Sparse Matrices in package Matrix and applications

Martin Maechler and Douglas Bates

Seminar für Statistik
ETH Zurich Switzerland

Department of Statistics
University of Madison, Wisconsin U.S.A.
(maechler|bates)@R-project.org (R-Core)

useR! 2009, Rennes
July 10, 2009

Outline

Introduction to Matrix and Sparse Matrices
  Sparse Matrices in package Matrix
  Matrix: Goals
  3D space of Matrix classes

Applications in Spatial Statistics
  Regression with Spatially Dependent Errors: SAR(1)

Application - Mixed Modelling (RE)ML in R

Who’s the best liked prof at ETH?

Introduction

▶ Matrix: the movie
▶ Matrix: the R package:
  ► Package Matrix: a recommended R package since R 2.9.0
  ► Infrastructure for other packages for several years, notably
    lme41
  ► CRAN nowadays lists direct “reverse dependencies”:

1lme4 := (Generalized–) (Non–) Linear Mixed Effect Modelling,
(using S4 | re-implemented from scratch the 4th time)

(reverse) Dependencies on Matrix

On June 26, 2008 (> one year ago), Matrix was not yet
recommended, and had the following CRAN dependency graph:

i.e., 14 + 1 directly dependent packages.
http://cran.r-project.org/web/packages/Matrix/

Matrix: Sparse and Dense Matrix Classes and Methods

Classes and methods for dense and sparse matrices and operations on them using Lapack and SuiteSparse.

Version: 0.999375-29
Priority: recommended
Depends: R (≥ 2.9.0), stats, methods, utils, lattice
Imports: graphics, lattice, grid, stats
Enhances: graph, SparseM
Author: Douglas Bates and Martin Maechler

Reverse dependencies:

- FAIR, FTICRMS, GOSim, MCMCglmm, Metabonomic, arm, arules, glmnet, klin,
- languageR, lme4, mlmRev, pedigreeMMM, qgen, ramps, spdep, surveyNG, svcm,
- systemfit, tpr, tsDyn
- arules, cba

Intro to Sparse Matrices in R package Matrix

- The R Package Matrix contains dozens of matrix classes and hundreds of method definitions.
- Has sub-hierarchies of denseMatrix and sparseMatrix.
- Very basic intro in some of sparse matrices:
The most obvious way to store a sparse matrix is the so called "Triplet" form; (virtual class \texttt{TsparseMatrix} in \texttt{Matrix}):

```r
> A <- spMatrix(10, 20, i = c(1,3:8),
                     + j = c(2,9,6:10),
                     + x = 7 * (1:7))
```

```
10 x 20 sparse Matrix of class "dgTMatrix"

[1,] .  7 . . . . . . . . . . . . . . . . . .
[2,] . . . . . . . . . . . . . . . . . . . .
[3,] . . . . . . . . 14 . . . . . . . . . .
[4,] . . . . . 21 . . . . . . . . . . . . .
[5,] . . . . . . 28 . . . . . . . . . . . .
[6,] . . . . . . . 35 . . . . . . . . . . .
[7,] . . . . . . . . 42 . . . . . . . . . .
[8,] . . . . . . . . . 49 . . . . . . . . . .
[9,] . . . . . . . . . . . . . . . . . . . .
[10,] . . . . . . . . . . . . . . . . . . . .
```

Less didactical, slightly more recommended:

```r
> A1 <- sparseMatrix(.....)
```

```r
> str(A) # note that *internally* 0-based indices (i,j) are used
```

```
Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
..@ i : int [1:7] 0 2 3 4 5 6 7
..@ j : int [1:7] 1 8 5 6 7 8 9
..@ Dim : int [1:2] 10 20
..@ Dimnames:List of 2
.. ..$ : NULL
.. ..$ : NULL
..@ x : num [1:7] 7 30 60 90 14 30 60 90 21 30 ...
..@ factors : list()
```

```r
> A[2:7, 12:20] <- rep(c(0,0,0,(3:1)*30,0), length = 6*9)
> A >= 20 # -> logical sparse; nice show() method
```

```
10 x 20 sparse Matrix of class "lgTMatrix"

[1,] . . . . . . . . . . . . . . . . . . . .
[2,] . . . . . . . . . . . . . | | | . . . .
[3,] . . . . . . . . . . . . . . | | | . . .
[4,] . . . . . | . . . . . . . . . | | | . .
[5,] . . . . . . | . . . | . . . | | | . |
[6,] . . . . . . . | . . | | . . . | | |
[7,] . . . . . . . . | . . . . . . . . . .
[8,] . . . . . . . . . | . . . . . . . . . .
[9,] . . . . . . . . . . . . . . . . . . . .
[10,] . . . . . . . . . . . . . . . . . . . .
```

The "column compressed" sparse representation:

```r
> Ac <- as(t(A), "CsparseMatrix")
```

```r
> str(Ac)
```

```
Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
..@ i : int [1:30] 1 13 14 15 8 14 15 16 5 15 ...
..@ p : int [1:11] 0 1 2 13 14 15 16 5 15 ...
..@ Dim : int [1:2] 20 10
..@ Dimnames:List of 2
.. ..$ : NULL
.. ..$ : NULL
..@ x : num [1:30] 7 30 60 90 14 30 60 90 21 30 ...
..@ factors : list()
```

Column index slot \texttt{j} replaced by a column \texttt{pointer} slot \texttt{p}.
R Package Matrix: Compelling reasons for S4

1. Classes for Matrices: well-defined inheritance hierarchies:
   1.1 Content kind: Classes dMatrix, lMatrix, nMatrix, (iMatrix, zMatrix) for contents of double, logical, pattern (and not yet integer and complex) Matrices, where nMatrix only stores the location of non-zero matrix entries (where as logical Matrices can also have NA entries)
   1.2 sparsity: denseMatrix, sparseMatrix
   1.3 structure: general, triangular, symmetric, diagonal Matrices

2. Inheritance: Visualisation via graphs

3. Multiple Inheritance (of classes)

4. Multiple Dispatch (of methods)

Goals of Matrix package

1. interface to LAPACK = state-of-the-art numerical linear algebra for dense matrices
   ▶ making use of special structure for symmetric or triangular matrices (e.g. when solving linear systems)
   ▶ setting and keep such properties allows more optimized code in these cases.

2. Sparse matrices for large designs: regression, mixed models, etc

3. . . . . . . [omitted in this talk]

Hence, quite a few different classes for matrices.

Multiple Dispatch in S4 .... for Matrix operations

Methods for "Matrix"-matrices: Often 2 matrices involved..

1. x %*% y
2. crossprod(x,y) — xᵀy
3. tcrossprod(x,y) — xyᵀ
4. x + y — "Arith" group methods
5. x <= y — "Compare" group methods

and many many more.

S4 >> S3

▶ S4 - multiple dispatch: Find method according to classes of both (or more) arguments.
▶ S3 - single dispatch: e.g., "ops.Matrix": only first argument counts.

many Matrix classes . . .

> library(Matrix)
> length(allCl <- getClasses("package:Matrix"))

[1] 98

> ## Those called "...Matrix"
> length(M.Cl <- grep("Matrix\$",allCl, value = TRUE))

[1] 70

i.e., many . . . , each inheriting from root class "Matrix"

> str(subs <- showExtends(getClassDef("Matrix")@subclasses, + printTo=FALSE))

List of 2

$ what: chr [1:76] "compMatrix" "triangularMatrix" "dMatrix" "iMatrix"
$ how: chr [1:76] "directly" "directly" "directly" "directly" "directly" ...

> ## even more... : All those above and these in addition:
> subs$what[ ! (subs$what %in% M.Cl)]

[1] "Cholesky" "pCholesky" "BunchKaufman" "pBunchKaufman"

. . . . . . a bit messy . . .
Logical organization of our Matrices: 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,

```r
setClass("dgCMatrix",
    contains= c("CsparseMatrix", "dsparseMatrix", "generalMatrix"),
    validity= function(..) .....
)
```

The three partitioning schemes are
1. **Content type**: Classes dMatrix, lMatrix, nMatrix, (iMatrix, zMatrix) for entries of type double, logical, pattern (and not yet integer 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

First two schemes: a slight generalization from LAPACK for dense matrices.

```r
str(M3cl <- grep("\^...Matrix\$",M.Cl, value = TRUE))
str(M3cl <- M3cl[M3cl != "corMatrix"] # corMatrix not desired in following)
```

Actual classes follow a "simple" terse naming convention:
```r
m <- grep("\^...Matrix\$",M.Cl, value = TRUE)
substr(m, 1, 3)
```
3D space of Matrix classes

dim1
dim2
dim3
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
●●●
d l n
general
symmetric
triangular
diagonal
dense
Csparse
Tsparse
Rsparse
ddi
dgC
dge
dgR
dgT
dpodpp
dsC
dsp
dsR
dsT
dsy
dtC
dtpdtr
dtR
dtT

Matrix 3d space: filled (2)

Matrix 3d space: filled (3)

Matrix 3d space: filled (4)
Spatially Dependent Errors — SAR(1)

Regression with spatially dependent errors; observations at locations \( i, i = 1, \ldots, n \) in the thousands, possibly 100,000s.

Simultaneous Autoregression

\[ y = X\beta + u \quad \text{where} \quad u = \lambda W u + \epsilon. \quad (1) \]

- \( W \): matrix \( (W_{ij}) \) of “distance-based contiguities” of locations \( i \) and \( j \) \( (W_{ii} \equiv 0) \).
- \( \lambda \): SAR(1) parameter; estimate via MLE, \( (\beta \) profiled out).
- \( u \sim N(0, \sigma^2(I - \lambda W)^{-1}) \)
- For log likelihood, need to compute determinant \( |I - \lambda W| \propto | -W + \frac{1}{\lambda} I| \) for many \( \lambda \)’s. \( (2) \)

Compute Cholesky / Determinant of \( A + \rho I \) for large sparse symmetric \( A \):

\[ \implies \text{Fast Cholesky Update} \]

SAR(1) – fast Likelihood from Cholesky Update

Data provided by Roger Bivand, as a relevant test case:

\[
\begin{align*}
\text{> } & \text{data(USCounties, package="Matrix")} \\
\text{> } & \text{dim(USCounties)} \\
\text{[1]} & \quad 3111 \quad 3111 \\
\text{> } & \text{(n <- ncol(USCounties))} \\
\text{[1]} & \quad 3111 \\
\text{> } & \text{IM <- .symDiagonal(n)} \\
\text{> } & \text{nWC <- -USCounties} \\
\text{> } & \text{set.seed(1)} \\
\text{> } & \text{rho <- sort(runif(50, 0, 1)) \# rho = 1 / lambda} \\
\text{and now compute determinant(A) := |A|} \\
\text{\quad |I - \lambda W| \propto | -W + \frac{1}{\lambda} I| \quad \text{for many } \lambda \text{'s.} \quad (2) }
\end{align*}
\]

SAR(1) – Cholesky Update – 2 –

\[
\begin{align*}
\text{> } & \text{## Determinant : Direct Computation} \\
\text{> } & \text{system.time(MJ <- sapply(rho, function(x)} \\
\text{+ \quad \text{determinant(IM - x * USCounties, logarithm = TRUE)$modulus})} \\
\text{> } & \text{user system elapsed} \\
\text{3.640} & \quad 0.124 \quad 4.062 \\
\text{> } & \text{## Determinant : "high-level" Update of the Cholesky \{Simplicial, Super\}} \\
\text{> } & \text{C1 <- Cholesky(nWC, Imult = 2)} \\
\text{> } & \text{system.time(MJ1 <- n * log(rho) +} \\
\text{+ \quad \text{sapply(rho, function(x) c(determinant(update(C1, nWC, 1/x))}$mod))} \\
\text{> } & \text{user system elapsed} \\
\text{0.692} & \quad 0.012 \quad 0.746 \\
\text{> } & \text{stopifnot(all.equal(MJ, MJ1))} \\
\text{> } & \text{C2 <- Cholesky(nWC, super = TRUE, Imult = 2) \# \quad "Supernodal\}} \\
\text{> } & \text{system.time(MJ2 <- n * log(rho) +} \\
\text{+ \quad \text{sapply(rho, function(x) c(determinant(update(C2, nWC, 1/x))}$mod))} \\
\text{> } & \text{user system elapsed} \\
\text{0.760} & \quad 0.060 \quad 0.888 \\
\text{> } & \text{stopifnot(all.equal(MJ, MJ2))} \\
\text{> } & \text{## Determinant : "low-level" Update of the Cholesky \{Simplicial, Super\}} \\
\text{> } & \text{system.time(MJ3 <- n*log(rho) + Matrix:::ldetL2up(C1, nWC,1/rho))} \\
\text{> } & \text{user system elapsed} \\
\text{0.400} & \quad 0.008 \quad 0.408 \\
\text{> } & \text{stopifnot(all.equal(MJ, MJ3))} \\
\text{> } & \text{system.time(MJ4 <- n*log(rho) + Matrix:::ldetL2up(C2, nWC,1/rho))} \\
\text{> } & \text{user system elapsed} \\
\text{0.404} & \quad 0.008 \quad 0.416 \\
\text{> } & \text{stopifnot(all.equal(MJ, MJ4))} \\
\text{Findings:} \\
\text{1. Using Cholesky update: order of magnitude faster} \\
\text{2. Simplicial \ (super= FALSE) } \leftrightarrow \text{ Supernodal \ (super= TRUE) : no big difference here} \\
\text{3. An even faster method for Det(Chol(.)) yields another 50% speed.} 
\end{align*}
\]
Mixed Modelling - (RE)ML Estimation in pure R

In (linear) mixed effects, the evaluation of the (RE) likelihood or equivalently deviance, needs repeated Cholesky decompositions of

$$U_\theta U_\theta^\top + I,$$

(3)

for many $\theta$ values (= the relative variance components) and (often very large), very sparse matrix $U_\theta$ where only the non-zeros of $U$ depend on $\theta$, i.e., the sparsity pattern is given (by the observational design).

Sophisticated (fill-reducing) Cholesky done in two phases:

1. “symbolic” decomposition: Determine the non-zero entries of $L (LL^\top = UU^\top + I)$,
2. numeric phase: compute these entries.

Phase 1: typically takes much longer; only needs to happen once.
Phase 2: “update the Cholesky Factorization”

Who’s the best liked prof at ETH?

- Private donation for encouraging excellent teaching at ETH
- Student union of ETH Zurich organizes survey to award prizes: Best lecturer — of ETH, and of each of the 14 departments.
- Smart Web-interface for survey: Each student sees the names of his/her professors from the last 4 semesters and all the lectures that applied.
- ratings in $\{1,2,3,4,5\}$.
- high response rate

Who’s the best prof — data

```r
> md <- within(read.csv("~/R/MM/Pkg-ex/lme4/puma-lmertest.csv"),
+ s <- factor(s) # Student_ID
+ d <- factor(d) # Lecturer_ID ("d"ozentIn)
+ dept <- factor(dept)
+ service <- factor(service)
+ studage <- ordered(studage)## *ordered* factors
+ lectage <- ordered(lectage) )
> str(md)
'data.frame': 73421 obs. of 7 variables:
$ s : Factor w/ 2972 levels "1","2","3","4",..: 1 1 1 1 2 2 3 3 3 3 ...
$ d : Factor w/ 1128 levels "1","6","7","8",..: 525 560 832 1068 ...
$ studage: Ord.factor w/ 4 levels "2"<"4"<"6"<"8": 1 1 1 1 1 1 1 1 1 1 ...
$ lectage: Ord.factor w/ 6 levels "1"<"2"<"3"<"4"<..: 2 1 2 2 1 1 1 1 ...
$ dept : Factor w/ 15 levels "1","2","3","4",..: 15 5 15 12 2 2 14 3 ...
$ y : int 5 2 5 3 2 4 4 5 5 4 ...
```

Modelling the ETH teacher ratings

Model: The rating depends on

- students (s) (rating subjectively)
- teacher (d) – main interest
- department (dept)
- “service” lecture or “own department student”, (service: 0/1).
- semester of student at time of rating (studage $\in \{2,4,6,8\}$).
- how many semesters back was the lecture (lectage).

Main question: **Who’s the best prof**

Hence, for “political” reasons, want $d$ as a **fixed** effect.
Model for ETH teacher ratings

Want d ("teacher_ID", ≈ 1000 levels) as fixed effect. Consequently, in

\[ y = X\beta + Zb + \epsilon \]

have \( X \) as \( n \times 1000 \) (roughly), \( Z \) as \( n \times 5000 \), \( n \approx 70'000 \).

\[ \text{> fm0 <- lmer2(y ~ d + dept*service + studage + lectage + (1|s), + data = md, sparseX = TRUE)} \]

\( \text{sparseX} = \text{TRUE}: \text{sparse} \ X \) (fixed effects) in addition to the indispensably sparse \( Z \) (random effects).

Unfortunately: Here, the above “sparseX - lmer” ends in

Error ... Cholmod error 'not positive definite' at file:../Cholesky/...

Good News: Newly in Matrix:

\[ \text{sparse.model.matrix()} \]

\[ \text{which \ lmer()} \text{ can use,} \]

\[ \text{or you can use for "truly sparse" least squares (i.e. no intermediately dense design matrix)} \]

\[ \text{something we plan to provide in Matrix 1.0-0.} \]

Summary

- Recommended R package "Matrix"
- Sparse Matrices: in increasing number of applications
- S4 classes and methods are the natural implementation tools
- lme4 is going to contain an alternative "pure R" version of ML and REML, you can pass to \text{nlminb()} \text{(or \text{optim() if you must :-). UseRs can easily extend these R functions to more flexible models or algorithms.}
- Matrix 1.0-0
  1. will happen
  2. will contain \text{sparse.model.matrix()}
  3. will contain truly sparse \text{lm(*, sparse=TRUE)}

That’s all folks — with thanks for your attention!