Package ‘SpatioTemporal’

February 19, 2015

Version 1.1.7
License GPL-2
Title Spatio-Temporal Model Estimation
Author Johan Lindstrom, Adam Szpiro, Paul D. Sampson, Silas Bergen, Assaf P. Oron
Maintainer Johan Lindstrom <johanl@maths.lth.se>
Description Utilities that estimate, predict and cross-validate the spatio-temporal model developed for MESA Air.
Encoding latin1
Date 2013-08-12
Depends R (>= 2.12.0), Matrix
Imports MASS
Suggests maps, plotrix, fields
Type Package
LazyLoad yes
Collate 'block_matrices.R' 'c_cov_matrices.R' 'c_F_mult.R'
  'c_lin_alg.R' 'CV_aux.R' 'CV_estimate.R' 'CV_predict.R'
  'deprecated.R' 'internal.R' 'loglikeST.R' 'loglikeST_aux.R'
  'loglikeST_derivatives.R' 'MCMC.R' 'SpatioTemporal-data.R'
  'SpatioTemporal-package.R' 'STdata.R' 'STdata_processing.R'
  'STmodel.R' 'STmodel_combine.R' 'STmodel_estimate.R'
  'STmodel_predict.R' 'STmodel_processing.R' 'STmodel_setup.R'
  'STmodel_simulate.R' 'temporal_SVD_incomplete_data.R' 'utils.R'
NeedsCompilation yes
Repository CRAN
Date/Publication 2013-08-12 16:51:44

R topics documented:

SpatioTemporal-package .......................... 4
R topics documented:

blockMult .......................... 7
boxplot.estCVSTmodel ................. 9
c.STmodel ................................ 10
calc.FX ................................ 12
calc.FXtF2 ................................ 14
calc.mu.B ................................ 15
calc.tFX ................................ 17
calc.tFXF ................................ 18
calcSmoothTrends ......................... 20
coef.estCVSTmodel ..................... 22
coef.estimateSTmodel .................. 23
computeLTA ................................ 24
convertCharToDate ....................... 25
createCV ................................ 26
createDataMatrix ......................... 28
createLUR ................................ 30
createSTdata ............................ 30
createSTmodel .......................... 32
crossDist ................................ 35
defaultList ................................ 36
density.mcmcSTmodel .................. 37
detrendSTdata ........................... 38
dropObservations ......................... 40
est.cv.mesa .............................. 41
est.mesa.model .......................... 43
estimate.STmodel ........... 45
estimateBetaFields ............ 48
estimateCV.STmodel ............ 50
evalCovFuns ............................. 53
expandF ................................ 54
genGradient ............................. 56
loglikeST ................................ 57
loglikeSTdim ............................ 58
loglikeSTgetPars ....................... 60
loglikeSTGrad ........................... 61
loglikeSTnames .......................... 63
make.sigma.B ............................ 64
makeCholBlock ......................... 68
makeSigmaB .............................. 70
makeSigmaNu ............................. 71
MCMC.mesa.model ...................... 73
MCMC.STmodel ........................... 74
mesa.data.raw ......................... 77
mesa.model .............................. 79
namesCovFuns ......................... 80
norm2 ................................... 82
parsCovFuns ............................. 83
plot.density.mcmcSTmodel ........... 84
topics documented:

- plot.mcmcSTmodel ........................................... 85
- plot.predCVSTmodel ........................................ 86
- plot.STdata .................................................. 89
- plot.SVDcv ................................................... 92
- pred.mesa.model ........................................... 93
- predict.STmodel ............................................ 94
- predictNaive .................................................. 97
- print.estCVSTmodel ......................................... 99
- print.estimateSTmodel ...................................... 100
- print.mcmcSTmodel ......................................... 101
- print.predCVSTmodel ....................................... 102
- print.predictSTmodel ....................................... 103
- print.STdata ................................................ 103
- print.STmodel ............................................... 104
- print.summary.estCVSTmodel ................................ 105
- print.summary.mcmcSTmodel ................................ 106
- print.summary.predCVSTmodel ................................ 106
- print.summary.STdata ....................................... 107
- print.summary.STmodel ..................................... 108
- print.SVDcv .................................................. 108
- processLocation ............................................. 109
- processLUR .................................................. 111
- qqnorm.predCVSTmodel ..................................... 113
- removeSTcovarMean ......................................... 115
- scatterPlot .................................................... 116
- scatterPlot.predCVSTmodel .................................. 117
- simulate.STmodel ........................................... 120
- stCheckClass ................................................ 121
- stCheckCovars .............................................. 122
- stCheckFields ............................................... 123
- stCheckObs ................................................... 124
- stCheckSTcovars ............................................ 125
- sumLogDiag .................................................. 126
- summary.estCVSTmodel ..................................... 127
- summary.mcmcSTmodel ..................................... 128
- summary.predCVSTmodel ................................... 129
- summary.STdata ............................................. 130
- summary.STmodel .......................................... 131
- SVDmiss ....................................................... 132
- SVDsmooth .................................................... 133
- updateCovf .................................................. 136
- updateTrend.STdata ......................................... 138

Index ............................................................. 141
SpatioTemporal-package

Spatio-Temporal Modelling

Description

Package for spatio-temporal modelling. Contains functions that estimate, simulate and predict from the model described in (Szpiro et.al., 2010; Sampson et.al., 2011; Lindström et.al., 2010). The package also contains functions that handle missing data SVD in accordance with (Fuentes et.al. 2006).

Package: SpatioTemporal
Type: Package
Version: 1.1.7
Date: 2013-08-12
License: GPL version 2 or newer
LazyLoad: yes

Examples in the package uses data from the Multi-Ethnic Study of Atherosclerosis and Air Pollution (MESA Air), (Cohen et.al., 2009).

Changelog

1.1.7 Updates: Handling of log-Gaussian fields

- Updated several functions to allow for prediction and CV of log-Gaussian fields. Updated functions: predict.STmodel, print.predictSTmodel, plot.predictSTmodel, predictCV.STmodel, print.predCVSTmodel, summary.predCVSTmodel, plot.predCVSTmodel, qnorm.predCVSTmodel, and scatterPlot.predCVSTmodel.
- Updated predict.STmodel to compute temporal averages, and return both prediction and variance of the averages. Both for Gaussian and log-Gaussian data.

1.1.6 Updates: sparse-Matrices and temporal basis functions

- Allows for sparse matrices in makeSigmaB and makeSigmaNu; this reduces the memory footprint and execution time for loglikeST, predict.STmodel, and estimate.STmodel.
- Added function that does regression estimates of the beta-coefficients: estimateBetaFields.
- Altered computation of CV-statistics in SVDsmoothCV.
- Added boxplot.SVDCv for illustration of CV-statistics from SVDsmoothCV.
- Replaced updateSTdataTrend with updateTrend.STdata and updateTrend.STmodel that also allows for temporal trends defined using functions.
- Updated SVDsmooth, SVDsmoothCV, and calcSmoothTrends to return both the trend and the smoothing function used to compute the trends, simplifying interpolation at unobserved time-points.
- Updated example data-sets.
• Added options for computation of temporal averages (incl. variances) to `predict.STmodel` and `predictCV.STmodel`.

1.1.5 Major bug fixes:
• In `predict.STmodel`, predictions now always use the trend given in object, ignoring the trend object in STdata. Prediction at dates in STdata are computed using the smoothing function that defines the trend; see `updateTrend.STmodel` for details.
• In `summary.predCVSTmodel`, code previously divided by the wrong variance when computing adjusted R2 using the pred.naive option.
• In `summary.predCVSTmodel`, code previously returned statistics even for dates without observations when using by.date=TRUE.
• In `plot.STdata` and `plot.STmodel` code now accounts for missing time-points when computing acf and pacf.

1.1.4 Added plot functions/Minor fixes:
• Added `scatterPlot.STdata`, `scatterPlot.STmodel`, and `scatterPlot.predCVSTmodel` for plotting observations/residuals against covariates.
• Added `plot.mcmcSTmodel`, `density.mcmcSTmodel`, and `plot.density.mcmcSTmodel` for plotting of MCMC results.
• Added `qqnorm.STdata`, `qqnorm.STmodel`, and `qqnorm.predCVSTmodel` for plotting of data and CV-prediction results.
• Added a restart option to `estimate.STmodel` allowing for restarts of optimisation in cases on bad optimisation.

1.1.3 Minor changes/Bug fixes:
• Fixed stupid mistake in `predictNaive` that caused computations to take unnecessarily long.

1.1.2 Minor changes/Bug fixes:
• Fixed a bug in `SVDsmooth`, that caused the values in the temporal smooths to depend on the number of unobserved time points. This also affects `calcSmoothTrends` and `updateSTdataTrend` when the option extra.dates is in use.
• Fixed bug in `simulate.STmodel` that caused NA values when simulating at unobserved sites.
• Fixed bug in `predict.STmodel` that could cause errors when predicting at unobserved sites.
• Fixed bug in `predictCV.STmodel` and `predict.STmodel`; these will now handle predictions at locations with incomplete nugget covariates.
• Updated `c.STmodel` and `predict.STmodel` to avoid errors/warnings due to more complex nugget models.
• Replaced warning in `createSTdata` when extra.dates!=NULL and n.basis=NULL with a message.

1.1.1 Bug fixes:
• `c.STmodel` will now combine STmodel objects with identical covariate scaling.

1.1.0 Major Changes:
• Changed the return of the variances for beta in `predict.STmodel`.
• Reduced the memory footprint of `predict.STmodel`. 
• Error checks in `c.STmodel` and `predict.STmodel`, combination of STmodel objects with different covariate scaling is **NOT** possible.

**1.0.7 Added:**

• New plot function: `plot.predCVSTmodel`.
• `coef.estimateSTmodel` and `coef.estCVSTmodel` functions that extract estimated parameters.
• Parameters for `predict.STmodel` and `predictCV.STmodel` can be specified using `estimateSTmodel` or `estCVSTmodel` objects.
• An `lwd` option to `plot.predictSTmodel`.
• A short introductory vignette as complement to the full tutorial.

**1.0.6 Bug fixes:**

• `predictNaive` now works for only one locations.
• `detrendSTdata` now works for different regions.

**1.0.5 Added packages** `maps` and `plotrix` to suggested packages.

**1.0.4 Bug fixes:**

• Prediction for leave-one-out CV.
• Stop `updateCovf` crashing in `Rscript/R CMD BATCH`.

**1.0.3 Minor bug fixes**

**1.0.2 Updated documentation and vignette**

**1.0.0 Major change, most old functions are now deprecated. New features:**

• Different covariance functions
• Nuggets in the beta-fields
• Different nuggets for different locations in the nu-field.
• Different coordinates for beta and nu-fields, allowing for precomputed deformations
• Covariates can be specified using `formula`-objects

**0.9.2 Minor updates - no user visible changes**

**0.9.0 First CRAN-release**

**0.1.0 First released version, short course at TIES-2010**

**Note**

Data used in the examples has been provided by the Multi-Ethnic Study of Atherosclerosis and Air Pollution (MESA Air). Details regarding the data can be found in Cohen et al. (2009).

Although the research described in this article has been funded wholly or in part by the United States Environmental Protection Agency through assistance agreement CR-834077101-0 and grant RD831697 to the University of Washington, it has not been subjected to the Agency’s required peer and policy review and therefore does not necessarily reflect the views of the Agency and no official endorsement should be inferred.

Travel for J. Lindström has been paid by STINT (The Swedish Foundation for International Cooperation in Research and Higher Education) Grant IG2005-2047.

Additional funding was provided by grants to the University of Washington from the Health Effects Institute (4749-RFA05-1A/06-10) and the National Institute of Environmental Health Sciences (P50 ES015915).
Author(s)
Johan Lindström, Adam Szpiro, Paul D. Sampson, Silas Bergen, Assaf P. Oron

References

Examples
### For a short introduction see:
```r
## Not run:
vignette("ST_intro",package="SpatioTemporal")

## End(Not run)
```

### For a worked out data-analysis example see the tutorial.
```r
##NOTE: This vignette is still work in progress
## Not run:
vignette("Tutorial",package="SpatioTemporal")

## End(Not run)
```

---

**blockMult**

*Multiplication of Block Diagonal Matrix and Vector*

**Description**
Computes the matrix product between a block diagonal square matrix and a column vector (or matrix).
Usage

\[ \text{blockMult}(\text{mat}, \ X, \ n\text{.blocks} = 1, \) \\
\text{block.sizes} = \text{rep}(\text{dim}(\text{mat})[1]/\text{n\text{.blocks}}, \ \text{n\text{.blocks}}) \]

Arguments

- **mat**: A block diagonal, square matrix.
- **x**: Vector or matrix to multiply by \( \text{mat} \); \( \text{length}(\text{x}) \) needs to be a multiple of \( \text{dim}(\text{mat})[1] \).
- **n.blocks**: Number of diagonal blocks in \( \text{mat} \) (or \( \text{R} \)). Defaults to 1 (i.e. a full matrix) if neither \( \text{n\text{.blocks}} \) nor \( \text{block.sizes} \) given, o.w. it defaults to \( \text{length}(\text{block.sizes}) \).
- **block.sizes**: A vector of length \( \text{n\text{.blocks}} \) with the size of each of the diagonal blocks. If not given it will assume equal-sized blocks.

Value

Returns \( \text{mat} \times \text{x} \).

Author(s)

Johan Lindström

See Also

Other basic linear algebra: `crossDist`, `dotProd`, `invCholBlock`, `makeCholBlock`, `norm2`, `solveTriBlock`, `sumLog`, `sumLogDiag`

Other block matrix functions: `calc.FX`, `calc.FXTF2`, `calc.iS.X`, `calc.mu.B`, `calc.tFX`, `calc.tFXF`, `calc.X.iS.X`, `invCholBlock`, `makeCholBlock`, `makeSigmaB`, `makeSigmaNu`, `solveTriBlock`

Examples

```r
# create a matrix
mat <- cbind(c(1,0,0), c(0,2,1), c(0,1,2))
# define the number of blocks and block sizes
block.sizes <- c(1,2)
n.blocks <- length(block.sizes)

# define a X vector
X <- matrix(c(1,2,3,1,1,1), 3, 2)

# compute mat %*% X
blockMult(mat, X, n.blocks, block.sizes)
# or the old fashioned way
mat %*% X
```
Description

boxplot method for class estCVSTmodel.

Usage

```r
## S3 method for class 'estCVSTmodel'
boxplot(x,
       plot.type = c("cov", "reg", "all"), ...)
```

Arguments

- `x` estCVSTmodel/STmodel object to boxplot.
- `plot.type` One of "cov", "reg", "all"; should we boxplot covariance, regression or all parameter estimates.
- `...` Additional parameters passed to `boxplot`.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other estCVSTmodel methods: `coef.estCVSTmodel, estimateCV, estimateCV.STmodel, predictCV, predictCV.STmodel, print.estCVSTmodel, print.summary.estCVSTmodel, summary.estCVSTmodel`

Examples

```r
##cross-validation load data
data(est.cv.mesa)
##...and old estimates
data(est.mesa.model)
##estimated parameters
par.cov <- coef(est.mesa.model, "cov")
par.all <- coef(est.mesa.model)

##boxplot of the different estimates from the CV
par(mfrow=c(1,1), mar=c(3,2,5,2), las=2)
boxplot( est.cv.mesa, plot.type="cov", boxwex=.5)
##compare with estimates for all data
points((1:length(par.cov$par))+.35, par.cov$par, pch=19, col=2)
```
c.STmodel

Combine Several STmodel/STdata Objects

Description

Combines several locations and covariates for several STmodel/STdata objects. Temporal trend, observations and covariance model (both spatial and spatio-temporal) are taken from the first object in the call. Any additional covariates/trends/observations not present in the first argument are dropped from the additional arguments without warning. Locations and covariates (both spatial and spatio-temporal) from additional objects are added to those in the first object.

Usage

```r
# S3 method for class 'STmodel'
c(..., recursive = FALSE)
```

Arguments

- `...` STmodel and STdata objects to combine, the first object has to be a STmodel.
- `recursive` For S3 compatibility; the function will ALWAYS run recursively

Details

For additional STdata objects the covariates are transformed according to STmodel$scale.covars of the first object, see `createSTmodel`.

For STmodel objects can not be combined if either has scaled covariates.

Value

An updated STmodel object.
Author(s)
Johan Lindström

See Also
Other STdata functions: `createDataMatrix`, `createSTdata`, `createSTmodel`, `detrendSTdata`, `estimateBetaFields`, `removeSTcovarMean`, `updateSTdataTrend`, `updateTrend.STdata`, `updateTrend.STmodel`


Examples
```r
# Load the data
data(mesa.data.raw)
# And create STdata-object
mesa.data <- createSTdata(mesa.data.raw$obs, mesa.data.raw$X, n.basis=2,
                           SpatioTemporal=mesa.data.raw$"lax.conc.1500")

# Keep only observations from the AQS sites
ID.AQS <- mesa.data$ covars$ID[ mesa.data$ covars$ type == "AQS" ]
mesa.data$obs <- mesa.data$obs[ mesa.data$obs$ID %in% ID.AQS,]

# Model specification
LUR <- list(~log10.m.to.a1 + s2000.pop.div.10000 + km.to.coast,
             ~km.to.coast, ~km.to.coast)
locations <- list(coords=c("x","y"), long.lat=c("long","lat"), others="type")

# Create reduced model, without and with a spatio-temporal covariate.
mesa.model <- createSTModel(mesa.data, LUR=LUR, locations=locations,
                              strip=TRUE)
mesa.model.ST <- createSTModel(mesa.data, LUR=LUR, ST=1,
                                locations=locations, strip=TRUE)

# And non stripped version
mesa.model.full <- createSTModel(mesa.data, LUR=LUR, ST=1,
                                  locations=locations)

# Combine, this adds the missing locations
mesa.model$locations$ID <- c(mesa.model, mesa.data)$locations$ID

# Or we could study the summary output
print(c(mesa.model.ST, mesa.data))

# No change since we're trying to adding existing sites
mesa.model.full$locations$ID <- c(mesa.model.full, mesa.data)$locations$ID
```
We can also combine two STmodels

```r
print(c(mesa.model, mesa.model.full))
```

---

### calc.FX

**Compute Matrix Product Between Temporal Trends and a LUR components**

#### Description

Computes the matrix products between a sparse matrix \( F \) containing the temporal trends and a list of land-use-regression components.

See the examples for details.

#### Usage

```r
calc.FX(F, LUR, loc.ind)
```

#### Arguments

- **F**: A (number of obs.) - by - (number of temporal trends) matrix containing the temporal trends. Usually `mesa.model$F`, where `mesa.model` is obtained from `createSTmodel`.
- **LUR**: A list of matrices, usually `mesa.model$X`. Each matrix in the list should have the same number of rows, but the number of columns may vary.
- **loc.ind**: A vector indicating which location each row in \( F \) corresponds to, usually `mesa.model$obs$idx`.

#### Value

Returns a matrix

#### Author(s)

Johan Lindström and Adam Szpiro

#### See Also

Other block matrix functions: `blockMult`, `calc.FXtF2`, `calc.iS.X`, `calc.mu.B`, `calc.tFX`, `calc.tFXF`, `calc.X.iS.X`, `invCholBlock`, `makeCholBlock`, `makeSigmaB`, `makeSigmaNu`, `solveTriBlock`

Other temporal trend functions: `calc.FXtF2`, `calc.tFX`, `calc.tFXF`, `expandF`
Examples

### This starts with a couple of simple examples, more elaborate examples with real data can be found further down.

#### create a trend
```r
trend <- cbind(1:5, sin(1:5))
```
#### an index of locations
```r
idx <- c(rep(1:3, 1:2), 2:3)
```
#### a list of time points for each location/observation
```r
T <- c(rep(1:3, each = 3), 4, 4, 5, 5)
```
#### create a list of matrices
```r
X <- list(matrix(1, 3, 1), matrix(runif(6), 3, 2))
```

#### expand the F matrix to match the locations/times in idx/T.
```r
F <- trend[T,]
```

#### compute F %*% X
```r
FX <- calc.FX(F, X, idx)
```

#### alternatively this can be computed as block matrix

#### times each (expanded) temporal trend
```r
Fexp <- expandF(F, idx)
```

#### Fexp is a sparse 'Matrix', we need to use cBind.
```r
FX.alt <- cBind(Fexp[,1:3] %*% X[[1]], Fexp[,4:6] %*% X[[2]])
```

#### compare results

#### some examples using real data
```r
data(mesa.model)
```

#### Some information about the size(s) of the model.
```r
dim <- loglikeStdim(mesa.model)
```

#### compute F %*% X
```r
FX <- calc.FX(mesa.model$F, mesa.model$LUR, mesa.model$obs$idx)
```

#### The resulting matrix is

#### (number of time points) - by - (number of land use covariates)
#### where the number of land use covariates are computed over all the
two + intercept temporal trends.
#### Each column contains the temporal trend for the observations
#### multiplied by the corresponding LUR-covariate
```r
par(mfrow=c(3,1))
plot(FX[,2])
points(mesa.model$LUR[[1]][mesa.model$obs$idx,2] * mesa.model$F[,1], col=2, pch=3)
plot(FX[,dim$p[1]+1])
points(mesa.model$LUR[[2]][mesa.model$obs$idx,1] * mesa.model$F[,2], col=2, pch=3)
```
points(mesa.model$LUR[3][2]*
      mesa.model$F[,3], col=2, pch=3)

## If the regression parameters, alpha, are known (or estimated)
## The intercept part of the model is given by FX %*% alpha

calc.FXF2  

Compute Quadratic Form Between Temporal Trends and Sigma B

Description

Computes the quadratic form between a sparse matrix \( F \) containing the temporal trends and the covariance matrix for the beta fields (Sigma_B). Or possibly the product between two different \( F \)'s and a cross-covariance matrix.

See the examples for details.

Usage

\[
\text{calc.FXF2}(F, \text{mat}, \text{loc.ind, F2 = F, loc.ind2 = loc.ind})
\]

Arguments

- \( F, F2 \)  
  (number of obs.) - by - (number of temporal trends) matrices containing the temporal trends. Usually \( \text{mesa.model}$F \), where \( \text{mesa.model} \) is obtained from \( \text{createSTmodel} \).
- \( \text{mat} \)  
  A block diagonal, with equal size blocks. The number of blocks need to equal \( \text{dim}(F)[2] \).
- \( \text{loc.ind, loc.ind2} \)  
  A vector indicating which location each row in \( F \) corresponds to, usually \( \text{mesa.model}$obs$idx \).

Value

Returns a square matrix with side \( \text{dim}(F)[1] \)

Author(s)

Johan Lindström and Adam Szpiro

See Also

Other block matrix functions: \( \text{blockMult, calc.FX, calc.iS.X, calc.mu.B, calc.tFX, calc.tXF, calc.X.iS.X, invCholBlock, makeCholBlock, makeSigmaB, makeSigmaNu, solveTriBlock} \)

Other temporal trend functions: \( \text{calc.FX, calc.tFX, calc.tXF, expandF} \)
Examples

# create a trend
trend <- cbind(1:5, sin(1:5))
# an index of locations
idx <- c(rep(1:3, 1:2, 2:3)
idx2 <- c(rep(1:3, 2:2)
# a list of time points for each location/observation
T <- c(rep(1:3, each=3), 4, 5, 5)
T2 <- c(rep(1:3, each=2), 4, 5)

# expand the F matrix to match the locations/times in idx/T.
F <- trend[T,]
F2 <- trend[T2,]

# first column gives time and second location for each observation
cbind(T, idx)
# ... and for the second set
cbind(T2, idx2)

# create a cross covariance matrix
C <- makeSigmaB(list(c(1,1), c(1,5)), crossDist(1:max(idx), 1:max(idx2)))

# compute F %*% X %*% F'
FXTF2 <- calc.FxtF2(F, C, loc.ind=idx, F2=F2, loc.ind2=idx2)

# which is equivalent to
FXTF2.alt <- expandF(F, idx) %*% C %*% t( expandF(F2, idx2) )
range(FXTF2 - FXTF2.alt)

---

calc.mu.B

Matrix Multiplication with Block Matrices

Description

Computes either the product between a block diagonal, square matrix iS and a block matrix X; the quadratic form of a block diagonal, square matrix, t(X)*iS*X; or a block matrix multiplied by a vector, X*alpha.

Usage

calc.mu.B(X, alpha)

calc.iS.X(X, iS)

calc.X.iS.X(X, iS.X)
Arguments

- **X**
  A list of \( m \) matrices with which to form the block matrix; each matrix should be \( p[i] \) - by - \( n \).

- **alpha**
  A list of \( m \) vectors, with the \( i \):th vector being of length \( p[i] \).

- **is**
  A block diagonal, square matrix, with \( dm \) blocks each of size \( n \) - by - \( dn \).

- **is.X**
  Matrix containing the product of \( is \) and \( X \). Output from \( \text{calc.is.X} \).

Value

- matrix containing \( is^X, X'is^X, \) or \( Xalpha \).

Author(s)

- Johan Lindström and Adam Szpiro

See Also

- Other block matrix functions: \( \text{blockMult, calc.FX, calc.FXf2, calc.tFX, calc.tFXf, invCholBlock, makeCholBlock, makeSigmaB, makeSigmaNu, solveTriBlock} \)
- Other likelihood utility functions: \( \text{loglikeSTdim, loglikeSTgetPars, loglikeSTnames} \)

Examples

```r
## Create a block diagonal matrix, ...
is <- rbind(c(2,1,0,0), c(1,3,0,0),
            c(0,0,3,2), c(0,0,2,4))
## ... a block matrix ...
X <- list(matrix(c(1,2)), matrix(c(2,2,3,4),2,2))
## ... with alternative form, ...
Xt <- rbind(cbind(X[[1]], matrix(0,2,2)),
            cbind(matrix(0,2,1), X[[2]]))
## ... and a vector alpha.
alpha <- list(c(1), c(-2,1))

## Compute is * X
is.X <- calc.is.X(X, is)
## or
is %*% Xt

## Compute X'* is * X
calc.X.is.X(X, is.X)
## or
t(Xt) %*% is %*% Xt

## Compute X* alpha
calc.mu.B(X, alpha)
## or
cbind(X[[1]] %*% alpha[[1]], X[[2]] %*% alpha[[2]])
```
**Calculate Matrix Product Between Temporal Trends and a Matrix**

**Description**
Computes the matrix products between the transpose of a sparse matrix \( F \) containing temporal trends and a vector/matrix.

See the examples for details.

**Usage**

```r
calc.tFX(F, X, loc.ind, n.loc = max(loc.ind))
```

**Arguments**
- **F**: A (number of obs.) - by - (number of temporal trends) matrix containing the temporal trends. Usually `mesa.model$F`, where `mesa.model` is obtained from `createSTmodel`.
- **X**: A vector or matrix; needs to be a multiple of \( \text{dim}(F)[1] \).
- **loc.ind**: A vector indicating which location each row in \( F \) corresponds to, usually `mesa.model$obs$idx`.
- **n.loc**: Number of locations.

**Value**
Returns a matrix of size \( n.loc \times \text{dim}(F)[2] \)-by-\( \text{coden}.x \).

**Author(s)**
Johan Lindström and Adam Szpiro

**See Also**
Other block matrix functions: `blockMult, calc.FX, calc.FXtF2, calc.iS.X, calc.mu.B, calc.tFXF, calc.X.iS.X, invCholBlock, makeCholBlock, makeSigmaB, makeSigmaNu, solveTriBlock`

Other temporal trend functions: `calc.FX, calc.FXtF2, calc.tFXF, expandF`

**Examples**

```r
# This starts with a couple of simple examples, more elaborate examples
# with real data can be found further down.

trend <- cbind(1:5, sin(1:5))
# an index of locations
idx <- c(rep(1:3,3),1:2,2:3)
```
```r
## a list of time points for each location/observation
T <- c(rep(1:3, each=3), 4, 4, 5, 5)
## create a random observations matrix
obs <- rnorm(length(T))

## expand the F matrix to match the locations/times in idx/T.
F <- trend[T,]

## compute tF %*% obs
Tobs <- calc.tFX(F, obs, idx)
## or possibly expanded if we have unobserved, trailing locations
Tobs.exp <- calc.tFX(F, obs, idx, 5)

## alternatively this can be computed as observations for each location
## multiplied by the trend function at the corresponding time points.
Tobs.alt <- t(expandF(F, idx)) %*% obs

## compare results
print(cbind(Tobs, Tobs.exp))

## some examples using real data
data(mesa.model)

## Some information about the size(s) of the model.
dim <- loglikeStdim(mesa.model)

## compute F' %*% obs
Tobs <- calc.tFX(mesa.model$F, mesa.model$obs$obs, mesa.model$obs$idx)

## The resulting matrix contains 75 elements (3 temporal trend at 25 # sites). The first element are the observations at the first site
## multiplied by the constant temporal trend, e.g.
print(tTobs[1])
print(sum(mesa.model$obs$obs[ Mesa.model$obs$idx==1]))

## The 27:th element are the observations at the second site (27-25)
## multiplied by the first temporal trend (second element in F)
print(tTobs[dim$n.obs+2])
print(sum(mesa.model$obs$obs[ Mesa.model$obs$idx==2] *
          mesa.model$F[ Mesa.model$obs$idx==2,2]))
```

calc.tFXF

**Calculate Quadratic Form Between Temporal Trends and Sigma nu**

**Description**

Computes the quadratic form between a sparse matrix \( F \) containing the temporal trends and the covariance matrix for the residual fields (\( \Sigma_{nu} \)).
calc.tFXF

See the examples for details.

Usage

calc.tFXF(F, mat, loc.ind, n.blocks = 1,
       block.sizes = rep(dim(mat)[1]/n.blocks, n.blocks),
       n.loc = max(loc.ind))

Arguments

F          A (number of obs.) by (number of temporal trends) matrix containing the
temporal trends. Usually mesa.model$F, where mesa.model is obtained from
createSTmodel.
mat        A block diagonal, square matrix.
loc.ind    A vector indicating which location each row in F corresponds to, usually
mesa.model$obs$idx.
n.blocks   Number of diagonal blocks in mat (or R). Defaults to 1 (i.e. a full matrix) if nei-
her n.blocks nor block.sizes given, o.w. it defaults to length(block.sizes)).
block.sizes A vector of length n.blocks with the size of each of the diagonal blocks. If not
given it will assume equal size blocks.
n.loc      Number of locations.

Value

Returns a square matrix with side dim(F)[2]*n.loc

Author(s)

Johan Lindström and Adam Szpiro

See Also

Other block matrix functions: blockMult, calc.FX, calc.FXtF2, calc.is.X, calc.mu.B, calc.tFX,
calc.X.is.X, invCholBlock, makeCholBlock, makeSigmaB, makeSigmaNu, solveTriBlock
Other temporal trend functions: calc.FX, calc.FXtF2, calc.tFX, expandF

Examples

##create a trend
trend <- cbind(1:5,sin(1:5))
##an index of locations
idx <- c(rep(1:3,3),1:2,2:3)
##a list of time points for each location/observation
T <- c(rep(1:3,each=3),4,4,5,5)

##expand the F matrix to match the locations/times in idx/T.
F <- trend[T,]
calcSmoothTrends

Smooth Basis Functions for a STdata Object

Description

A front end function for calling SVDsmooth (and SVDsmoothCV), with either a STdata object or vectors containing observations, dates and locations.

Usage

calcSmoothTrends(STdata = NULL, obs = STdata$obs$obs, date = STdata$obs$date, ID = STdata$obs$ID, subset = NULL, extra.dates = NULL, n.basis = 2, cv = FALSE, ...)

Arguments

STdata A STdata/STmodel data structure containing observations, see mesa.data.raw. Use either this or the obs, date, and ID inputs.
obs A vector of observations.
date A vector of observation times.
ID A vector of observation locations.
subset A subset of locations to extract the data matrix for. A warning is given for each name not found in ID.
extra.dates Additional dates for which smooth trends should be computed (any duplicates will be removed).
n.basis Number of basis functions to compute, see SVDsmooth.
cv Also compute smooth functions using leave one out cross-validation, see SVDsmoothCV.
... Additional parameters passed to SVDsmooth and SVDsmoothCV; except fnc, which is always TRUE.
calcSmoothTrends

Details

The function uses createDataMatrix to create a data matrix which is passed to SVDsmooth (and SVDsmoothCV). The output can be used as

```r
STdata$trend = calcSmoothTrends(...)$trend, or
STdata$trend = calcSmoothTrends(...)$trend.cv[[i]]. However, it is recommended to use updateTrend.STdata.
```

Value

Returns a list with

```r
trend A data.frame containing the smooth trends and the dates. This can be used as the trend in STdata$trend.
trend.cv If cv==TRUE a list of data.frames; each one containing the smooth trend obtained when leaving one site out. Similar to SVDsmoothCV(data)$smoothSVD[[1]]).
trend.fnc,trend.fnc.cv Functions that produce the content of the above data.frames, see SVDsmooth.
```

Author(s)

Johan Lindström and Paul D. Sampson

See Also

Other SVD for missing data: boxplot.SVDcv, plot.SVDcv, print.SVDcv, summary.SVDcv, SVDmiss, SVDsmooth, SVDsmoothCV, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel

Examples

```r
#Let's load some data
data(mesa.model)

#Let's compute two smooth trend functions
trend <- calcSmoothTrends(mesa.model, n.basis=2)

##or with some other parameters for the splines
trend.alt <- calcSmoothTrends(mesa.model, n.basis=2, df=100)

##and study the trends
par(mfrow=c(2,1), mar=c(2.5,2.5,.5,.5))
plot(trend$trend$date, trend$trend$V1, type="1", ylab="", xlab="", ylim=range(c(trend$trend$V1, trend$trend$V2)))
lines(trend$trend$date, trend$trend$V2, col=2)
plot(trend.alt$trend$date, trend.alt$trend$V1, type="1", ylab="", xlab="", ylim=range(c(trend.alt$trend$V1, trend.alt$trend$V2)))
lines(trend.alt$trend$date, trend.alt$trend$V2, col=2)

#Let's exclude locations with fewer than 100 observations
```
## coef.estCVSTmodel

Returns estimated parameters for each CV-group.

### Description

`coef` method for class `estCVSTmodel`.

### Usage

```r
## S3 method for class 'estCVSTmodel'
coef(object, 
  pars = c("all", "cov", "reg"), ...) 
```

### Arguments

- **object**: `estCVSTmodel` object from which to extract estimated parameters.
- **pars**: One of "cov", "reg", "all"; which parameters to extract.
- ...: Ignored additional arguments.
**Value**

Nothing

**Author(s)**

Johan Lindström

**See Also**

Other estCVSTmodel methods: `boxplot.estCVSTmodel, estimateCV, estimateCV.STmodel, predictCV, predictCV.STmodel, print.estCVSTmodel, print.summary.estCVSTmodel, summary.estCVSTmodel`

**Examples**

```r
# load data
data(est.cv.mesa)
# extract all parameters
coef(est.cv.mesa)
# extract only covariance parameters
coef(est.cv.mesa, pars="cov")
```

---

**Description**

`coef` method for class estimateSTmodel.

**Usage**

```r
## S3 method for class 'estimateSTmodel'
coef(object, 
     pars = c("all", "cov", "reg"), ...)
```

**Arguments**

- `object`: estimateSTmodel object from which to extract estimated parameters.
- `pars`: One of "cov", "reg", "all"; which parameters to extract.
- `...`: Ignored additional arguments.

**Value**

Estimated parameters.

**Author(s)**

Johan Lindström
computeLTA

### Description
Computes the long term average of observations and cross-validated predictions for each of the sites in object. The long term averages are computed using only timepoints that have observations, this applies to both the observed and predicted. Also the function allows for a transformation: if requested the transformation is applied before the averaging.

### Usage
```r
computeLTA(object, transform = function(x) {
  return(x) })
```

### Arguments
- **object**: A predCVSTmodel object, the result of `predictCV.STmodel`.
- **transform**: Transform observations (without bias correction) and predictions before computing averages; e.g. `transform=exp` gives the long term averages as `mean(exp(obs))` and `mean(exp(pred))`.

### Value
Returns a (number of locations) - by - 4 matrix with the observed and predicted value (using the three different model parts) for each location.

### Author(s)
Johan Lindström

### See Also
Other cross-validation functions: `createCV`, `dropObservations`, `estimateCV`, `estimateCV.STmodel`, `predictCV`, `predictCV.STmodel`, `predictNaive`
Other predCVSTmodel functions: `estimateCV`, `estimateCV.STmodel`, `predictCV`, `predictCV.STmodel`
Examples

```r
# load data
data(pred.cv.mesa)

# compute long term averages of predictions and observations
pred.lta <- computeLTA(pred.cv.mesa)

## we can now compare observed and predicted averages at each site
plot(pred.lta[, "obs"], pred.lta[, "EX.mu"], pch=1,
     xlim=range(pred.lta), ylim=range(pred.lta),
     xlab="obs", ylab="predictions")
## for the different model components
points(pred.lta[, "obs"], pred.lta[, "EX.mu.beta"], pch=3, col=2)
points(pred.lta[, "obs"], pred.lta[, "EX"], pch=4, col=3)
abline(0,1)

## we could also try computations on the original scale
pred.lta <- computeLTA(pred.cv.mesa, exp)

## compare observed and predicted averages
plot(pred.lta[, "obs"], pred.lta[, "EX.mu"], pch=1,
     xlim=range(pred.lta), ylim=range(pred.lta),
     xlab="obs", ylab="predictions")
points(pred.lta[, "obs"], pred.lta[, "EX.mu.beta"], pch=3, col=2)
points(pred.lta[, "obs"], pred.lta[, "EX"], pch=4, col=3)
abline(0,1)
```

---

**convertCharToDate**

**Convert Character to Dates**

**Description**

Attempts to convert input vector to Date, if that fails tries to convert to double. If conversion induces NA the function returns NULL indicating a failure.

**Usage**

```r
convertCharToDate(x)
```

**Arguments**

- `x`  
  character vector to convert to dates

**Value**

- a vector of dates, or of doubles or NULL.

**Author(s)**

Johan Lindström
createCV

Define Cross-Validation Groups

Description

Creates a matrix that specifies cross-validation schemes.

Usage

```r
createCV(SStModel, groups = 10, min.dist = 0.1,
random = FALSE, subset = NA,
option = c("all", "fixed", "comco", "snapshot", "home"),
icv.vector = TRUE)
```

Arguments

- **SStModel**: Model object for which to determine cross-validation.
- **groups**: Number of cross-validation groups, zero gives leave-one-out cross-validation.
- **min.dist**: Minimum distance between locations for them to end up in separate groups. Points closer than min.dist will be forced into the same group. A high value for min.dist can result in fewer cross-validation groups than specified in groups.
- **random**: If FALSE repeated calls to the function will return the same grouping, if TRUE repeated calls will give different CV-groupings. Ensures that simulation studies are reproducible.
- **subset**: A subset of locations for which to define the cross-validation setup. Only sites listed in subset are dropped from one of the cross-validation groups; in other words sites not in subset are used for estimation and prediction of all cross-validation groups. This option is ignored if option!="all".

See Also

Other utility functions: `defaultList`

Examples

```r
# a vector of dates is returned as is
convertCharToDate(seq(as.Date("2012-01-01"),as.Date("2012-01-31"),by=5))

# if given as character vectors Date is returned
convertCharToDate(c("2012-01-01","2012-01-05","2012-01-10","2012-01-12"))

# if given as character vectors Date is returned
convertCharToDate(c("a","b","c"))
convertCharToDate(c("2012-01-01", "2012-01-05", "a", "2012-01-12"))
convertCharToDate(c(1,2,3,"d"))
```
createCV

option              For internal MESA Air usage, see Details below.

icv.vector          Attempt to return a vector instead of a matrix. If the same observation is in
                     several groups a matrix will still be returned.

Details

The number of observations left out of each group can be rather uneven; the main goal of createCV
is to create CV-groups such that the groups contain roughly the same number of locations ignoring
the number of observations at each location. If there are large differences in the number of observ-
ations at different locations one could use the subset option to create different CV-groupings for
different types of locations. If icv.vector=FALSE, the groups can then be combined as

\[ I_{\text{final}} = I_{1.1} | I_{1.2} | I_{1.3}. \]

The option input determines which sites to include in the cross-validation. Possible options are
"all", "fixed", "comco", "snapshot" and "home".

all               Uses all available sites, possibly subset according to subset. The sites will be grouped with
sites separated by less than \( \text{min.dist} \) being put in the same CV-group.

fixed            Uses only sites that have
                     \( \text{STmodel}\$\text{locations}\$\text{type} \%in\% c("AQS","FIXED"). \) Given the subsettting the sites will
                     be grouped as for all.

home             Uses only sites that have
                     \( \text{STmodel}\$\text{locations}\$\text{type} \%in\% c("HOME"). \) Given the subsettting the sites will be
                     grouped as for all.

comco            Uses only sites that have
                     \( \text{STmodel}\$\text{locations}\$\text{type} \%in\% c("COMCO"). \) The sites will be grouped together if they
                     are from the same road gradient. The road gradients are grouped by studying the name of
                     the sites. With "?" denoting one or more letters and "#" denoting one or more digits the names
                     are expected to follow "?.?#?#", for random sites, and "?.#?#?" for the gradients (with all but
                     the last letter being the same for the entire gradient).

Value

Return a vector, with each element giving the CV-group (as an integer) of each observation; Or a
(number or observations) - by - (groups) logical matrix; each column defines a cross-validation set
with the TRUE values marking the observations to be left out.

Author(s)

Johan Lindström

See Also

Other cross-validation functions: computeLTA, dropObservations, estimateCV, estimateCV.STmodel,
predictCV, predictCV.STmodel, predictNaive

Other STmodel functions: createDataMatrix, createSTmodel, dropObservations, estimateBetaFields,
loglikeST, loglikeSTdim, loglikeSTnaive, predictNaive, processLocation, processLUR,
processST, updateCovf, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel
createDataMatrix

Create a Data Matrix

description

Creates a data matrix from a STdata/STmodel object. Missing observations are marked as NA.
createDataMatrix

Usage

createDataMatrix(STdata = NULL, obs = STdata$obs$obs, date = STdata$obs$date, ID = STdata$obs$ID, subset = NULL)

Arguments

STdata A STdata/STmodel object containing observations. Use either this or the obs, date, and ID inputs.
obs A vector of observations.
date A vector of observation times.
ID A vector of observation locations.
subset A subset of locations to extract the data matrix for. A warning is given for each name not found in ID.

Value

Returns a matrix with dimensions (number of timepoints)-by-(number of locations). Row and column names of the matrix are taken as ID and sort(unique(date)) respectively.

Author(s)

Johan Lindström

See Also

Other data matrix: estimateBetaFields, mesa.data.raw, SVDmiss, SVDsmooth, SVDsmoothCV
Other STdata functions: c.STmodel, createSTdata, createSTmodel, detrendSTdata, estimateBetaFields, removeSTcovarMean, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel
Other STmodel functions: createCV, createSTmodel, dropObservations, estimateBetaFields, loglikeST, loglikeSTdim, loglikeSTnaive, predictNaive, processLocation, processLUR, processST, updateCovf, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel

Examples

##load the data
data(mesa.model)

##create a data matrix
M1 <- createDataMatrix(mesa.model)
dim(M1)
head(M1)

##create data matrix for only a few locations
M2 <- createDataMatrix(mesa.model, subset = c("60370002","60370016","60370113","60371002", "60371103","60371201","L001","L002"))
dim(M2)
createSTdata

**Description**

Creates a **STdata** object that can be used as input for `createSTmodel`. Names and dates are derived from the input data, either using predefined fields or `rownames / colnames`; for details see the sub-functions linked under the relevant Arguments.

**Usage**

```r
createSTdata(obs, covars, SpatioTemporal = NULL, 
transform.obs = function(x) { return(x) },
mean.0.ST = FALSE, n.basis = 0, extra.dates = NULL,
..., detrend = FALSE, region = NULL, method = NULL)
```

**Arguments**

- `obs`: Optional matrix with observations, e.g., sum of interest rates.
- `covars`: Optional matrix with covariates, e.g., market interest rates.
- `SpatioTemporal`: Optional matrix with spatio-temporal covariates, e.g., weather conditions.
- `transform.obs`: A function to transform the observations.
- `mean.0.ST`: Logical; whether to center the spatio-temporal data.
- `n.basis`: Integer; number of basis functions to use.
- `extra.dates`: Vector of dates for additional information.
- `...`: Additional arguments passed to the `createSTmodel` function.
- `detrend`: Logical; whether to detrend the data.
- `region`: Argument for spatial data handling.

**Value**

A **STdata** object with added fields `lur`, or `ST` and `ST.all`.

**Author(s)**

Johan Lindström

---

createLUR

**Add Covariate Fields to STdata Object.**

**Description**

Extracts the requested geographic and spatio-temporal covariates from a **STmodel** object and formats them into suitable matrices. For **INTERNAL** use by `createSTmodel`.

**Usage**

```r
createLUR(STdata, LUR.list)
createST(STdata, ST.list)
```

**Arguments**

- `STdata`: **STdata** object with observations, covariates, trends, etc; see `mesa.data.raw`.
- `LUR.list`: Specification of covariates; e.g. output from `processLUR`.
- `ST.list`: Specification of spatio-temporal covariates; e.g. output from `processST`.

**Value**

**STdata** with added fields `LUR`, or `ST` and `ST.all`.

**Author(s)**

Johan Lindström

---

head(M2)
createSTdata

Arguments

obs Either a data.frame with fields obs, date, ID giving observations, time-points and location names; or a matrix, e.g. output from createDataMatrix.
covars matrix/data.frame of covariates; should include both geographic covariates and coordinates of all locations, see stCheckCovars.
SpatioTemporal possible spatio-temporal covariate, see stCheckSTcovars.
transform.obs function to apply to the observations, defaults to an identity transform. Possible options are log, sqrt, and exp.
mean.O ST Call removeSTcovarMean to produce a mean-zero spatio-temporal covariate?
n.basis Number of temporal components in the smooth trends computed by updateTrend.STdata, if NULL no trend is computed (implies only a constant).
extra.dates Additional dates for which smooth trends should be computed, used by updateTrend.STdata. If n.basis=NULL this will force n.basis=0; since the dates are stored in the trend..
... Additional parameters passed to updateTrend.STdata.
detrend Use detrendSTdata to remove a temporal trend from the observations; requires n.basis!=NULL.
region,method Additional parameters passed to detrendSTdata.

Value

A STdata object with, some or all of, the following elements:
covars Geographic covariates, locations and names of the observation locations (the later in covars$ID), createSTmodel will extract covariates (land use regressors), observations locations, etc from this data.frame when constructing the model specification.
trend The temporal trends with one of the columns being named date, preferably of class Date providing the time alignment for the temporal trends.
obs A data.frame with columns:
  obs The value of each observation.
  date The observations time, preferably of class Date.
  ID A character-class giving observation locations; should match elements in locations$ID.
SpatioTemporal A 3D-array of spatio-temporal covariates, or NULL if no covariates exist. The array should be (number of timepoints) - by - (number of locations) - by - (number of covariates) and provide spatio-temporal covariates for all space-time locations, even unobserved ones (needed for prediction). The rownames of the array should represent dates/times and colnames should match the observation location names in covars$ID.
old.trend,fit.trend Additional components added if the observations have been detrended, see detrendSTdata.
Author(s)

Johan Lindström and Assaf P. Oron

See Also

Other STdata functions: c.STmodel, createDataMatrix, createSTmodel, detrendSTdata, estimateBetaFields, removeSTcovarMean, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel

Other STdata methods: plot.STdata, plot.STmodel, print.STdata, print.summary.STdata, qqnorm.predCVSTmodel, qqnorm.STdata, qqnorm.STmodel, scatterPlot.predCVSTmodel, scatterPlot.STdata, scatterPlot.STmodel, summary.STdata

Examples

```r
## load the raw data
data(mesa.data.raw)

## extract observations and covariates
obs <- mesa.data.raw$obs
covars <- mesa.data.raw$x

## list with the spatio-temporal covariates
ST.list <- list(lax.conc.1500=mesa.data.raw$lax.conc.1500)

## create STdata object
mesa.data <- createSTdata(obs, covars, SpatioTemporal=ST.list)
print(mesa.data)

## create object with mean 0 spatio temporal covariate
mesa.data.2 <- createSTdata(obs, covars, SpatioTemporal=ST.list,
                              mean.0=TRUE)
print(mesa.data.2)

## create object with mean 0 spatio temporal covariate, and
## trend with two components, and additional dates (every seventh day)
extra.dates <- seq(min(as.Date(rownames(obs))),
                   max(as.Date(rownames(obs))), by=7)
mesa.data.3 <- createSTdata(obs, covars, n.basis=2, extra.dates=extra.dates)
print(mesa.data.3)
```

createSTmodel

Construct STmodel Object

Description

Creates a STmodel object that can be for estimation and prediction. For details see the sub-functions linked under the relevant Arguments.
Usage

createSTmodel(STdata, LUR = NULL, ST = NULL,
cov.beta = list(covf = "exp", nugget = FALSE),
cov.nu = list(covf = "exp", nugget = TRUE, random.effect = FALSE),
locations = list(coords = c("x", "y"), long.lat = NULL, coords.beta = NULL,
  coords.nu = NULL, others = NULL),
strip = FALSE, scale = FALSE, scale.covars = NULL)

Arguments

STdata       STdata object with observations, covariates, trends, etc; see createSTdata or
mesa.data.raw for an example.
LUR          Specification of covariates for the beta-fields, see processLUR.
ST           Specification of spatio-temporal covariates, see processST.
cov.beta, cov.nu
  Specification of the covariance functions, see updateCovf.
locations    Specification of the sites (both monitored and un-monitored), see processLocation.
strip        Should unobserved locations be dropped?
scale        Scale the covariates? If TRUE all non-factor covariates are scaled after the loca-
  tions have been extracted but before constructing the covariate matrix for the
  beta-fields. (NOTE: If set to TRUE this scales the LUR.all elements to mean=0,
  sd=1).
scale.covars  list with elements mean and sd giving the mean and standard deviation to use
  when scaling the covariates. Computed from STdata$covars if not given.

Details

The object holds observations, trends, geographic, and spatio-temporal covariates, as well as a
number of precomputed fields that speed up log-likelihood evaluations. To improve performance
the locations are also reorder so that observed locations come before unobserved.

Value

A STmodel object, see mesa.model for an example.

Author(s)

Johan Lindström

See Also

Other STdata functions: c.STmodel, createDataMatrix, createSTdata, detrendSTdata, estimateBetaFields,
removeSTcovarMean, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel
Other STmodel functions: createCV, createDataMatrix, dropObservations, estimateBetaFields,
loglikeST, loglikeSTdim, loglikeSTnaive, predictNaive, processLocation, processLUR,
processST, updateCovf, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel

Examples

```r
# load the data
data(mesa.data.raw)
## and create STdata-object
mesa.data <- createSTdata(mesa.data.raw$obs, mesa.data.raw$X, n.basis=2,
                          SpatioTemporal=mesa.data.raw$"lax.conc.1500")

## define land-use covariates
LUR <- list(-log10.m.to.a1+s2000.pop.div.10000+km.to.coast,
             ~km.to.coast, ~km.to.coast)
## and covariance model
cov.beta <- list(covf="exp", nugget=FALSE)
cov.nu <- list(covf="exp", nugget=TRUE, random.effect=FALSE)
## which locations to use
locations <- list(coords=c("x","y"), long.lat=c("long","lat"), others="type")

## create object
mesa.model <- createSTmodel(mesa.data, LUR=LUR, ST="lax.conc.1500",
cov.beta=cov.beta, cov.nu=cov.nu,
locations=locations)
print(mesa.model)
## This is the same as data(mesa.model)

## lets try some alternatives:
model.none <- createSTmodel(mesa.data, LUR=NULL, ST=NULL)
print(model.none)

## Specify LUR:s using numbers
names(mesa.data$covars)
model.diff <- createSTmodel(mesa.data, LUR=list(c(7,10,11,12),11:12,11:12),
                          ST=1)
print(model.diff)

## Same covariates for all temporal trends, calling by name
## but with different covariance models for each trend, and nugget that depends
## on monitor type
model.same <- createSTmodel(mesa.data, LUR=c("log10.m.to.a1", "log10.m.to.road",
                                "km.to.coast","s2000.pop.div.10000"),
                        ST="lax.conc.1500", cov.nu=list(nugget="type"),
cov.beta=list(covf=c("exp",exp2","iid"),
              nugget=c(FALSE, FALSE, TRUE))
print(model.same)
```
crossDist

Computed the Euclidian Distance Matrix

Description
Computed the Euclidian distance matrix between to sets of points.

Usage
\[ \text{crossDist(coord1, coord2 = coord1)} \]

Arguments
coord1, coord2 Matrices with the coordinates of locations, between which distances are to be computed.

Value
A \( \text{dim(coord1)[1]} \)-by-\( \text{dim(coord2)[1]} \) distance matrix.

Author(s)
Johan Lindström

See Also
Other basic linear algebra: blockMult, dotProd, invCholBlock, makeCholBlock, norm2, solveTriBlock, sumLog, sumLogDiag
Other covariance functions: evalCovFuns, makeSigmaB, makeSigmaNu, namesCovFuns, parsCovFuns, updateCovf

Examples
```r
## First create some random locations
x <- rnorm(5)
y <- rnorm(5)

## Compute distance matrix
D <- crossDist( cbind(x,y) )

## Or distance between different locations
X <- matrix(rnorm(6),3,2)
Y <- rbind(X, matrix(rnorm(8),4,2))
Dcross <- crossDist(X, Y)

## Or distances between coordinates in R3
C1 <- matrix(rnorm(9),3,3)
C2 <- matrix(rnorm(12),4,3)
Dcross.R3 <- crossDist(C1, C2)
```
**defaultList**  
*Add Default Elements to Incomplete list*

**Description**

Given two lists elements (by name) missing from the first are copied from the second list (if present). Is used to create default lists, ensuring that all elements expected in the list are present with reasonable values (if not user specified).

**Usage**

```r
defaultList(x, prototype = list())
```

**Arguments**

- `x`  
  A list

- `prototype`  
  A list with named elements, any elements missing from `x` are replaced with corresponding elements from prototype.

**Value**

Updated version of `x`

**Author(s)**

Johan Lindström

**See Also**

Other utility functions: `convertCharToDate`

**Examples**

```r
defaultList(list(a=1,b=4), list(a=3,c="a",d=4))
```
Description

density method for class mcmcSTmodel.

Usage

## S3 method for class 'mcmcSTmodel'
density(x, BurnIn = 0,
estSTmodel = NULL, ...)

Arguments

- **x**: mcmcSTmodel object
- **BurnIn**: Number of initial points to ignore.
- **estSTmodel**: Either a estimateSTmodel object from estimate.STmodel or a matrix with parameter-estimates and standard deviations, such as the output from coef.estimateSTmodel. If given as a matrix, it should have columns named "par" and "sd", and rows named after the parameters.
- **...**: Additional parameters passed to density.

Details

Computes kernel density estimates for the MCMC-parameters; as well as approximate Gaussian densities based on the Fischer-information.

Value

List containing density estimate and Gaussian densities for all model parameters.

Author(s)

Johan Lindström

See Also

Other mcmcSTmodel methods: MCMC, MCMC.STmodel, plot.density.mcmcSTmodel, plot.mcmcSTmodel, print.mcmcSTmodel, print.summary.mcmcSTmodel, summary.mcmcSTmodel
Examples

```r
# load estimation results
data(est.mesa.model)
# and MCMC results instead
data(MCMC.mesa.model)

# compute density estimates for the results, and use the Gaussian approximation
# based on Fischer information as reference.
dens <- density(MCMC.mesa.model, estSTmodel=est.mesa.model)

# all the estimated densities
str(dens,1)

# or results for one parameter
dens[[1]]

# plot density functions
plot(dens)

# for a different parameter, along with Gaussian approx
plot(dens, 3, norm.col="red")

# all covariance parameters
par(mfrow=c(3,3),mar=c(4,4,2.5,.5))
for(i in 9:17){
  plot(dens, i, norm.col="red")
}
```

detrendSTdata

Removes Temporal Trend from Observations in a STdata Object

Description

Removes an estimated time-trend from the observations in a STdata object. Returns a modified STdata object with no trend; the new object can be used to fit a simpler model.

Usage

```r
detrendSTdata(STdata, region = NULL, method = lm, ...)
```

Arguments

- **STdata**
  - A STdata object, see `mesa.data.raw`.
- **region**
  - Vector of the same length and order as STdata$covars$ID. Indicates region(s) in which different trends are to be fitted and removed.
- **method**
  - Method for fitting the trend (set to method=lm if is.null(method)); should produce output that allows the use of `predict`. Possible options include `lm`, `rlm`, or `lqs`.
- **...**
  - Additional parameters passed to method.
**Details**

Sometimes there is no apparent spatial structure to the time-trend amplitude, or there is not enough identifiability in the data to properly model the structure. In that case, it is possible, at least as a sensitivity analysis, to de-trend the observations and run a model with a spatial field for the intercept only (apart from the spatio-temporal residual field).

`detrendSTdata` will remove the trends from the observations, using `STdata$trend`. `method` is applied as

```r
method(STdata$obs$obs ~ F, ...) # where F is the temporal trend from STdata$trend for each observation; or as
```

```r
method(STdata$obs$obs ~ F*obs.region, ...)
```

where

```r
obs.region = factor(region[match(STdata$obs$ID, STdata$covars$ID)]). allowing for different trends in different region (i.e. interaction between the time trend(s) and region identifiers).
```

`predict(method(...))` is then subtracted from `STdata$obs$obs`, detrending the data.

**Value**

Returns a modified version of the input, with detrended observations and some changes:

- `STdata$obs` Has an additional column, removed.trend, with the amount subtracted per observation.
- `STdata$trend` Is reduced to only the date column, indicating a constant trend.
- `STdata$old.trend` The previous `STdata$trend`, which was used for detrending.
- `STdata$fit.trend` The result of `method`; the trend component removed for each observations can be obtained as `predict(STdata$fit.trend)`. NOTE: Additional functions, such as `createSTmodel`, might reorder `STdata$obs` implying that `STdata$obs$removed.trend != predict(STdata$fit.trend)`.

**Author(s)**

Assaf P. Oron and Johan Lindström

**See Also**

Other `STdata` functions: `c.STmodel, createDataMatrix, createSTdata, createSTmodel, estimateBetaFields, removeSTcovarMean, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel`

**Examples**

```r
# load the data
data(mesa.data.raw)
# and create STdata-object
mesa.data <- createSTdata(mesa.data.raw$obs, mesa.data.raw$X, n.basis=2, SpatioTemporal=mesa.data.raw"lax.conc.1500")

# plot time-series for the first site,
```
par(mfrow=c(3,2), mar=c(2.5,2.5,3,1))
plot(mesa.data, "obs", ID=1)
# And combined for all sites
plot(mesa.data, "loc.obs", legend.loc="bottomleft")

# Attempt to detrend
mesa.data.detrend <- detrendSTdata(mesa.data)
# Examine object, note the trends
mesa.data.detrend

# Plot detrended time-series for the first site,
plot(mesa.data.detrend, "obs", ID=1)
# And combined for all sites
plot(mesa.data.detrend, "loc.obs", legend.loc="bottomleft")

# Use different detrending for different types of locations
mesa.data.detrend2 <- detrendSTdata(mesa.data, region=mesa.data$covars$type)
# Examine object, note the trends
mesa.data.detrend2
# Plot for the first site,
plot(mesa.data.detrend2, "obs", ID=1)
plot(mesa.data.detrend2, "loc.obs", legend.loc="bottomleft")

# Compare the two fitted and removed trends
print(mesa.data.detrend$fit.trend)
print(mesa.data.detrend2$fit.trend)

---

dropObservations  

*Drop Observations from a STmodel*

**Description**

Drops observations from STmodel, removing marked observations along with the corresponding locations and recomputes a number of relevant elements.

**Usage**

```r
dropObservations(STmodel, Ind.cv)
```

**Arguments**

- `STmodel`: Model object from which to drop observations.
- `Ind.cv`: A logical vector with one element per observation in `STmodel$obs`. Observations marked with the TRUE will be dropped from the data structure. Use `createCV` to create the logical vector.

**Value**

Returns the STmodel without the observations marked by `Ind.cv`. Only observed locations are retained.
**Author(s)**

Johan Lindström

**See Also**

Other cross-validation functions: `computeLTA`, `createCV`, `estimateCV`, `estimateCV.STmodel`, `predictCV`, `predictCV.STmodel`, `predictNaive`

Other STmodel functions: `createCV`, `createDataMatrix`, `createSTmodel`, `estimateBetaFields`, `loglikeST`, `loglikeSTdim`, `loglikeSTnaive`, `predictNaive`, `processLocation`, `processLUR`, `processST`, `updateCovf`, `updateSTdataTrend`, `updateTrend`, `updateTrend.STdata`, `updateTrend.STmodel`

**Examples**

```r
# load data
data(mesa.model)

# Mark 30% of observations
I <- runif(dim(mesa.model$obs)[1]) < .3
# drop these observations
mesa.model.new <- dropObservations(mesa.model, I)

# This reduces the remaining number of observations
print(mesa.model)
print(mesa.model.new)

# create cross validation structure
icv <- createCV(mesa.model, groups=10)

# drop observations from the second CV group
mesa.model.new <- dropObservations(mesa.model, icv==2)

# This reduces the remaining number of observations (and locations)
print(mesa.model)
print(mesa.model.new)
```

---

**est.cv.mesa**

*Example of estCVSTmodel and predCVSTmodel structures*

**Description**

Example of 10-fold cross-validated for the model in `mesa.model` using `estimateCV.STmodel` and `predictCV.STmodel`. 
**Format**

A list with elements, see the return description in `estimateCV.STmodel` and `predictCV.STmodel`.

**Source**

Contains parameter estimates for the Spatio-Temporal model applied to monitoring data from the MESA Air project, see Cohen et.al. (2009) and `mesa.data.raw` for details.

**References**


**See Also**

`estimateCV.STmodel` and `predictCV.STmodel` for cross-validation. `createSTmodel` for creation of the originating STmodel object.

Other example data: `est.mesa.model, MCMC.mesa.model, mesa.data.raw, mesa.model, pred.mesa.model`

**Examples**

```r
# load data
data(mesa.model)
data(est.mesa.model)

# estimateCV
# create the CV structure defining 10 different CV-groups
Ind.cv <- createCV(mesa.model, groups=10, min.dist=.1)

# use the best parameters and their starting values as
x.init <- coef(est.mesa.model, pars="cov")[,c("par","init")]

# Not run:
# estimate different parameters for each CV-group
est.cv.mesa <- estimateCV(mesa.model, x.init, Ind.cv)

# End(Not run)
# let's load precomputed results instead
data(est.cv.mesa)

# examine the estimation results
print(est.cv.mesa)
# estimated parameters for each CV-group
coef(est.cv.mesa, pars="cov")
```

---------------------
## predictCV ##

### Not run: ###

Do cross-validated predictions using the just estimated parameters

Ind.cv is inferred from est.cv.mesa as est.cv.mesa$Ind.cv

```r
pred.cv.mesa <- predictCV(mesa.model, est.cv.mesa, LTA=TRUE)
```

### End(Not run) ###

Let's load precomputed results instead

data(pred.cv.mesa)

### Prediction results ###

```r
print(pred.cv.mesa)
```

### and CV-statistics ###

```r
print(summary(pred.cv.mesa, LTA=TRUE))
```

### Not run: ###

A faster option is to only consider the observations and not to compute

variances

```r
pred.cv.fast <- predictCV(mesa.model, est.cv.mesa, only.obs=TRUE, pred.var=FALSE)
```

```r
print(pred.cv.fast)
```

```r
summary(pred.cv.fast)
```

### End(Not run) ###

---

**est.mesa.model**

### Examples of estimateSTmodel structure ###

**Description**

Example of a model structure holding parameter estimates for the model in `mesa.model` using `estimate.STmodel`. Estimation results are also provided for models including spatio-temporal covariates.

**Format**

A list with elements, see the return description in `estimate.STmodel`.

**Source**

Contains parameter estimates for the Spatio-Temporal model applied to monitoring data from the MESA Air project, see Cohen et.al. (2009) and `mesa.data.raw` for details.
References


See Also

estimate.STmodel for parameter estimation.
createSTmodel for creation of the originating STmodel object.

Other example data: est.cv.mesa, MCMC.mesa.model, mesa.data.raw, mesa.model, pred.mesa.model

Examples

```r
##load a model object
data(mesa.model)

##create vector of initial values
dim <- loglikeSTdim(mesa.model)
x.init <- cbind(c(rep(2, dim$mparam.cov-1), 0),
               c(rep(c(1,-3), dim$m+1), -3, 0))
rownames(x.init) <- loglikeSTnames(mesa.model, all=FALSE)

## Not run:
##estimate parameters
est.mesa.model <- estimate(mesa.model, x.init, hessian.all=TRUE)

## End(Not run)

##time consuming estimation, load pre-computed results instead
data(est.mesa.model)

#estimation results
print(est.mesa.model)

##compare the estimated parameters for the two starting points
est.mesa.model$summary$par.all
##and values of the likelihood (and convergence info)
est.mesa.model$summary$status

##extract the estimated parameters and approximate uncertainties
x <- coef(est.mesa.model)

##compare estimated parameters
#plot the estimated parameters with uncertainties
par(mfrow=c(1,1),mar=c(3.5,2.5,.5,.5))
with(x, plot(par, ylim=range(c(par-1.96*sd, par+1.96*sd)),
       xlab="", xaxt="n")
with(x, points(par - 1.96*sd, pch=3))
with(x, points(par + 1.96*sd, pch=3))
```
estimates STmodel

Estimation of the Spatio-Temporal Model

Description

Estimates parameters of the spatio-temporal model using maximum-likelihood, profile maximum likelihood or restricted maximum likelihood (REML). The function uses the \textit{L-BFGS-B} method in \texttt{optim} to maximise \texttt{loglikeST}.

Usage

\texttt{## S3 method for class 'STmodel'}
\texttt{estimate(object, x, x.fixed = NULL,}
\texttt{  type = "p", h = 0.001, diff.type = 1,}
\texttt{  hessian.all = FALSE, lower = -15, upper = 15,}
\texttt{  method = "L-BFGS-B",}
\texttt{  control = list(trace = 3, maxit = 1000), restart = 0,}
\texttt{  ...)}

\texttt{estimate(object, x, ...)}

Arguments

\textbf{object} \texttt{STmodel} object for which to estimate parameters.

\textbf{x} Vector or matrix of starting point(s) for the optimisation. A vector will be treated as a single starting point. If \texttt{x} is a matrix the optimisation will be run using each column as a separate starting point. If \texttt{x} is a single integer then multiple starting points will be created as a set of constant vectors with the values of each starting point taken as \texttt{seq(-5, 5, length.out=x)}. See details below.

\textbf{x.fixed} Vector with parameter to be held fixed; parameters marked as NA will still be estimated.

\textbf{type} A single character indicating the type of log-likelihood to use. Valid options are "f", "p", and "r", for \textit{full}, \textit{profile} or \textit{restricted maximum likelihood} (REML).
h, diff.type  Step length and type of finite difference to use when computing gradients, see loglikeSTGrad.

hessian.all  If type!="f" computes hessian (and uncertainties) for both regression and log-covariance parameters, not only for log-covariance parameters. See value below.

lower, upper, method  Parameter bound and optimisation method, passed to optim.

control  A list of control parameters for the optimisation. See optim for details; setting trace=0 eliminates all output.

restart  Number of times to restart each optimisation if optim fails to converge; can sometimes resolve issues with L-BFGS-B line search.

...  Ignored additional arguments.

Details

The starting point(s) for the optimisation can either contain both regression parameters and log-covariances parameters for a total of loglikeSTdim(object)$nparam parameters or only contain the log-covariances covariances parameters i.e. loglikeSTdim(object)$nparam.cov parameters.

If regression parameters are given but not needed (type!="f") they are dropped; if they are needed but not given they are inferred through a generalised least squares (GLS) computation, obtained by calling predict.STmodel.

If multiple starting points are used this function returns all optimisation results, along with an indication of the best result. The best result is determined by first evaluating which of the optimisations have converged. Convergence is determined by checking that the output from optim has convergence==0 and that the hessian is negative definite, i.e. all(eigen(hessian)$value < -1e-10).

Among the converged optimisations the one with the highest log-likelihood value is then selected as the best result.

If none of the optimisations have converged the result with the highest log-likelihood value is selected as the best result.

Most of the elements in res.best (and in res.all[[i]]) are obtained from optim. The following is a brief description:

par  The best set of parameters found.

value  Log-likelihood value corresponding to par.

counts  The number of function/gradient calls.

convergence  \(0\) indicates successful convergence, see optim.

message  Additional information returned by optim.

hessian  A symmetric matrix giving the finite difference Hessian of the function par.

conv  A logical variable indicating convergence; \(\text{TRUE}\) if convergence==0 and hessian is negative definite, see details above.

par.init  The initial parameters used for this optimisation.

par.all  All parameters (both regression and log-covariance). Identical to par if type="f".
**estimate.STmodel**

**hessian.all** The hessian for all parameters (both regression and log-covariance).

*NOTE:* Due to computational considerations hessian.all is computed *only* for res.best.

**Value**

estimate.STmodel object containing:

**res.best** A list containing the best optimisation result; elements are described below. Selection of the best result is described in details above.

**res.all** A list with all the optimisations results, each element contains (almost) the same information as res.best e.g. res.all[[i]] contains optimisation results for the i:th starting point.

**summary** A list with parameter estimates and convergence information for all starting points.

**Author(s)**

Johan Lindström

**See Also**

Other estimate.STmodel methods: coef.estimate.STmodel, print.estimate.STmodel


**Examples**

```r
#Load a model object
data(mesa.model)

#Create vector of initial values
dim <- loglikeSTdim(mesa.model)
x.init <- cbind(c( rep(2, dim$nparam.cov-1), 0),
               c( rep(c(1,-3), dim$m+1), -3, 0))
rownames(x.init) <- loglikeSTnames(mesa.model, all=FALSE)

#Not run:
#Estimate parameters
est.mesa.model <- estimate(mesa.model, x.init, hessian.all=TRUE)

#End (Not run)

#Time consuming estimation, load pre-computed results instead
data(est.mesa.model)

#Estimation results
```
print(est.mesa.model)

## compare the estimated parameters for the two starting points
est.mesa.model$summary$par.all
## and values of the likelihood (and convergence info)
est.mesa.model$summary$status

## extract the estimated parameters and approximate uncertainties
x <- coef(est.mesa.model)

## compare estimated parameters
## plot the estimated parameters with uncertainties
par(mfrow=c(1,1),mar=c(3.5,2.5,.5,.5))
with(x, plot(par, ylim=range(c(par-1.96*sd, par+1.96*sd)),
       xlab="", xaxt="n"))
with(x, points(par - 1.96*sd, pch=3))
with(x, points(par + 1.96*sd, pch=3))

abline(h=0, col="grey")
## add axis labels
axis(1, 1:length(x$par), rownames(x), las=2)

## Not run:
## example using a few fixed parameters
x.fixed <- coef(est.mesa.model)$par
x.fixed[c(1,2,5:9)] <- NA
est.fix <- estimate(mesa.model, x.init, x.fixed, type="p")

## End(Not run)

estimateBetaFields Regression Estimates of beta-Fields

Description

Estimates the latent-beta fields for a STdata/STmodel object by regressing the observations for each site on the temporal trends.

Usage

estimateBetaFields(STdata = NULL, subset = NULL)

Arguments

STdata A STdata/STmodel object containing observations. Use either this or the obs, date, and ID inputs.

subset A subset of locations for which to estimate the beta-fields. A warning is given for each name not found in ID.
**Value**

A list with two matrices; the estimated beta-coefficients and standard deviations of the estimates.

**Author(s)**

Johan Lindström

**See Also**

Other data matrix: `createDataMatrix, mesa.data.raw, SVDmiss, SVDsmooth, SVDsmoothCV`

Other STdata functions: `c.STmodel, createDataMatrix, createSTdata, createSTmodel, detrendSTdata, removeSTcovarMean, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel`

Other STmodel functions: `createCV, createDataMatrix, createSTmodel, dropObservations, loglikeST, loglikeSTdim, loglikeSTnaive, predictNaive, processLocation, processLUR, processST, updateCovf, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel`

**Examples**

```r
require(plotrix)
# load data
data(mesa.model)

# Regression based estimate of the beta-fields
beta <- estimateBetaFields(mesa.model)

# check regression coefficients
summary(beta$beta)

# or plot as a function of distance to coast, with uncertainties
par(mfrow=c(2,2))
for(i in 1:3){
    plotCI(mesa.model$LUR[[1]]$"log0.m.to.a1", beta$beta[,i],
           uiw=1.96*beta$beta.sd[,i],
           ylab=colnames(beta$beta)[i])
}

# or compare to the fields from predict.STmodel
data(pred.mesa.model)

# Study the results
# Start by comparing beta fields
par(mfcol=c(1,1), mar=c(4.5,4.5,2,.5), pty="s")
plotCI(x=beta$beta[,1], y=pred.mesa.model$beta$EX[,1],
       uiw=1.96*sqrt(pred.mesa.model$beta$VX[,1]),
       main="Temporal Intercept",
       xlab="Empirical estimate",
       ylab="Spatio-Temporal Model")
plotCI(x=beta$beta[,1], y=pred.mesa.model$beta$EX[,1],
       uiw=1.96*beta$beta.sd[,1], add=TRUE, err="x")
```
estimateCV.STmodel

Cross-Validated Estimation and Prediction

Description

Functions that perform cross-validated parameter estimation and prediction for the spatio-temporal model.

Usage

```r
## S3 method for class 'STmodel'
estimateCV(object, x, Ind.cv, 
            control = list(trace = 3), verbose.res = FALSE, ...)

estimateCV(object, x, Ind.cv, ...)

## S3 method for class 'STmodel'
predictCV(object, x, Ind.cv = NULL, 
          ..., silent = TRUE, LTA = FALSE)
predictCV(object, x, Ind.cv, ...)
```

Arguments

- **object**: STmodel object for which to perform cross-validation.
- **x**: Either a vector or matrix of starting point(s) for the optimisation, see `estimate.STmodel`; or a matrix with parameters, the i:th row being used for prediction of the i:th cross-validation set. For prediction either a estCVSTmodel or estimateSTmodel object, results from `estimateCV.STmodel` or `estimate.STmodel`, can be used.
- **Ind.cv**: Ind.cv defines the cross-validation scheme. Either a (number or observations) - by - (groups) logical matrix or an integer valued vector with length equal to (number or observations). For predictCV.STmodel Ind.cv can be inferred from x if x is a estCVSTmodel object See further `createCV`.
- **control**: A list of control parameters for the optimisation. See `optim` for details; setting trace=0 eliminates all output.
- **verbose.res**: A TRUE/FALSE variable indicating if full results from `estimate.STmodel` for each CV group should be returned; defaults to FALSE.
- **...**: All additional parameters for `estimate.STmodel` or `predict.STmodel`. For `predict.STmodel` a number of parameters are set in predictCV.STmodel and can NOT be overridden, these are nugget.unobs, only.pars=FALSE, and combine.data=FALSE.
silent Show status after each iteration?
LTA TRUE/FALSE, compute long-term temporal averages, similar to computeLTA, but with the option of including the uncertainty; see predict.STmodel.

Details

For predictCV.STmodel the parameters used to compute predictions for the left out observations can be either a single vector or a matrix. For a single vector the same parameter values will be used for all cross-validation predictions; for a matrix the parameters in \( x[,i] \) will be used for the predictions of the \( i \)-th cross-validation set (i.e. for \( \text{Ind.cv}[,i] \)). Suitable matrices are provided in the output from estimateCV.STmodel.

The cross-validation groups are given by \( \text{Ind.cv} \). \( \text{Ind.cv} \) should be either a (number of observations) - by - (groups) logical matrix or an integer valued vector with length equal to (number of observations). If a matrix then each column defines a cross-validation set with the TRUE values marking the observations to be left out. If a vector then \( 1:s \) denote observations to be dropped in the first cross-validation set, \( 2:s \) observations to be dropped in the second set, etc. Observations marked by values \( < \) are never dropped. See createCV for details.

Value

Either a estCVSTmodel object with elements:

- **status** Data frame with convergence information and best function value for each cross-validation group.
- **Ind.cv** The cross-validation grouping.
- **x.fixed** Fixed parameters in the estimation, see estimate.STmodel.
- **x.init** Matrix of initial values used, i.e. \( x \) from the input.
- **par.all, par.cov** Matrices with estimated parameters for each cross-validation group.
- **par.all.sd, par.cov.sd** Standard deviations computed from the Hessian/information matrix for set of estimated parameters.
- **res.all** Estimation results for each cross-validation group, contains the output from the estimate.STmodel calls, only included if verbose.res=TRUE.

Or a predCVSTmodel object with elements:

- **opts** Copy of the opts field in the output from predict.STmodel.
- **Ind.cv** The cross-validation grouping.
- **pred.obs** A data frame with a copy of observations from object$obs, predictions (for different model components), variances, and residuals. Variance field will be missing if pred.var=FALSE.
- **pred.all** A list with time-by-location data frames containing predictions and variances for all space-time locations as well as predictions and variances for the beta-fields. Unobserved points are NA for the option only.obs=TRUE.
estimateCV.STmodel

Author(s)
Johan Lindström

See Also
Other cross-validation functions: computeLTA, createCV, dropObservations, predictNaive
Other estCVSTmodel methods: boxplot.estCVSTmodel, coef.estCVSTmodel, print.estCVSTmodel, print.summary.estCVSTmodel, summary.estCVSTmodel
Other predCVSTmodel functions: computeLTA
Other predCVSTmodel methods: plot.predCVSTmodel, plot.predictSTmodel, print.predCVSTmodel, print.summary.predCVSTmodel, qqnorm.predCVSTmodel, qqnorm.STdata, qqnorm.STmodel, scatterPlot.predCVSTmodel, summary.predCVSTmodel

Examples

```r
# load data
data(mesa.model)
data(est.mesa.model)

# estimateCV

# create the CV structure defining 10 different CV-groups
Ind.cv <- createCV(mesa.model, groups=10, min.dist=.1)

# use the best parameters and there starting values as
x.init <- coef(est.mesa.model, pars="cov")[,c("par","init")]

# Not run:
## estimate different parameters for each CV-group
est.cv.mesa <- estimateCV(mesa.model, x.init, Ind.cv)

# End(Not run)
# let's load precomputed results instead
data(est.cv.mesa)

# examine the estimation results
print( est.cv.mesa )
## estimated parameters for each CV-group
coef(est.cv.mesa, pars="cov")

# predictCV

# Not run:
## Do cross-validated predictions using the just estimated parameters```
### evalCovFuns

**Description**
Computes covariance functions (excluding nugget) for a given vector or matrix of distances.

**Usage**
```r
evalCovFuns(type = "exp", pars = c(1, 1),
d = seq(0, 10, length.out = 100))
```

**Arguments**
- **type**
  Name of covariance functions, see `namesCovFuns`.
- **pars**
  Parameter for the covariance function, see `parsCovFuns`.
- **d**
  Vector/matrix for which to compute the covariance function.

**Value**
Covariance function computed for all elements in d.

**Author(s)**
Johan Lindström
See Also

Other covariance functions: `crossDist`, `makeSigmaB`, `makeSigmaNu`, `namesCovFuns`, `parsCovFuns`, `updateCovf`

Examples

```r
## vector of distances
d <- seq(0, 10, length.out = 10);
## just the simplest case (exponential, range=2, sill=0.7)
plot(d, evalCovFuns("exp", c(2, 0.7), d), type = "l")

## create list of ranges
range <- c(1, 2, 3.5, 5);
## list names
name <- list("exp", "exp2", "spherical", "cauchy", "matern", "matern")
## and list of shapes
shape <- c(vector("list", 4), list(1, 5, .25, 5))

## matrix holding results
covf <- array(NA, c(length(d), length(name), length(range)))

## compute a few covariance functions
for (i in 1:length(name)) {
  for (j in 1:length(range)) {
    pars <- c(range[j], 1, shape[[i]])
    covf[i, j] <- evalCovFuns(name[[i]], pars, d)
  }
}

## plot the covariance function for comparison
par(mfrow = c(2, 2))
for (j in 1:length(range)) {
  plot(0, 0, type = "n", main = range[j],
       xlim = range(d), ylim = range(covf[,, j], na.rm = TRUE))
  for (i in 1:length(name)) {
    lines(d, covf[i, j], col = i)
  }
  abline(v = range[j])
  if (j == 1) {
    legend("topright", lty = 1, col = 1:length(name),
           legend = paste("covf: ", sapply(name, as.character),
                           sapply(shape, function(x){
                             if (is.null(x)) "{" else as.character(x)})))
  }
}
```
Description

Expands the temporal trends in F to a full matrix (with lots of zeros). Mainly used for testing, and illustration in examples.

Usage

```
expandf(F, loc.ind, n.loc = max(loc.ind), sparse = TRUE)
```

Arguments

- **F**: A (number of obs.) - by - (number of temporal trends) matrix containing the temporal trends. Usually `mesa.model$F`, where `mesa.model` is obtained from `createSTmodel`.
- **loc.ind**: A vector indicating which location each row in F corresponds to, usually `mesa.model$obs$idx`.
- **n.loc**: Number of locations.
- **sparse**: Should the returned matrix be sparse (uses the Matrix-package, see `sparseMatrix`)

Value

Returns the expanded F, a `dim(F)[1]`-by-`n.loc`\*`dim(F)[2]` matrix

Author(s)

Johan Lindström and Adam Szpiro

See Also

Other temporal trend functions: `calc.FX`, `calc.FXF2`, `calc.tFX`, `calc.tFXF`

Examples

```r
##create a trend
trend <- cbind(1:5, sin(1:5))
##an index of locations
idx <- c(rep(1:3, 1:2, 2:3)
##a list of time points for each location/observation
T <- c(rep(1:3, each=3),4,4,5,5)

##expand the F matrix to match the locations/times in idx/T.
F <- trend[T,]

##compute the expanded matrix
expandf(F, idx)

##compute the expanded matrix, assuming additional locations
expandf(F, idx, 5)

##or as a full matrix
expandf(F, idx, 5, sparse=FALSE)
```
genGradient

Compute Finite Difference Gradient and Hessians.

Description

Computes finite difference gradient and/or hessian. genGradient function does forward, backward or central differences, the genHessian function uses only central differences.

Usage

gengradient(x, func, h = 0.001, diff.type = 0)
genhessian(x, func, h = 0.001)

Arguments

x
Point at which to compute the gradient or hessian.

func
function that takes only x as an input argument. Use function(x)(my.func(x, other.input)) to create a temporary function, see the example.

h
Step length for the finite difference.

diff.type
Type of finite difference, diff.type>0 gives forward differences, diff.type=0 gives central differences, and diff.type<0 gives backward differences.

Value

gradient vector or Hessian matrix.

Author(s)

Johan Lindström

See Also

Other numerical derivatives: loglikeSTGrad, loglikeSTHessian, loglikeSTnaiveGrad, loglikeSTnaiveHessian

Examples

# create a two variable function
f.test <- function(x)(sin(x[1])*cos(x[2]))

# compute the gradient using forward difference
genGradient(c(.5,.5), f.test, diff.type=1)
# and central difference
genGradient(c(.5,.5), f.test, diff.type=0)
# compared to the true value
\(c(\cos(.5)\cos(.5), -\sin(.5)\sin(.5))\)

# Compute the Hessian
genHessian(c(.5, .5), f.test, h=1e-4)
# and compare to the true value
matrix(c(-\sin(.5)\cos(.5), -\cos(.5)\sin(.5),
       -\cos(.5)\sin(.5), -\sin(.5)\cos(.5)), 2, 2)

---

**loglikeST**

*Compute the Log-likelihood for the Spatio-Temporal Model*

**Description**

Computes the log-likelihood for the spatio-temporal model. `loglikeST` uses an optimised version of the log-likelihood, while `loglikeSTnaive` uses the naive (slow) version and is included mainly for testing and speed checks.

**Usage**

```r
loglikeST(x = NULL, STmodel, type = "p", x.fixed = NULL)
loglikeSTnaive(x = NULL, STmodel, type = "p",
               x.fixed = NULL)
```

**Arguments**

- **x**
  - Point at which to compute the log-likelihood, should be only log-covariance parameters if `type=c("p","r")` and regression parameters followed by log-covariance parameters if `type="f"`. If `x=NULL` the function acts as an alias for `loglikeSTnames` returning the expected names of the input parameters.

- **STmodel**
  - STmodel object with the model for which to compute the log-likelihood.

- **type**
  - A single character indicating the type of log-likelihood to compute. Valid options are "f", "p", and "r", for full, profile or restricted maximum likelihood (REML).

- **x.fixed**
  - Parameters to keep fixed, NA values in this vector is replaced by values from `x` and the result is used as `x`, ie.
  ```r
  x.fixed[ is.na(x.fixed) ] <- x
  x <- x \cdot fixed.
  ```

**Value**

Returns the log-likelihood of the spatio temporal model.
loglikeSTdim

Warning

loglikeSTnaive may take long to run. However for some problems with many locations and short
time series loglikeSTnaive could be faster than loglikeST.

Author(s)

Johan Lindström

See Also

Other likelihood functions: loglikeSTGrad, loglikeSTHessian, loglikeSTnaiveGrad, loglikeSTnaiveHessian
Other STmodel functions: createCV, createDataMatrix, createSTmodel, dropObservations,
estimateBetaFields, loglikeSTdim, predictNaive, processLocation, processLUR, processST,
updateCovf, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel

Examples

```r
# load the data
data(mesa.model)

# Compute dimensions for the data structure
dim <- loglikeSTdim(mesa.model)

# Find out in which order parameters should be given
loglikeST(NULL, mesa.model)

# Let's create random vectors of values
x <- runif(dim$nparam.cov)
x.all <- runif(dim$nparam)

# Evaluate the log-likelihood for these values
loglikeST(x.all, mesa.model, "f")
loglikeST(x, mesa.model, "p")
```

loglikeSTdim Dimensions of the STmodel Structure

Description

Function that computes the dimension of several objects in a STmodel object.

Usage

loglikeSTdim(STmodel)

Arguments

STmodel STmodel object for which dimensions are to be computed.
Value

list containing:

T  Number of observation times.
m  Number of temporal basis functions, including the intercept.
n  Number of distinct locations in the data.
n.obs  Number of observed locations.
p  vector of length m; number of geographic covariates for each temporal basis functions.
L  Number of spatio-temporal covariates

npars.beta.covf  vector of length m; number of parameters for each covariance-function for the beta-fields.
npars.beta.tot  vector of length m; total number of parameters for each beta-fields, including nugget(s).
npars.nu.covf, npars.nu.tot  number of parameters for the nu-field, same distinction as above.
nparam  Total number of parameters, including regression parameters.
nparam.cov  Number of covariance parameters.

Author(s)

Johan Lindström

See Also

Other likelihood utility functions: calc.is.x, calc.mu.b, calc.x.is.x, loglikeStgetPars, loglikeStnames

Other STmodel functions: createCV, createDataMatrix, createSTmodel, dropObservations, estimateBetaFields, loglikeST, loglikeSTnaive, predictNaive, processLocation, processLUR, processST, updateCovf, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel

Examples

# load the data
data(mesa.model)

# compute dimensions for the data structure
loglikeStdim(mesa.model)
Description

Extracts parameters for the log-likelihood from a parameter vector and separates regression parameters and log-covariance parameters.

Usage

loglikeSTgetPars(x, STmodel)

Arguments

x
A vector containing regression (optionally) and log-covariance parameters. The ordering of has to be exactly that indicated by loglikeSTnames.

STmodel
STmodel STmodel object describing the problem.

Value

list containing:

gamma
Regression coefficients for the spatio-temporal covariate(s).

alpha
A list of regression coefficients for geographic covariates.

cov.beta
A list containing a list of pars and vector of nuggets. See makesigmab.

cov.nu
A list of covariance parameters for the nu-field, as pars, nugget and random.effect respectively.

Covariance parameters are also back-transformed from log-scale.

Author(s)

Johan Lindström

See Also

Other likelihood utility functions: calc.iS.X, calc.mu.B, calc.X.iS.X, loglikeSTdim, loglikeSTnames

Examples

# Load the data
data(mesa.model)

# Compute dimensions for the data structure
dim <- loglikeSTdim(mesa.model)

# Let's create random parameter vectors ...
x <- runif(dim$mparam.cov)
loglikeSTgrad

Computes finite difference gradients and hessians for the log-likelihood functions loglikeST and loglikeSTnaive. Uses genGradient and genHessian to compute finite difference derivatives of the log-likelihood function in loglikeST and loglikeSTnaive.

Usage

\[
\text{loglikeSTGrad}(x, S\text{model}, \text{type} = "p", x.\text{fixed} = \text{NULL}, h = 0.001, \text{diff.type} = 0)
\]

\[
\text{loglikeSTHessian}(x, S\text{model}, \text{type} = "p", x.\text{fixed} = \text{NULL}, h = 0.001)
\]

\[
\text{loglikeSTnaiveGrad}(x, S\text{model}, \text{type} = "p", x.\text{fixed} = \text{NULL}, h = 0.001, \text{diff.type} = 0)
\]

\[
\text{loglikeSTnaiveHessian}(x, S\text{model}, \text{type} = "p", x.\text{fixed} = \text{NULL}, h = 0.001)
\]
Arguments

- **x**: Point at which to compute the gradient or hessian, see `loglikeST`.
- **stmodel**: STmodel object with the model for which to compute derivatives of the log-likelihood.
- **type**: A single character indicating the type of log-likelihood to compute. Valid options are "f", "p", and "r", for full, profile or restricted maximum likelihood (REML).
- **x.fixed**: Parameters to keep fixed, see `loglikeST`.
- **h.diff.type**: Step length and type of finite difference to use when computing gradients, see `genGradient`.

Value

Returns the gradient or Hessian for the `loglikeST` and `loglikeSTnaive` functions.

Warning

`loglikeSTnaiveGrad` and `loglikeSTnaiveHessian` may take very long time to run, use with extreme caution.

Author(s)

Johan Lindström

See Also

Other likelihood functions: `loglikeST`, `loglikeSTnaive`

Other numerical derivatives: `genGradient`, `genHessian`

Examples

```R
## Not run:
###load the data
data(mesa.model)

###Compute dimensions for the data structure
dim <- loglikeSTdim(mesa.model)

###Let's create random vectors of values
x <- runif(dim$nparam$cov)
x.all <- runif(dim$nparam)

###Compute the gradients
Gf <- loglikeSTGrad(x.all, mesa.model, "f")
Gp <- loglikeSTGrad(x, mesa.model, "p")
Gr <- loglikeSTGrad(x, mesa.model, "r")

##And the Hessian, this may take some time...
Hf <- loglikeSTHessian(x.all, mesa.model, "f")
```
loglikeSTnames

Create Names for Log-likelihood Parameters for STmodel objects

Description

Function that creates a character vector with names for the parameters expected by log-likelihood functions. Names are created by extracting names from the STmodel structure.

Usage

loglikeSTnames(STmodel, all = TRUE)

Arguments

- **STmodel**: STmodel object for which parameter names are to be computed.
- **all**: compute all parameter names (regression and covariance) or only covariance parameters.

Value

Returns names of the parameters expected by the log-likelihood functions. Regression parameter names start with gamma/alpha (spatio-temporal/geographic covariate), followed by name of beta-field, and the name of covariate. The covariance parameters follow, log (reminder that parameter is log-scale), covariance parameter name, name of field, type of covariance function.

Author(s)

Johan Lindström

See Also

Other likelihood utility functions: `calc.i.S.X, calc.mu.B, calc.X.i.S.X, loglikeSTdim, loglikeSTgetPars`

Examples

```r
Hp <- loglikeSTHessian(x, mesa.model, "p")
Hr <- loglikeSTHessian(x, mesa.model, "r")

## End(Not run)

# load the data
data(mesa.model)

## Find out in which order parameters should be given
loglikeSTnames(mesa.model)
## ...and for only the covariance parameters.
loglikeSTnames(mesa.model, FALSE)
```
make.sigma.B  

*Deprecated functions, use replacements!*

**Description**

Deprecated functions, use replacements!

**Usage**

```r
make.sigma.B(...)  
make.sigma.B.full(...)  
make.sigma.nu(...)  
make.sigma.nu.cross.cov(...)  
calc.tF.times.mat(...)  
calc.F.times.X(...)  
calc.tF.mat.F(...)  
block.mult(...)  
dot.prod(...)  
SVD.miss(...)  
SVD.smooth(...)  
SVD.smooth.cv(...)  
calc.smooth.trends(...)  
setupSTdataset(...)  
printMesaDataNbrObs(...)  
plotMonitoringLoc(...)  
plotMesaData(...)  
create.data.matrix(...)  
remove.ST.mean(...)  
```
detrend.data(...)  
create.data.model(...)  
default.LUR.list(...)  
default.ST.list(...)  
construct.LUR.basis(...)  
construct.ST.basis(...)  
loglike.dim(...)  
loglike.var.names(...)  
get.params(...)  
loglike(...)  
loglike.naive(...)  

gen.gradient(...)  
gen.hessian(...)  
loglike.grad(...)  
loglike.hessian(...)  
loglike.naive.grad(...)  
loglike.naive.hessian(...)  
fit.mesa.model(...)  
cond.expectation(...)  
simulateMesaData(...)  
combineMesaData(...)  
drop.observations(...)  
plotPrediction(...)  
tstat(...)
compute.ltaCV(...)  
CVbasics(...)  
summaryStatsCV(...)  
run.MCMC(...)  
plotCV(...)  
CVresiduals.qqnorm(...)  
CVresiduals.scatter(...)  

**Arguments**  
... Unused, for compatibility.

**Details**  
Functions have been rename/replaced as:

- `block.mult` to `blockMult`  
- `calc.F.times.X` to `calc.FX`  
- `calc.smooth.trends` to `calcSmoothTrends` and `updateTrend`  
- `calc.tF.mat.F` to `calc.tFXF`  
- `calc.tF.times.mat` to `calc.tFXF`, see also `expandF`.  
- `combineMesaData` to `c.STmodel`  
- `compute.LtaCV` to `computeLTA`  
- `cond.expectation` to `predict.STmodel`  
- `construct.LUR.basis` to `createLUR`  
- `construct.ST.basis` to `createST`  
- `create.data.matrix` to `createDataMatrix`  
- `create.data.model` to `createSTmodel`, see also `updateCovf` and `processLocation`.  
- `CVbasics` Included in `estimateCV.STmodel`.  
- `CVresiduals.qqnorm` to `qqnorm.STdata.qqnorm.STmodel`, or `qqnorm.predCVSTmodel`  
- `CVresiduals.scatter` to `scatterPlot.STdata.scatterPlot.STmodel`, or `scatterPlot.predCVSTmodel`  
- `default.LUR.list` to `processLUR`  
- `default.ST.list` to `processST`  
- `detrend.data` to `detrendSTdata`  
- `dot.prod` to `dotProd`  
- `drop.observations` to `dropObservations`  
- `fit.mesa.model` to `estimate.STmodel`
Value

Does not return.

Author(s)

Johan Lindström
Computations for Block Diagonal Matrices

Description

Provides block diagonal version of the base package functions `chol`, `chol2inv`, and `backsolve`.

Computes the Cholesky factor, the matrix inverse and solves matrix equation systems for block diagonal matrices.

Usage

```r
makeCholBlock(mat, n.blocks = 1,
               block.sizes = rep(dim(mat)[1]/n.blocks, n.blocks))

invCholBlock(R, n.blocks = 1,
             block.sizes = rep(dim(R)[1]/n.blocks, n.blocks))

solveTriBlock(R, B, n.blocks = 1,
              block.sizes = rep(dim(R)[1]/n.blocks, n.blocks),
              transpose = FALSE)
```

Arguments

- **mat**: A block diagonal, square, positive definite matrix.
- **R**: Upper right block diagonal Cholesky factor. The output from `chol` or `makeCholBlock`.
- **n.blocks**: Number of diagonal blocks in `mat` (or `R`). Defaults to 1 (i.e. a full matrix) if neither `n.blocks` nor `block.sizes` given, o.w. it defaults to `length(block.sizes)`.
- **block.sizes**: A vector of length `n.blocks` with the size of each of the diagonal blocks. If not given it will assume equal size blocks.
- **B**: Vector or matrix containing the right hand side of the equations system to be solved; needs to be a multiple of `dim(R)[1]`.
- **transpose**: Transpose `R` before solving the equation system. Controls if we solve the equations system given by `R*x = B` or `R'*x=B`.

Details

- `makeCholBlock` computes the Cholesky factor of a block diagonal matrix using the block diagonal structure to speed up computations.
- `invCholBlock` uses the Cholesky factor from `makeCholBlock` to compute the inverse of `mat`.
- `solveTriBlock` solves equation systems based on the Cholesky factor, using the block diagonal structure to speed up computations (c.f. `backsolve`). The function solves equations of the form `R*x = B`, and `R'*x = B` with respect to `x`, where the transpose is controlled by the parameter `transpose`. Applying the function twice solves `mat*x=B`, see the examples.
For all three functions the block diagonal structure of the matrix is defined by two input variables, the number of blocks \( n \), blocks, and the size of each block \( block.sizes \). The size of the matrices must match the total number of blocks, i.e. \( \sum(block.sizes) \) must equal \( \text{dim(mat)} \).

The functions can be used for full matrices by setting the number of blocks to 1.

**Value**

\( \text{makeCholBlock} \) gives the Cholesky factor and \( \text{invCholBlock} \) gives the inverse of the matrix \( \text{mat} \). \( \text{solveTriBlock} \) gives the answer to the equation system.

**Author(s)**

Johan Lindström and Adam Szpiro

**See Also**

Other basic linear algebra: \texttt{blockMult, crossDist, dotProd, norm2, sumLog, sumLogDiag}

Other block matrix functions: \texttt{blockMult, calc.FX, calc.FXtF2, calc.iS.X, calc.mu.B, calc.tFX, calc.tFXF, calc.X.iS.X, makeSigmaB, makeSigmaNu}

**Examples**

```r
## create a matrix
mat <- cbind(c(1,0,0),c(0,2,1),c(0,1,2))
## define the number of blocks and block sizes
block.sizes <- c(1,2)
n.blocks <- length(block.sizes)

## Compute the Cholesky factor
R <- makeCholBlock(mat, n.blocks, block.sizes)
## and the matrix inverse
i.mat <- invCholBlock(R, n.blocks, block.sizes)
## compare to the alternative
i.mat-solve(mat)

## define a B vector
B <- c(1,2,3)
## solve the equation system (we need n.x since B is not a matrix)
x1 <- solveTriBlock(R, B, n.blocks, block.sizes, tr=TRUE)
x2 <- solveTriBlock(R, x1, n.blocks, block.sizes, tr=FALSE)
print(x2)
## compare to the alternative
print(solve(mat,B))
range(x2-solve(mat,B))

## compute the quadratic form B'\times i.mat\times B
norm2(x1)
## compare to the alternative
t(B) %*% i.mat %*% B
```
Function that creates a block covariance matrix with equal sized blocks. Used to construct the Sigma_B matrix.

Usage
makeSigmaB(pars, dist, type = "exp", nugget = 0,
    symmetry = dim(dist)[1] == dim(dist)[2],
    ind2.to.1 = 1:dim(dist)[2], sparse = FALSE)

Arguments
- pars: List of parameters for each block; if not a list a single block matrix is assumed. Should match parameters suggested by parsCovFuns.
- dist: Distance matrix.
- type: Name(s) of covariance functions, see namesCovFuns.
- nugget: Vector of nugget(s) to add to the diagonal of each matrix.
- symmetry: TRUE/FALSE flag if the dist is symmetric, resulting in a symmetric covariance matrix.
- ind2.to.1: Vectors, that for each index along the second dimension gives a first dimension index, used only if symmetry=FALSE to determine which covariances should have an added nugget (collocated sites).
- sparse: If TRUE, return a block diagonal sparse matrix, see bdiag.

Details
Any parameters given as scalars will be rep-ed to match length(pars).

Value
Block covariance matrix of size dim(dist)*n.blocks.

Author(s)
Johan Lindström

See Also
Other block matrix functions: blockMult, calc.FX, calc.FXF2, calc.is.X, calc.mu.B, calc.tFX, calc.tFXF, calc.X.is.X, invCholBlock, makeCholBlock, makeSigmaNu, solveTriBlock
Other covariance functions: crossDist, evalCovFuns, makeSigmaNu, namesCovFuns, parsCovFuns, updateCovf
Examples

```r
## First create some random locations
x <- rnorm(5)
y <- rnorm(5)

## Compute distance matrix
d <- crossDist(cbind(x, y))

## Create a block diagonal matrix exponential covariance matrix
## with different range, sill, and nugget
pars <- list(c(3,2), c(2,1), c(1,3))
nugget <- c(5, 0, 1)

Sigma1 <- makeSigmaB(pars, D, type="exp", nugget=nugget)

## Or using different covariance functions for each block
Sigma2 <- makeSigmaB(pars, D, type=c("exp","exp2","cubic"),
                     nugget=nugget)

## Make a cross-covariance matrix
Dcross <- D[1:3, c(1,1,2,2)]
Sigma.cross <- makeSigmaB(pars, Dcross, type="exp", nugget=nugget,
                          ind2.to.1=c(1,1,2,2))
```

---

**makeSigmaNu**  
Create Block Covariance Matrix (Unequal Block Sizes)

**Description**

Function that creates a block covariance matrix with unequally sized blocks. Used to construct the Sigma_nu matrix.

**Usage**

```r
makeSigmaNu(pars, dist, type = "exp", nugget = 0,
            random.effect = 0,
            symmetry = dim(dist)[1] == dim(dist)[2],
            blocks1 = dim(dist)[1], blocks2 = dim(dist)[2],
            ind1 = 1:dim(dist)[1], ind2 = 1:dim(dist)[2],
            ind2.to.1 = 1:dim(dist)[2], sparse = FALSE)
```

**Arguments**

- **pars**: Vector of parameters, as suggested by parsCovFuns.
- **dist**: Distance matrix.
- **type**: Name of the covariance function to use, see namesCovFuns.
**nugget**
A value of the nugget or a vector of length dim(dist)[1] giving (possibly) location specific nuggets.

**random.effect**
A constant variance to add to the covariance matrix, can be interpreted as either and partial sill with infinite range or as a random effect with variance given by random.effect for the mean value.

**symmetry**
TRUE/FALSE flag if the dist matrix is symmetric. If also ind1==ind2 and blocks1==blocks2 the resulting covariance matrix will be symmetric.

**blocks1,blocks2**
Vectors with the size(s) of each of the diagonal blocks, usually mesa.model$nt. If symmetry=TRUE and then blocks2 defaults to blocks1 if missing.

**ind1,ind2**
Vectors indicating the location of each element in the covariance matrix, used to index the dist-matrix to determine the distance between locations, usually mesa.model$obs$idx. If symmetry=TRUE and then ind2 defaults to ind1 if missing.

**ind2.to.1**
Vectors, that for each index along the second dimension, ind2, gives a first dimension index, ind1, used only if symmetry=FALSE to determine which covariances should have an added nugget (collocated sites).

**sparse**
If TRUE, return a block diagonal sparse matrix, see bdiag.

**Value**
Block covariance matrix of size length(ind1)-by-length(ind2).

**Author(s)**
Johan Lindström

**See Also**
Other block matrix functions: blockMult, calc.FX, calc.FXtF2, calc.iS.X, calc.mu.B, calc.tFX, calc.tFXF, calc.X.iS.X, invCholBlock, makeCholBlock, makeSigmaB, solveTriBlock

Other covariance functions: crossDist, evalCovFuns, makeSigmaB, namesCovFuns, parsCovFuns, updateCovf

**Examples**

```R
##First create some random locations
x <- rnorm(5)
y <- rnorm(5)

##compute distance matrix
D <- crossDist( cbind(x,y) )

# a vector of locations
I <- c(1,2,3,1,4,4,3,2,1,1)
T <- c(1,1,1,2,2,3,3,3,4)

##create a block diagonal matrix consisting of four parts with
```
### Example of an mcmcStModel structure

**Description**

The output from a Metropolis-Hastings algorithm, implemented in `MCMC.STmodel()`, run for the model in `mesa.model`.

**Format**

A list with elements, see the return description in `MCMC.STmodel`.

**Source**

Contains parametere estimates for the Spatio-Temporal model applied to monitoring data from the MESA Air project, see Cohen et.al. (2009) and `mesa.data.raw` for details.

**References**


**See Also**

`createSTmodel` for creation of the originating STmodel object.

Other example data: `est.cv.mesa`, `est.mesa.model`, `mesa.data.raw`, `mesa.model`, `pred.mesa.model`
**Examples**

```r
## load data
data(mesa.model)
## and results of estimation
data(est.mesa.model)

## starting point
x <- coef(est.mesa.model)
## Hessian, for use as proposal matrix
H <- est.mesa.model$res.best$hessian.all

## Not run:
## run MCMC
MCMC.mesa.model <- MCMC(mesa.model, x$par, N = 2500, Hessian.prop = H)

## End(Not run)
## lets load precomputed results instead
data(MCMC.mesa.model)

## Examine the results
print(MCMC.mesa.model)

## and contents of result vector
names(MCMC.mesa.model)

## Summary
summary(MCMC.mesa.model)

## MCMC tracks for four of the parameters
par(mfrow=c(5,1),mar=c(2,2,2,5,.5))
plot(MCMC.mesa.model, ylab=quot;", xlab=quot;", type=quot;1")
for(i in c(4,9,13,15)){
  plot(MCMC.mesa.model, i, ylab=quot;", xlab=quot;", type=quot;1")
}
```

---

**MCMC STmodel**

**MCMC Inference of Parameters in the Spatio-Temporal Model**

---

**Description**

Estimates parameters and parameter uncertainties for the spatio-temporal model using a Metropolis-Hastings based Markov Chain Monte Carlo (MCMC) algorithm.

The function runs uses a Metropolis-Hastings algorithm (Hastings, 1970) to sample from the parameters of the spatio-temporal model, assuming flat priors for all the parameters (flat on the log-scale for the covariance parameters).

**Usage**

```r
## S3 method for class 'STmodel'
MCMC(object, x, x.fixed = NULL,
```
type = "f", N = 1000, Hessian.prop = NULL, Sigma.prop = NULL, info = min(ceiling(N/50), 100), ...)

MCMC(object, ...)

Arguments

object

STmodel for which to run MCMC.

x

Point at which to start the MCMC. Could be either only log-covariance parameters or regression and log-covariance parameters. If regression parameters are given but not needed they are dropped, if they are needed but not given they are inferred by calling predict.STmodel with only.pars=TRUE.

x.fixed

Vector with parameter to be held fixed; parameters marked as NA will still be estimated.

type

A single character indicating the type of log-likelihood to compute. Valid options are "f" or "r", for full, or restricted maximum likelihood (REML). Since profile is not a proper likelihood type="p" will revert (with a warning) to using the full log-likelihood.

N

Number of MCMC iterations to run.

Hessian.prop

Hessian (information) matrix for the log-likelihood, can be used to create a proposal matrix for the MCMC.

Sigma.prop

Proposal matrix for the MCMC.

info

Outputs status information every info:th iteration. If info=0 no output.

... ignored additional arguments.

Details

At each iteration of the MCMC new parameters are proposed using a random-walk with a proposal covariance matrix. The proposal matrix is determined as:

1 If Sigma.prop is given then this is used.
2 If Sigma.prop=NULL then we follow Roberts et.al. (1997) and compute
c <- 2.38*2.38/dim(Hessian.prop)[1]
Sigma.prop <- -c*solve(Hessian.prop).
3 If both Sigma.prop=NULL and Hessian.prop=NULL then the Hessian is computed using loglikeSTHessian
and Sigma.prop is computed according to point 2.

The resulting proposal matrix is checked to ensure that it is positive definite before proceeding.
all(eigen(Sigma.prop)$value > 1e-10).

Value

mcmcSTmodel object with elements:

par

A N - by - (number of parameters) matrix with trajectories of the parameters.

log.like

A vector of length N with the log-likelihood values at each iteration.
acceptance A vector of length N with the acceptance probability for each iteration.
Sigma.prop, chol.prop
Proposal matrix and its Cholesky factor.
x.fixed Any fixed parameters.

Author(s)
Johan Lindström

See Also
Other mcmcSTmodel methods: density.mcmcSTmodel, plot.density.mcmcSTmodel, plot.mcmcSTmodel,
print.mcmcSTmodel, print.summary.mcmcSTmodel, summary.mcmcSTmodel

Other STmodel methods: c.STmodel, createSTmodel, estimate, estimate.STmodel, estimateCV,
estimateCV.STmodel, plot.STdata, plot.STmodel, predict.STmodel, predictCV, predictCV.STmodel,
print.STmodel, print.summary.STmodel, qqnorm.predictCVSTmodel, qqnorm.STdata, qqnorm.STmodel,
deployPlot.predictCVSTmodel, deployPlot.STdata, deployPlot.STmodel, simulate.STmodel,
summary.STmodel

Examples

```r
# load data
data(mesa.model)

# and results of estimation
data(est.mesa.model)

# strating point
x <- coef(est.mesa.model)
# Hessian, for use as proposal matrix
H <- est.mesa.model$res.best$hessian.all

# Not run:
  #run MCMC
  MCMC.mesa.model <- MCMC(mesa.model, x$par, N = 2500, Hessian.prop = H)

# End(Not run)
# lets load precomputed results instead
data(MCMC.mesa.model)

# Examine the results
print(MCMC.mesa.model)

# and contents of result vector
names(MCMC.mesa.model)

# Summary
summary(MCMC.mesa.model)

# MCMC tracks for four of the parameters
par(mfrow=c(5,1),mar=c(2,2,2.5,.5))
plot(MCMC.mesa.model, ylab="", xlab="", type="l")
for(i in c(4,9,13,15)){
```
### Description

The raw data that was used to create the `mesa.model` structures.

The data structure contains raw data from the MESA Air project. The example below describes how to create the `mesa.model` structure from raw data.

### Format

The structure contains observations, temporal trends, locations, geographic covariates, and spatio-temporal covariates. The data is stored as a list with elements:

- **X** A data.frame containing names, locations, and (geographic) covariates for all the (observation) locations.
- **obs** A time-by-location matrix for the observed data, missing data marked as NA
- **lax.conc.1500** A time-by-location matrix of a spatio-temporal covariate based on output from Caline3QHC.

### Source

Contains monitoring data from the MESA Air project, see Cohen et.al. (2009) for details.

### References


### See Also

- `createSTdata` for creation of STdata objects.
- Other data matrix: `createDataMatrix, estimateBetaFields, SVDmiss, SVDsmooth, SVDsmoothCV`
- Other example data: `est.cv.mesa, est.mesa.model, MCMC.mesa.model, mesa.model, pred.mesa.model`
Examples

```r
# Load the data
data(mesa.data.raw)

# Extract matrix of observations (missing marked by NA)
obs.mat <- mesa.data.raw$obs
head(obs.mat)

# Optionally observations can be given as a data.frame
obs <- data.frame(obs=c(obs.mat),
                   date=rep(rownames(obs.mat), dim(obs.mat)[2]),
                   ID=rep(colnames(obs.mat), each=dim(obs.mat)[1]))

# Force date format
obs$date <- as.Date(obs$date)

# Drop unobserved
obs <- obs[!is.na(obs$obs),,drop=FALSE]

# Create a 3D-array for the spatio-temporal covariate
ST <- array(mesa.data.raw$lax.conc.1500, dim =
            c(dim(mesa.data.raw$lax.conc.1500),1))
dimnames(ST) <- list(rownames(mesa.data.raw$lax.conc),
                     colnames(mesa.data.raw$lax.conc),
                     "lax.conc.1500")

# Or use a list of matrices
ST.list <- list(lax.conc.1500=mesa.data.raw$lax.conc.1500)

# Create STdata object
# Create the data object
mesa.data <- createSTdata(obs.mat, mesa.data.raw$X, n.basis=2,
                           SpatioTemporal=ST)
mesa.data.2 <- createSTdata(obs, mesa.data.raw$X, n.basis=2,
                           SpatioTemporal=ST.list)

# This should yield equal structures,
# which are also the same as data(mesa.data)
all.equal(mesa.data, mesa.data.2)

# Create STmodel object
# Define land-use covariates, for intercept and trends
LUR <- list(~log10.m.to.ai+s2000.pop.div.10000+km.to.coast,
             ~km.to.coast, ~km.to.coast)
# And covariance model
cov.beta <- list(covf="exp", nugget=FALSE)
cov.nu <- list(covf="exp", nugget="type", random.effect=FALSE)
# Which locations to use
locations <- list(coords=c("x","y"), long.lat=c("long","lat"), others="type")
# Create object
```

Example of a STmodel structure

Description

Example of a model structure holding observations, geographic covariates, observation locations, smooth temporal trends, spatio-temporal covariates, and covariance specifications for the model.

Format

A list with elements, a detailed description of each elements is given in details below.

Details

A STmodel object consists of a list with, some or all of, the following elements:

- **obs**: A data.frame with columns:
  - **obs**: The value of each observation.
  - **date**: The observations time, preferably of class Date.
  - **ID**: A character-class giving observation locations; should match elements in locations$ID.
  - **idx**: match between obs$ID and locations$ID for faster computations.

  The data.frame is sorted by date and idx.

- **locations.list,locations** Specification of locations and data.frame with locations for observations (and predictions), see processLocation.

- **D.nu,D.beta**: Distance matrices for the locations in the, possibly different coordinate systems for beta- and nu-fields. See processLocation.

- **cov.beta,cov.nu**: Covariance structure for beta- and nu-fields, see updateCovf.

- **LUR.list,LUR**: Specification of covariates for the beta-fields and a list with covariates for each of the beta-fields, see processLUR and createLUR.

- **trend,trend.fnc**: The temporal trends with one of the columns being named date, preferably of class Date providing the time alignment for the temporal trends.

- **F**: A matrix containing smooth temporal trends for each observation; elements taken from trend.

- **ST.list,ST,ST.all**: Spatio-temporal covariates, NULL if no covariates. For the observations and all space-time locations respectively, see processST and createST.

- **old.trend,fit.trend**: Additional components added if the observations have been detrended, see detrendSTdata.
Source

Contains monitoring data from the MESA Air project, see Cohen et.al. (2009) and mesa.data.raw for details.

References


See Also

createSTmodel for creation of STmodel objects.
createSTdata for creation of the originating STdata object.

Other example data: est.cv.mesa, est.mesa.model, MCMC.mesa.model, mesa.data.raw, pred.mesa.model

Examples

```r
# load the data
data(mesa.model)

# examine components
names(mesa.model)
print(mesa.model)
summary(mesa.model)

# requested geographic and spatio-temporal covariates
mesa.model$LUR.list
mesa.model$ST.list

# covariates for the temporal intercept
head(mesa.model$LUR$const)
#... and the two smooth temporal trends
head(mesa.model$LUR$V1)
head(mesa.model$LUR$V2)

# Some important dimensions of the model
loglikeStdim(mesa.model)
```

---

namesCovFuns

### Available covariance functions

<table>
<thead>
<tr>
<th>namesCovFuns</th>
<th>Available covariance functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

Returns a list of possible covariance function names
Usage

namesCovFuns()

Details

Available covariance functions (d is the distance between points):

- **exp**, exponential: Exponential covariance:
  \[ \sigma^2 \exp\left(\frac{-d}{\rho}\right) \]

- **exp2**, exponential2, gaussian: Gaussian/double exponential covariance:
  \[ \sigma^2 \exp\left(\frac{-d}{\rho}^2\right) \]

- **cubic**: Cubic covariance:
  \[ \sigma^2 \left(1 - 7\left(\frac{d}{\rho}\right)^2 + 8.75\left(\frac{d}{\rho}\right)^3 - 3.5\left(\frac{d}{\rho}\right)^5 + 0.75\left(\frac{d}{\rho}\right)^7 \right) \]
  if \( d < \rho \).

- **spherical**: Spherical covariance:
  \[ \sigma^2 \left(1 - 1.5\left(\frac{d}{\rho}\right) + 0.5\left(\frac{d}{\rho}\right)^3 \right) \]
  if \( d < \rho \).

- **matern**: Matern covariance:
  \[ \frac{\sigma^2}{\Gamma(\nu)2^{\nu-1}} \left(\frac{d\sqrt{8\nu}}{\rho}\right)^\nu K_\nu \left(\frac{d\sqrt{8\nu}}{\rho}\right) \]

- **cauchy**: Cauchy covariance:
  \[ \frac{\sigma^2}{(1 + (d/\rho)^2)^\nu} \]

- **iid**: IID covariance, i.e. zero matrix since nugget is added afterwards.
  \[ 0 \]

Value

Character vector with valid covariance function names.

Author(s)

Johan Lindström

See Also

Other covariance functions: `crossDist`, `evalCovFuns`, `makeSigmaB`, `makeSigmaNu`, `parsCovFuns`, `updateCovf`

Examples

namesCovFuns()
Description

\texttt{dotProd} computes the inner (or dot/scalar) product between two vectors.

\texttt{norm2} computes the squared 2-norm of all the elements in a matrix or vector.

If the vectors are of unequal length \texttt{dotProd} will give a warning and then truncates the longer vector, discarding any excess elements before the computations.

Usage

\begin{verbatim}
\texttt{norm2(v1)}
\end{verbatim}

\begin{verbatim}
\texttt{dotProd(v1, v2)}
\end{verbatim}

Arguments

\begin{verbatim}
v1, v2               Two vectors
\end{verbatim}

Value

\texttt{dotProd} returns the inner product of \texttt{v1} and \texttt{v2}. \texttt{norm2} returns the squared 2-norm of all elements in \texttt{v1}.

Author(s)

Johan Lindström

See Also

Other basic linear algebra: \texttt{blockMult, crossDist, invCholBlock, makeCholBlock, solveTriBlock, sumLog, sumLogDiag}

Examples

\begin{verbatim}
## Create two vectors of equal length
v1 <- rnorm(10)
v2 <- rnorm(10)

## Compute the inner product between the vectors
dotProd(v1, v2)
## or
sum(v1*v2)

## Compute the square 2-norm of \texttt{v1}
norm2(v1)
\end{verbatim}
parsCovFuns

## Description

Provides a list of parameter names for the given covariance function(s), excluding the nugget which is added elsewhere.

## Usage

```
parsCovFuns(type = namesCovFuns(), list = FALSE)
```

## Arguments

- **type**: Name(s) of covariance functions, see `namesCovFuns`.
- **list**: Always return a list (if FALSE returns a vector if possible)

## Value

Character vector with parameter names (excluding the nugget), NULL if the name is unknown. Returns a list if type contains more than one element.

## Author(s)

Johan Lindström

## See Also

Other covariance functions: `crossDist`, `evalCovFuns`, `makeSigmaB`, `makeSigmaNu`, `namesCovFuns`, `updateCovf`

## Examples

```
## all possible parameters
parsCovFuns()
## just one covariance function
parsCovFuns("exp")
## non existant covariance function
parsCovFuns("bad.name")
```
plot.density.mcmcSTmodel

Plots for an density.mcmcSTmodel object

Description

plot method for class density.mcmcSTmodel. Plots results from density.mcmcSTmodel.

Usage

\#
S3 method for class 'density.mcmcSTmodel'
plot(x, y = 1,
    add = FALSE, norm.col = 0, main = NULL, ylim = NULL,
    ...
)

Arguments

x density.mcmcSTmodel object to plot.
y Name/index of parameter for which to plot the density.
add Add to existing plot using lines.
norm.col Add the Gaussian density using a line with colour norm.col, if norm.col=0 do not add the Gaussian.
main Parameter passed as main to plot.density, defaults to the parameter-name if not given.
ylim Additional parameters passed to plot.density.
... Additional parameters passed to plot.density or lines.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other mcmcSTmodel methods: density.mcmcSTmodel, MCMC, MCMC.STmodel, plot.mcmcSTmodel, print.mcmcSTmodel, print.summary.mcmcSTmodel, summary.mcmcSTmodel
Examples

```r
# load estimation results
data(est.mesa.model)
# and MCMC results instead
data(MCMC.mesa.model)

# compute density estimates for the results, and use the Gaussian approximation
# based on Fischer information as reference.
dens <- density(MCMC.mesa.model, estSTmodel=est.mesa.model)

# all the estimated densities
str(dens,1)

# or results for one parameter
dens[[1]]

# plot density functions
plot(dens)

# for a different parameter, along with Gaussian approx
plot(dens, 3, norm.col="red")

# all covariance parameters
par(mfrow=c(3,3),mar=c(4,4,2.5,5))
for(i in 9:17){
  plot(dens, i, norm.col="red")
}
```

Description

`plot` method for class `mcmcSTmodel`.

Usage

```r
## S3 method for class 'mcmcSTmodel'
plot(x, y = "like", add = FALSE,
     main = NULL, ...)
```

Arguments

- `x`: `mcmcSTmodel` object to plot.
- `y`: Type of plot, options are "like", "alpha", or name/index number of a parameter.
- `add`: Add to existing plot using `lines`
- `main`: Parameter passed as `main` to `plot`, defaults to the parameter-name if not given.
- `...`: Additional parameters passed to `plot` or `lines`
plot.predCVSTmodel

Details

Plots results from `MCMC.STmodel`. Either parameter paths or the log-likelihood for the mcmc simulations.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other mcmcSTmodel methods: `density.mcmcSTmodel`, `MCMC, MCMC.STmodel.plot.density.mcmcSTmodel`, `print.mcmcSTmodel`, `print.summary.mcmcSTmodel`, `summary.mcmcSTmodel`

Examples

```r
# load MCMC results instead
data(MCMC.mesa.model)

# plot the log-likelihood
plot(MCMC.mesa.model, ylab="", xlab="", type="l")

# and MCMC tracks for four of the parameters
par(mfrow=c(4,1), mar=c(2,2,2.5,.5))
for(i in c(4,9,13,15)){
  plot(MCMC.mesa.model, i, ylab="", xlab="", type="l")
}

# or by name
par(mfrow=c(1,1), mar=c(2,2,2.5,.5))
plot(MCMC.mesa.model, "nu.log.range.exp", ylab="", xlab="", type="l",
     main="all range estimates", ylim=c(-14,10))

# all ranges in one plot
plot(MCMC.mesa.model, "log.range.const.exp", add=TRUE, col=2)
plot(MCMC.mesa.model, "log.range.V1.exp", add=TRUE, col=3)
plot(MCMC.mesa.model, "log.range.V2.exp", add=TRUE, col=4)
```

plot.predCVSTmodel

Plots for `predictSTmodel` and `predCVSTmodel` Objects

Description

`plot` method for classes `predictSTmodel` and `predCVSTmodel`. Provides several different plots of the data.
Usage

## S3 method for class 'predCVSTmodel'

```r
plot(x, y = "time",
     ID = colnames(x$pred.all$EX)[1],
     col = c("black", "red", "grey"), pch = c(NA, NA),
     cex = c(1, 1), lty = c(1, 1), lwd = c(1, 1), p = 0.95,
     pred.type = "EX", pred.var = TRUE, add = FALSE, ...)
```

## S3 method for class 'predictSTmodel'

```r
plot(x, y = "time",
     STmodel = NULL, ID = x$ID[1],
     col = c("black", "red", "grey"), pch = c(NA, NA),
     cex = c(1, 1), lty = c(1, 1), lwd = c(1, 1), p = 0.95,
     pred.type = "EX", pred.var = FALSE, add = FALSE, ...)
```

Arguments

- `x` predictSTmodel or predCVSTmodel object to plot.
- `y` Plot predictions as a function of either "time" or "obs"ervations.
- `STmodel` STdata/STmodel object containing observations with which to compare the predictions (not used for plot.predCVSTmodel).
- `ID` The location for which we want to plot predictions. A string matching names in colnames(x$EX) (or x$ID, number(s) which are used as ID = colnames(x$EX)[ID], or "all" in which case all predictions are used. If several locations are given (or "all") then y must be "obs".
- `col` A vector of three colours: The first is the colour of the predictions, second for the observations and third for the polygon illustrating the confidence bands. For y="obs" the colours are 1) colour of the points, 2) colour of the 1-1 line, and 3) colour of the polygon. If ID="all", picking col[1]="ID" will colour code the observations-prediction points by site ID.
- `pch`, `cex`, `lty`, `lwd` Vectors with two elements giving the point type, size, line type and line width to use when plotting the predictions and observations respectively. Setting a value to NA will give no points/lines for the predictions/observations. When plotting predictions as a function of observations lty[2] is used for the addition of `abline(0,1, lty=lty[2], col=col[2], lwd=lwd[2]); pch[2]` and `cex[2]` are ignored.
- `p` Width of the plotted confidence bands (as coverage percentage, used to find appropriate two-sided normal quantiles).
- `pred.type` Which type of prediction to plot, one of "EX", "EX.mu", "EX.mu.beta", or "EX.pred"; see the output from predict.STmodel.
- `pred.var` Should we plot confidence bands based on prediction (TRUE) or confidence intervals (FALSE), see predict.STmodel. Only relevant if pred.type="EX" or pred.type="EX preds". **NOTE:** The default differs for plot.predictSTmodel and plot.predCVSTmodel.
- `add` Add to existing plot?
Additional parameters passed to `plot`.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other predCVSTmodel methods: `estimateCV`, `estimateCV.STmodel`, `predictCV`, `predictCV.STmodel`, `print.predCVSTmodel`, `print.summary.predCVSTmodel`, `qqnorm.predCVSTmodel`, `qqnorm.STdata`, `qqnorm.STmodel`, `scatterPlot.predCVSTmodel`, `scatterPlot.STdata`, `scatterPlot.STmodel`, `summary.predCVSTmodel`

Other predictSTmodel methods: `predict.STmodel`, `print.predictSTmodel`

Examples

```r
### plot predictions for a given site ###
### load data
load(mesa.model)
### load predictions
load(pred.mesa.model)

par(mfrow=c(2,1))
plot(pred.mesa.model)
### different site with data and prediction variances
plot(pred.mesa.model, STmodel=mesa.model, ID="L001", pred.var=TRUE)

### compare the different contributions to the predictions
plot(pred.mesa.model)
plot(pred.mesa.model, pred.type="EX.mu", col="red", add=TRUE)
plot(pred.mesa.model, pred.type="EX.mu.beta", col="green", add=TRUE)

### compare the two confidence and prediction intervals
plot(pred.mesa.model, ID=3, pred.var=TRUE, col=c(0,0,"darkgrey"))
plot(pred.mesa.model, ID=3, STmodel=mesa.model,
     col=c("black","red","lightgrey"), add=TRUE)

### plot predictions as function of observations
par(mfrow=c(2,2))
plot(pred.mesa.model, y="obs", STmodel=mesa.model, pred.var=TRUE)

### all data, using points and colour coded by site
plot(pred.mesa.model, y="obs", STmodel=mesa.model, ID="all",
     lty=c(NA,1), pch=c(19,NA), col=c("ID", "red", "grey"),
     cex=.25, pred.var=TRUE)
```
plot.STdata

Different Plots for STdata/STmodel object

Description

plot method for class STdata or STmodel. Provides several different plots of the data. When called for STmodel, STmodel$locations acts as STdata$covars.
Usage

```r
## S3 method for class 'STdata'
plot(x,
     y = c("obs", "res", "acf", "pacf", "loc", "loc.obs"),
     ID = x$ covars$ ID[1], type = x$ covars$ type, col = NULL,
     pch = NULL, cex = NULL, lty = NULL,
     legend.loc = "topleft", legend.names = NULL,
     add = FALSE, ...)
```

```r
## S3 method for class 'STmodel'
plot(x, y = "obs",
     ID = x$ locations$ ID[1], type = x$ locations$ type, ...)
```

Arguments

- `x`: STdata/STmodel object to plot.
- `y`: Type of plot, options are "obs", "res", "acf", "pacf", "loc", or "loc.obs", see details below.
- `ID`: The location for which we want to plot observations. Either a string matching the names in x$ covars$ ID or an integer; if an integer the functions will plot data from ID=x$ covars$ ID[1].
- `type`: Factorial of length(x$ covars$ type), used by "loc" and "loc.obs" to determine how many groups should be plotted and colour/type coded.
- `col, pch, cex, lty`: Colour, type of points, size of points, and type of lines. Exact meaning depends on value of `y`, see Details.
- `legend.loc`: The location of the legend, for "loc" and "loc.obs". See `legend`.
- `legend.names`: A vector of character strings to be used in the legend, for "loc" and for "loc.obs"
- `add`: Add to existing plot, only relevant if `y` is "obs", "res", "loc", or "loc.obs".
- `...`: Additional parameters passed to `plot` or `plot.acf`.

Details

Performs a variety of different plots determined by `y`:

- "obs": Plot observations for location `ID`, along with the fitted temporal trend.
- "res": Plot residuals for the fitted temporal trend at location `ID`; adds the y=0 line for reference.
- "acf": Plot autocorrelation function for the residuals from the fitted temporal trend at location `ID`.
- "pacf": Plot partial autocorrelation function for the residuals from the fitted temporal trend at location `ID`.
- "loc": Plot the observation location index number as a function of the observation date, for all observations. Possibly coded by the type of observations locations.
- "loc.obs": Plot the observation value as a function of the observation date, for all observations. Possibly coded by the type of observations locations.
For `y=c("obs","res")` the first element of `col,pch,cex,lty` is used to specify plotting of the observations, and the second element is used to specify plotting of the fitted temporal trend, or 0-line for "res". Defaults: `col=1,pch=c(1,NA),cex=1,lty=c(NA,1)`. Elements of length one are repeated.

For `y=c("acf","pacf")` `col,pch,cex,lty` are ignored.

For `y=c("loc","loc.obs")` `col,pch,cex` are used to specify the points for each of the different levels in `type` and should be of length 1 or length(levels(type)). `lty` is ignored. Default: `col=1:length(levels(type)),pch=19,cex=.1`

For `y=c("loc","loc.obs")` a legend is added if `legend.loc=NULL`. The vector `legend.names` should have length equal to the number of unique location types. The default legend names are `levels(type)`. 

**Value**

Nothing

**Author(s)**

Johan Lindström and Assaf P. Oron

**See Also**

Other `STdata` methods: `createSTdata,print.STdata,print.summary.STdata,qqnorm.predCVSTmodel,qqnorm.STdata,qqnorm.STmodel,scatterPlot.predCVSTmodel,scatterPlot.STdata,scatterPlot.STmodel,summary.STdata`


**Examples**

```r
#load data
data(mesa.model)

#default plot
plot(mesa.model)

#plot monitor locations
plot(mesa.model, "loc")

#different names/colours/etc
plot(mesa.model, "loc", main="A nice plot", col=c("green","blue"),
legend.names=c("Sites of one type","...and of the other"),
legend.loc="topleft")

#composite time-trend
plot(mesa.model, "loc.obs", legend.loc="bottomleft", cex=.5, pch=c(3,4))
```
plot.SVDcv

Plot and Boxplot cross-validation statistics for SVDcv object

Description

plot and boxplot methods for class SVDcv. Plots summary statistics for the cross-validation. Plots include RMSE, R2, BIC, and scatter plots of BIC for each column.

Usage

```r
## S3 method for class 'SVDcv'
plot(x,
     y = c("all", "MSE", "R2", "AIC", "BIC"), pairs = FALSE,
     sd = FALSE, ...)
```

```r
## S3 method for class 'SVDcv'
boxplot(x,
     y = c("all", "MSE", "R2", "AIC", "BIC"), ...)
```

Arguments

- `x` SVDcv object to plot.
- `y` Which CV-statistic to plot. For pairs "all" implies "BIC".
- `pairs` TRUE/FALSE plot cross-validation statistics, or scatter plot of individual BIC:s.
- `sd` TRUE/FALSE add uncertainty to each CV-statistic.
- `...` Additional parameters passed to `plot` or `pairs`.  

---

```r
# plot tim-series for the first site,
layout(matrix(c(1,2,3,1,2,4),3,2))
plot(mesa.model, "obs", ID=1, col=c("red", "black"))
# residuals from the temporal trends,
plot(mesa.model, "res", ID=1, col=c("black", "grey"))
# afc
plot(mesa.model, "acf", ID=1)
# ... and pacf for the residuals
plot(mesa.model, "pacf", ID=1, ci.col="red")
```

```r
# Different site and with no temporal trend.
mesa.model <- updateTrend(mesa.model, n.basis=0)
layout(matrix(c(1,2,3,1,2,4),3,2))
plot(mesa.model, "obs", ID="60370016")
plot(mesa.model, "res", ID="60370016")
plot(mesa.model, "acf", ID="60370016")
plot(mesa.model, "pacf", ID="60370016")
```
pred.mesa.model

Value
Nothing

Author(s)
Johan Lindström

See Also
Other SVD for missing data: calcSmoothTrends, print.SVDCv, summary.SVDCv, SVDmiss, SVDSmooth, SVDSmoothCV, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel
Other SVDcv methods: print.SVDCv, summary.SVDCv, SVDSmooth, SVDSmoothCV

Examples

# See SVDsmooth example

pred.mesa.model  Example of a predictSTmodel structure

Description
Example of a predictions for the model in mesa.model using predict.STmodel. Two sets of predictions are presented, pred.mesa.model and pred.mesa.model.obs.

Format
A list with elements, see the return description in predict.STmodel.

Source
Contains parameter estimates for the Spatio-Temporal model applied to monitoring data from the MESA Air project, see Cohen et.al. (2009) and mesa.data.raw for details.

References

See Also
predict.STmodel for prediction.
createSTmodel for creation of the originating STmodel object.
Other example data: est.cv.mesa, est.mesa.model, MCMC.mesa.model, mesa.data.raw, mesa.model
Examples

```r
# Load data
data(mesa.model)
data(est.mesa.model)

# Find regression parameters using GLS
x.reg <- predict(mesa.model, est.mesa.model, only.pars = TRUE)
str(x.reg$pars)

# Not run:
# Compute predictions at all locations, including beta-fields
pred.mesa.model <- predict(mesa.model, est.mesa.model,
                           pred.var=TRUE)

# End(Not run)
# Let's load precomputed results instead.
data(pred.mesa.model)

# Study results
print(pred.mesa.model)
```

predict.STmodel  Computes Conditional Expectation at Unobserved Locations

Description

Compute the conditional expectations (i.e., predictions) at the unobserved space-time locations. Predictions are computed for the space-time locations in `object` and/or `STdata`, conditional on the observations (and temporal trends) in `object` and parameters given in `x`.

Usage

```r
## S3 method for class 'STmodel'
predict(object, x, STdata = NULL,
        Nmax = 1000, only.pars = FALSE, nugget.unobs = 0,
        only.obs = FALSE, pred.var = TRUE, pred.covar = FALSE,
        beta.covar = FALSE, combine.data = FALSE, type = "p",
        LTA = FALSE, transform = c("none", "unbiased", "mspe"),
        ...)
```

Arguments

- `object`  STmodel object for which to compute predictions.
- `x` Model parameters for which to compute the conditional expectation. Either as a vector/matrix or an estimate STmodel from `estimate.STmodel`.
- `STdata` STdata/STmodel object with locations/times for which to predict. If not given predictions are computed for locations/times in `object`
\texttt{predict.STmodel}

- \texttt{Nmax} Limits the size of matrices constructed when computing expectations. Use a smaller value if memory becomes a problem.

- \texttt{only.pars} Compute only the regression parameters (using GLS) along with the related variance.

- \texttt{nugget.unobs} Value of nugget at unonserved locations, either a scalar or a vector with one element per unobserved site. \textbf{NOTE:} All sites in \texttt{STdata} are considered unobserved!

- \texttt{only.obs} Compute predictions at only locations specified by observations in \texttt{STdata}. Used to limit computations when doing cross-validation. \texttt{only.obs=TRUE implies pred.covar=FALSE and combine.data=FALSE}. Further \texttt{createSTmodel} will be called on any \texttt{STdata} input, possibly \texttt{reordering the observations}.

- \texttt{pred.var,pred.covar} Compute point-wise prediction variances; or compute covariance matrices for the predicted time series at each location. \texttt{pred.covar=TRUE implies pred.var=TRUE} and sets \texttt{Nmax} equal to the number of timepoints.

- \texttt{beta.covar} Compute the full covariance matrix for the latent beta-fields, otherwise only the diagonal elements of \texttt{V(beta|obs)} are computed.

- \texttt{combine.data} Combine object and \texttt{STdata} and predict for the joint set of points, see \texttt{c.STmodel}.

- \texttt{type} A single character indicating the type of prediction to compute. Valid options are "f", "p", and "r", for \texttt{full, profile or restricted maximum likelihood} (REML). For profile and full the predictions are computed assuming that both covariance parameters and regression parameters are known, e.g. \texttt{E(X|Y, cov_par, reg_par)}; for REML predictions are compute assuming only covariance parameters known, e.g. \texttt{E(X|Y, cov_par)}. The main difference is that REML will have \texttt{larger variances} due to the additional uncertainty in the regression parameters.

- \texttt{transform} Regard field as log-Gaussian and apply exponential transformation to predictions. For the final expectations two options exist, either a unbiased prediction or the (biased) mean-squared error predictions.

- \texttt{LTA} Compute long-term temporal averages. Either a logical value or a list; if \texttt{TRUE} then averages at each location (and variances if \texttt{pred.var=TRUE}) are computed; otherwise this should be a list with elements named after locations and each element containing a vector (or list of vectors) with dates over which to compute averages. If \texttt{only.obs=TRUE} averages are computed over only the observations.

- \texttt{...} Ignored additional arguments.

\textbf{Details}

In addition to computing the conditional expectation at a number of space-time locations the function also computes predictions based on only the regression part of the model as well as the latent beta-fields.

Prediction are computed as the conditional expectation of a latent field given observations. This implies that \( \mathbb{E}(X_i \mid Y_i) \neq Y_i \), with the difference being due to smoothing over the nugget. Further two possible variance can be computed (see below), \( \mathbb{V}(X_i \mid Y_i) \) and \( \mathbb{V}(X_i \mid Y_i) + \text{nugget}_i \). Here the nugget for unobserved locations needs to be specified as an additional argument \texttt{nugget.nobs}. The two variances correspond, loosely, to confidence and prediction intervals.
Variance are computed if `pred.var = TRUE` point-wise variances for the predictions (and the latent beta-fields) are computed. If instead `pred.cov = TRUE` the full covariance matrices for each predicted time series is computed; this implies that the covariances between temporal predictions at the same location are calculated but *not*, due to memory restrictions, any covariances between locations. `beta.cov = TRUE` gives the full covariance matrices for the latent beta-fields.

If `transform! = "none"` the field is assumed to be log-Gaussian and expectations are transformed, and if `pred.var = TRUE` the mean squared prediction errors are given.

## Value

The function returns a list containing (objects not computed will be missing):

- **opts**
  Copy of options used in the function call.

- **pars**
  A list with regression parameters and related variances. `pars` contain `gamma.E` and `alpha.E` with regression coefficients for the spatio-temporal model and land-use covaraiates; variances are found in `gamma.V` and `alpha.V`; cross-covariance between gamma and alpha in `gamma.alpha.C`.

- **beta**
  A list with estimates of the beta-fields, including the regression mean `mu`, conditional expectations `EX`, possibly variances `VX`, and the full covariance matrix `VX.full`.

- **EX.mu**
  Predictions based on the regression parameters, geographic covariates, and temporal trends. i.e. only the deterministic part of the spatio-temporal model.

- **EX.mu.beta**
  Predictions based on the latent-beta fields, but excluding the residual nu field.

- **EX**
  Full predictions at the space-time locations in `object` and/or `STdata`.

- **EX.pred**
  Only for `transform! = "none"`, full predictions including bias correction for prediction error.

- **VX, VX.pred**
  Pointwise variances and prediction variances (i.e. incl. contribution from `nugget.unobs`) for all locations in `EX`.

- **VX.full**
  A list with (number of locations) elements, each element is a (number of time-points) - by - (number of timepoints) temporal covariance matrix for the time-series at each location.

- **MSPE, MSPE.pred**
  Pointwise mean-square prediction errors for the log-Gaussian fields.

- **log.EX, log.VX.pred, log.VX**
  Pointwise predictions and variances for the un-transformed fields when `transform! = "none"`

- **LTA**
  A data.frame with temporal averages for locations specified by `LTA`.

- **I**
  A vector with the locations of the observations in `object` or `STdata`. To extract predictions at the observations locations use `EX[I]`.

## Author(s)

Johan Lindström
predictNaive

### Description

Computes naive predictions that are based on a few sites. These predictions can then be used, e.g. in `summary.predCVSTmodel`, to evaluate how much better the spatial-temporal model performs compared to simple (temporal) predictions.

The function requires one of `location` and `type` to be specified, if both are given `location` will be used over `type`. If `type` is given locations such that `as.character(STmodel$locations$type)` will be used.

### Usage

```r
predictNaive(STmodel, locations = NULL, type = NULL)
```
Arguments

*STmodel*  
STmodel object for which to compute simple predictions.

*locations*  
Locations on which to base the naive predictions.

*type*  
The type of sites to base the predictions on, uses the (optional) field `STmodel$locations$type`.

Details

Given a set of locations the function computes 4 sets of naive prediction for the observations in `STmodel`:

- **smooth.fixed**  
The smooth trend in `STmodel$trend` is fit to all observations at the sites in `locations` using a linear regression. The resulting smooth is used as a naive prediction for all locations.

- **avg.fixed**  
The temporal average over sites in `locations` is used as a naive prediction.

- **smooth.closest.fixed**  
This fits the smooth trend in `STmodel$trend` to each site in `locations`; using the smooth at the closest fixed site as a naive prediction.

- **closest.fixed**  
This uses the observations at the closest site in `locations` to predict observations at all other sites.

Value

A list with items:

- **pred**  
A (number of observations) - by - (6) data.frame containing the four naive predictions described under details, along with dates and IDs.

- **locations**  
The locations used for the naive predictions.

Author(s)

Johan Lindström

See Also

Other cross-validation functions: `computeLTA`, `createCV`, `dropObservations`, `estimateCV`, `estimateCV.STmodel`, `predictCV`, `predictCV.STmodel`

Other `STmodel` functions: `createCV`, `createDataMatrix`, `createSTmodel`, `dropObservations`, `estimateBetaFields`, `loglikeST`, `loglikeSTdim`, `loglikeSTnaive`, `processLocation`, `processLUR`, `processST`, `updateCovf`, `updateSTdataTrend`, `updateTrend`, `updateTrend.STdata`, `updateTrend.STmodel`

Examples

```r
# load data
data(mesa.model)

# naive predictions based on either AQS,
pred.aqs <- predictNaive(mesa.model, type="AQS")
#...or only one sites,
pred.1site <- predictNaive(mesa.model, locations="60372005")
```
print.estCVSTmodel

#plot the predictions - The two cases that are constant in space
par(mfcol=c(2,1), mar=c(4.5,4.5,1,.5))

#observations as a function of date
plot(mesa.model, "loc.obs", type=as.factor(mesa.model$locations$ID),
     legend.loc=NULL, pch=19, cex=.25)
##Add the predictions based on the smooth fitted to all sites
with(pred.aqs$pred, lines(date, smooth.fixed, col=1, lwd=2))
with(pred.$site$pred, lines(date, smooth.fixed, col=2, lwd=2))

#plot the predictions - One of the cases that vary in space, i.e. the smooth
##fit to the closest site.
D <- with(pred.aqs$pred, createDataMatrix(obs=smooth.closest.fixed,
                                          date=date, ID=1))

#observations as a function of date
##only five sites for clarity
mesa.model <- dropObservations(mesa.model, !(mesa.model$obs$idx %in% c(1,2,3,23,24)))
plot(mesa.model, "loc.obs", type=as.factor(mesa.model$locations$ID),
     legend.loc=NULL, pch=19, cex=.25)
##Add the predictions based on the smooth
##fitted to the closest site
for(i in 1:5){
  lines(as.Date(rownames(D)), D[,mesa.model$locations$ID[i]], col=i, lwd=2)
}

print.estCVSTmodel  Print details for estCVSTmodel object

Description

print method for class estCVSTmodel.

Usage

## S3 method for class 'estCVSTmodel'
print(x, ...)

Arguments

x estCVSTmodel object to print information for.
...
  Ignored additional arguments.

Value

Nothing
Author(s)
Johan Lindström

See Also
Other estCVSTmodel methods: boxplot.estCVSTmodel, coef.estCVSTmodel, estimateCV, estimateCV.STmodel, predictCV, predictCV.STmodel, print.summary.estCVSTmodel, summary.estCVSTmodel

Examples
```r
# load some data
data(est.cv.mesa)
# print basic information for the CV-predictions
print(est.cv.mesa)
```

print.estimateSTmodel  
*Print details for estimateSTmodel object*

Description
`print` method for class `estimateSTmodel`.

Usage
```r
## S3 method for class 'estimateSTmodel'
print(x, ...)
```

Arguments
- `x`: `estimateSTmodel` object to print information for.
- `...`: Ignored additional arguments.

Value
Nothing

Author(s)
Johan Lindström

See Also
Other `estimateSTmodel` methods: `coef.estimateSTmodel, estimate, estimate.STmodel`

Examples
```r
# load data
data(est.mesa.model)
print(est.mesa.model)
```
print.mcmcSTmodel  

Print details for mcmcSTmodel object

Description

print method for class mcmcSTmodel.

Usage

## S3 method for class 'mcmcSTmodel'
print(x, ...)

Arguments

x  
mcmcSTmodel object to print information for.

...  
Ignored additional arguments.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other mcmcSTmodel methods: density.mcmcSTmodel, MCMC, MCMC.STmodel, plot.density.mcmcSTmodel, plot.mcmcSTmodel, print.summary.mcmcSTmodel, summary.mcmcSTmodel

Examples

# load data
data(MCMC.mesa.model)
print(MCMC.mesa.model)
print.predCVSTmodel  

Description

`print` method for class `predCVSTmodel`.

Usage

```r
## S3 method for class 'predCVSTmodel'
print(x, ...)
```

Arguments

- `x`: `predCVSTmodel` object to print information for.
- `...`: Ignored additional arguments.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other `predCVSTmodel` methods: `estimateCV`, `estimateCV.STmodel`, `plot.predCVSTmodel`, `plot.predictSTmodel`, `predictCV`, `predictCV.STmodel`, `print.summary.predCVSTmodel`, `qqnorm.predCVSTmodel`, `qqnorm.STdata`, `qqnorm.STmodel`, `scatterPlot.predCVSTmodel`, `scatterPlot.STdata`, `scatterPlot.STmodel`, `summary.predCVSTmodel`

Examples

```r
## load some data
data(pred.cv.mesa)
## print basic information for the CV-predictions
print(pred.cv.mesa)
```


Description

`print` method for class `predictSTmodel`.

Usage

```r
## S3 method for class 'predictSTmodel'
print(x, ...)
```

Arguments

- `x` predictSTmodel object to print information for.
- `...` Ignored additional arguments.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other predictSTmodel methods: `plot.predCVSTmodel, plot.predictSTmodel, predict.STmodel`

Examples

```r
# load data
data(pred.mesa.model)
print(pred.mesa.model)
```

Description

`print` method for class `STdata`.

Usage

```r
## S3 method for class 'STdata'
print(x, type = x$covars$type, ...)
```
Arguments

- **x**: STdata object to print information for.
- **type**: Factorial of `length(x$covars$ID)`, if not NULL the output also presents summaries of number of sites and observations as well as time periods per type of site.
- ... Ignored additional arguments.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other STdata methods: `createSTdata`, `plot.STdata`, `plot.STmodel`, `print.summary.STdata`, `qqnorm.predCVSTmodel`, `qqnorm.STdata`, `qqnorm.STmodel`, `scatterPlot.predCVSTmodel`, `scatterPlot.STdata`, `scatterPlot.STmodel`, `summary.STdata`

---

**print.STmodel**

Print details for STmodel object

Description

`print` method for class `STmodel`.

Usage

```r
## S3 method for class 'STmodel'
print(x, type = x$locations$type, ...)
```

Arguments

- **x**: STmodel object to print information for.
- **type**: Factorial of `length(x$locations$ID)`, if not NULL the output also presents summaries of number of sites and observations as well as time periods per type of site.
- ... Ignored additional arguments.

Value

Nothing
Author(s)
Johan Lindström

See Also

Examples

```r
# load some data
data(mesa.model)
# print basic information regarding obs, locations, dates, etc
print(mesa.model)
```

Description

Print details for summary.estCVSTmodel object

Usage

```r
# S3 method for class 'summary.estCVSTmodel'
print(x, ...)
```

Arguments

- `x` summary.estCVSTmodel object to print information for.
- `...` Additional arguments, passed to `print.table`.

Value

Nothing

Author(s)
Johan Lindström

See Also
Other estCVSTmodel methods: boxplot.estCVSTmodel, coef.estCVSTmodel, estimateCV, estimateCV.STmodel, predictCV, predictCV.STmodel, print.estCVSTmodel, summary.estCVSTmodel
print.summary.mcmcSTmodel

Print details for `summary.mcmcSTmodel` object

Description

`print` method for class `summary.mcmcSTmodel`.

Usage

```r
### S3 method for class 'summary.mcmcSTmodel'
print(x, ...)
```

Arguments

- `x`  
  `summary.mcmcSTmodel` object to print information for.
- `...`  
  Additional arguments, passed to `print.table`.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other `mcmcSTmodel` methods:  
- `density.mcmcSTmodel`  
- `MCMC.MCMC.STmodel`  
- `plot.density.mcmcSTmodel`  
- `plot.mcmcSTmodel`  
- `print.mcmcSTmodel`  
- `summary.mcmcSTmodel`
Arguments

x summary.predCVSTmodel object to print information for.
... Additional arguments, passed to print.table.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other predCVSTmodel methods: estimateCV, estimateCV.STmodel, plot.predCVSTmodel, plot.predictSTmodel, predictCV, predictCV.STmodel, print.predCVSTmodel, qqnorm.predCVSTmodel, qqnorm.STdata, qqnorm.STmodel, scatterPlot.predCVSTmodel, scatterPlot.STdata, scatterPlot.STmodel, summary.predCVSTmodel

print.summary.STdata  Print details for summary.STdata object

Description

print method for class summary.STdata.

Usage

## S3 method for class 'summary.STdata'
print(x, ...)

Arguments

x summary.STdata object to print information for.
... Ignored additional arguments.

Value

Nothing

Author(s)

Johan Lindström

See Also

Other STdata methods: createSTdata, plot.STdata, plot.STmodel, print.STdata, qqnorm.predCVSTmodel, qqnorm.STdata, qqnorm.STmodel, scatterPlot.predCVSTmodel, scatterPlot.STdata, scatterPlot.STmodel, summary.STdata
print.summary.STmodel  

Print details for summary.STmodel object

Description

print method for class summary.STmodel.

Usage

## S3 method for class 'summary.STmodel'
print(x, ...)

Arguments

x  
summary.STmodel object to print information for.

...  
Ignored additional arguments.

Value

Nothing

Author(s)

Johan Lindström

See Also


print.SVDcv  

Print details for SVDcv object

Description

print and summary methods for class SVDcv, prints cross-validation statistics.

Usage

## S3 method for class 'SVDcv'
print(x, ...)

## S3 method for class 'SVDcv'
summary(object, ...)
**processLocation**

**Arguments**

- `x` SVDCv object to print information for.
- `...` ignored additional arguments.
- `object` SVDCv object to compute summary for.

**Value**

Nothing

**Author(s)**

Johan Lindström

**See Also**

Other SVD for missing data: `boxplot.SVDCv`, `calcSmoothTrends`, `plot.SVDCv`, `SVDmiss`, `SVDSmooth`, `SVDSmoothCV`, `updateSTdataTrend`, `updateTrend`, `updateTrend.STdata`, `updateTrend.STmodel`

Other SVDCv methods: `boxplot.SVDCv`, `plot.SVDCv`, `SVDSmooth`, `SVDSmoothCV`

**Examples**

```r
## See SVDSmooth example
```

---

**Description**

Function that creates a data.frame of locations (and auxiliary information) from STdata$covars, used by `createSTmodel`.

**Usage**

```r
processLocation(STdata, locations)
```

**Arguments**

- `STdata` STdata object with observations, covariates, trends, etc; see `mesa.data.raw`.
- `locations` A list specifying which fields in STdata$covars that should be used for what in the location data.frame, see details.
Details

The `locations` list specifies what should go in the locations data.frame, in addition to thing listed below `STdata$covars$ID` is always added. Each of the fields below should contain names (as character) of columns in `STdata$covars`:

- **coords**  The x,y-coordinates for monitors
- **coords.beta,coords.nu**  Alternative x,y-coordinates for monitors, used when computing distance-matrices for the beta- and nu-fields. Allows the use of non-stationary covariance structures through the deformation method of Damian (2003), given a precomputed deformation.
- **long.lat**  The long,lat-coordinates for monitors
- **others**  Additional fields in `STdata$covars` that should be added to the location data.frame

Value

A data.frame with location information for all the sites.

Author(s)

Johan Lindström

References


See Also

Other STmodel functions: `createCV`, `createDataMatrix`, `createSTmodel`, `dropObservations`, `estimateBetaFields`, `loglikeST`, `loglikeSTdim`, `loglikeSTnaive`, `predictNaive`, `processLUR`, `processST`, `updateCovf`, `updateSTdataTrend`, `updateTrend`, `updateTrend.STdata`, `updateTrend.STmodel`

Examples

```r
# load the data
data(mesa.data.raw)

# and create STdata-object
mesa.data <- createSTdata(mesa.data.raw$obs, mesa.data.raw$X, n.basis=2,
                          SpatioTemporal=mesa.data.raw["lax.conc.1500"])

# specify locations, using x/y and specifying long/lat and picking
# type as an additional field
loc.spec <- list(coords=c("x","y"), long.lat=c("long","lat"), others="type")
# create the location data.frame
str( processLocation(mesa.data, loc.spec) )

# specify only locations
str( processLocation(mesa.data, list(coords=c("x","y"))) )

# different coordinates for beta and nu fields
loc.spec <- list(coords=c("x","y"), coords.nu=c("long","lat"))
str( processLocation(mesa.data, loc.spec) )
```
**processLUR**

*Internal Function that do Covariate Selection*

**Description**

Function that create covariate specifications for `createSTmodel`, and compare the covariates requested (both geographic and spatio-temporal) with those available in `stdata`.

**Usage**

```r
processLUR(STdata, LUR.in)
processST(STdata, ST.in)
```

**Arguments**

- **STdata**: `STdata` object with observations, covariates, trends, etc; see `mesa.data.raw`.
- **LUR.in**: A vector or list indicating which geographic covariates to use.
- **ST.in**: A vector indicating which spatio-temporal covariates to use.

**Details**

Several options exist for `LUR.in`

- `LUR.in=NULL` Only an intercept for all beta-fields.
- `LUR.in="all"` Use all elements in `STdata$covars`, *NOT* recommended.
- `LUR.in=list(...)` Use different covariates for each, specified by the different components of the list.
- `LUR.in=vector` Use the same covariates for all beta-field.

For the two last options the vector/list-elements can contain either:

- **integer**: This will be used as `names(STdata$covars)[int]` to extract a character vector (see below) of covariates.
- **character**: The character vector will be used to create a `formula` (see below), through:
  ```r
  as.formula(paste("~", paste(unique(chars), collapse="+")), env=.GlobalEnv)
  ```

- **formula**: The formula will be used as `model.matrix(formula, STdata$covars)` to create a covariate matrix.

Setting any element(s) of the list to NULL implies *only an intercept* for the corresponding temporal trend(s).

`ST.in` should be a vector specifying the spatio-temporal covariates to use; the vector either give names or layers in `STdata$SpatioTemporal` to use, compare character and integer options for `LUR.in` above.

If covariates are specified using names these should match `dimnames(STdata$SpatioTemporal)[[3]]`, unmatched elements are dropped with a warning.
Value

A list of LUR specifications, as `formula`; or a ST specification as a character vector.

Author(s)

Johan Lindström

See Also

Other STmodel functions: `createCV`, `createDataMatrix`, `createSTmodel`, `dropObservations`, `estimateBetaFields`, `loglikeST`, `loglikeSTdim`, `loglikeSTnaive`, `predictNaive`, `processLocation`, `updateCovf`, `updateSTdataTrend`, `updateTrend`, `updateTrend.STdata`, `updateTrend.STmodel`

Examples

```R
# load the data
data(mesa.data.raw)
# and create ST-data-object
mesa.data <- createSTdata(mesa.data.raw$obs, mesa.data.raw$X, n.basis=2,
                          SpatioTemporal=mesa.data.raw["lax.conc.1500"])

# create a simple set of covariates
processLUR(mesa.data, list(c(7:9),7,8))

# or a structure with the same covariates for all
temporal trends
processLUR(mesa.data, c(7,11))

# or a structure with only intercept for the temporal trends
processLUR(mesa.data, list(c(7:9),NULL,NULL))

# Ask for covariates by name
processLUR(mesa.data, list(c("log10.m.to.a1","log10.m.to.a2"),
                          "log10.m.to.a1","log10.m.to.a1"))

# use formula for part of it
processLUR(mesa.data, list(~log10.m.to.a1+log10.m.to.a2+log10.m.to.a1*km.to.coast,
                          "log10.m.to.a1", "log10.m.to.a1"))

# Ask for non-existent covariate by name or formula, or location
# for each temporal trend
try(processLUR(mesa.data, list("log10.m.to.a4",~log10.m.to.a1+log10.m.to.a4, 25)))

# create a simple set of spatio-temporal covariates
processST(mesa.data, 1)
# or create a empty set of spatio-temporal covariates
processST(mesa.data, NULL)
# by name
processST(mesa.data, "lax.conc.1500")
```
Description

`qqnorm` method for classes `Stdata/Stmodel/predCVSTmodel`. Used for data and residual analysis of the cross validation.

Usage

```r
## S3 method for class 'predCVSTmodel'
qqnorm(y, ID = "all",
       main = "Q-Q plot for CV residuals", group = NULL,
       col = 1, norm = FALSE, line = 0, org.scale = TRUE, ...)

## S3 method for class 'Stdata'
qqnorm(y, ID = "all",
       main = "Q-Q plot for observations", group = NULL,
       col = 1, line = 0, ...)

## S3 method for class 'Stmodel'
qqnorm(y, ID = "all",
       main = "Q-Q plot for observations", group = NULL,
       col = 1, line = 0, ...)
```

Arguments

- **norm** TRUE/FALSE, plot normalised (mean=0, sd=1) or raw cross-validation residuals. If `norm=TRUE` a 0-1 line is added, to indicate what normalised residuals should look like.
- **org.scale** TRUE/FALSE scatter plots on the original untransformed scale, or using `exp(y)`. Only relevant if `x` was computed using `transform` in `predictCVSTmodel` (as pass through argument to `predictSTmodel`)
- **y** `Stdata/Stmodel/predCVSTmodel` object for the `qqnorm`.
- **ID** The location for which we want to norm-plot observations/residuals or "all" to plot for all locations.
- **main** Title of the plot
- **group** Do the norm-plot both for all data and then for each subset defined by the factor/levels in group variable.
- **col** Colour of points in the plot, either a scalar or a vector with length matching the number of observations/residuals.
- **line** If non-zero add a `qqline` with `lty=line`, to the plot; if 0 do not add a line.
- **...** Arguments passed on to the plotting function, `qqnorm`
qqnorm.predCVSTmodel

Value

Nothing

Author(s)

Johan Lindström

See Also


Other STdata methods: createSTdata, plot, plot.STdata, plot.STmodel, print, print.STdata, plot, plot.STdata, plot.STmodel, print, print.STdata, plot, plot.STdata, plot.STmodel, summary, print


Examples

#########################################################################
## Example for STdata/STmodel ##
#########################################################################
# load data
data(mesa.model)

# standard plot
qqnorm(mesa.model)
# add a line, and group (and colour) by AQs/FIXED
par(mfrow=c(2,2))
obs.type <- mesa.model$locations$type[match(mesa.model$obs$ID, mesa.model$locations$ID)]
qqnorm(mesa.model, line=1, group=obs.type, col=obs.type)

# colour code by season and split by type
# first create a vector dividing data into four seasons
I.season <- as.factor(as.POSIXlt(mesa.model$obs$date)$mon+1)
levels(I.season) <- c(rep("Winter", 2), rep("Spring", 3), rep("Summer", 3), rep("Fall", 3), "Winter")
par(mfrow=c(2,2))
qqnorm(mesa.model, line=1, col=I.season, group=obs.type)
legend("bottomright", legend=as.character(levels(I.season)), pch=1, col=1:nlevels(I.season))

#########################################################################
## Example for predCVSTmodel ##
#########################################################################
# load data
removeSTcovarMean

Mean-Centre the Spatio-Temporal Covariate

Description

Removes the temporal mean at each location for the spatio-temporal covariates. The means are added to the covar field in the returned object and can be used as geographic covariates.

Usage

removeSTcovarMean(STdata)

Arguments

STdata A STdata object, see mesa.data.raw.

Value

Returns a modified version of the input, where the spatio-temporal covariates have been expanded to include covariates where the site by site temporal average has been removed. The averages are seen as geographic covariates and added to STdata$covars.

Author(s)

Johan Lindström

See Also

Other STdata functions: c.STmodel, createDataMatrix, createSTdata, createSTmodel, detrendSTdata, estimateBetaFields, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel
Examples

```r
# load the data
data(mesa.data.raw)
# and create STdata-object
mesa.data <- createSTdata(mesa.data.raw$obs, mesa.data.raw$x, n.basis=2,
                         SpatioTemporal=mesa.data.raw["lax.conc.1500"])

mesa.data.mean0 <- removeSTcovarMean(mesa.data)

# compare the data structures
# geographic covariates
summary(mesa.data$covars)
summary(mesa.data.mean0$covars)

# mean of the spatio-temporal covariate, note that the new
# contains both mean-zero and original
cbind(colMeans(mesa.data$SpatioTemporal),
     colMeans(mesa.data.mean0$SpatioTemporal))
```

dataset

**Description**

Scatter plot of data in x

**Usage**

```r
cscatterPlot(x, ...)
```

**Arguments**

- `x` object to scatter plot
- `...` additional parameters

**Value**

Nothing

**Author(s)**

Johan Lindström
scatterPlot.predCVSTmodel

scatterPlot for STdata/STmodel/predCVSTmodel objects

Description

Does a scatterPlot of observations/residuals against covariates (either geographic or temporal trends), adding a spline fit (similar to `scatter.smooth`).

Usage

```r
## S3 method for class 'predCVSTmodel'
s ScatterPlot(x, covar = NULL,
        trend = NULL, pch = 1, col = 1, cex = 1, lty = 1,
        subset = NULL, group = NULL, add = FALSE,
        smooth.args = NULL, STdata,
        type = c("obs", "res", "res.norm"), org.scale = TRUE,
        ...)  

## S3 method for class 'STdata'
s ScatterPlot(x, covar = NULL,
        trend = NULL, pch = 1, col = 1, cex = 1, lty = 1,
        subset = NULL, group = NULL, add = FALSE,
        smooth.args = NULL, ...)  

## S3 method for class 'STmodel'
s ScatterPlot(x, covar = NULL,
        trend = NULL, pch = 1, col = 1, cex = 1, lty = 1,
        subset = NULL, group = NULL, add = FALSE,
        smooth.args = NULL, ...)  
```

Arguments

- **type**
  - What to use in the scatter plot, valid options are "obs" for observations, "res" residuals, and "res.norm" for normalised residuals.

- **STdata**
  - STdata or STmodel containing covariates and trend against which to plot.

- **org.scale**
  - TRUE/FALSE scatter plots on the original untransformed scale, or using exp(y).
  - Only relevant if x was computed using transform in `predictCV.STmodel` (as pass through argument to `predict.STmodel`)

- **x**
  - STdata/STmodel/predCVSTmodel object to plot.

- **covar,trend**
  - Plot observations as a function of? Only one of these should be not NULL. covar uses location covariates, and trend uses temporal trend (or dates); trend=0 uses a temporal intercept (i.e. a constant).

- **pch,cex**
  - Point and point size for the plot, a single value or nlevels(group)
col, lty

Color of points and smooth lines. A single value or nlevels(group)+1 values; the last value is used for fitting a line to all data. Use lty=NA to suppress smooth lines.

subset

A subset of locations for which to plot observations as a function of covariates.

group

A vector of factors of the same length as the number of observations (typically length(x$obs$obs), or length(x$pred.obs$obs)) used to group data and fit different smooths to each group.

add

Add to existing plot

smooth.args

List of arguments for loess.smooth.

Value

Nothing

Author(s)

Johan Lindström

See Also


Other STdata methods: createSTdata, plot.STdata, plot.STmodel, print.STdata, print.summary.STdata, qqnorm.predCVSTmodel, qqnorm.STdata, qqnorm.STmodel, summary.STdata


Examples

### Example for STdata/STmodel ###

```r
# load data
data(mesa.model)

par(mfrow=c(2,2))

# plot observations as a function of longitude for an STmodel object
scatterPlot(mesa.model, covar="long")

# as a function of the first temporal trend, subset to only AQS sites
# and fit for each location
scatterPlot(mesa.model, trend=1, col=c(1:25,1), pch=19, cex=.1,
group=mesa.model$obs$ID, lty=rep(2,25),1),
subset=with(mesa.model$locations, ID[type=="AQS"])
```
if plotting against the distance to coast, we might have to change the
smoothing.
suppressWarnings( scatterPlot(mesa.model, covar="km.to.coast") )

better
scatterPlot(mesa.model, covar="km.to.coast", col=c(NA,2), add=TRUE,
smooth.args=list(span=4/5, degree=2))

Lets group data by season
First create a vector dividing data into four seasons
I.season <- as.factor(as.POSIXlt(mesa.model$obs$date)$mon+1)
levels(I.season) <- c(rep("Winter",2), rep("Spring",3),
rep("Summer",3), rep("Fall",3), "Winter")
scatterPlot(mesa.model, covar="log10.m.to.a1", col=c(2:5,1),
group=I.season)
legend("bottomleft", c(levels(I.season),"All"), col=c(2:5,1), pch=1)

Example for predCVSTmodel

# load data
data(pred.cv.mesa)

simple case of residuals against temporal trends
par(mfrow=c(2,1))
scatterPlot(pred.cv.mesa, trend=1, stdata=mesa.model, type="res")

colour coded by season
I.season <- as.factor(as.POSIXlt(pred.cv.mesa$pred.obs$date)$mon+1)
levels(I.season) <- c(rep("Winter",2), rep("Spring",3),
rep("Summer",3), rep("Fall",3), "Winter")
scatterPlot(pred.cv.mesa, trend=1, stdata=mesa.model, type="res",
group=I.season, col=c(2:5,1), lty=c(1,1,1,1,2),
smooth.args=list(span=.1, degree=2))

or as function of covariates
par(mfcol=c(2,2))
scatterPlot(pred.cv.mesa, , type="res", covar="log10.m.to.a1", stdata=mesa.model, group=I.season, col=c(2:5,1))
scatterPlot(pred.cv.mesa, type="res", covar="km.to.coast", stdata=mesa.model, group=I.season, col=c(2:5,1),
smooth.args=list(span=4/5, degree=1))

lets compare to the original observations
scatterPlot(pred.cv.mesa, covar="log10.m.to.a1", stdata=mesa.model, group=I.season, col=c(2:5,1), type="obs")
scatterPlot(pred.cv.mesa, covar="km.to.coast", stdata=mesa.model, group=I.season, col=c(2:5,1), type="obs",
smooth.args=list(span=4/5, degree=1))
Simulate Data from the Spatio-Temporal Model

Description
Data is simulated for the space-time locations in `object` using the parameters in `x`.

Usage
```r
## S3 method for class 'STmodel'
simulate(object, nsim = 1,
         seed = NULL, x, nugget.unobs = 0, ...)
```

Arguments
- `object`: A `STmodel` object to perform unconditional simulation from.
- `nsim`: Number of replicates to simulate.
- `seed`: if !=NULL used in a call to `set.seed`, allowing for replicatable simulation studies.
- `x`: Parameters to use when simulating the data; both regression and covariance parameters must be given, see `loglikeSTgetPars`.
- `nugget.unobs`: Value of nugget at unonserved locations, either a scalar or a vector with one element per unobserved site.
- `...`: Additional parameters for `set.seed`

Value
A list containing:
- `param`: Parameters used in the simulation, i.e. `x`.
- `B`: The simulated beta fields in a (number of locations) - by - (number of temporal trends) - by - (number of replicates) array.
- `X`: The simulated spatio-temporal fields in a (number of timepoints) - by - (number of locations) - by - (number of replicates) array. Row and column names indicate the time and locations for each point.
- `obs`: A list with one element per replicate, containing the simulated observations extracted at space-time locations matching those in `object$obs`. To replace the observations with the i:th simulated values do: `object$obs <- res$obs[[i]]`.

Author(s)
Johan Lindström
**stCheckClass**

**Test if an object belongs to given class(es).**

**Description**

Test if an object belongs to given class(es), and produce reasonable error message if not.

**Usage**

```r
stCheckClass(x, what, name = "Object")
```

---

**See Also**

stCheckCovars

Arguments

- **x**: Object to test.
- **what**: A character vector naming classes.
- **name**: Character string to be pasted into the error message describing x.

Value

- Nothing

Author(s)

- Johan Lindström

See Also

- Similar to inherits
- Other object checking utilities: stCheckCovars, stCheckFields, stCheckObs, stCheckSTcovars

Examples

```r
# create a basic object
x <- 1
class(x) <- "test"
# should be ok
stCheckClass(x, "test", "x")
# this fails
try(stCheckClass(x, "other", "x"))
```

Description

Checks that data.frame of covariates is valid, making sure that all locations specified in ID.unique exist. The function will attempt to name each row in covars using 1) covars$ID, 2) rownames(covars), and 3) as.character(1:dim(covars)[1]). The field covars$ID is added if missing and rownames are removed.

Usage

```r
stCheckCovars(covars, ID.unique = character(0))
```

Arguments

- **covars**: data.frame containing covariates, to be checked.
- **ID.unique**: vector with unique IDs that HAVE to be present in the covariates, typically the observation locations.
stCheckFields

Value

Updated covars data.frame.

Author(s)

Johan Lindström

See Also

Other object checking utilities: stCheckClass, stCheckFields, stCheckObs, stCheckSTcovars

Examples

```r
# load data
data(mesa.model)

# check covariates
tmp <- stCheckCovars( mesa.model$locations, mesa.model$locations$ID )
str(tmp)

# require non-existant site
try( stCheckCovars( mesa.model$locations, "Bad.Site" ) )

# drop the ID
mesa.model$locations$ID <- NULL
tmp <- stCheckCovars( mesa.model$locations )

# ID:s infered from rownames (1-25)
str(tmp)
```

stCheckFields

Test if fields exist in an object.

Description

Test if named fields exist in name(x), if not the function fails with a suitable error message.

Usage

```r
stCheckFields(x, what, name = "Object")
```

Arguments

- **x**: Object to test.
- **what**: A character vector naming that should occur in names(x).
- **name**: Character string to be pasted into the error message describing x.

Value

- **Nothing**
**Author(s)**

Johan Lindström

**See Also**

Other object checking utilities: `stCheckClass`, `stCheckCovars`, `stCheckObs`, `stCheckSTcovars`

**Examples**

```r
# load data
data(mesa.model)
# names present in data
names(mesa.model$locations)

# check for some names
stCheckFields(mesa.model$locations, c("ID","x","lat"))
# check for non-existant names
try(stCheckFields(mesa.model$locations, c("ID","x","test")))
```

---

**stCheckObs**

*Check an obs data.frame.*

**Description**

Checks that an observation data.frame is valid.

**Usage**

```r
stCheckObs(obs)
```

**Arguments**

- `obs` data.frame to be checked.

**Details**

A valid observation data.frame needs to fulfill:

- Contains fields `obs`, `date`, and `ID`
- All elements in `obs$obs` are finite
- `obs$date` is one of `Date`, `numeric`, or `integer`
- `obs$ID` is character
- No duplicated observations (same `ID` and `date`)

**Value**

Nothing
stCheckSTcovars

Author(s)
Johan Lindström

See Also
Other object checking utilities: stCheckClass, stCheckCovars, stCheckFields, stCheckSTcovars

Examples

```r
# load data
data(mesa.model)

# check observations
stCheckObs( mesa.model$obs )
# some possible failures
mesa.model$obs <- rbind(mesa.model$obs, mesa.model$obs[1,])
try( stCheckObs( mesa.model$obs ) )
mesa.model$obs$obs[1] <- NaN
try( stCheckObs( mesa.model$obs ) )
mesa.model$obs$date <- as.character( mesa.model$obs$date )
try( stCheckObs( mesa.model$obs ) )
mesa.model$obs$date <- NULL
try( stCheckObs( mesa.model$obs ) )
```

stCheckSTcovars

Check an Array/List of Spatio-Temporal Covariates

Description

Checks that array/list of spatio-temporal covariates is valid, making sure that at least all locations specified in `ID.unique` exist. The function will attempt to name extract locations ID’s from `colnames(ST)` and observation dates from `rownames(ST)` (using `convertCharToDate`).

Usage

```r
stCheckSTcovars(ST, ID.unique = character(0),
                 date.unique = integer(0))
```

Arguments

- **ST** A 3D-array containing the ST-covariates, or a list of arrays, the list elements have to be of matching sizes and have the same rownames and colnames; list elements are stacked to form a 3D-array.
- **date.unique** vector with unique dates/times that HAVE to be present in the ST-covariates, typically the observation time-points.
- **ID.unique** vector with unique IDs that HAVE to be present in the ST-covariates, typically the observation locations and un-observation locations for predictions.
sumLogDiag

Value

Updated ST array

Author(s)

Johan Lindström

See Also

Other object checking utilities: stCheckClass, stCheckCovars, stCheckFields, stCheckObs

Examples

```r
# load data
data(mesa.model)

c # check covariates
tmp <- stCheckSTcovars( mesa.model$ST.all, mesa.model$locations$ID )
str(tmp)
# require non-existant site
try( stCheckSTcovars( mesa.model$ST.all, "Bad.Site" ) )
# require non-existant site
try( stCheckSTcovars( mesa.model$ST.all, date.unique=1 ) )
```

Sum the Logarithm of (Diagonal) Elements

Description

Computes the sum of the logarithm of the diagonal elements in a matrix, or of elements in a vector. This corresponds to the logarithm of the determinant for a Cholesky factor. Behaviour is undefined for any elements that are <=0.

Usage

```r
sumLogDiag(mat)

sumLog(v)
```

Arguments

- `mat`: A square matrix (preferably a Cholesky factor).
- `v`: A vector

Value

Sum of the logarithm of the (diagonal) elements.
**summary.estCVSTmodel**

**Computes summary details for estCVSTmodel object**

**Description**

`summary` method for class `estCVSTmodel`.

**Usage**

```r
## S3 method for class 'estCVSTmodel'
summary(object, ...)```

**Arguments**

- `object` estCVSTmodel object to compute summary information for.
- `...` Ignored additional arguments.

**Value**

A `summary.estCVSTmodel` object.

**Author(s)**

Johan Lindström
See Also

Other estCVSTmodel methods: boxplot.estCVSTmodel, coef.estCVSTmodel, estimateCV, estimateCV.STmodel, predictCV, predictCV.STmodel, print.estCVSTmodel, print.summary.estCVSTmodel

Examples

```r
# load some data
data(est.cv.mesa)
# print basic information for the CV-predictions
summary(est.cv.mesa)

summary.mcmcSTmodel  Computes summary details for mcmcSTmodel object

Description

`summary` method for class mcmcSTmodel.

Usage

```r
## S3 method for class 'mcmcSTmodel'
summary(object, burnIn = 0, ...)
```

Arguments

- `object`: mcmcSTmodel object to compute summary information for.
- `burnIn`: Number of initial iterations to drop.
- `...`: Ignored additional arguments.

Value

A `summary.mcmcSTmodel` object.

Author(s)

Johan Lindström

See Also

Other mcmcSTmodel methods: density.mcmcSTmodel, MCMC, MCMC.STmodel, plot.density.mcmcSTmodel, plot.mcmcSTmodel, print.mcmcSTmodel, print.summary.mcmcSTmodel

Examples

```r
# load data
data(MCMC.mesa.model)
summary(MCMC.mesa.model)
```
Computes summary details for predCVSTmodel object

Description

summary method for class predCVSTmodel.

Usage

## S3 method for class 'predCVSTmodel'
summary(object,
pred.naive = NULL, by.date = FALSE, p = 0.95,
transform = function(x) { return(x) }, LTA = FALSE,
...)

Arguments

object  predCVSTmodel object to compute summary information for; the output from predictCV.STmodel.
pred.naive Result of naive prediction; used to compute modified R2 values. The output from predictNaive.
by.date Compute individual cross-validation statistics for each time-point. May lead to very many statistics.
p Approximate coverage of the computed confidence bands; the confidence bands are used when computing coverage of the cross-validated predictions.
transform Transform observations and predictions (without bias correction) before computing statistics; see also computeLTA. Redundant if option transform was used in predictCV.STmodel (as pass through argument to predict.STmodel)
LTA Compute cross-validation statistics for the long term averages at each site, uses computeLTA to compute the averages. transform is passed to computeLTA. This is redundant if option LTA=TRUE was uses in predictCV.STmodel.
... Ignored additional arguments.

Details

Computes summary statistics for cross validation. Statistics that are computed include RMSE, R2, and coverage of CI:s; both for all observations and (possibly) stratified by date.

Value

A summary.predCVSTmodel object.

Author(s)

Johan Lindström
See Also

Other predCVSTmodel methods: estimateCV, estimateCV.STmodel, plot.predCVSTmodel, plot.predictSTmodel, predictCV, predictCV.STmodel, print.predCVSTmodel, print.summary.predCVSTmodel, qnorm.predCVSTmodel, qnorm.STdata, qnorm.STmodel, scatterPlot.predCVSTmodel, scatterPlot.STdata, scatterPlot.STmodel

Examples

```r
# load some data
data(pred.cv.mesa)

# basic summary statistics
summary(pred.cv.mesa)
```

---

**summary.STdata**

*Computes summary details for STdata object*

Description

`summary` method for class `STdata`.

Usage

```r
## S3 method for class 'STdata'
summary(object, 
   type = object$covars$type, ...) 
```

Arguments

- `object` STdata object to compute summary information for.
- `type` Factorial of `length(x$covars$ID)`, if not NULL summaries for the observations are computed per type of site.
- `...` Ignored additional arguments.

Value

A `summary.STdata` object.

Author(s)

Johan Lindström

See Also

Other STdata methods: `createSTdata`, `plot.STdata`, `plot.STmodel`, `print.STdata`, `print.summary.STdata`, `qnorm.predCVSTmodel`, `qnorm.STdata`, `qnorm.STmodel`, `scatterPlot.predCVSTmodel`, `scatterPlot.STdata`, `scatterPlot.STmodel`
**summary.STmodel**  
Computes summary details for STmodel object

**Description**

*summary* method for class STmodel.

**Usage**

```r
## S3 method for class 'STmodel'
summary(object, 
     type = object$covars$type, ...)
```

**Arguments**

- `object` STmodel object to compute summary information for.
- `type` Factorial of length(x$locations$ID), if not NULL the output also presents summaries of number of sites and observations as well as time periods per type of site.
- `...` Ignored additional arguments.

**Value**

A `summary.STmodel` object.

**Author(s)**

Johan Lindström

**See Also**


**Examples**

```r
# load some data
data(mesa.model)
# Summary of data fields.
summary(mesa.model)
```
**SVDmiss**  

*Missing Data SVD*

**Description**

Function that completes a data matrix using iterative svd as described in Fuentes et. al. (2006). The function iterates between computing the svd for the matrix and replacing the missing values by linear regression of the columns onto the first ncomp svd components. As initial replacement for the missing values regression on the column averages are used. The function will fail if entire rows and/or columns are missing from the data matrix.

**Usage**

```r
SVDmiss(X, niter = 25, ncomp = min(4, dim(X)[2]), conv.reldiff = 0.001)
```

**Arguments**

- `X` Data matrix, with missing values marked by NA.
- `niter` Maximum number of iterations to run before exiting. Inf will run until the conv.reldiff criteria is met.
- `ncomp` Number of SVD components to use in the reconstruction (>0).
- `conv.reldiff` Assume the iterative procedure has converged when the relative difference between two consecutive iterations is less than conv.reldiff.

**Value**

A list with the following components:

- `Xfill` The completed data matrix with missing values replaced by fitting the data to the ncomp most important svd components.
- `svd` The result of svd on the completed data matrix, i.e. svd(Xfill).
- `status` A vector of status variables: diff, the absolute difference between the two last iterations; rel.diff, the relative difference; n.iter, the number of iterations; and max.iter, the requested maximum number of iterations.

**Author(s)**

Paul D. Sampson and Johan Lindström

**References**

SVDsmooth

See Also

Other data matrix: createDataMatrix, estimateBetaFields, mesa.data.raw, SVDsmooth, SVDsmoothCV

Other SVD for missing data: boxplot.SDcv, calcSmoothTrends, plot.SDcv, print.SDcv, summary.SDcv, SVDsmooth, SVDsmoothCV, updateSTdataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel

Examples

```r
# create a data matrix
x <- seq(0, 4*pi, len=50)
x.org <- cbind(cos(t), sin(2*t))  # matrix(rnorm(20), 2, 10)

# add some normal errors
x <- x.org + .25*rnorm(length(x.org))

# and mark some data as missing
x[runif(length(x))<.25] <- NA

# Ensure that we have complete columns/rows
while( any(rowSums(is.na(x)) == dim(x)[2]) || any(colSums(is.na(x)) == dim(x)[1]) ){
  x <- x.org + .25*rnorm(length(x.org))
  x[runif(length(x))<.25] <- NA
}

# run the missing data svd
res <- SVDmiss(x, niter=100, ncomp=2)

# look at the status
res$status

# plot the first four columns of the data matrix
par(mfrow=c(2,2))
for(i in 1:4){
  plot(t, x[,i])
  lines(t, res$xfill[,i])
  lines(t, x.org[,i], col=2)
}
```

SVDsmooth

Smooth Basis Functions for Data Matrix with Missing Values

Description

Function that computes smooth functions for a data matrix with missing values, as described in Fuentes et. al. (2006), or does cross validation to determine a suitable number of basis functions. The function uses SVDmiss to complete the matrix and then computes smooth basis functions by applying smooth.spline to the SVD of the completed data matrix.
Usage

SVDSmooth(X, n.basis = min(2, dim(X)[2]),
   date.ind = NULL, scale = TRUE, niter = 100,
   conv.reldiff = 0.001, df = NULL, spar = NULL,
   fnc = FALSE)

SVDSmoothCV(X, n.basis, ...)

Arguments

X                Data matrix, with missing values marked by NA (use createDataMatrix). Rows
                and/or columns that are completely missing will be dropped (with a message),
                for the rows the smooths will be interpolated using predict.smooth.spline.

n.basis          Number of smooth basis functions to compute, will be passed as ncomp to
                SVDmiss; for SVDSmoothCV a vector with the different number of basis func-
                tions to evaluate (including 0).

date.ind         Vector giving the observation time of each row in X, used as x in
                smooth.spline when computing the smooth basis functions. If missing convertCharToDate
                is used to coerce the rownames(X).

scale            If TRUE, will use scale to scale X before calling SVDmiss.

niter,conv.reldiff Controls convergence, passed to SVDmiss.

df,spar          The desired degrees of freedom/smoothing parameter for the spline,
                see smooth.spline

fnc              If TRUE return a function instead of the trend-matrix, see Value below.

...              Additional parameters passed to SVDSmooth; i.e. date.ind, scale, niter,
                conv.reldiff, df, spar, and/or fnc.

Details

SVDSmoothCV uses leave-one-column-out cross-validation; holding one column out from X, calling
SVDSmooth, and then regressing the held out column on the resulting smooth functions. Cross-
validation statistics computed for each of these regressions include MSE, R-squared, AIC and BIC.
The weighted average (weighted by number of observations in the column) is then reported as CV-
statistics.

Value

Depends on the function:

SVDSmooth       A matrix (if fnc==FALSE) where each column is a smooth basis function based
                on the SVD of the completed data matrix. The left most column contains the
                smooth of the most important SVD. If fnc==TRUE a function that will create
                the data matrix if called as fnc(date.ind), fnc(1:dim(X)[1]), or fnc(convertCharToDate(
                rownames(X) )).

SVDSmoothCV      A list of class SVDcv with components:
SVsmooth

CV.stat, CV.sd data.frames with mean and standard deviation of the CV statistics for each of the number of basis functions evaluated.

MSE.all, R2.all, AIC.all, BIC.all data.frames with the individual MSE, R2, AIC, and BIC values for each column in the data matrix and for each number of basis functions evaluated.

smoothSVD A list with length(n.basis) components. If fnc==FALSE each component contains an array where smoothSVD[[j]][,i] is the result of svdsmooth applied to x[-i] with n.basis[j] smooth functions; if fnc==FALSE each component contains a list of functions as smoothSVD[[j]][[i]].

Author(s)
Paul D. Sampson and Johan Lindström

References

See Also
Other data matrix: createDataMatrix, estimateBetaFields, mesa.data.raw, SVDmiss
Other SVD for missing data: boxplot.SVdcv, calcSmoothTrends, plot.SVdcv, print.SVdcv, summary.SVdcv, SVDmiss, updateStDataTrend, updateTrend, updateTrend.STdata, updateTrend.STmodel
Other SVDcv methods: boxplot.SVdcv, plot.SVdcv, print.SVdcv, summary.SVdcv

Examples
## create a data matrix
t <- seq(0, 4*pi, len=50)
X.org <- cbind(cos(t), sin(2*t)) %% matrix(rnorm(10), 2, 5)

## add some normal errors
X <- X.org + .25*runorm(length(X.org))
## and mark some data as missing
X[runif(length(X))<.25] <- NA

## Ensure that we have complete columns/rows
while( any(rowSums(is.na(X))===dim(X)[2]) || any(colSums(is.na(X))===dim(X)[1]) ){
    X <- X.org + .25*runorm(length(X.org))
    X[runif(length(X))<.25] <- NA
}

## compute two smooth basis functions
res <- SVsmooth(X, n.basis=2, niter=100)

## or compute the function that gives the basis functions
res.fnc <- SVsmooth(X, n.basis=2, niter=100, fnc=TRUE)
## and they are equal
summary(res.fnc()-res)

# plot the two smooth basis functions
par(mfcol=c(3,2), mar=c(4,4,.5,.5))
plot(t, res[,1], ylim=range(res), type="l")
lines(t, res[,2], col=2)
## and some of the data fitted to the smooths
for(i in 1:5){
  plot(t, X[,,i])
  lines(t, predict.lm(lm(X[,,i]-res, data.frame(res))) )
  lines(t, X.org[,,i], col=2)
}

# compute cross-validation for 1 to 4 basis functions
res.cv <- SVDsmoothCV(X, n.basis=0:4, niter=100)
# study cross-validation results
print(res.cv)
summary(res.cv)

# plot cross-validation statistics
plot(res.cv, sd=TRUE)
# boxplot of CV statistics for each column
boxplot(res.cv)
# plot the BIC for each column
plot(res.cv, "BIC", pairs=TRUE)

---

**updateCovf**

*Update Covariance Functions in STmodel Objects*

**Description**

Updates/sets the covariance functions for STmodel objects. Used by `createSTmodel`.

**Usage**

```r
updateCovf(STmodel, cov.beta = STmodel$cov.beta,
            cov.nu = STmodel$cov.nu)
```

**Arguments**

- `STModel` STmodel object with observations, covariates, trends, etc; see `mesa.model`.
- `cov.beta, cov.nu` Covariance specification for the beta- and nu-fields should contain fields `covf, nugget, and random.effect` (for the nu field); see details for description of
updateCovf

these fields. For cov.beta the fields should contain one element for each smooth-temporal trend/beta-field if the fields have only one element, these elements are repeated implying the same covariance for all beta-fields.

Details

The covariance function is specified using lists for cov.beta and cov.nu. The lists should contain the following elements:

- **covf**: The type of covariance function(s), see `namesCovFuns`.
- **nugget**: For the beta-fields: a vector of TRUE/FALSE indicating if each beta-field should contain a nugget.
  - For the nu-field: Either TRUE/FALSE for constant nugget/no nugget; a formula; or length=1 character vector. For the latter two the nugget is allowed to vary as `exp(B*theta)` where:
    - `nugget = as.formula(paste("~", paste(cov.nu$nugget, collapse="+")))`
    - `covars = model.frame(nugget, covars, drop.unused.levels=TRUE)`
    - `B=model.matrix(nugget, covars)`
    - `B=as.matrix(B)`
  - The resulting regression matrix is stored as `STmodel$cov.nu$nugget.matrix` giving nugget for the observed locations. Unobserved locations are assumed to have a zero nugget.

- **random.effect**: Only used for cov.nu, TRUE/FALSE indicating if a random.effect for the mean value should be included, see `makeSigmaNu`.

Value

updated version of STmodel with new covariance specifications.

Author(s)

Johan Lindström

See Also

Other covariance functions: `crossDist`, `evalCovFuns`, `makeSigmaB`, `makeSigmaNu`, `namesCovFuns`, `parsCovFuns`

Other STmodel functions: `createCV`, `createDataMatrix`, `createSTmodel`, `dropObservations`, `estimateBetaFields`, `loglikeST`, `loglikeSTdim`, `loglikeSTnaive`, `predictNaive`, `processLocation`, `processLUR`, `processST`, `updateSTdataTrend`, `updateTrend`, `updateTrend.STdata`, `updateTrend.STmodel`

Examples

```r
# load the data
data(mesa.model)

# covariance specification:
cov.beta <- list(covf="exp", nugget=FALSE)
cov.nu <- list(covf="exp", nugget=TRUE, random.effect=FALSE)

# simple covariance structure
updateCovf(mesa.model, cov.beta, cov.nu)
```
updateTrend.STdata  

**Description**

Updates/sets the temporal trend for STdata or STmodel objects. It also checks that the spatio-temporal covariate exists for all dates in the trend, mainly an issue if extra.dates!=NULL adds additional times at which to do predictions.

**Usage**

```r
## S3 method for class 'STdata'
updateTrend(object, n.basis = 0,
            fnc = NULL, extra.dates = NULL, ...)

## S3 method for class 'STmodel'
updateTrend(object, n.basis = 0,
            fnc = NULL, ...)

updateSTdataTrend(object, n.basis = 0,
                   extra.dates = NULL, fnc = NULL, ...)
```

**Arguments**

- **object**: A STdata or STmodel object, see `mesa.data.raw`.
- **n.basis**: number of basis functions for the temporal trend
- **extra.dates**: Additional dates for which smooth trends should be computed (otherwise only those in object$obs$date are used); only for STdata.
- **fnc**: Function that defines the trend, see Details and Example.
- **...**: Additional parameters passed to `calcSmoothTrends`. 

---

```r
# different behaviour for different beta:s

cov.beta <- list(covf=c("exp","exp2","matern"), nugget=c(TRUE,FALSE,FALSE))
updateCovf(mesa.model, cov.beta, cov.nu)

# Spatially varying nugget

cov.nu <- list(covf="exp", nugget="type", random.effect=FALSE)
print(tmp <- updateCovf(mesa.model, cov.beta, cov.nu))

# lets study the regression matrix for the nugget
str(tmp$cov.nu$nugget.matrix)
head(tmp$cov.nu$nugget.matrix)
```
**Details**

If `n.basis` is given this will use `calcSmoothTrends` to compute smoothed SVDs of data for use as temporal trends. If `fnc` is given, `n.basis` is ignored and `fnc` should be a function that, given a vector of dates, returns an object that can be coerced to a data.frame with *numeric* temporal trends; recall that an intercept is always added.

For a STmodel object the new trend must have no more components than the existing trend; if a function is given colnames of the new trend must match those of the existing trend. In both cases the returned STdata or STmodel object will have both a `$trend` and `$trend.fnc` field.

Function `updateSTdataTrend` is deprecated and will be removed in future versions of the package.

**Value**

Returns a modified version of the input, with an added/altered trend.

**Author(s)**

Johan Lindström

**See Also**

Other STdata functions: `c.STmodel, createDataMatrix, createSTdata, createSTmodel, detrendSTdata, estimateBetaFields, removeSTcovarMean`

Other STmodel functions: `createCV, createDataMatrix, createSTmodel, dropObservations, estimateBetaFields, loglikeST, loglikeSTdim, loglikeSTnaive, predictNaive, processLocation, processLUR, processST, updateCovf`

Other SVD for missing data: `boxplot.SVDcv, calcSmoothTrends, plot.SVDcv, print.SVDcv, summary.SVDcv, SVDmiss, SVDSmooth, SVDSmoothCV`

**Examples**

```r
# load data
data(mesa.model)

# default data and time trend for one location
par(mfrow=c(3,1), mar=c(2.5,2.5,3,1))
plot(mesa.model)

# let's try with no trend
mesa.model.0 <- updateTrend(mesa.model, n.basis=0)
plot(mesa.model.0)

# ... and two basis functions, based on only AQS sites and much less smooth
subset <- mesa.model$locations$ID[mesa.model$locations$type=="AQS"]
mesa.model.2 <- updateTrend(mesa.model, n.basis=2, subset=subset, df=100)
plot(mesa.model.2)

# Compute trends based on only 10 sites (and compute the cross-validated
# trends leaving each of the sites out
smooth.trend <- calcSmoothTrends(mesa.model, n.basis=2, cv=TRUE,
```
# update trends using the function definition
mesa.model <- updateTrend(mesa.model, fnc=smooth.trend$trend.fnc)

# and create objects with each of the trends.
mesa.model.cv <- vector("list", length(smooth.trend$trend.fnc.cv))
for(i in 1:length(mesa.model.cv)){
  suppressMessages(mesa.model.cv[[i]] <- updateTrend(mesa.model, fnc=smooth.trend$trend.fnc.cv[[i]]))
}

# plot
par(mfrow=c(1,1),mar=c(2.5,2.5,3,1))
plot(mesa.model)
for(i in 1:length(mesa.model.cv)){
  plot(mesa.model.cv[[i]], add=TRUE, col=i, pch=NA, lty=c(NA,2))
}
Index

*Topic **datasets**
  est.cv.mesa, 41
est.mesa.model, 43
MCMC.mesa.model, 73
mesa.data.raw, 77
mesa.model, 79
pred.mesa.model, 93
*Topic **package**
  SpatioTemporal-package, 4

abline, 87
backsolve, 68
bdig, 70, 72
block.mult (make.sigma.B), 64
blockMult, 7, 12, 14, 16, 17, 19, 35, 66, 69, 70, 72, 82, 127
boxplot, 9, 92
boxplot.estCVSTmodel, 9, 23, 52, 100, 105, 128
boxplot.SVDCv, 4, 21, 109, 133, 135, 139
boxplot.SVDCv(plot.SVDCv), 92
c.STmodel, 5, 6, 10, 29, 32–34, 39, 47, 49, 52, 66, 76, 91, 95, 97, 105, 108, 114, 115, 118, 121, 131, 139
calc.F.times.X (make.sigma.B), 64
calc.FX, 8, 12, 14, 16, 17, 19, 55, 66, 69, 70, 72
calc.FXtf2, 8, 12, 14, 16, 17, 19, 55, 67, 69, 70, 72
calc.isX, 8, 12, 14, 17, 19, 59, 60, 63, 69, 70, 72
calc.isX(calc.mu.B), 15
calc.mu.B, 8, 12, 14, 15, 17, 19, 59, 60, 63, 69, 70, 72
calc.smooth.trends (make.sigma.B), 64
calc.tf.mat.F (make.sigma.B), 64
calc.tf.times.mat (make.sigma.B), 64
calc.tFX, 8, 12, 14, 16, 17, 19, 55, 66, 69, 70, 72
calc.tF XF, 8, 12, 14, 16, 17, 18, 55, 66, 69, 70, 72
calc.X. is.X, 8, 12, 14, 17, 19, 59, 60, 63, 69, 70, 72
calc.X. is.X (calc.mu.B), 15
calcSmoothTrends, 4, 5, 20, 66, 93, 109, 133, 135, 138, 139
chol, 68
chol2inv, 68
coeff, 22, 23
coeff.estCVSTmodel, 6, 9, 22, 52, 100, 105, 128
coeff, 22, 23
coeff.estimateSTmodel, 6, 23, 37, 47, 100
combineMesaData (make.sigma.B), 64
compute.ltaCV (make.sigma.B), 64
computeST, 24, 27, 41, 51, 52, 66, 98, 129
cond. expectation (make.sigma.B), 64
create.LUR.basis (make.sigma.B), 64
create.ST.basis (make.sigma.B), 64
cventCharToDate, 25, 36, 125, 134
create.data.matrix (make.sigma.B), 64
create.data.model (make.sigma.B), 64
createCV, 24, 26, 29, 33, 40, 41, 49–52, 58, 59, 98, 110, 112, 137, 139
createDataMatrix, 11, 21, 27, 28, 31–33, 39, 41, 49, 58, 59, 66, 77, 98, 110, 112, 115, 133–135, 137, 139
createLUR, 30, 66, 79
createST, 66, 79
createST (createLUR), 30
createSTdata, 5, 11, 29, 30, 33, 39, 49, 67, 77, 80, 91, 104, 107, 114, 115, 118, 130, 139
createSTmodel, 10–12, 14, 17, 19, 27, 29–32, 32, 39, 41, 42, 44, 47, 49, 52, 55, 58, 59, 66, 73, 76, 80, 91, 93, 95, 97, 98, 105, 108–112, 114, 115, 118, 121,
MCMC.mesa.model, 42, 44, 73, 77, 80, 93
MCMC.STmodel, 11, 34, 37, 47, 52, 67, 73, 74,
84, 86, 91, 97, 101, 105, 106, 108,
114, 118, 121, 128, 131
mesa.data.raw, 20, 29, 30, 33, 38, 42–44, 49,
73, 77, 80, 93, 109, 111, 115, 133,
135, 138
mesa.model, 12, 14, 17, 19, 33, 41–44, 55, 72,
73, 77, 93, 136
model.matrix, 111

namesCovFuns, 35, 53, 54, 67, 70–72, 80, 83,
137
norm2, 8, 35, 69, 82, 127

optim, 45, 46, 50
pairs, 92
parsCovFuns, 35, 53, 54, 67, 70, 72, 81, 83,
137
plot, 84–86, 88–90, 92, 118
plot.acf, 90
plot.density, 84
plot.density.mcmcSTmodel, 5, 37, 76, 84,
86, 101, 106, 128
plot.mcmcSTmodel, 5, 37, 76, 84, 85, 101,
106, 128
plot.predictCVSTmodel, 4, 6, 52, 67, 86, 97,
102, 103, 107, 114, 118, 130
plot.predictSTmodel, 4, 6, 52, 67, 97, 102,
103, 107, 114, 118, 130
plot.predictSTmodel
(plot.predictCVSTmodel), 86
plot.STdata, 5, 11, 32, 34, 47, 52, 67, 76, 89,
97, 104, 105, 107, 108, 114, 118,
121, 130, 131
plot.STmodel, 5, 11, 32, 34, 47, 52, 67, 76,
97, 104, 105, 107, 108, 114, 118,
121, 130, 131
plot.STmodel(plot.STdata), 89
plot.SVDCv, 21, 67, 93, 108, 133, 135, 139
plotCV(make.sigma.B), 64
plotMesaData(make.sigma.B), 64
plotMonitoringLoc(make.sigma.B), 64
plotPrediction(make.sigma.B), 64
pred.cv.mesa(est.cv.mesa), 41
pred.mesa.model, 42, 44, 73, 77, 80, 93
predict, 38
predict.smooth.spline, 134
predict.STmodel, 4–6, 11, 34, 46, 47, 50–52,
66, 67, 75, 76, 87, 88, 91, 93, 94,
103, 105, 108, 113, 114, 117, 118,
121, 129, 131
predictCV, 9, 11, 23, 24, 27, 34, 41, 47, 76,
88, 91, 97, 98, 100, 102, 105, 107,
108, 114, 118, 121, 128, 130, 131
predictCV(estimateCV.STmodel), 50
predictCV.STmodel, 4–6, 9, 11, 23, 24, 27,
34, 41, 42, 47, 76, 88, 91, 97, 98,
100, 102, 105, 107, 113, 114,
117, 118, 121, 128–131
predictNaive, 5, 6, 24, 27, 29, 33, 41, 49, 52,
58, 59, 97, 110, 112, 129, 137, 139
print, 99–108
print.estCVSTmodel, 9, 23, 52, 99, 105, 128
print.estimateSTmodel, 24, 47, 100
print.mcmcSTmodel, 37, 76, 84, 86, 101, 106,
128
print.predictCVSTmodel, 4, 52, 88, 102, 107,
114, 118, 130
print.predictSTmodel, 4, 88, 97, 103
print.STdata, 32, 67, 91, 103, 107, 114, 118,
130
print.STmodel, 11, 34, 47, 52, 67, 76, 91, 97,
104, 108, 114, 118, 121, 131
print.summary.estCVSTmodel, 9, 23, 52,
100, 105, 128
print.summary.mcmcSTmodel, 37, 76, 84, 86,
101, 106, 128
print.summary.predictCVSTmodel, 52, 88, 102,
106, 114, 118, 130
print.summary.STdata, 32, 91, 104, 107,
114, 118, 130
print.summary.STmodel, 11, 34, 47, 52, 76,
91, 97, 105, 108, 114, 121, 131
print.SVDCv, 21, 67, 93, 108, 133, 135, 139
print.table, 105–107
printMesaDataNbrObs(make.sigma.B), 64
processLocation, 27, 29, 33, 41, 49, 58, 59,
66, 79, 98, 109, 112, 137, 139
processLUR, 27, 29, 30, 33, 41, 49, 58, 59, 66,
79, 98, 110, 111, 137, 139
processST, 27, 29, 30, 33, 41, 49, 58, 59, 66,
79, 98, 110, 137, 139
processST(processLUR), 111

qqline, 113
qqnorm, 113
qqnorm.predCVSTmodel, 4, 5, 11, 32, 34, 47, 52, 66, 76, 88, 91, 97, 102, 104, 105, 107, 108, 113, 118, 121, 130, 131
qqnorm.Stdata, 5, 11, 32, 34, 47, 52, 66, 76, 88, 91, 97, 102, 104, 105, 107, 108, 118, 121, 130, 131
qqnorm.Stdata(qqnorm.predCVSTmodel), 113
qqnorm.Stmodel, 5, 11, 32, 34, 47, 52, 66, 76, 88, 91, 97, 102, 104, 105, 107, 108, 118, 121, 130, 131
qqnorm.Stmodel(qqnorm.predCVSTmodel), 113
remove. ST.mean (make.sigma.B), 64
removeSTCovarMean, 11, 29, 31–33, 39, 49, 67, 115, 139
rlm, 38
run.MCMC (make.sigma.B), 64
scale, 134
scatter.smooth, 117
scatterPlot, 116
scatterPlot.predCVSTmodel, 4, 5, 11, 32, 34, 47, 52, 66, 76, 88, 91, 97, 102, 104, 105, 107, 108, 114, 117, 121, 130, 131
scatterPlot.Stdata, 5, 11, 32, 34, 47, 52, 66, 76, 88, 91, 97, 102, 104, 105, 107, 108, 114, 121, 130, 131
scatterPlot.Stdata(scatterPlot.predCVSTmodel), 117
scatterPlot.Stmodel, 5, 11, 32, 34, 47, 52, 66, 76, 88, 91, 97, 102, 104, 105, 107, 108, 114, 121, 130, 131
scatterPlot.Stmodel(scatterPlot.predCVSTmodel), 117
set.seed, 120
setupSTdataset (make.sigma.B), 64
simulate.STmodel, 5, 11, 34, 47, 52, 67, 76, 91, 97, 105, 108, 114, 118, 120, 131
simulateMesaData (make.sigma.B), 64
smooth.spline, 133, 134
solveTriBlock, 8, 12, 14, 16, 17, 19, 35, 70, 72, 82, 127
solveTriBlock (makeCholBlock), 68
sparseMatrix, 55

SpatioTemporal
(SpatioTemporal-package), 4
SpatioTemporal-package, 4
sqrt, 31
stCheckClass, 121, 123–126
stCheckCovars, 31, 122, 124–126
stCheckFields, 122, 123, 125, 126
stCheckObs, 122–124, 126
stCheckSTcvars, 31, 122–125, 125
sumLog, 8, 35, 69, 82
sumLog (sumLogDiag), 126
sumLogDiag, 8, 35, 69, 82, 126
summary, 108, 127–131
summary.estCVSTmodel, 9, 23, 52, 100, 105, 127
summary.mcmcSTmodel, 37, 76, 84, 86, 101, 106, 128
summary.predCVSTmodel, 4, 5, 52, 67, 88, 97, 102, 107, 114, 118, 129
summary.Stdata, 32, 67, 91, 104, 107, 114, 118, 130
summary.STmodel, 11, 34, 47, 52, 67, 76, 91, 97, 105, 108, 114, 118, 121, 131
summary.SVDCv, 21, 93, 133, 135, 139
summary.SVDCv (print.SVDCv), 108
summaryStatsCV (make.sigma.B), 64
SVD.miss (make.sigma.B), 64
SVD.smooth (make.sigma.B), 64
SVDmiss, 21, 29, 49, 67, 77, 93, 109, 132, 133–135, 139
SVDsmoth, 4, 5, 20, 21, 29, 49, 67, 77, 93, 109, 133, 139
SVDsmoothCV, 4, 20, 21, 29, 49, 67, 77, 93, 109, 133, 139
SVDsmoothCV (SVDsmooth), 133
tstat (make.sigma.B), 64
updateCovf, 6, 27, 29, 33, 35, 41, 49, 54, 58, 59, 66, 70, 72, 79, 81, 83, 93, 98, 110, 112, 136, 139
updateSTdataTrend, 4, 5, 11, 21, 27, 29, 32, 33, 39, 41, 49, 58, 59, 93, 98, 109, 110, 112, 115, 133, 135, 137
updateSTdataTrend (updateTrend.STdata), 138
updateTrend, 11, 21, 27, 29, 32, 33, 39, 41, 49, 58, 59, 66, 93, 98, 109, 110, 112, 115, 133, 135, 137
updateTrend (updateTrend.STdata), 138
updateTrend.STmodel, 4, 5, 11, 21, 27, 29, 32, 33, 39, 41, 49, 58, 59, 93, 98, 109, 110, 112, 115, 133, 135, 137