Sparse Model Matrices for Generalized Linear Models

Martin Maechler and Douglas Bates

(maechler|bates)@R-project.org (R-Core)

Seminar für Statistik
ETH Zurich Switzerland Department of Statistics
University of Madison, Wisconsin U.S.A.

useR! 2010, Gaithersburg
July 21, 2010
Outline

Sparse Matrices

Sparse Model Matrices

modelMatrix → General Linear Prediction Models

Mixed Modelling in R: lme4
Introduction

- Package Matrix: a recommended R package → part of every R.

- Infrastructure for other packages for several years, notably lme4

- Reverse depends (2010-07-18): ChainLadder, CollocInfer, EquiNorm, FAiR, FTICRMS, GLMMarp, GOSim, GrassmannOptim, HGLMMM, MCMCglmm, Metabonomic, amer, arm, arules, diffusionMap, expm, gamlss.util, gamm4, glmnet, klin, languageR, lme4, mclogit, mediation, mi, mlmRev, optimbase, pedigree, pedigreemm, phybase, qgen, ramps, recommenderlab, spdep, speedglm, sphet, surveillance, surveyNG, svcm, systemfit, tsDyn, Ringo

- Reverse suggests: another dozen . . .

\[^{1}\text{lme4} := (\text{Generalized–}) (\text{Non–}) \text{ Linear Mixed Effect Modelling, (using S4 | re-implemented from scratch the 4^{th} time)}\]
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.
- Quick intro in some of sparse matrices:
simple example — Triplet form

The most obvious way to store a sparse matrix is the so called "Triplet" form; (virtual class TsparseMatrix in Matrix):

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

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: A1 <- sparseMatrix(.....)
> 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 14 21 28 35 42 49
  ..@ factors : list()

> A[2:7, 12:20] <- rep(c(0,0,0,(3:1)*30,0), length = 6*9)

What to expect from a comparison on a sparse matrix?
> A >= 20
probably a logical sparse matrix . . .:
>` A >= 20         # -> logical sparse. Observe show() method:

10 x 20 sparse Matrix of class "lgTMatrix"

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

Note `:` a "non-structural" FALSE, logical analog of non-structural zeros printed as "0" as opposed to ".":

>` 1*(A >= 20)

```
[2,] . . . . . . . . . . . . . 1 1 1 1 . . .
[3,] . . . . . . . . 0 . . . . 1 1 1 1 . . .
[4,] . . . . . 1 . . . . . . . . . 1 1 1 1 . . .
[5,] . . . . . . 1 . . . . 1 . . . . 1 1 1 1 . . .
[6,] . . . . . . . 1 . . . 1 1 . . . . 1 1 1 . . .
[7,] . . . . . . . . 1 . . . . . . . . 1 1 1 1 . . .
[8,] . . . . . . . . . 1 . . . . . . . . . .
[9,] . . . . . . . . . . . . . . . . . . . .
[10,] . . . . . . . . . . . . . . . . . . . .
```

Note ":", a "non-structural" FALSE, logical analog of non-structural zeros printed as "0" as opposed to ".":
Triplet representation: easy for us humans, but can be both made smaller and more efficient for (column-access heavy) operations:
The “column compressed” sparse representation:

```r
> Ac <- as(t(A), "CsparseMatrix")
> 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 4 8 12 17 23 29 30 30 ...  
  ..@ 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 j
replaced by a column pointer slot p.
CHANGE since talk (July 21, 2010):

- model.Matrix(),
- its result classes,
- all subsequent modeling classes,
- glm4(), etc

have been “factored out” into (new) package MatrixModels.
(2010, End of July on R-forge; Aug. 6 on CRAN)
Sparse Model Matrices

New model matrix classes, generalizing R’s standard `model.matrix()`:

```r
> str(dd <- data.frame(a = gl(3,4), b = gl(4,1,12)))# balanced 2-way

'data.frame': 12 obs. of 2 variables:
  $ a: Factor w/ 3 levels "1","2","3": 1 1 1 1 2 2 2 2 3 3 ...
  $ b: Factor w/ 4 levels "1","2","3","4": 1 2 3 4 1 2 3 4 ...

> model.matrix(~ 0+ a + b, dd)

   a1 a2 a3 b2 b3 b4
1  1  0  0  0  0  0
2  1  0  0  1  0  0
3  1  0  0  0  1  0
4  1  0  0  0  0  1
5  0  1  0  0  0  0
6  0  1  0  1  0  0
7  0  1  0  0  1  0
8  0  1  0  0  0  1
9  0  0  1  0  0  0
10 0  0  1  1  0  0
11 0  0  1  0  1  0
12 0  0  1  0  0  1
```

attr( "assign" )
attr( "contrasts" )
```
Sparse Model Matrices

The model matrix above

- has many zeros, and
- ratio ((zeros) : (non-zeros)) increases dramatically with many-level factors
- even more zeros for factor interactions:

```r
> model.matrix(~ 0+ a * b, dd)

a1 a2 a3 b2 b3 b4 a2:b2 a3:b2 a2:b3 a3:b3 a2:b4 a3:b4
1  1  0  0  0  0  0   0   0   0   0   0   0
2  1  0  0  1  0  0   0   0   0   0   0   0
3  1  0  0  0  1  0   0   0   0   0   0   0
4  1  0  0  0  0  1   0   0   0   0   0   0
5  0  1  0  0  0  0   0   0   0   0   0   0
6  0  1  0  1  0  0   1   0   0   0   0   0
7  0  1  0  0  1  0   0   1   0   0   0   0
8  0  1  0  0  0  1   0   0   0   0   1   0
9  0  0  1  0  0  0   0   0   0   0   0   0
10 0  0  1  1  0  0   0   1   0   0   0   0
11 0  0  1  0  1  0   0   0   1   0   0   0
12 0  0  1  0  0  1   0   0   0   0   0   1
attr("assign")
[1] 1 1 1 2 2 2 3 3 3 3 3 3
```
Sparse Model Matrices in 'MatrixModels'

- These matrices can become very large: Both many rows (large \( n \)), and many columns, large \( p \).
- Eg., in Linear Mixed Effects Models,

\[
E(Y | \mathcal{B} = b) = X\beta + Zb,
\]

- the \( Z \) matrix is often large and very sparse, and in lme4 has always been stored as "sparseMatrix" ("dgCMatrix", specifically).
- Sometimes, \( X \), (fixed effect matrix) is large, too.
  → optionally also "sparseMatrix" in lme4\(^2\).

- We’ve extended `model.matrix()` to `model.Matrix()` in package MatrixModels with optional argument `sparse = TRUE`.

\(^2\)the development version of lme4, currently called lme4a.
Sparse Model Matrix Classes in 'MatrixModels'

```r
setClass("modelMatrix",
    representation(assign = "integer",
                   contrasts = "list", "VIRTUAL"),
    contains = "Matrix",
    validity = function(object) { ............ })

setClass("sparseModelMatrix", representation("VIRTUAL"),
    contains = c("CsparseMatrix", "modelMatrix"))
setClass("denseModelMatrix", representation("VIRTUAL"),
    contains = c("denseMatrix", "modelMatrix"))
```

## The ‘‘actual’’ *ModelMatrix classes:
```
setClass("dsparseModelMatrix",
    contains = c("dgCMatrix", "sparseModelMatrix"))
setClass("ddenseModelMatrix", contains =
    c("dgeMatrix", "ddenseMatrix", "denseModelMatrix"))
```

("ddenseMatrix": *not* for slots, but consistent superclass ordering)
model.Matrix(*, sparse=TRUE)

Constructing **sparse** models matrices (**MatrixModels** package):

> model.Matrix(~ 0+ a * b, dd, sparse=TRUE)

"dsparseModelMatrix": 12 x 12 sparse Matrix of class "dgCMatrix"

```plaintext
1 1 . . . . . . . . . . .
2 1 . 1 . . . . . . . . .
3 1 . . 1 . . . . . . . .
4 1 . . . 1 . . . . . . .
5 . 1 . . . . . . . . . .
6 . 1 . . 1 . . . 1 . . .
7 . 1 . . . 1 . . . . 1 .
8 . 1 . . . 1 . . . . . .
9 . . 1 . . . . . . . . .
10 . . 1 1 . . . 1 . . .
11 . . 1 1 . . . 1 . . .
12 . . 1 . . 1 . . . . . 1
@ assign: 1 1 1 2 2 2 3 3 3 3 3 3
@ contrasts:
$a
[1] "contr.treatment"

$b
```
Ideas: Very general setup for statistical models based on linear predictors

Class "glpModel" :- General Linear Prediction Models

```r
definitions:::
  setClass("Model", representation(call = "call", fitProps = "list" "VIRTUAL"))
  setClass("glpModel", representation(resp = "respModule", 
                                       pred = "predModule"),
           contains = "Model")
```

Two main ingredients:

1. Response module "respModule"
2. (Linear) Prediction module "predModule"
(1) Response Module

"respModule": Response modules for models with a linear predictor, which can include linear models (lm), generalized linear models (glm), nonlinear models (nls) and generalized nonlinear models (nglm):

```r
setClass("respModule",
    representation(mu = "numeric", # of length n
                         offset = "numeric", # of length n * s
                         sqrtXwt = "matrix", # of dim(.) == (n, s)
                         sqrtrwt = "numeric", # sqrt(residual weights)
                         weights = "numeric", # prior weights
                         wtres = "numeric",
                         y = "numeric"),
    validity = function(object) { ....... })

setClass("glmRespMod",
    representation(family = "family",
                         eta = "numeric",
                         n = "numeric"), # for evaluation of the
    contains = "respModule", validity=function(object) { .... })

setClass("nlsRespMod",
    representation(nlenv = "environment", .....), .......

setClass("nglmRespMod", contains = c("glmRespMod", "nlsRespMod")
```

(2) Prediction Module

"predModule": Linear predictor module consists of

- the model matrix \( X \),
- the coefficient vector \( \text{coef} \),
- a triangular factor of the weighted model matrix \( \text{fac} \),
- \((Vtr = V^T r, \text{where } r = \text{residuals (typically)})\)

currently in dense and sparse flavor:

```r
setClass("predModule", 
    representation(X = "modelMatrix", \text{coef} = "numeric", 
    Vtr = "numeric", \text{fac} = "CholeskyFactorization" 
    "VIRTUAL"))

## sub classes: more specific classes for the two non-trivial slots:
setClass("dPredModule", \{contains = "predModule", 
representation(X = "ddenseModelMatrix", \text{fac} = "Cholesky")

setClass("sPredModule", \{contains = "predModule", 
representation(X = "dsparseModelMatrix", \text{fac} = "CHMfactor")
```
Fitting all “glpModel”’s with One IRLS algorithm

Fitting via IRLS (Iteratively Reweighted Least Squares), where the prediction and response module parts each update “themselves”.

These 3 Steps are iterated till convergence:

1. prediction module (PM) only passes $X \%\% \text{coef} = X\beta$ to the response module (RM)
2. from that, the RM
   - updates its $\mu$,
   - then its weighted residuals and “$X$ weights”
3. these two are in turn passed to PM which
   - reweights itself and
   - solve()s for $\Delta \beta$, the increment of $\beta$.

Convergence only if Bates-Watts orthogonality criterion is fulfilled.
Mixed Modelling - (RE)ML Estimation

In (linear) mixed effects,

\[(Y|\mathcal{B} = b) \sim \mathcal{N}(X\beta + Zb, \sigma^2 I)\]
\[\mathcal{B} \sim \mathcal{N}(0, \Sigma_{\theta}), \quad \text{and} \]
\[\Sigma_{\theta} = \sigma^2 \Lambda_{\theta} \Lambda_{\theta}^T, \quad (1)\]

the evaluation of the (RE) likelihood or equivalently deviance, needs repeated Cholesky decompositions (including fill-reducing permutation \(P\))

\[L_{\theta} L_{\theta}^T = P \left( \Lambda_{\theta}^T Z^T Z \Lambda_{\theta} + I_q \right) P^T, \quad (2)\]

for many \(\theta\)'s and often very large, very sparse matrices \(Z\) and \(\Lambda_{\theta}\) where only the non-zeros of \(\Lambda\) depend on \(\theta\), i.e., the sparsity pattern (incl. fill-reducing permutation \(P\)) and \(f\) is given (by the observational design).
Mixed Modelling - (RE)ML Estimation

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

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

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

- **Sparse Matrices**: used in increasing number of applications and R packages.
- **Matrix** (in every R since 2.9.0)
  1. has `model.Matrix(formula, ....... , sparse = TRUE/FALSE)`
  2. has class "glpModel" for linear prediction modeling
  3. has (currently hidden) function `glm4()`; a proof of concept, (allowing “glm” with **sparse** `X`), using very general `IRLS()` function [convergence check by stringent Bates and Watts (1988) orthogonality criterion]
- **lme4a** on R-forge (= next generation of package lme4) is providing
  1. `lmer()`, `glmer()`, `nlmer()`, and eventually `gnlmer()`, all making use of modular classes (prediction [= fixed eff. + random eff.] and response modules) and generic algorithms (e.g. “PIRLS”).
  2. All with **sparse** (random effect) matrices $Z$ and $\Lambda_\theta$ (where $\text{Var}(\mathcal{B}) = \sigma^2 \Lambda_\theta \Lambda_\theta^T$),
  3. and optionally (**sparseX** = TRUE) sparse fixed effect matrix, $X$.