Sparse Matrices Methods and Kriging

Applications to Large Spatial Data Sets

SAMSI

July 28 – August 1, 2009

Reinhard Furrer, University of Zurich

Outline

What are sparse matrices?

How to work with sparse matrices?

Sparse positive definite matrices in statistics.

Sparse matrices and `fields`.

Sparse Matrices

What is "sparse" or a sparse matrix?

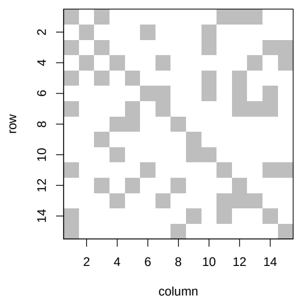
According to Wiktionary/Wikipedia:

Sparse: (Adjective)
1. Having widely spaced intervals
2. Not dense; meager

Sparse matrix:
a matrix populated primarily with zeros.

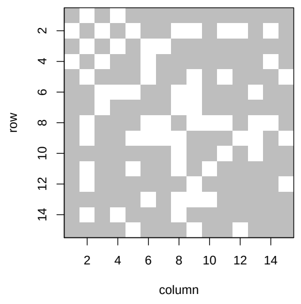
Sparse Matrices

```
R>  n <- 15
R>  A <- array( runif(n^2), c(n,n)) + diag(n)
R>  A[A < 0.75] <- 0
```



Sparse Matrices

```
R>  n <- 15
R>  A <- array( runif(n^2), c(n,n)) + diag(n)
R>  A[A < 0.75] <- 0
R>  AtA <- t(A) %*% A
```



Sparse Matrices

Why should we use sparse matrices?

1. Savings in storage
   nonzeros vs total elements

2. Savings in computing time
   0.066sec vs 0.003sec for 1,000 × 1,000 matrix multiplication

To exploit the savings need to exploit the sparsity.

We need a clever storage format and fast algorithms.

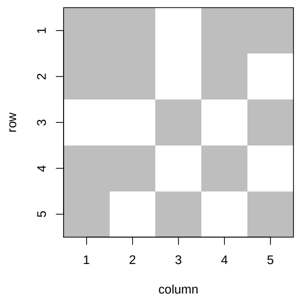
Storage Formats

Let $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{n\times m}$ and z the number of its nonzero elements.

1. Naive/"traditional"/classic format:
   one vector of length $n \times m$ and a dimension attribute.
2. Triplet format:
   three vectors of length z, (i, j, a_{ij}) and a dimension attribute.
3. Compressed sparse row (CSR) format:
   eliminate redundant row indices.
4. and about 10 more . . .

Storage Formats, Example

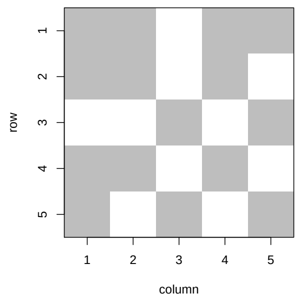
$$A = \begin{pmatrix} 1 & 0.1 & 0 & 0.2 & 0.3 \\ 0.4 & 2 & 0 & 0.5 & 0 \\ 0 & 0 & 3 & 0 & 0.6 \\ 0.7 & 0.8 & 0 & 4 & 0 \\ 0.9 & 0 & 0.0 & 0 & 5 \end{pmatrix}$$



Naive/traditional/classic:
1, .4, 0, .7, .9, .1, 2, 0, .8, 0, 0, 0, 3, 0, .0, .2, .5, 0, 4, 0, .3, 0, .6, 0, 5

Storage Formats, Example

$$A = \begin{pmatrix} 1 & 0.1 & 0 & 0.2 & 0.3 \\ 0.4 & 2 & 0 & 0.5 & 0 \\ 0 & 0 & 3 & 0 & 0.6 \\ 0.7 & 0.8 & 0 & 4 & 0 \\ 0.9 & 0 & 0.0 & 0 & 5 \end{pmatrix}$$

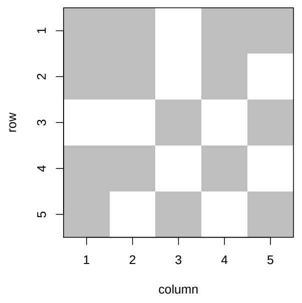


Triplet:

i :	1	1	1	1	2	2	2	3	3	4	4	4	5	5	5
j :	1	2	4	5	1	2	4	2	3	1	2	4	1	3	5
a_{ij} :	1	.1	.2	.3	.4	2	.5	3	.6	.7	.8	4	.9	.0	5

Storage Formats, Example

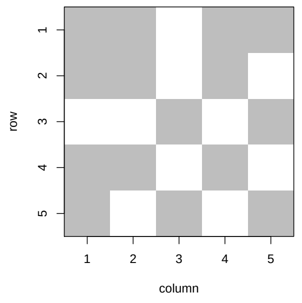
$$A = \begin{pmatrix} 1 & 0.1 & 0 & 0.2 & 0.3 \\ 0.4 & 2 & 0 & 0.5 & 0 \\ 0 & 0 & 3 & 0 & 0.6 \\ 0.7 & 0.8 & 0 & 4 & 0 \\ 0.9 & 0 & 0.0 & 0 & 5 \end{pmatrix}$$



i :	1				2			3		4			5		
j :	1	2	4	5	1	2	4	2	3	1	2	4	1	3	5
a_{ij} :	1	.1	.2	.3	.4	2	.5	3	.6	.7	.8	4	.9	.0	5

Storage Formats, Example

$$A = \begin{pmatrix} 1 & 0.1 & 0 & 0.2 & 0.3 \\ 0.4 & 2 & 0 & 0.5 & 0 \\ 0 & 0 & 3 & 0 & 0.6 \\ 0.7 & 0.8 & 0 & 4 & 0 \\ 0.9 & 0 & 0.0 & 0 & 5 \end{pmatrix}$$



CSR:

ptr :	1				5			8		10			13			16
j :	1	2	4	5	1	2	4	2	3	1	2	4	1	3	5	
a_{ij} :	1	.1	.2	.3	.4	2	.5	3	.6	.7	.8	4	.9	.0	5	

Compressed Sparse Row Format

1. the nonzero values row by row
2. the (ordered) column indices of nonzero values
3. the position in the previous two vectors corresponding to new rows, given as pointers
4. the column dimension of the matrix.

CSR:

ptr :	1				5			8		10			13			16
j :	1	2	4	5	1	2	4	2	3	1	2	4	1	3	5	
a_{ij} :	1	.1	.2	.3	.4	2	.5	3	.6	.7	.8	4	.9	.0	5	

(Dis)Advantages

1. Naive format:
   No savings in storage and computation (for sparse matrices)
   Status quo
2. Triplet format:
   Savings in storage and computation for sparse matrices
   Loss in storage and computation for full matrices
   Intuitive
3. Compressed sparse row (CSR) format:
   Apart from intuitive, same as triplet
   Faster element access
   Many available algorithms
   Arbitrary choice for "row" vs "column" format (CSC)

Implications

With a new storage format new "algorithms" are required . . .

Is it worthwhile???

Timing

Setup:

```
R> timing <- function(expr)
+   as.vector( system.time( for (i in 1:N) expr)[1])

R> N <- 1000       # how many operations
R> n <- 999        # matrix dimension
R> cutoff <- 0.9   # what will be set to 0

R> A <- array( runif(n^2), c(n,n))
R> A[A < cutoff] <- 0

R> S <- somecalltomagicfunctiontogetsparseformat( A)
```

Compare timing for different operations on `A` and `S`.

Timing

Setup:

```
R> timing <- function(expr)
+   as.vector( system.time( for (i in 1:N) expr)[1])

R> N <- 1000       # how many operations
R> n <- 999        # matrix dimension
R> cutoff <- 0.9   # what will be set to 0

R> A <- array( runif(n^2), c(n,n))
R> A[A < cutoff] <- 0

R> S <- as.spam( A)
```

Compare timing for different operations on `A` and `S`.

Timing

```
R> timing( A + sqrt(A) )
[1] 0.058
R> timing( S + sqrt(S) )
[1] 0.065

R> timing( AtA <- t(A) %*% A )
[1] 1.567
R> timing( StS <- t(S) %*% S )
[1] 3.422
```

Timing

```
R> timing( A[1,2] <- .5 )
[1] 0.007
R> timing( S[1,2] <- .5 )
[1] 0.018

R> timing( A[n,n-1] <- .5 )
[1] 0.001
R> timing( S[n,n-1] <- .5 )
[1] 0.012
```

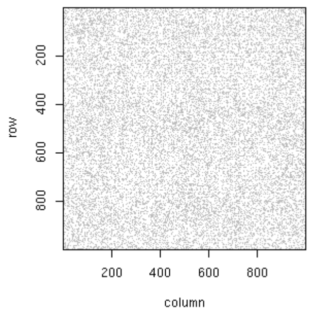
Timing

```
R> timing( xA <- solve(AtA, rep(1,n)) )
[1] 0.626
R> timing( xS <- solve(StS, rep(1,n)) )
[1] 0.831

R> timing( RA <- chol(AtA) )
[1] 0.301
R> timing( RS <- chol(StS) )
[1] 0.824
```

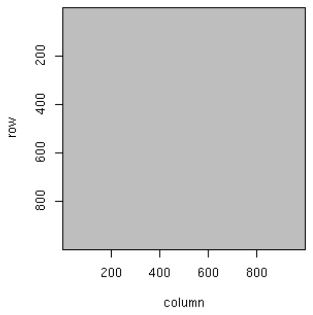
Is it really worthwhile? What is going on?

Timing

Matrix `S`



Matrix `StS`



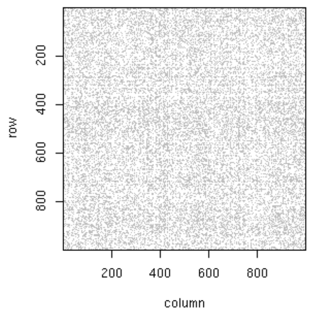
Timing

With cutoff 0.99:

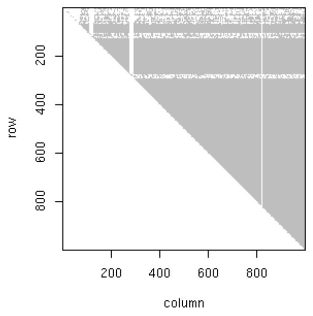
```
R> timing( AtA <- t(A) %*% A )
[1] 1.536
R> timing( StS <- t(S) %*% S )
[1] 0.076

R> timing( RA <- chol(AtA) )
[1] 0.304
R> timing( RS <- chol(StS) )
[1] 0.451
```

Timing

Matrix `StS`



Matrix `chol(StS)`



Density of the factor is 41% with fill-in ratio 7.2.

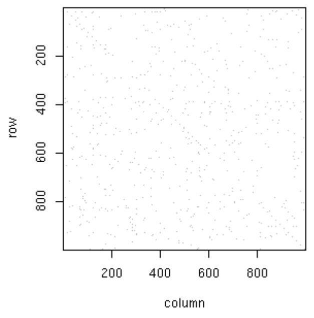
Timing

With cutoff 0.999:

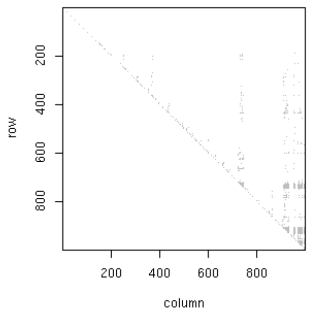
```
R> timing( AtA <- t(A) %*% A )
[1] 1.549
R> timing( StS <- t(S) %*% S )
[1] 0.004

R> timing( RA <- chol(AtA) )
[1] 0.304
R> timing( RS <- chol(StS) )
[1] 0.004
```

Timing

Matrix `StS`



Matrix `chol(StS)`



Density of the factor is .6% with fill-in ratio 2.3.

Implications

With a new storage format new “algorithms” are required . . .

Is it worthwhile??? Yes!

Especially since

`spam`: R package for sparse matrix algebra.

What is `spam`?

- an R package for sparse matrix algebra
  - publicly available from CRAN, 0.15-4
  - 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
  - `?xyz.spam` for help
  - `S4` and `S3` syntax
- situated between `SparseM` and `Matrix`

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, diag=TRUE, upper=NULL)
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-b2) * nearest.dist(x, delta=1, upper=NULL) +
+    b2 * nearest.dist(x, delta=sqrt(2), upper=NULL)
```

— irregular grids: using incidence list and `spam`

```
R> incidence <- list( i=..., j=..., values)
R> C <- spam( incidence, n, n)
```

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 $\mathbf{P}^\mathsf{T}\mathbf{AP}$

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

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

Cholesky

BTW:
efficient Cholesky factorization

$\Longleftrightarrow$

efficient determinant calculation:

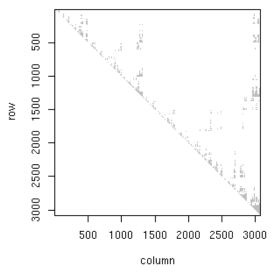
$$\det(\mathbf{C}) = \det(\mathbf{U}^\mathsf{T})\det(\mathbf{U}) = \prod_{i=1}^{n} \mathbf{U}_{ii}^2$$

Cholesky

`spam` knows Cholesky!

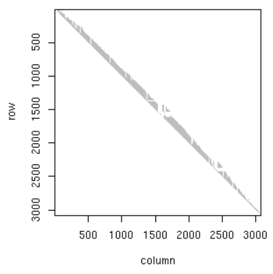
— Several methods to construct permutation matrices $\mathbf{P}$

Cholesky factor with `MMD`



Density: 1.5%, fill-in: 4.7

`RCM` ordering



Density: 2.7%, fill-in: 8.1

Cholesky

`spam` knows Cholesky!

— Several methods to construct permutation matrices $\mathbf{P}$

— `update` to perform only 'partial' Cholesky factors

```
update( choleskyobject, covariancematrix)
```

Performing the numerical decomposition only, [3]

Cholesky

spam knows Cholesky!

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

— update to perform only 'partial' Cholesky factors

— Passing arguments and flags for avoiding sanity checks

Using the specific call chol.spam

Passing memory=list(nnzR=..., nnzcolindices=...), pivot=...

Options safemode=c(FALSE,FALSE,FALSE), cholsymmetrycheck=FALSE, cholpivotcheck=FALSE

Options in spam

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

```
R> powerboost()

R> noquote( 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 of spam

What can spam not do (yet)?

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

But, please, comments to reinhard.furrer@math.uzh.ch!

Sparse Matrices in Statistics

Where do large matrices occur?

- Location matrices
- Design matrices
- Covariance matrices
- Precision matrices

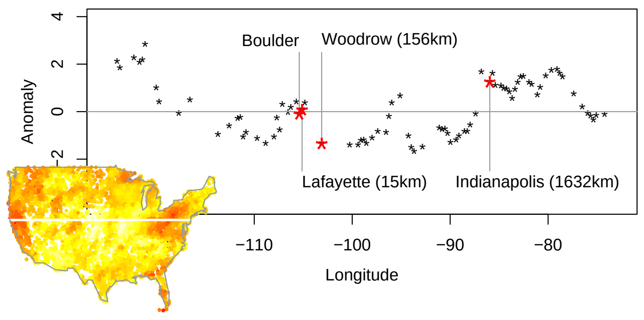
Sparse Matrices in Statistics

- Covariance matrices:
  Compactly supported covariance functions
  Tapering
- Precision matrices:
  (Gaussian) Markov random fields
  (Tapering???)

We have symmetric positive definite (spd) matrices.

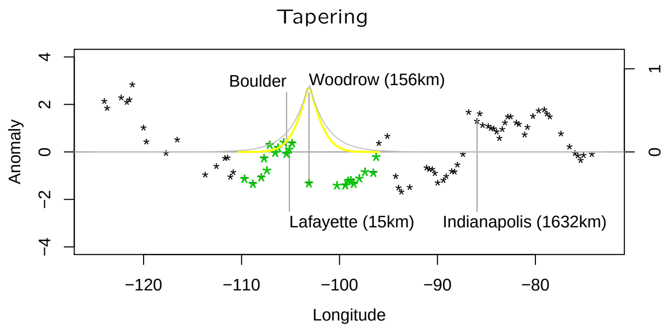
Motivation

Precipitation anomaly along 40° lat.

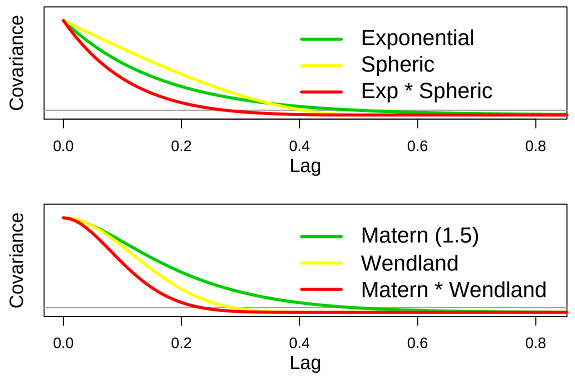


Motivation

Precipitation anomaly along 40° lat.



Examples



Objective

For an isotropic and stationary process
with Matérn covariance $C_0(h)$,
find a taper $C_\theta(h)$,
such that kriging with the product $C_0(h)C_\theta(h)$
is asymptotically optimal.

$$\frac{\mathrm{MSE}(\mathbf{x}^*, C_0C_\theta)}{\mathrm{MSE}(\mathbf{x}^*, C_0)} \to 1 \qquad \frac{\varrho(\mathbf{x}^*, C_0C_\theta)}{\mathrm{MSE}(\mathbf{x}^*, C_0)} \to \gamma$$

$$\varrho(\mathbf{x}^*, C) = C(0) - \mathbf{c}^{*\mathsf{T}}\mathbf{C}^{-1}\mathbf{c}^*$$

Misspecified Covariances

In a series of (Annals) papers, Stein gives asymptotic results for misspecified covariances.

Under appropriate conditions, tapering is a form of misspecification.

The taper has to be

- as differentiable at the origin as the original covariance
- more differentiable throughout the domain than at the origin

Tapering

Tapering is an (asymptotically and computationally) efficient technique to create sparse covariance matrices.

Taper range can be justified by computing resources. However, 20–30 locations within the taper range is often sufficient.

"Classical" tapers are:

- spherical: $C_\theta(h) = \left(1 - \frac{|h|}{\theta}\right)_+^2 \left(1 + \frac{|h|}{\theta}\right)$
- Wendland-type: $C_\theta(h) = \left(1 - \frac{|h|}{\theta}\right)_+^{\ell+k} \cdot$ polynomial in $\frac{|h|}{\theta}$ of deg k

Positive Definite Matrices

A (large) covariance (often) appears in:

- drawing from a multivariate normal distribution
- calculating/maximizing the (log-)likelihood
- linear/quadratic discrimination analysis
- PCA, EOF, . . .

But all boils down to solving a linear system and possibly calculating the determinant . . .

'Sparse PCA' is sparse in a different sense . . .

Sparse Matrices and `fields`

- `fields` is not bound to a specific sparse matrix format
- All heavy lifting is done in `mKrig` or `Krig.engine.fixed`
- For a specific sparse format, requires the methods:
  `chol`, `backsolve`, `forwardsolve` and `diag`
  as well as elementary matrix operations
  need to exist
- If available uses operators to handle diagonal matrices quickly

⇝ The covariance matrix has to stem from particular class.

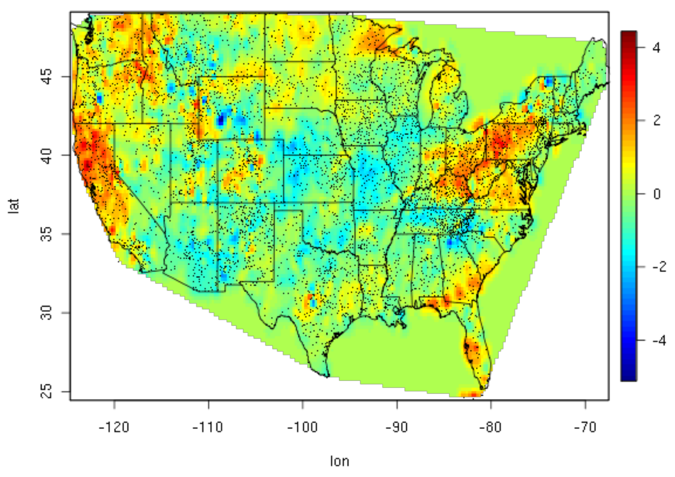
`fields` uses `spam` as default package!

Example `mKrig`

With appropriate covariance function:

```
R> x <- USprecip[ precipsubset, 1:2] # locations
R> Y <- USprecip[ precipsubset, 4]   # anomaly

R> out <- mKrig(x,Y, m=1, cov.function="wendland.cov",theta=1.5)


R> out.p <- predict.surface( out, nx=220, ny=120)
R> surface(out.p, type='I')
R> US(add=T)
R> points(x,pch='.')
```

Example `mKrig`



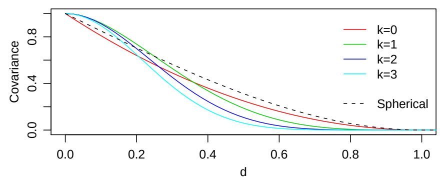
Example `Krig`

```
R> out <- Krig( x,Y, m=1, cov.function="wendland.cov",theta=1.5,
+              lambda=0)

R> out.q <- predict.surface( out, nx=220, ny=120)

R> sum( ( out.q$z-out.p$z)^2, na.rm=T)
[1] 1.616783e-20
```

`Krig/predict` is slower (here 2.1/3.7 vs 10.4/3.9 seconds).

Wendland family

`wendland.cov` (based on `Wendland`) produces a `spam` matrix.
All matrix functions are appropriately overloaded . . .



To create sparse covariance matrices based on other covariance functions, use `wendland.cov` as skeleton.

Tapering

Tapering can be performed with `stationary.taper.cov`.
Arguments are (selection):

`Covariance = "Exponential"`

`Taper = "Wendland"`

`Taper.args = NULL`: arguments for the taper function

`Dist.args = NULL`: arguments passed to `nearest.dist`

`...` : arguments passed to covariance function

All arguments can also be passed from `mKrig/Krig`

Tapering

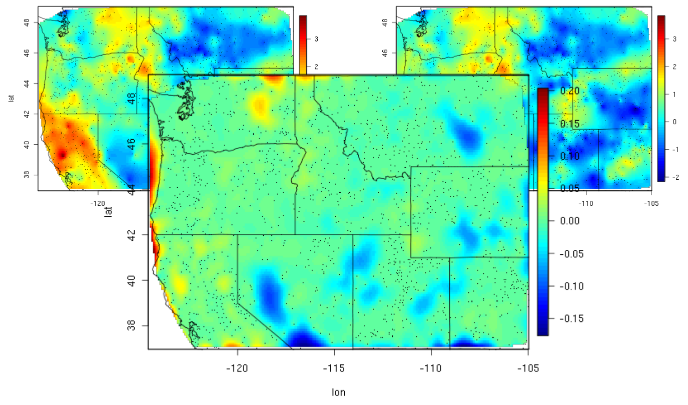
Compare the predicted surfaces without and with tapering:

```
R>  out1 <- mKrig( xr,Yr, m=1, theta=1.5) )
R>  out1.p <- predict.surface( out1, nx=220, ny=120) )

R>  out2 <- mKrig( xr,Yr, m=1, theta=1.5,
+                  cov.function="stationary.taper.cov",
+                  Taper.args = list(k=0, theta=3)))
R>  out2.p <- predict.surface( out2, nx=220, ny=120))
```

(timing yields 4/22 and 1/9 seconds)

Tapering

Exponential covariance and with tapering



Asides . . .

The following arguments of `mKrig`/`Krig` are linked to `spam`:

`Dist.args` : arguments passed to `nearest.dist`

`chol.args` : arguments passed to `chol`

Use their help for fine tuning.

`predict.se.Krig`, `predict.surface.se.Krig` are very inefficient because `nrow(x)` equations need to be solved.

How Big is Big?

Upper limit to create a large matrix is the minimum of:

(1) available memory (machine and OS/shell dependent)
Error: `'cannot allocate vector of size'`

(2) addressing capacity ($2^{31} - 1$)
Error: `'cannot allocate vector of length'`

However, R is based on passing by value, calls create local copies (often 3–4 times the space of the object is used).

```
R>  help("Memory-limits")
```

And Beyond?

Parallelization:
`nws`, `snow`, `Rmpi`, . . .

Memory "Outsourcing":
Matrices are not (entirely) kept in memory:
`ff`, `filehash`, `biglm`, . . .

(S+ has the library `BufferedMatrix`)

References

Furrer, R., Genton, M. G. and Nychka, D. (2006). Covariance Tapering for Interpolation of Large Spatial Datasets. *Journal of Computational and Graphical Statistics*, 15(3), 502–523.

Furrer, R. and Sain, S. R. (2009). Spatial Model Fitting for Large Datasets with Applications to Climate and Microarray Problems. *Statistics and Computing*, 19(2), 113–128, doi:10.1007/s11222-008-9075-x.

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