Inlet one:

`spam`: a Sparse Matrix R Package

with Emphasis on MCMC Methods for Gaussian Markov Random Fields

NCAR – August 2008

Reinhard Furrer

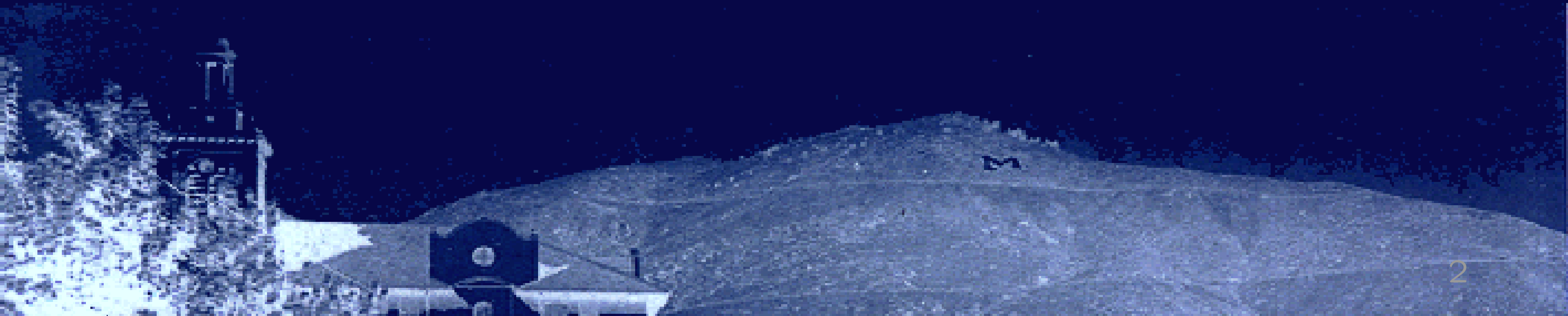
What is `spam`?

- 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?

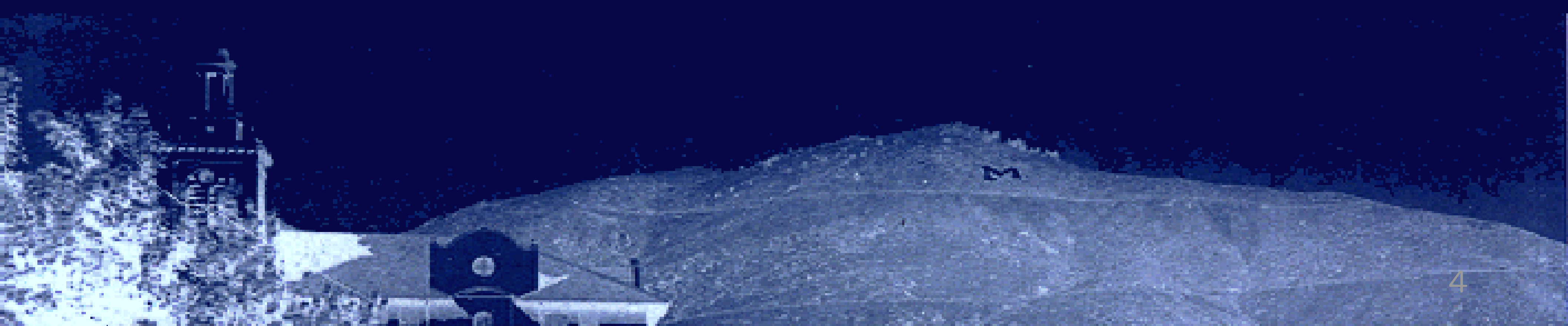
```
Package:  spam
Version:  0.15-0
Date:     2008-06-10
Author:   Reinhard Furrer
Maintainer:  Reinhard Furrer <rfurrer@mines.edu>
Depends: R (>= 2.4), methods
Suggests:  SparseM (>= 0.72), Matrix
Description: Set of function for sparse matrix algebra.
  Differences with SparseM/Matrix are:
  (1) we 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/
```

Representation of Sparse Matrices

`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> A
     [,1] [,2] [,3] [,4] [,5]
[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> slotNames(A)
[1] "entries"     "colindices"  "rowpointers" "dimension"
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> A@colindices
 [1] 1 2 4 5 1 2 4 3 5 1 2 4 1 3 5
R> A@rowpointers
[1]  1  5  8 10 13 16
R> A@dimension
[1] 5 5
```

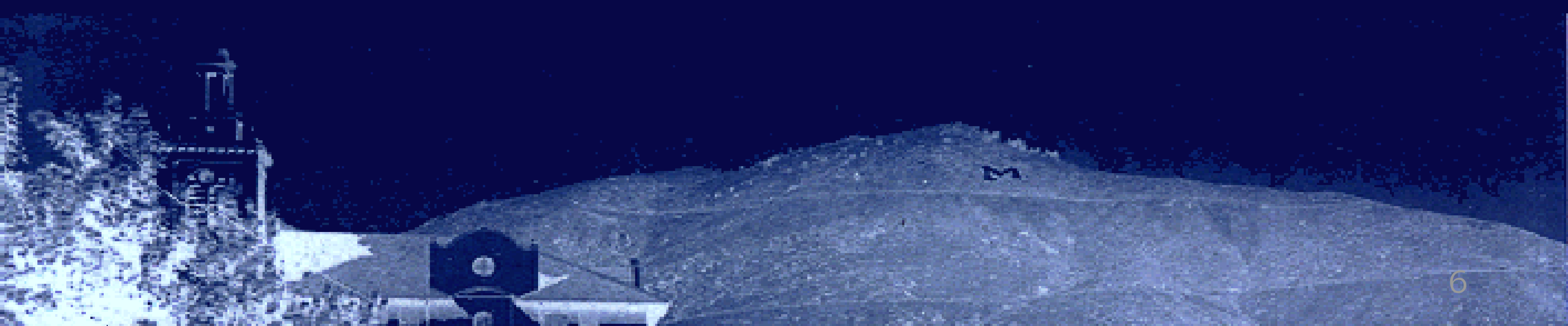
Creating Sparse Matrices

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:

`nearest.dist` and applying a covariance function:

```
R> C <- nearest.dist(x)
R> C@entries <- Wendland( C@entries, dim=2, k=1)
```

Precision matrix (GMRF):

— regular grids: `nearest.dist` with different cutoffs

```
R> diag.spam(n) + b1 * nearest.dist(x, delta=1) +
+            b2 * nearest.dist(x, delta=sqrt(2))
```

— irregular grids: using incidence list and `spam`

Solving Linear Systems

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

To solve the system $\mathbf{Ax} = \mathbf{b}$, we

- perform a Cholesky factorisation $\mathbf{A} = \mathbf{U}^\mathsf{T}\mathbf{U}$
- solve two triangular systems $\mathbf{U}^\mathsf{T}\mathbf{z} = \mathbf{b}$ and $\mathbf{Ux} = \mathbf{z}$

But we need to “ensure” that $\mathbf{U}$ is as sparse as possible!

Solving Linear Systems

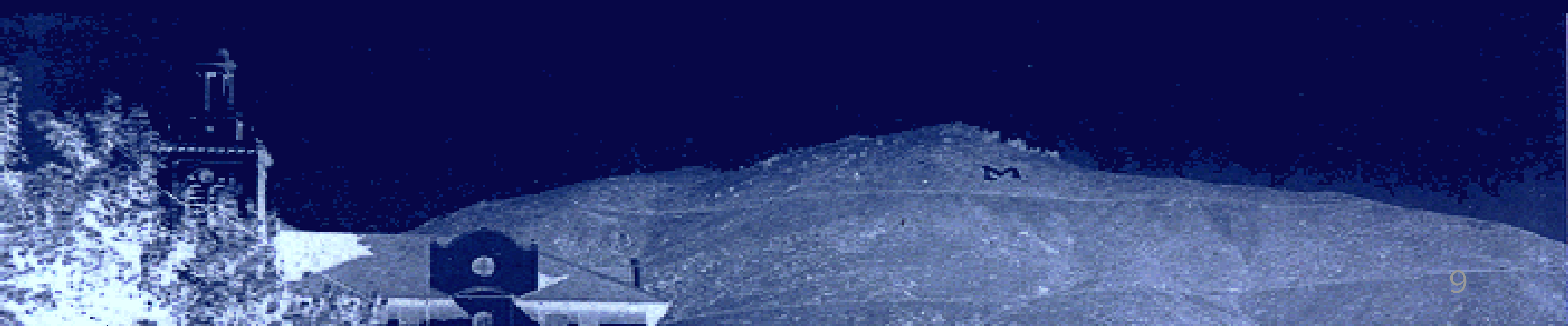
A key feature of `spam` is to solve efficiently linear systems.

To solve the system $\mathbf{Ax} = \mathbf{b}$, we

- perform a Cholesky factorisation $\mathbf{A} = \mathbf{U}^\mathsf{T}\mathbf{U}$
- solve two triangular systems $\mathbf{U}^\mathsf{T}\mathbf{z} = \mathbf{b}$ and $\mathbf{Ux} = \mathbf{z}$

But we need to “ensure” that $\mathbf{U}$ is as sparse as possible!

Permute the rows and columns of $\mathbf{A}$: $\mathbf{P}^\mathsf{T}\mathbf{AP} = \mathbf{U}^\mathsf{T}\mathbf{U}$.



Cholesky

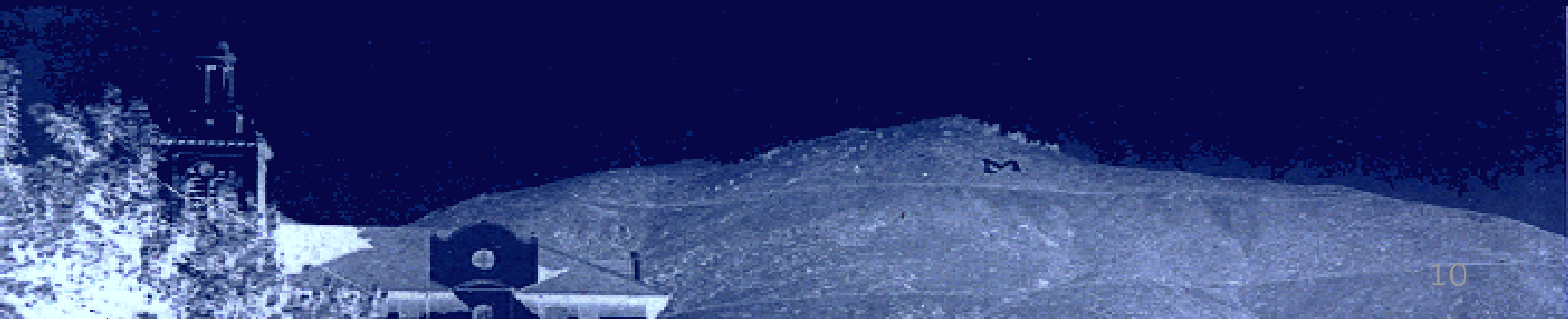
Some technical details about a Cholesky decomposition:

---

[1] Determine permutation and
permute the input matrix $\mathbf{A}$ to obtain $\mathbf{P}^{\mathsf{T}}\mathbf{A}\mathbf{P}$

[2] Symbolic factorization:
the sparsity structure of $\mathbf{U}$ is constructed

[3] Numeric factorization:
the elements of $\mathbf{U}$ are computed



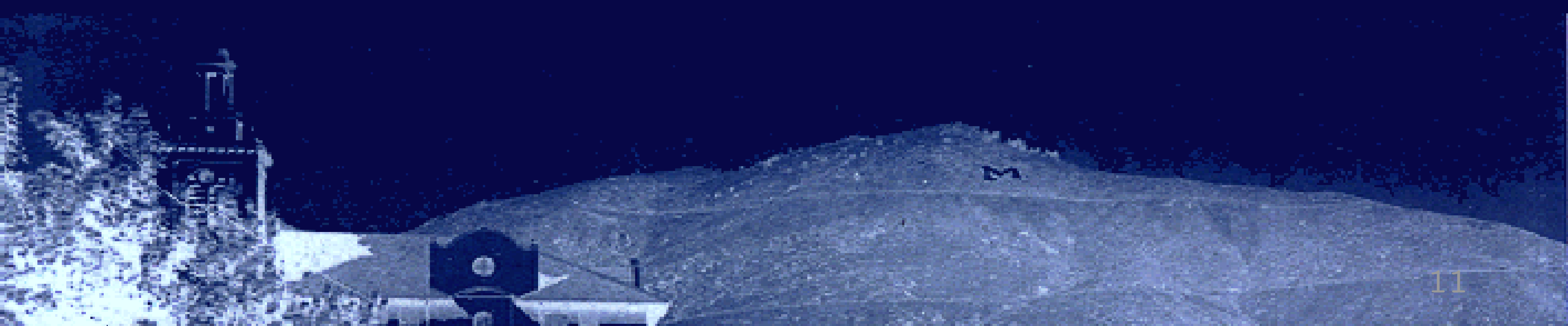
Cholesky

`spam` knows Cholesky!

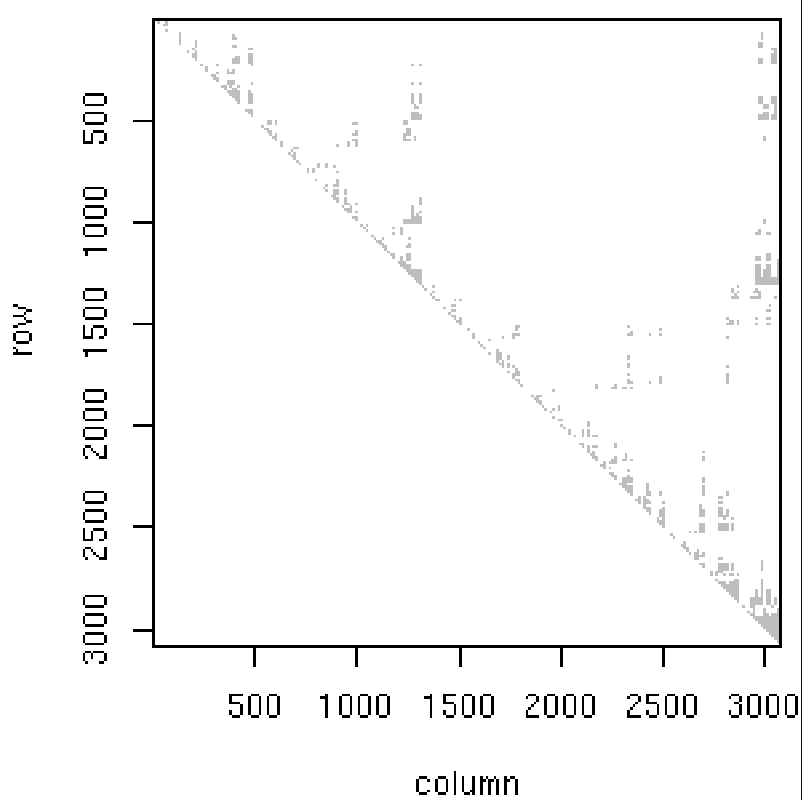
— Several methods to construct permutation matrices **P**

— `update` to perform only ‘partial’ Cholesky factors

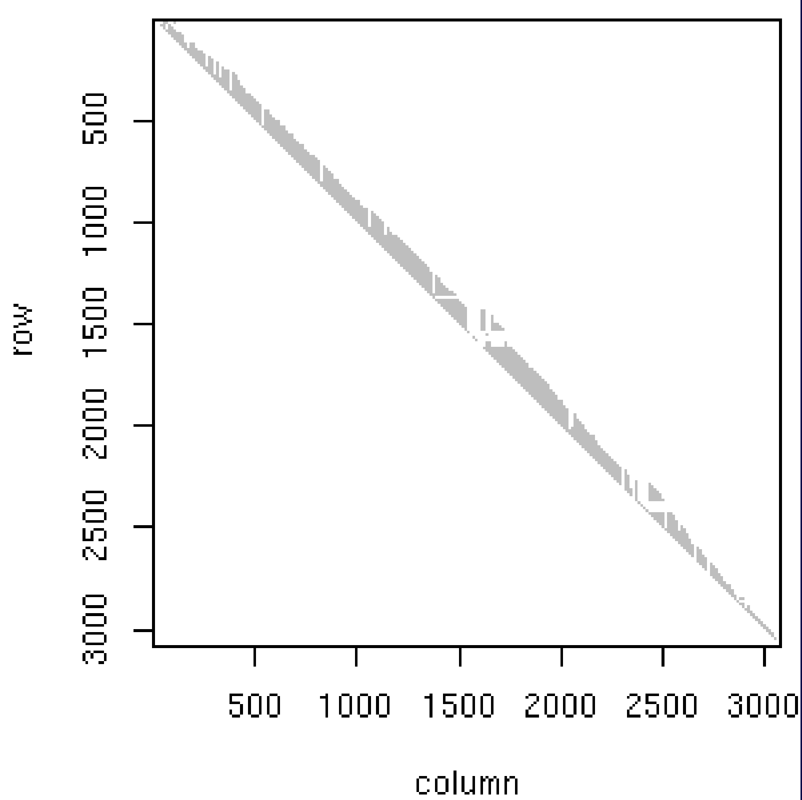
— Flags for avoiding sanity checks



Cholesky

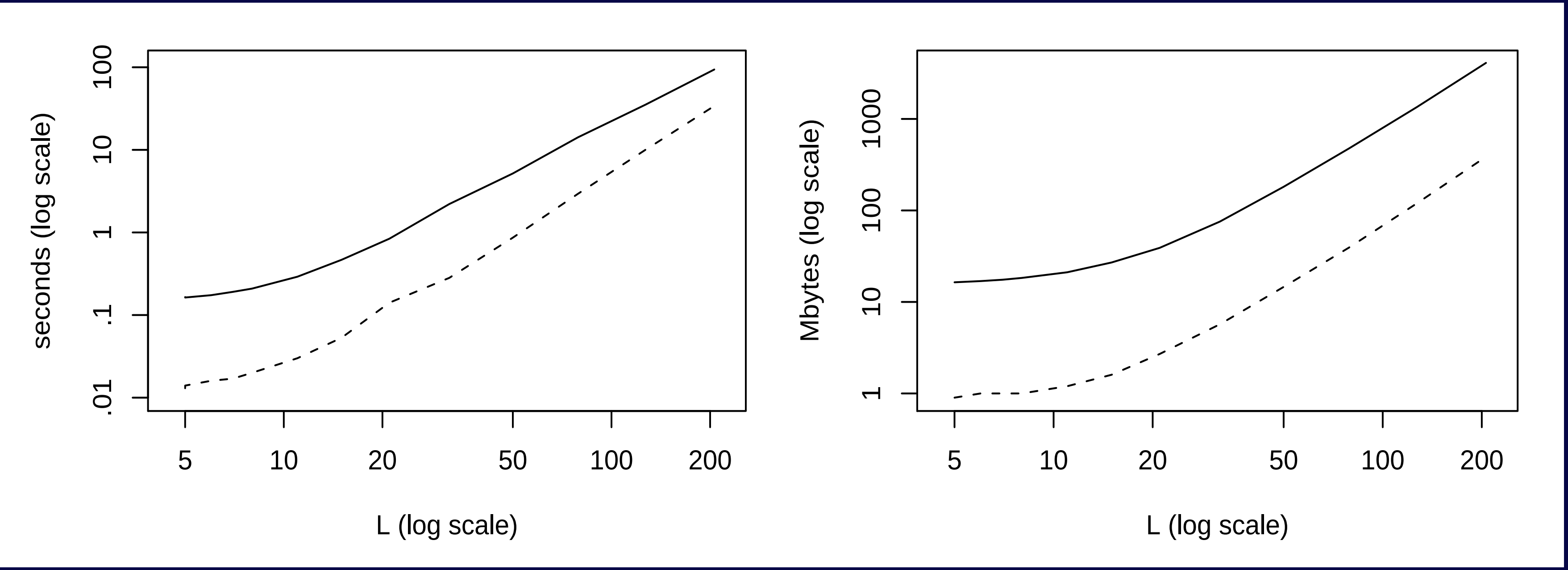


row
500
1000
1500
2000
2500
3000
column



row
500
1000
1500
2000
2500
3000
column

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 50×50 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.

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

Options

For "experts", flags to speed up the code . . .

```
R> noquote(unlist(format(spam.options())) )
                      eps                 drop            printsize
             2.220446e-16                FALSE                  100
                imagesize            trivalues                  cex
                    10000                FALSE                 1200
                 safemode           dopivoting    cholsymmetrycheck
         TRUE, TRUE, TRUE                 TRUE                 TRUE
           cholpivotcheck cholupdatesingular  cholincreasefactor
                     TRUE              warning           1.25, 1.25
nearestdistincreasefactor       nearestdistnnz
                     1.25          160000, 400
```

Limits

What can `spam` not do (yet)?

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

But, please, comments to `rfurrer@mines.edu`!

Reference

For example:

Furrer, R. and Sain, S. R. (2008). spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields. Submitted.

Furrer, R. and Sain, S. R. (2008). Spatial Model Fitting for Large Datasets with Applications to Climate and Microarray Problems. *Statistics and Computing*, doi:10.1007/s11222-008-9075-x.