Package ‘saeRobust’

March 27, 2018

Title  Robust Small Area Estimation
Version  0.2.0
Author  Sebastian Warnholz [aut, cre]
Maintainer  Sebastian Warnholz <Sebastian.Warnholz@fu-berlin.de>
Description  Methods to fit robust alternatives to commonly used models used in Small Area Estimation. The methods here used are based on best linear unbiased predictions and linear mixed models. At this time available models include area level models incorporating spatial and temporal correlation in the random effects.

BugReports  https://github.com/wahani/saeRobust/issues
Depends  R (>= 3.3.0), methods, aoos
Imports  assertthat, ggplot2, Matrix, magrittr, MASS, modules, memoise, pbapply, Rcpp, spdep
License  MIT + file LICENSE
LazyData  true
Suggests  knitr, rmarkdown, sae, saeSim, testthat
VignetteBuilder  knitr
Encoding  UTF-8
ByteCompile  true
RoxygenNote  6.0.1
LinkingTo  Rcpp, RcppArmadillo
NeedsCompilation  yes
Repository  CRAN
Date/Publication  2018-03-27 20:43:34 UTC
**R topics documented:**

- bootstrap .............................................. 2
- corSAR1-class ........................................ 3
- fitrfh .................................................. 4
- fixedPoint ............................................ 6
- makeXY ................................................. 7
- matU .................................................... 8
- mse ...................................................... 10
- plot.rfh ............................................... 11
- psiOne ............................................... 12
- rfh ..................................................... 13
- score .................................................. 15
- testMatX ................................................ 16
- update.rfh-method .................................... 17
- variance ............................................... 18

**Index**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap</td>
<td>2</td>
</tr>
<tr>
<td>corSAR1-class</td>
<td>3</td>
</tr>
<tr>
<td>fitrfh</td>
<td>4</td>
</tr>
<tr>
<td>fixedPoint</td>
<td>6</td>
</tr>
<tr>
<td>makeXY</td>
<td>7</td>
</tr>
<tr>
<td>matU</td>
<td>8</td>
</tr>
<tr>
<td>mse</td>
<td>10</td>
</tr>
<tr>
<td>plot.rfh</td>
<td>11</td>
</tr>
<tr>
<td>psiOne</td>
<td>12</td>
</tr>
<tr>
<td>rfh</td>
<td>13</td>
</tr>
<tr>
<td>score</td>
<td>15</td>
</tr>
<tr>
<td>testMatX</td>
<td>16</td>
</tr>
<tr>
<td>update.rfh-method</td>
<td>17</td>
</tr>
<tr>
<td>variance</td>
<td>18</td>
</tr>
</tbody>
</table>

**bootstrap**  
*Fit model on Bootstrap sample*

**Description**

These functions help to repeatedly fit a rfh model on bootstrap samples. Use `bootstrap` as a user interface. `boot` can be used to extend the framework but is not meant to be used interactively. If you are interested in the parameteric bootstrap for a `rfh` model you can use the implementation in `mse`.

**Usage**

```r
bootstrap(object, matV = variance(object), B = NULL, ...)

boot(object, matV, B, ...)
```

```r
## S4 method for signature 'ANY,ANY,integerORnumeric'
boot(object, matV, B, filter = NULL,
     postProcessing = identity, ...)
```

```r
## S4 method for signature 'rfh,rfhVariance,'NULL''
boot(object, matV, B, ...)
```

**Arguments**

object  
a fitted object

matV  
the variance of a fitted object used to draw samples. In most cases this is `object`. Alternatively it may be useful to use a non-robust model.

B  
the number of repetitions
corSAR1-class

... arguments passed down to methods

filter a vector indicating which elements in the fitted object to keep in each repetition.

postProcessing a function to process the results. Is applied before the filter.

Examples

data(milk, package = "sae")
milk$samplingVar <- milk$SD^2
modelFit <- rfh(yi ~ as.factor(MajorArea), milk, "samplingVar")
bootstrapCoefs <- bootstrap(modelFit, B = 2, filter = "coefficients")
do.call(rbind, unlist(bootstrapCoefs, FALSE))

---

Correlation Structure

corSAR1-class

Description

Various correlation structures. They can be used inside the rfh function to supply an alternative variance structure to be fitted. For examples see the documentation of rfh.

Usage

corSAR1(W, ...)
corAR1(nTime, ...)
corSAR1AR1(...)

Arguments

W the row-standardised proximity matrix

... arguments passed to new. In the case of corSAR1AR1 arguments W and nTime are expected.
nTime (numeric) number of time periods

Details

corSAR1 can be used to model a simultaneous autoregressive process of order one: spatial correlation.
corAR1 can be used to model a autoregressive process of order one: temporal correlation.
corSAR1AR1 can be used to model spatial and temporal correlation

Slots

W the row-standardised proximity matrix

nTime (numeric) number of time periods
fitrfh

Fitting Procedures

Description
Several fitting procedures. The arguments can be passed to these functions using the interface in rfh. The functions here listed are the low level implementations and are not intended for interactive use.

Usage
fitrfh(y, x, samplingVar, ...)
fitrsfh(y, x, samplingVar, W, x0Var = c(0.01, 1), ...)
fitrtfh(y, x, samplingVar, nTime, x0Var = c(0.01, 1, 1), ...)
fitrstfh(y, x, samplingVar, W, nTime, x0Var = c(0.01, 0.01, 1, 1), ...)
fitGenericModel(y, x, matVFun, fixedPointParam, k = 1.345, K = getK(k), x0Co = NULL, x0Var = 1, x0Re = NULL, tol = 1e-06, maxIter = 100, maxIterParam = 10, maxIterRe = 100, convCrit = convCritRelative(tol), ...)

## S4 method for signature 'numeric, matrix OR Matrix, numeric, 'NULL''
rfh(formula, data,
    samplingVar, correlation = NULL, ...)

## S4 method for signature 'numeric, matrix OR Matrix, numeric, corSAR1'
rfh(formula, data,
    samplingVar, correlation = NULL, ...)

## S4 method for signature 'numeric, matrix OR Matrix, numeric, corAR1'
rfh(formula, data,
    samplingVar, correlation = NULL, ...)

## S4 method for signature 'numeric, matrix OR Matrix, numeric, corSAR1AR1'
rfh(formula, data,
    samplingVar, correlation = NULL, ...)

Arguments
y (numeric) response vector
x ([m|M]atrix) the design matrix
samplingVar (numeric) vector with sampling variances
... arguments passed to fitGenericModel
fitrfh

W (matrix) proximity matrix
x0Var (numeric) starting values for variance parameters
nTime (integer) number of time periods
matVFun (function) a function with one argument - the variance parameters - constructing something like variance

fixedPointParam (function) a function with one argument. The vector of model parameters. Returns a list of results of the next iteration in the overall algorithm.

k (numeric) tuning constant
K (numeric) scaling constant
x0Coeff (numeric) starting values for regression coefficients
x0Re (numeric) starting values for random effects
tol (numeric) numerical tolerance to be used during optimisation
maxIter (integer) the maximum number of iterations for model parameters.
maxIterParam (integer) the maximum number of iterations for each parameter in each overall iteration
maxIterRe (integer) the maximum number of iterations for fitting the random effects
convCrit (function) a function defining the stopping rule
formula (formula) a formula specifying the fixed effects part of the model.
data (data.frame) a data set.
correlation an optional correlation structure, e.g. corSAR1, for the random effects part of the model. Default is no correlation, i.e. a random intercept.

Details

fitrfh implements the robust Fay-Herriot model; fitrsfh the spatial, fitrtfh the temporal, and fitrstfh the spatio-temporal extension to this model type. See rfh how to fit such models. fitGenericModel is used by all these implementations and can be used for possible extensions of the framework.

Examples

data(milk, package = "sae")
x <- matrix(1, nrow = NROW(milk))
y <- milk$yi
samplingVar <- milk$SD^2
fitrfh(y, x, samplingVar)
Description
A fixed-point function supplied by the user is iteratively evaluated. addAverageDamp can be used to add average damping to the function - this may have a positive effect on the speed of convergence.

Usage
fixedPoint(fun, x0, convCrit)
addAverageDamp(fun)
addConstraintMin(fun, value)
addConstraintMax(fun, value)
convCritAbsolute(tolerance = 1e-06)
convCritRelative(tolerance = 1e-06)
addMaxIter(fun, maxIter)
addCounter(fun)
addHistory(fun)
addStorage(fun)
newtonRaphson(funList, ...)
newtonRaphsonFunction(funList)

Arguments
fun the function to be evaluated in the algorithm
x0 starting value
convCrit a function returning a logical scalar. Is called with two arguments; the first is the value from iteration n; the second is the value from iteration n-1
value (numeric)
tolerance a numeric value > 0
maxIter maximum number of iterations
funList (list) the functions to be evaluated in the algorithm. First element is typically the score function, second is the derivative of the score.
... arguments passed to fixedPoint
Details

addAverageDamp adds average damping to an arbitrary fixed point function.
addConstraintMin takes care that values are not below a minimum value.
addConstraintMax takes care that values are not larger than maximum value.
convCritAbsolute absolute difference as convergence criterion.
convCritRelative relative (to previous iteration) absolute difference as convergence criterion. If value is smaller than 1, absolute difference is used.
addMaxIter can be used to modify convergence criterion functions.
addCounter can be used to count the number of calls of a function.
addHistory can be used to save a history of results of a function. The history is stored as a matrix, so this works best if the return value of fun is numeric.
addStorage will add a storage to a function. The storage is a list in which each result is stored. The function will coerce the return value into a numeric.
newtonRaphson finds zeroes of a function. The user can supply the function and its first derivative. Note that the Newton Raphson Algorithm is a special case of a fixed point algorithm thus it is implemented using fixedPoint and is only a convenience.

Examples

## Not run:
vignette("fixedPoint", "saeRobust")

## End(Not run)

makeXY

Description

Extract response vector and design matrix from data with given formula.

Usage

makeXY(.formula, .data)

Arguments

.formula        (formula)
.data           (data.frame) data from which design matrix and response to extract from

Examples

set.seed(1)
makeXY(y ~ I(x^2), data.frame(x = rnorm(10), y = rnorm(10)))
Matrix constructor functions

Description
These functions construct different parts of matrix components. They are used internally. If you are interested in the weights of a model fitted using rfh please try to use weights.fitrfh on that object.

Usage
matU(.V)
matTrace(x)
matB(y, X, beta, re, matV, psi)
matBConst(y, X, beta, matV, psi)
matA(y, X, beta, matV, psi)
matAConst(y, X, matV, psi)
matW(y, X, beta, re, matV, psi)
matWbc(y, reblup, W, samplingVar, c = 1)
matTZ(.nDomains, .nTime)
matTZ1(.nDomains = 10, .nTime = 10)

Arguments
.V (Matrix) variance matrix
x (Matrix) a matrix
y (numeric) response
X (Matrix) design matrix
beta (numeric) vector of regression coefficients
re (numeric) vector of random effects
matV (list of functions) see matVFH for an example
psi (function) the influence function
reblup (numeric) vector with robust best linear unbiased predictions
W (Matrix) the weighting matrix
samplingVar (numeric) the vector of sampling variances
c (numeric) scalar
Details

matU computes \( U \). \( U \) is the matrix containing only the diagonal elements of \( V \). This function returns a list of functions which can be called to compute specific transformations of \( U \).

matTrace computes the trace of a matrix.

matB computes the matrix \( B \) which is used to compute the weights in the pseudo linearised representation of the REBLUP.

matBConst returns a function with one argument, \( u \), to compute the matrix \( B \). This function is used internally to compute \( B \) in the fixed point algorithm.

matA computes the matrix \( A \) which is used to compute the weights in the pseudo linearized representation of the REBLUP.

matAConst returns a function with one argument, \( \beta \), to compute the matrix \( A \). This function is used internally to compute \( A \) in the fixed point algorithm for \( \beta \).

matW returns a matrix containing the weights as they are defined for the pseudo linear form, such that \( \text{matW} \%\%\% y \) is the REBLUP.

matWbc returns a matrix containing the weights as they are defined for the pseudo linear form, such that \( \text{matWbc} \%\%\% y \) is the bias-corrected REBLUP. \( c \) is a multiplier for the standard deviation.

matTZ constructs the \( Z \) matrix in a linear mixed model with autocorrelated random effects.

matTZ1 constructs the \( Z_1 \) matrix in a linear mixed model with autocorrelated random effects.

References


Examples

data("grapes", package = "sae")
data("grapesprox", package = "sae")

fitRFH <- rfh(
grapehect ~ area + workdays - 1,
data = grapes,
samplingVar = "var"
)

matV <- variance(fitRFH)

# matU:
matU(matV$V())$U()
matU(matV$V())$sqrt()
matU(matV$V())$sqrtInv()

# matB (and matA + matW accordingly):
mse

Compute the Mean Squared Error of an Estimator

Description

A generic function to compute the mean squared error of the predicted values under the estimated model. See also rfh for examples.

Usage

mse(object, ...)

## S3 method for class 'fitrfh'
mse(object, type = "pseudo", predType = "reblupbc", B = 100, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>(see methods) an object containing the estimation result, e.g. rfh</td>
</tr>
<tr>
<td>...</td>
<td>arguments passed to methods</td>
</tr>
<tr>
<td>type</td>
<td>(character) the type of the MSE. Available are 'pseudo' and 'boot'</td>
</tr>
<tr>
<td>predType</td>
<td>(character) the type of prediction: c(&quot;reblup&quot;,&quot;reblupbc&quot;)</td>
</tr>
<tr>
<td>B</td>
<td>(numeric) number of bootstrap repetitions</td>
</tr>
</tbody>
</table>
Details

Type pseudo is an approximation of the MSE based on a pseudo linearisation approach by Chambers, et. al. (2011). The specifics can be found in Warnholz (2016). Type boot implements a parametric bootstrap for these methods.

References


Examples

data("grapes", package = "sae")
data("grapesprox", package = "sae")

fitRFH <- rfh(
    grapehect ~ area + workdays - 1,
    data = grapes,
    samplingVar = "var"
)

mseRFH <- mse(fitRFH)
plot(mseRFH)

plot.rfh

Plots

Description

Various implementations of diagnostic plots. They are linked together using the plot generic function.

Usage

## S3 method for class 'rfh'
plot(x, y, ...)

## S3 method for class 'prediction.fitrfh'
plot(x, y, alpha = 0.05, ...)

## S3 method for class 'mse.fitrfh'
plot(x, y = "pseudo", xlim = NULL, ylim = NULL, ...)

dqPlot(sample)

blandAltmanPlot(x, y, alpha = 0.05)
Arguments

- **x**: an object
- **y**: for mse estimates a filter for the predictors; otherwise ignored
- **...**: ignored
- **alpha**: (numeric) between 0 and 1 - used in computation of confidence interval
- **xlim, ylim**: arguments are passed to `coord_cartesian` and `coord_flip`.
- **sample**: (numeric) a vector

Details

- `qqPlot` a QQ-Plot using ggplot2
- `blandAltmanPlot` a Bland-Altman plot. Solid line is the mean. Dashed lines are the upper and lower bound of the limits-of-agreements: $z$-quantile * sd(x - y) – not the standard error. The alpha level can be set using `alpha`. This plot is otherwise known as Tukey’s mean-difference plot.

Examples

```r
qqPlot(rnorm(10))
blandAltmanPlot(rnorm(10), rnorm(10))
```

Description

`psiOne` is a Huber influence function. `getK` function to compute capital K – used internally.

Usage

```r
psiOne(u, k = 1.345, deriv = FALSE)
getK(k)
```

Arguments

- **u**: standardized residuals
- **k**: tuning constant
- **deriv**: if TRUE returns the derivative

Examples

```r
set.seed(1)
u <- rnorm(10)
psiOne(u, k = 1.345, deriv = FALSE)
```
**rfh**

---

**Robust Fay Herriot Model**

---

**Description**

User interface to fit robust Fay-Herriot type models. These models are here framed as linear mixed models. The parameter estimation is robust against outliers. Available models are the standard FH model, a spatial extension, a temporal extension and a spatio-temporal extension.

**Usage**

rfh(formula, data, samplingVar, correlation = NULL, ...)

```r
## S4 method for signature 'formula, data.frame, character, ANY'
rfh(formula, data, samplingVar, correlation, ...)

## S3 method for class 'rfh'
predict(object, type = "reblup", c = 1, ...)
```

**Arguments**

- **formula** *(formula)* a formula specifying the fixed effects part of the model.
- **data** *(data.frame)* a data set.
- **samplingVar** *(character)* the name of the variable in data containing the sampling variances.
- **correlation** an optional correlation structure, e.g. corSAR1, for the random effects part of the model. Default is no correlation, i.e. a random intercept.
- **...** arguments passed fitGenericModel
- **object** *(rfh)* an object of class rfh
- **type** *(character)* one or more in c("linear", "reblup", "reblupbc")
- **c** *(numeric)* scalar; a multiplier constant used in the bias correction. Default is to make no correction for realisations of direct estimator within c = 1 times the standard deviation of direct estimator.

**Details**

To trigger the spatial and temporal extensions you can supply an argument `correlation`. When corSAR1 is used the model of Petrucci and Salvati (2006); for corAR1 the model of Rao and Yu (1994) is used; and for corSAR1AR1 the model of Marhuenda et al. (2013).

The methods introducing the robust framework underpinning this implementation can be found in Warnholz (2016). They are based on the results by Sinha and Rao (2009) and Richardson and Welsh (1995).
Value

A list with the following elements:

- call (language) the call generating the value
- formula (formula) the formula passed as argument
- samplingVar (numeric) the vector of sampling variances
- coefficients (numeric) the vector of regression coefficients
- variance (numeric) the vector of fitted variance parameter(s)
- iterations (list) reporting each step in the optimisation
- tol (numeric) the tolerance level used
- maxIter (numeric) maximum overall allowed iterations
- maxIterParam (numeric) maximum allowed iterations for model parameters in each overall iteration
- maxIterRe (numeric) maximum allowed iterations for fitting the random effects
- k (numeric) tuning constant in influence function
- K (numeric) additional tuning constant; often derived from k to scale down the residual variance
- y (numeric) the response vector
- x (Matrix) the design matrix
- re (numeric) the fitted random effects. Can be c(re1, re2)
- reblup (numeric) the robust best linear unbiased prediction under the fitted model
- residuals (numeric) the realised sampling errors
- fitted (numeric) the fitted values using only the fixed effects part

References


Examples

```r
# Non-temporal models:
data("grapes", package = "sae")
data("grapesprox", package = "sae")

fitRFH <- rfh(
  grapehect ~ area + workdays - 1,
  data = grapes,
  samplingVar = "var"
)

fitRFH
summary(fitRFH)

plot(fitRFH)
plot(predict(fitRFH))
plot(mse(fitRFH))

# Not run:
# And the same including a spatial structure:
fitRSFH <- rfh(
  grapehect ~ area + workdays - 1,
  data = grapes,
  samplingVar = "var",
  corSAR1(as.matrix(grapesprox))
)

# Use the same methods, e.g. plot, for all these implementations:
data("spacetime", package = "sae")
data("spacetimeprox", package = "sae")
nTime <- length(unique(spacetime$Time))

fitRTFH <- rfh(
  Y ~ X1 + X2,
  spacetime,
  "Var",
  corAR1(nTime = nTime)
)

fitRSTFH <- rfh(
  Y ~ X1 + X2,
  spacetime,
  "Var",
  corSAR1AR1(W = as.matrix(spacetimeprox), nTime = nTime)
)

## End(Not run)
```

---

**score**  
Compute values of robust score functions
Description

Can be used to compute the values of the robust estimation equations at their 'solution'.

Usage

\[ \text{score(object, filter, ...)} \]

Arguments

- **object**: a fitted object
- **filter**: (character) a selection of values to be computed
- **...**: arguments passed to methods

Examples

```r
data("grapes", package = "sae")

fitRFH <- rfh(
  grapehect ~ area + workdays - 1,
  data = grapes,
  samplingVar = "var"
)

score(fitRFH)
```

---

**testMatX**

*Construction of test data*

Description

Construction of test data

Usage

```r
testMatX(...)  

testResponse0(x, beta = rep(1, ncol(x)))  
testResponse(y0, k = 1:4, .sd = sd(y0))  
testRook(n)
```
**update.rfh-method**

**Arguments**

- ...  matrices
- x  a matrix
- beta  a vector with parameters
- y  a response vector (numeric)
- k  values in 1 to 4 (integer)
- .sd  the standard deviation used for random numbers
- n  dimension

**References**


**Examples**

```r
## Examples from Weihs et al. (2014) p. 108
library("Matrix")
testMatX(Matrix(998), Matrix(998))
Z <- Matrix(c(998, 0, 0, 0), 2, 2)
testMatX(Z, Z)
testResponse0(testMatX(Matrix(1)))
library("magrittr")
Matrix(1) %>% testMatX %>% testResponse0 %>% testResponse
```

**Description**

This is a method which can be used to update a rfh result object and refit it. The fitted parameter values from the current object are used as starting values, then update.default is called.

**Usage**

```r
## S4 method for signature 'rfh'
update(object, formula, ..., where = parent.frame(2))

## S4 method for signature 'fitrfh'
update(object, ...)
```

**Arguments**

- object  (rfh) an object fitted by rfh
- formula  see update.formula
- ...  arguments passed to update.default
- where  (environment) should not be specified by the user
**Description**

A generic function to construct the different variance components of an object. You may want to use this in conjunction with bootstrap.

**Usage**

```r
variance(.object, ...)
```

## S3 method for class 'fitrfh'
```r
variance(.object, ...)
```

## S3 method for class 'fitrsfh'
```r
variance(.object, ...)
```

## S3 method for class 'fitrfh'
```r
variance(.object, ...)
```

## S3 method for class 'fitrsthf'
```r
variance(.object, ...)
```

## S3 method for class 'fitrfh'
```r
weights(object, c = 1, ...)
```

**Arguments**

- `.object`, object
  - an object
- `...` arguments passed to method
- `c` numeric scalar

**Examples**

```r
data("grapes", package = "sae")
data("grapesprox", package = "sae")

fitRFH <- rfh(
  grapehect ~ area + workdays - 1,
  data = grapes,
  samplingVar = "var"
)

# The variance component of a mixed linear model:
matV <- variance(fitRFH)
# The full variance matrix:
```

```r
```
matV$V()

# The sampling error component
matV$Ve()

# the random effects component
matV$Vu()
Index

addAverageDamp (fixedPoint), 6
addConstraintMax (fixedPoint), 6
addConstraintMin (fixedPoint), 6
addCounter (fixedPoint), 6
addHistory (fixedPoint), 6
addMaxIter (fixedPoint), 6
addStorage (fixedPoint), 6

blandAltmanPlot (plot.rfh), 11
boot (bootstrap), 2
boot, ANY, ANY, integer OR numeric-method
  (bootstrap), 2
boot, rfh, rfhVariance, NULL-method
  (bootstrap), 2
bootstrap, 2, 18

convCritAbsolute (fixedPoint), 6
convCritRelative (fixedPoint), 6
coord_cartesian, 12
coord_flip, 12
corAR1 (corSAR1-class), 3
corAR1-class (corSAR1-class), 3
corSAR1, 5, 13
corSAR1 (corSAR1-class), 3
corSAR1-class, 3
corSAR1AR1 (corSAR1-class), 3
corSAR1AR1-class (corSAR1-class), 3

fitGenericModel, 13
fitGenericModel (fitrfh), 4
fitrfh, 4
fitrsfh (fitrfh), 4
fitrstfh (fitrfh), 4
fitsrfh (fitrfh), 4

fixedPoint, 6, 6, 7

getK (psiOne), 12

makeXY, 7
matA (matU), 8
matAConst (matU), 8
matB (matU), 8
matBConst (matU), 8
matTrace (matU), 8
matTZ (matU), 8
matTZ! (matU), 8
matU, 8
matW (matU), 8
matWbc (matU), 8
mse, 2, 10

newtonRaphson (fixedPoint), 6
newtonRaphsonFunction (fixedPoint), 6

plot, 11
plot.mse.fitrfh (plot.rfh), 11
plot.prediction.fitrfh (plot.rfh), 11
plot.rfh, 11
predict.fitrfh (rfh), 13
psiOne, 12

qqPlot (plot.rfh), 11

rfh, 2–5, 8, 10, 13, 17
rfh.formula.data.frame.character, ANY-method
  (rfh), 13
rfh, numeric, matrixORMatrix, numeric, corAR1-method
  (fitrfh), 4
rfh, numeric, matrixORMatrix, numeric, corSAR1-method
  (fitrfh), 4
rfh, numeric, matrixORMatrix, numeric, corSAR1AR1-method
  (fitrfh), 4
rfh, numeric, matrixORMatrix, numeric, NULL-method
  (fitrfh), 4

score, 15

testMatX, 16
testResponse (testMatX), 16
testResponse0 (testMatX), 16
testRook (testMatX), 16
update, fitrfh-method
(update, rfh-method), 17
update, rfh-method, 17
update.default, 17
update.formula, 17

variance, 5, 18

weights.fitrfh, 8
weights.fitrfh(variance), 18