2nd Introduction to the Matrix package

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

September 2006 (typeset on July 3, 2015)

Abstract

Linear algebra is at the core of many areas of statistical computing and from its inception the S language has supported numerical linear algebra via a matrix data type and several functions and operators, such as `%*%`, `qr`, `chol`, and `solve`. However, these data types and functions do not provide direct access to all of the facilities for efficient manipulation of dense matrices, as provided by the Lapack subroutines, and they do not provide for manipulation of sparse matrices.

The **Matrix** package provides a set of S4 classes for dense and sparse matrices that extend the basic matrix data type. Methods for a wide variety of functions and operators applied to objects from these classes provide efficient access to BLAS (Basic Linear Algebra Subroutines), Lapack (dense matrix), CHOLMOD including AMD and COLAMD and `Csparse` (sparse matrix) routines. One notable characteristic of the package is that whenever a matrix is factored, the factorization is stored as part of the original matrix so that further operations on the matrix can reuse this factorization.

1 Introduction

The most automatic way to use the **Matrix** package is via the `Matrix()` function which is very similar to the standard R function `matrix()`,

```
> library(Matrix)
> M <- Matrix(10 + 1:28, 4, 7)
> M

4 x 7 Matrix of class "dgeMatrix"
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   11   15   19   23   27   31   35
[2,]   12   16   20   24   28   32   36
[3,]   13   17   21   25   29   33   37
[4,]   14   18   22   26   30   34   38

> tM <- t(M)
```

Such a matrix can be appended to (using `cBind()` or `rBind()` with capital "B") or indexed,

```
> (M2 <- cBind(-1, M))

4 x 8 Matrix of class "dgeMatrix"
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]   -1   11   15   19   23   27   31   35
[2,]   -1   12   16   20   24   28   32   36
[3,]   -1   13   17   21   25   29   33   37
[4,]   -1   14   18   22   26   30   34   38
```

```
> M[2, 1]

[1] 12

> M[4, ]

[1] 14 18 22 26 30 34 38
```

where the last two statements show customary matrix indexing, returning a simple numeric vector each[1]. We assign 0 to some columns and rows to "sparsify" it, and some NAs (typically "missing values" in data analysis) in order to demonstrate how they are dealt with; note how we can *"subassign"* as usual, for classical R matrices (i.e., single entries or whole slices at once),

```
> M2[, c(2,4:6)] <- 0
> M2[2, ] <- 0
> M2 <- rBind(0, M2, 0)
> M2[1:2,2] <- M2[3,4:5] <- NA
```

and then coerce it to a sparse matrix,

```
> sM <- as(M2, "sparseMatrix")
> 10 * sM

6 x 8 sparse Matrix of class "dgCMatrix"

[1,]   . NA   .  .  . .   .   .
[2,] -10 NA 150  .  . . 310 350
[3,]   .  .   . NA NA .   .   .
[4,] -10  . 170  .  . . 330 370
[5,] -10  . 180  .  . . 340 380
[6,]   .  .   .  .  . .   .   .

> identical(sM * 2, sM + sM)

[1] TRUE

> is(sM / 10  +  M2 %/% 2, "sparseMatrix")

[1] TRUE
```

where the last three calls show that multiplication by a scalar keeps sparcity, as does other arithmetic, but addition to a "dense" object does not, as you might have expected after some thought about "sensible" behavior:

```
> sM + 10

6 x 8 Matrix of class "dgeMatrix"
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]   10   NA   10   10   10   10   10   10
[2,]    9   NA   25   10   10   10   41   45
[3,]   10   10   10   NA   NA   10   10   10
[4,]    9   10   27   10   10   10   43   47
[5,]    9   10   28   10   10   10   44   48
[6,]   10   10   10   10   10   10   10   10
```

[1] because there's an additional default argument to indexing, `drop = TRUE`. If you add " `, drop = FALSE` " you will get submatrices instead of simple vectors.

Operations on our classed matrices include (componentwise) arithmetic (+, −, ∗, /, etc) as partly seen above, comparison (>, ≤, etc), e.g.,

```
> Mg2 <- (sM > 2)
> Mg2
6 x 8 sparse Matrix of class "lgCMatrix"

[1,] . N . . . . . .
[2,] : N | . . . | |
[3,] . . . N N . . .
[4,] : . | . . . | |
[5,] : . | . . . | |
[6,] . . . . . . . .
```

returning a logical sparse matrix. When interested in the internal **str**ucture, `str()` comes handy, and we have been using it ourselves more regulary than `print()`ing (or `show()`ing as it happens) our matrices; alternatively, `summary()` gives output similar to Matlab's printing of sparse matrices.

```
> str(Mg2)
Formal class 'lgCMatrix' [package "Matrix"] with 6 slots
  ..@ i       : int [1:16] 1 3 4 0 1 1 3 4 2 2 ...
  ..@ p       : int [1:9] 0 3 5 8 9 10 10 13 16
  ..@ Dim     : int [1:2] 6 8
  ..@ Dimnames:List of 2
  .. ..$ : NULL
  .. ..$ : NULL
  ..@ x       : logi [1:16] FALSE FALSE FALSE NA NA TRUE ...
  ..@ factors : list()
```

```
> summary(Mg2)
6 x 8 sparse Matrix of class "lgCMatrix", with 16 entries
   i j     x
1  2 1 FALSE
2  4 1 FALSE
3  5 1 FALSE
4  1 2    NA
5  2 2    NA
6  2 3  TRUE
7  4 3  TRUE
8  5 3  TRUE
9  3 4    NA
10 3 5    NA
11 2 7  TRUE
12 4 7  TRUE
13 5 7  TRUE
14 2 8  TRUE
15 4 8  TRUE
16 5 8  TRUE
```

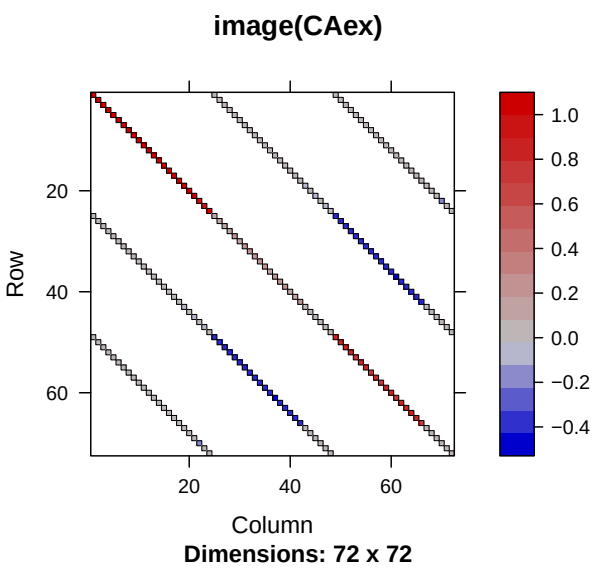
As you see from both of these, `Mg2` contains "extra zero" (here `FALSE`) entries; such sparse matrices may be created for different reasons, and you can use `drop0()` to remove ("drop") these extra zeros. This should *never* matter for functionality, and does not even show differently for logical sparse matrices, but the internal structure is more compact:

```
> Mg2 <- drop0(Mg2)
> str(Mg2@x) # length 13, was 16

 logi [1:13] NA NA TRUE TRUE TRUE NA ...
```

For large sparse matrices, visualization (of the sparsity pattern) is important, and we provide `image()` methods for that, e.g.,

```
> data(CAex)
> print(image(CAex, main = "image(CAex)")) # print(.) needed for Sweave
```



Further, i.e., in addition to the above implicitly mentioned `"Ops"` operators (`+`, `*`,. . . , `<=`,`>`,. . . , `&` which all work with our matrices, notably in conjunction with scalars and traditional matrices), the `"Math"`-operations (such as `exp()`, `sin()` or `gamma()`) and `"Math2"` (`round()` etc) and the `"Summary"` group of functions, `min()`, `range()`, `sum()`, all work on our matrices as they should. Note that all these are implemented via so called *group methods*, see e.g., `?Arith` in R. The intention is that sparse matrices remain sparse whenever sensible, given the matrix *classes* and operators involved, but not content specifically. E.g., <sparse> + <dense> gives <dense> even for the rare cases where it would be advantageous to get a <sparse> result.

These classed matrices can be "indexed" (more technically "subset") as traditional S language (and hence R) matrices, as partly seen above. This also includes the idiom `M [ M` ⟨*op*⟩ ⟨num⟩ `]` which returns simple vectors,

```
> sM[sM > 2]

 [1] NA NA 15 17 18 NA NA 31 33 34 35 37 38

> sml <- sM[sM <= 2]
> sml

 [1]  0 -1  0 -1 -1  0 NA NA  0  0  0  0  0  0  0  0  0 NA  0  0  0  0  0
[24] NA  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

and *"subassign"*ment similarly works in the same generality as for traditional S language matrices.

1.1 Matrix package for numerical linear algebra

Linear algebra is at the core of many statistical computing techniques and, from its inception, the S language has supported numerical linear algebra via a matrix data type and several functions and operators, such as `%*%`, `qr`, `chol`, and `solve`. Initially the numerical linear algebra functions in R called underlying Fortran routines from the Linpack (Dongarra et al., 1979) and Eispack (Smith et al., 1976) libraries but over the years most of these functions have been switched to use routines from the Lapack (Anderson et al., 1999) library which is the state-of-the-art implementation of numerical dense linear algebra. Furthermore, R can be configured to use accelerated BLAS (Basic Linear Algebra Subroutines), such as those from the Atlas (Whaley et al., 2001) project or other ones, see the R manual "Installation and Administration".

Lapack provides routines for operating on several special forms of matrices, such as triangular matrices and symmetric matrices. Furthermore, matrix decompositions like the QR decompositions produce multiple output components that should be regarded as parts of a single object. There is some support in R for operations on special forms of matrices (e.g. the `backsolve`, `forwardsolve` and `chol2inv` functions) and for special structures (e.g. a QR structure is implicitly defined as a list by the `qr`, `qr.qy`, `qr.qty`, and related functions) but it is not as fully developed as it could be.

Also there is no direct support for sparse matrices in R although Koenker and Ng (2003) have developed the **SparseM** package for sparse matrices based on SparseKit.

The **Matrix** package provides S4 classes and methods for dense and sparse matrices. The methods for dense matrices use Lapack and BLAS. The sparse matrix methods use CHOLMOD (Davis, 2005a), CSparse (Davis, 2005b) and other parts (AMD, COLAMD) of Tim Davis' "SuiteSparse" collection of sparse matrix libraries, many of which also use BLAS.

TODO: `triu()`, `tril()`, `diag()`, ... *and* `as(.,.)` , *but of course only when they've seen a few different ones.*

TODO: *matrix operators include* `%*%`, `crossprod()`, `tcrossprod()`, `solve()`

TODO: `expm()` *is the matrix exponential ... ...*

TODO: `symmpart()` *and* `skewpart()` *compute the symmetric part,* `(x + t(x))/2` *and the skew-symmetric part,* `(x - t(x))/2` *of a matrix* `x`.

TODO: *factorizations include* `Cholesky()` *(or* `chol()`*),* `lu()`, `qr()` *(not yet for dense)*

TODO: *Although generally the result of an operation on dense matrices is a dgeMatrix, certain operations return matrices of special types.*

TODO: *E.g. show the distinction between* `t(mm) %*% mm` *and* `crossprod(mm)`.

2 Matrix Classes

The **Matrix** package provides classes for real (stored as double precision), logical and so-called "pattern" (binary) dense and sparse matrices. There are provisions to also provide integer and complex (stored as double precision complex) matrices.

Note that in R, `logical` means entries `TRUE`, `FALSE`, or `NA`. To store just the non-zero pattern for typical sparse matrix algorithms, the pattern matrices are *binary*, i.e., conceptually just `TRUE` or `FALSE`. In **Matrix**, the pattern matrices all have class names starting with `"n"` (patter**n**).

2.1 Classes for dense matrices

For the sake of brevity, we restrict ourselves to the *real* (**d**ouble) classes, but they are paralleled by **l**ogical and patter**n** matrices for all but the positive definite ones.

dgeMatrix Real matrices in general storage mode

dsyMatrix Symmetric real matrices in non-packed storage

dspMatrix Symmetric real matrices in packed storage (one triangle only)

dtrMatrix Triangular real matrices in non-packed storage

dtpMatrix Triangular real matrices in packed storage (triangle only)

dpoMatrix Positive semi-definite symmetric real matrices in non-packed storage

dppMatrix ditto in packed storage

Methods for these classes include coercion between these classes, when appropriate, and coercion to the `matrix` class; methods for matrix multiplication (`%*%`); cross products (`crossprod`), matrix norm (`norm`); reciprocal condition number (`rcond`); LU factorization (`lu`) or, for the `poMatrix` class, the Cholesky decomposition (`chol`); and solutions of linear systems of equations (`solve`).

Whenever a factorization or a decomposition is calculated it is preserved as a (list) element in the `factors` slot of the original object. In this way a sequence of operations, such as determining the condition number of a matrix then solving a linear system based on the matrix, do not require multiple factorizations of the same matrix nor do they require the user to store the intermediate results.

2.2 Classes for sparse matrices

Used for large matrices in which most of the elements are known to be zero (or `FALSE` for logical and binary ("pattern") matrices).

Sparse matrices are automatically built from `Matrix()` whenever the majority of entries is zero (or `FALSE` respectively). Alternatively, `sparseMatrix()` builds sparse matrices from their non-zero entries and is typically recommended to construct large sparse matrices, rather than direct calls of `new()`.

TODO: *E.g. model matrices created from factors with a large number of levels*
TODO: *or from spline basis functions (e.g. COBS, package* **cobs***), etc.*
TODO: *Other uses include representations of graphs. indeed; good you mentioned it! particularly since we still have the interface to the* **graph** *package. I think I'd like to draw one graph in that article — maybe the undirected graph corresponding to a crossprod() result of dimension ca.* 50^2
TODO: *Specialized algorithms can give substantial savings in amount of storage used and execution time of operations.*
TODO: *Our implementation is based on the CHOLMOD and CSparse libraries by Tim Davis.*

2.3 Representations of sparse matrices

2.3.1 Triplet representation (`TsparseMatrix`)

Conceptually, the simplest representation of a sparse matrix is as a triplet of an integer vector `i` giving the row numbers, an integer vector `j` giving the column numbers, and a numeric vector `x` giving the non-zero values in the matrix.[2] In **Matrix**, the `TsparseMatrix` class is the virtual class of all sparse matrices in triplet representation. Its main use is for easy input or transfer to other classes.

As for the dense matrices, the class of the `x` slot may vary, and the subclasses may be triangular, symmetric or unspecified ("general"), such that the `TsparseMatrix` class has several[3] 'actual" subclasses, the most typical (numeric, general) is `dgTMatrix`:

```
> getClass("TsparseMatrix") # (i,j, Dim, Dimnames) slots are common to all

Virtual Class "TsparseMatrix" [package "Matrix"]

Slots:
```

[2] For efficiency reasons, we use "zero-based" indexing in the **Matrix** package, i.e., the row indices `i` are in `0:(nrow(.)-1)` and the column indices `j` accordingly.

[3] the 3×3 actual subclasses of `TsparseMatrix` are the three structural kinds, namely **t**riangular, **s**ymmetric and **g**eneral, times three entry classes, **d**ouble, **l**ogical, and patter**n**.

```
Name:         i        j      Dim Dimnames
Class:  integer  integer  integer     list

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

Known Subclasses: "dgTMatrix", "dtTMatrix", "dsTMatrix", "lgTMatrix", "ltTMatrix",
"lsTMatrix", "ngTMatrix", "ntTMatrix", "nsTMatrix"

> getClass("dgTMatrix")

Class "dgTMatrix" [package "Matrix"]

Slots:

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

Extends:
Class "TsparseMatrix", directly
Class "dsparseMatrix", directly
Class "generalMatrix", 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 "TsparseMatrix", distance 3
Class "mMatrix", by class "Matrix", distance 4
```

Note that the *order* of the entries in the `(i,j,x)` vectors does not matter; consequently, such matrices are not unique in their representation. [4]

2.3.2 Compressed representations: `CsparseMatrix` and `RsparseMatrix`

For most sparse operations we use the compressed column-oriented representation (virtual class `CsparseMatrix`) (also known as "csc", "compressed sparse column"). Here, instead of storing all column indices `j`, only the *start* index of every column is stored.

Analogously, there is also a compressed sparse row (csr) representation, which e.g. is used in in the **SparseM** package, and we provide the `RsparseMatrix` for compatibility and completeness purposes, in addition to basic coercion (`(as(., "`*cl*`")`) between the classes. These compressed representations remove the redundant row (column) indices and provide faster access to a given location in the matrix because you only need to check one row (column).

There are certain advantages [5] to csc in systems like R, Octave and Matlab where dense matrices are stored in column-major order, therefore it is used in sparse matrix libraries such as CHOLMOD or CSparse of which we make use. For this reason, the `CsparseMatrix` class and subclasses are the principal classes for sparse matrices in the **Matrix** package.

The Matrix package provides the following classes for sparse matrices

. . . FIXME *many mor — maybe ex plain namin scheme? . . .*

[4] Furthermore, there can be *repeated* `(i,j)` entries with the customary convention that the corresponding `x` entries are *added* to form the matrix element m_{ij}.

[5] routines can make use of high-level ("level-3") BLAS in certain sparse matrix computations

dgTMatrix general, numeric, sparse matrices in (a possibly redundant) triplet form. This can be a convenient form in which to construct sparse matrices.

dgCMatrix general, numeric, sparse matrices in the (sorted) compressed sparse column format.

dsCMatrix symmetric, real, sparse matrices in the (sorted) compressed sparse column format. Only the upper or the lower triangle is stored. Although there is provision for both forms, the lower triangle form works best with TAUCS.

dtCMatrix triangular, real, sparse matrices in the (sorted) compressed sparse column format.

TODO: *Can also read and write the Matrix Market and read the Harwell-Boeing representations.*
TODO: *Can convert from a dense matrix to a sparse matrix (or use the Matrix function) but going through an intermediate dense matrix may cause problems with the amount of memory required.*
TODO: *similar range of operations as for the dense matrix classes.*

3 More detailed examples of "Matrix" operations

Have seen `drop0()` above, showe a nice double example (where you see "." and "0").
Show the use of `dim<-` for *resizing* a (sparse) matrix.
Maybe mention `nearPD()`.
TODO: *Solve a sparse least squares problem and demonstrate memory / speed gain*
TODO: *mention* `lme4` *and* `lmer()`*, maybe use one example to show the matrix sizes.*

4 Notes about S4 classes and methods implementation

Maybe we could give some glimpses of implementations at least on the R level ones?
TODO: *The class hierarchy: a non-trivial tree where only the leaves are "actual" classes.*
TODO: *The main advantage of the multi-level hierarchy is that methods can often be defined on a higher (virtual class) level which ensures consistency [and saves from "cut & paste" and forgetting things]*
TODO: *Using Group Methods*

5 Session Info

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

- R version 3.2.1 Patched (2015-06-23 r68582), `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=de_CH.UTF-8`, `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, utils
- Other packages: Matrix 1.2-2
- Loaded via a namespace (and not attached): grid 3.2.1, lattice 0.20-31, tools 3.2.1

References

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, 3rd edition, 1999.

Tim Davis. CHOLMOD: sparse supernodal Cholesky factorization and update/downdate. http://www.cise.ufl.edu/research/sparse/cholmod, 2005a.

Tim Davis. CSparse: a concise sparse matrix package. http://www.cise.ufl.edu/research/sparse/CSparse, 2005b.

Jack Dongarra, Cleve Moler, Bunch, and G.W. Stewart. *Linpack Users' Guide*. SIAM, 1979.

Roger Koenker and Pin Ng. SparseM: A sparse matrix package for R. *J. of Statistical Software*, 8(6), 2003.

B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines. EISPACK Guide*, volume 6 of *Lecture Notes in Computer Science*. Springer-Verlag, New York, 1976.

R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).