3.1