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 March 14, 2008)

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 ... FIXME ...

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 we see above 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 thought about "sensible" behavior:

```
> sM + 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.

```
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
```

Operations on our classed matrices include (componentwise) arithmetic ($+$, $-$, $*$, $/$, etc) as partly seen above, comparison ($>$, $\leq$, 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 NA 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
```

Further, i.e., in addition to the above `"Ops"` operators, 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

These classed matrices can be "indexed" (more technically "subset") as normal ones, as partly seen above. This also includes the idiom `M [ M` ⟨*cop*⟩ ⟨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
```

. . . FIXME *2nd introduction – maybe keep some?* . . .

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`. 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 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)`.

... ... The following is the old 'Introduction.Rnw' ... FIXME ... ...

2 Classes for dense matrices

The `Matrix` package provides classes for real (stored as double precision) and logical dense (and sparse) matrices. There are provisions to also provide integer and complex (stored as double precision complex) matrices. The basic real classes are

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`).

Further, group methods have been defined for the `Arith` (basic arithmetic, including with scalar numbers) and the `Math` (basic mathematical functions) group..

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.

3 Classes for sparse matrices

Used for large matrices in which most of the elements are known to be zero.
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.*

3.1 Representations of sparse matrices

3.1.1 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.

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

3.1.2 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.

There are certain advantages to csc in systems like R 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.

The Matrix package provides the following classes for sparse matrices

. . . FIXME *many more — maybe explain naming scheme? . . .*

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 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.*

4 More detailed examples of "Matrix" operations

Mention `drop0()` showing a nice example
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.*

5 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*

6 Session Info

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

- R version 2.6.2 Patched (2008-03-13 r44752), `x86_64-unknown-linux-gnu`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils
- Other packages: Matrix 0.999375-6, lattice 0.17-6
- Loaded via a namespace (and not attached): grid 2.6.2

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.

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

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`).