Inlet one:

*spam*: a Sparse Matrix R Package

with Emphasis on MCMC Methods for Gaussian Markov Random Fields

NCAR – August 2008

Reinhard Furrer
What is *spam*?

- an R package for **sparse** matrix algebra
  - publicly available from CRAN
  - platform independent and documented
What is spam?

- an R package for **sparse** matrix algebra
  - publicly available from CRAN
  - platform independent and documented

- storage economical and fast
  - uses “old Yale sparse format”
  - most routines are in Fortran, adapted for spam
  - balance between readability and overhead
  - flags for “expert” users
What is **spam**?

- an R package for sparse matrix algebra
  - publicly available from CRAN
  - platform independent and documented

- storage economical and fast
  - uses “old Yale sparse format”
  - most routines are in Fortran, adapted for **spam**
  - balance between readability and overhead
  - flags for “expert” users

- versatile, intuitive and simple
  - wrap an `as.spam()` and go
  - **S4** and **S3** syntax
What is spam?

- an R package for **sparse** matrix algebra
  - publicly available from CRAN
  - platform independent and documented

- storage economical and fast
  - uses “old Yale sparse format”
  - most routines are in Fortran, adapted for **spam**
  - balance between readability and overhead
  - flags for “expert” users

- versatile, intuitive and simple
  - wrap an `as.spam()` and go
  - S4 and S3 syntax

- situated between SparseM and Matrix
What is spam?

Package: spam
Version: 0.15-0
Date: 2008-06-10
Author: Reinhard Furrer
Maintainer: Reinhard Furrer <rfurrer@mines.edu>
Depends: R (>= 2.4), methods
Suggests: SparseM (>= 0.72), Matrix
Description: Set of function for sparse matrix algebra.
   Differences with SparseM/Matrix are:
   (1) we only support (essentially) one sparse matrix format,
   (2) based on transparent and simple structure(s),
   (3) tailored for MCMC calculations within GMRF.
   (4) S3 and S4 like-"compatible" ... and it is fast.
LazyLoad: Yes
LazyData: Yes
License: GPL | file LICENSE
Title: SPArse Matrix
URL: http://www.mines.edu/~rfurrer/software/spam/
spam defines a S4 class spam containing the vectors: “entries”, “colindices”, “rowpointers” and ”dimension”.

R> slotNames("spam")
[1] "entries" "colindices" "rowpointers" "dimension"

R> getSlots( "spam")
entries colindices rowpointers dimension
"numeric" "integer" "integer" "integer"
### Representation of Sparse Matrices

```r
R> A
[1,] 1.0 0.1 0 0.2 0.3
[2,] 0.6 2.0 0 0.5 0.0
[3,] 0.0 0.0 3 0.0 0.6
[4,] 0.7 0.8 0 4.0 0.0
[5,] 0.9 0.0 1 0.0 5.0
```

Class `‘spam’`

```r
R> slotNames(A)
[1] "entries" "colindices" "rowpointers" "dimension"
```

```r
R> A@entries
[1] 1.0 0.1 0.2 0.3 0.6 2.0 0.5 3.0 0.6 0.7 0.8 4.0 0.9 1.0 5.0
```

```r
R> A@colindices
[1] 1 2 4 5 1 2 4 3 5 1 2 4 1 3 5
```

```r
R> A@rowpointers
[1] 1 5 8 10 13 16
```

```r
R> A@dimension
[1] 5 5
```
Creating Sparse Matrices

Similar coercion techniques as with matrix:

- spam(...)
- as.spam(...)

Special functions:

- diag.spam(...)
- nearest.dist(...)
Methods for spam

- Similar behavior as with matrices
  plot; dim; determinant; \%*\%; +; ...

- Slightly enhanced behavior
  print; dim<-; chol;

- Specific behavior
  Math; Math2; Summary; ...

- New methods
  display; ordering;
Create Covariance Matrices

Covariance matrix:

\[ \text{nearest.dist} \text{ and applying a covariance function:} \]

\[
\begin{align*}
R & \quad \text{C} <- \text{nearest.dist(x)} \\
R & \quad \text{C@entries} <- \text{Wendland( C@entries, dim=2, k=1)}
\end{align*}
\]

Precision matrix (GMRF):

— regular grids: \text{nearest.dist} with different cutoffs

\[
\begin{align*}
& R \quad \text{diag.spam(n)} + b1 * \text{nearest.dist(x, delta=1)} + \\
& \quad b2 * \text{nearest.dist(x, delta=sqrt(2))}
\end{align*}
\]

— irregular grids: using incidence list and spam
A key feature of spam is to solve efficiently linear systems.

To solve the system $Ax = b$, we

- perform a Cholesky factorisation $A = U^T U$
- solve two triangular systems $U^T z = b$ and $Ux = z$

But we need to “ensure” that $U$ is as sparse as possible!
Solving Linear Systems

A key feature of spam is to solve efficiently linear systems.

To solve the system $Ax = b$, we

- perform a Cholesky factorisation $A = U^T U$
- solve two triangular systems $U^T z = b$ and $U x = z$

But we need to “ensure” that $U$ is as sparse as possible!

Permute the rows and columns of $A$: $P^T A P = U^T U$. 
## Cholesky

Some technical details about a Cholesky decomposition:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>Determine permutation and permute the input matrix $A$ to obtain $P^TAP$</td>
</tr>
<tr>
<td>[2]</td>
<td>Symbolic factorization: the sparsity structure of $U$ is constructed</td>
</tr>
<tr>
<td>[3]</td>
<td>Numeric factorization: the elements of $U$ are computed</td>
</tr>
</tbody>
</table>
Cholesky

spam knows Cholesky!

— Several methods to construct permutation matrices $P$
— update to perform only ‘partial’ Cholesky factors
— Flags for avoiding sanity checks
Cholesky
Time and memory usage for 101 Cholesky factorizations (solid) and one factorization and 100 updates (dashed) of a precision matrix from different sizes $L$ of regular $L \times L$ grids with a second order neighbor structure.

(The precision matrix from $L = 200$ has $L^4 = 1.6 \cdot 10^9$ elements)
### Cholesky

Gain of time and memory usage with different options and arguments in the case of a second order neighbor structure of a regular $50 \times 50$ grid and of the US counties. The time and memory usage for the generic call `chol` are 6.2 seconds, 174.5 Mbytes and 15.1 seconds, 416.6 Mbytes, respectively.

<table>
<thead>
<tr>
<th>Options or arguments</th>
<th>Regular grid time</th>
<th>Regular grid memory</th>
<th>US counties time</th>
<th>US counties memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the specific call <code>chol.spam</code></td>
<td>1.001</td>
<td>0.992</td>
<td>0.954</td>
<td>1.004</td>
</tr>
<tr>
<td>Option <code>safemode=c(FALSE,FALSE,FALSE)</code></td>
<td>0.961</td>
<td>1.002</td>
<td>0.988</td>
<td>0.997</td>
</tr>
<tr>
<td>Option <code>cholsymmetrycheck=False</code></td>
<td>0.568</td>
<td>0.524</td>
<td>0.646</td>
<td>0.493</td>
</tr>
<tr>
<td>Passing <code>memory=list(nnzR=..., nnzcolindices=...)</code></td>
<td>0.969</td>
<td>0.979</td>
<td>0.928</td>
<td>0.972</td>
</tr>
<tr>
<td>All of the above</td>
<td>0.561</td>
<td>0.508</td>
<td>0.618</td>
<td>0.490</td>
</tr>
<tr>
<td>All of the above and passing <code>pivot=...</code> to <code>chol.spam</code></td>
<td>0.542</td>
<td>0.528</td>
<td>0.572</td>
<td>0.496</td>
</tr>
<tr>
<td>All of the above and option <code>cholpivotcheck=False</code></td>
<td>0.510</td>
<td>0.511</td>
<td>0.557</td>
<td>0.489</td>
</tr>
<tr>
<td>Numeric update only using <code>update</code></td>
<td>0.132</td>
<td>0.070</td>
<td>0.170</td>
<td>0.063</td>
</tr>
</tbody>
</table>
Options

For “experts”, flags to speed up the code ...

R> noquote(unlist(format(spam.options())))

<table>
<thead>
<tr>
<th>eps</th>
<th>drop</th>
<th>printsize</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.220446e-16</td>
<td>FALSE</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>imagesize</th>
<th>trivalues</th>
<th>cex</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>FALSE</td>
<td>1200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>safemode</th>
<th>dopivoting</th>
<th>cholsymmetrycheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cholpivotcheck</th>
<th>cholupdatesingular</th>
<th>cholincreasefactor</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>1.25, 1.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nearestdistincreasefactor</th>
<th>nearestdistnnz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.25</td>
<td>160000, 400</td>
</tr>
</tbody>
</table>
Limits

What can spam not do (yet)?

- LU/SVD decompositions
- Eigendecompositions
- Non double elements
- ...

But, please, comments to rfurrer@mines.edu!
For example:
