Package 'stsm.class'

February 20, 2015

Version 1.3

Date 2014-07-21

Title Class and Methods for Structural Time Series Models

Description This package defines an S4 class for structural time series models and provides some basic methods to work with it.

Author Javier López-de-Lacalle <javlacalle@yahoo.es>

Maintainer Javier López-de-Lacalle <javlacalle@yahoo.es>

Depends R (>= 3.0.0), methods

Suggests numDeriv

NeedsCompilation no

Encoding UTF-8

License GPL-2

Repository CRAN

Date/Publication 2014-07-26 15:53:50

R topics documented:

stsm.class-package *Class and Methods for Structural Time Series Models*

Description

This package defines an S4 class for structural time series models and provides some basic methods to work with it.

Details

The class defined in this package is used by the package **stsm** as the base framework to implement statistical algorithms related to structural time series models.

References

Christophe Genolini. *A (Not So) Short Introduction to S4. Object Oriented Programming in* R. V0.5.1. August 20, 2008.

Author(s)

Javier López-de-Lacalle <javlacalle@yahoo.es>

stsm-char2numeric-methods

State Space Representation of Objects of Class stsm

Description

This method returns the state space representation of time series models defined in the class stsm.

Usage

```
## S4 method for signature 'stsm'
char2numeric(x, P0cov = FALSE, rescale = FALSE)
```

Arguments

x	an object of class stsm.
P0cov	logical. If TRUE the values of the elements outside the diagonal in the initial covariance matrix of the state vector are set equal to the values in the diagonal. Otherwise values outside the diagonal are set equal to zero.
rescale	logical. If TRUE, relative variance parameters are rescaled into absolute variances. Otherwise, relative variances are used. Ignored if x@cpar is null.

Details

This method uses the information from the slots `pars`, `nopars` and `cpar` in order to build the numeric representation of the matrices.

For details about the argument `rescale` see the details section in `stsm-get-methods` and the examples below.

A previous version of this method employed the information in the slot `ss`. This slot contains the matrices of the state space form of the model but instead of inserting the parameter values, character strings indicating the location of the parameters are placed in the corresponding cells. This method performed the mapping from the character to the numeric matrices by means of a internal function called `ss.fill`. Currently the slot `ss` and the matrices are directly built depending on the model that was selected among those available in `stsm.model`. The current approach is straightforward and faster. The previous approach may still be interesting to allow the user to define additional models just by translating the notation of the model into character matrices. The usefulness of enhancing this approach will be assessed in future versions of the package.

Value

A list of class `stsmSS` containing the following numeric matrices and vectors:

`Z`	observation matrix.
`T`	transition matrix.
`H`	observation variance.
`R`	selection matrix.
`V`	state vector variance-covariance matrix.
`Q`	RVR'.
`a0`	initial state vector.
`P0`	initial state vector uncertainty matrix.

The list contains also two vectors, `Vid` and `Qid`, with the indices of those cells where the variance parameters are located respectively in the matrices V and Q. The first element in a matrix is indexed as 0.

State space representation

The general univariate linear Gaussian state space model is defined as follows:

$$y[t] = Za[t] + e[t], e[t] \sim \tilde{} N(0, H)$$

$$a[t+1] = Ta[t] + Rw[t], w[t] \sim \tilde{} N(0, V)$$

for $t = 1, \ldots, n$ and $a[1] \sim \tilde{} N(a0, P0)$. Z is a matrix of dimension $1 \times m$; H is 1×1; T is $m \times m$; R is $m \times r$; V is $r \times r$; $a0$ is $m \times 1$ and $P0$ is $m \times m$, where m is the dimension of the state vector a and r is the number of variance parameters in the state vector.

See Also

`stsm-class`, `stsm.model`.

Examples

```
# sample model with arbitrary parameter values
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 6), nopars = c("var3" = 12))
ss1 <- char2numeric(m)
c(get.pars(m), get.nopars(m), get.cpar(m))
# character notation of the covariance matrix of the state vector
m@ss$Q
# information from the slots 'pars', 'nopars' and 'cpar'
# is used to retrieve the numeric representation of 'm@ss$Q'
ss1$Q

# same as above but with P0cov=TRUE
# the only change is in the initial covariance matrix of
# the state vector 'P0'
ss2 <- char2numeric(m, P0cov = TRUE)
ss1$P0
ss2$P0

# if a non-standard parameterization is used,
# the values in the slot 'pars' are transformed accordingly
# and the actual variance parameters are returned;
# notice that the transformation of parameters applies only
# to the parameters defined in the slot 'pars'
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 6), nopars = c("var3" = 12),
  transPars = "square")
c(get.pars(m), get.nopars(m), get.cpar(m))[1:3]
ss <- char2numeric(m)
ss$H
ss$Q

# model defined in terms of relative variances,
# the variances in 'pars' are relative to the scaling parameter 'cpar',
# in this example 'cpar' is chosen to be the variance 'var1'
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var2" = 3, "var3" = 6), cpar = c("var1" = 2),
  transPars = NULL)
# the state space representation can be done with
# relative variances (no rescaling)
ss <- char2numeric(m, rescale = FALSE)
ss$H
ss$Q
# or with absolute variances (rescaling)
ss <- char2numeric(m, rescale = TRUE)
ss$H
ss$Q

# in a model where the parameters are the relative variances
# and with non-null 'transPars', the transformation is applied to
# the relative variances, not to the absolute variances, i.e.,
# the relative variances are first transformed and afterwards they are
```

```
# rescaled back to absolute variances if requested
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var2" = 3, "var3" = 6), cpar = c("var1" = 2),
  transPars = "square")
# the state space representation can be done with
# relative variances (no rescaling)
ss <- char2numeric(m, rescale = FALSE)
ss$H
ss$Q
# or with absolute variances (rescaling)
ss <- char2numeric(m, rescale = TRUE)
ss$H
ss$Q
```

`stsm-class` *Class* `stsm` *for Structural Time Series Models*

Description

This class defines a structural time series model.

Slots

`call` Object of class `call`. Call to `stsm.model`.

`model` Object of class `character`. Name or label for the selected model (see `stsm.model` for available models).

`y` Object of class `ts`. Original time series.

`diffy` Object of class `ts`. Differenced series `y`. The differencing operator that renders stationarity in the `model` is applied to the series `y`.

`xreg` An optinal matrix or numeric vector of external regressors.

`fdiff` Object of class `function`. Function with arguments `x`: a `ts` object, `s`: periodicity of the data. This function applies the differencing operator that renders stationarity in the `model` to a `ts` object passed to it.

`ss` Object of class `list`. Matrices of the state space form of the structural model.

`pars` Object of class `numeric`. Named vector with the parameters of the model.

`nopars` Optional object of class `numeric`. An optional named vector with the remaining parameters of the model not included in `pars`. This slot is not affected by the transformation of parameters `transPars`. These parameters are considered fixed in the optimization procedures implemented in package **stsm**.

`cpar` Optional object of class `numeric`. Named vector of length one containing the parameter that is concentrated out of the likelihood function (if any).

`lower` Object of class `numeric`. Named vector with the lower bounds for `pars`.

`upper` Object of class `numeric`. Named vector with the upper bounds for `pars`.

`transPars` Character string referring to the parameterization of the model, see `transPars`.

`ssd` Optional object of class `numeric`. Sample spectral density (periodogram) of the differenced series `diffy`.

`sgfc` Optional object of class `matrix`. Constant elements in the spectral generating function of the `model` (for pure variance models).

Methods

`char2numeric` Return a list containing the matrices of the state space representation of the model. The matrices are the same as those in the slot `ss` but the characters are replaced by the corresponding numeric values defined in `pars`, `nopars` and `cpar`.

`checkbounds` Check whether the values of `pars` lie within the `lower` and `upper` bounds.

`get.pars` Return the slot `pars`, the parameters of the model. If the model is parameterized in terms of a set of auxiliar parameters such as those considered in `transPars`, then the transformed parameters are returned. Thus, when the slot `transPars` is not `NULL` `x@pars` will not be equal to `get.pars(x)`.

`get.cpar` Return the slot `cpar`.

`get.nopars` Return the slot `nopars`.

`set.cpar` Set or modify the value of the slot `cpar`

`set.nopars` Set or modify the value of the slot `nopars`.

`set.pars` Set or modify the value of the slot `pars`.

`set.sgfc` Compute and set the value of the slot `sgfc`.

`set.xreg` Set or modify the value of the slot `xreg`.

`setValidity` Check the validity of the arguments passed to the function.

`show` Show a brief summary of the object.

`transPars` Transform the parameters of the model according to the parameterization defined in the slot `transPars`.

See Also

`stsm.model`.

---

stsm-get-methods *Getter Methods for Class* `stsm`

Description

Get access to the information stored in the slots `cpar`, `nopars` and `pars` in objects of class `stsm`.

Usage

```
## S4 method for signature 'stsm'
get.cpar(x, rescale = FALSE)
## S4 method for signature 'stsm'
get.nopars(x, rescale = FALSE)
## S4 method for signature 'stsm'
get.pars(x, rescale = FALSE)
```

Arguments

x	an object of class stsm.
rescale	logical. If TRUE, relative variance parameters are rescaled into absolute variances. Ignored if x@cpar is null.

Details

Transformation of the parameters of the model. The method transPars allows parameterizing the model in terms of an auxiliar vector of parameters. The output of get.pars is returned in terms of the actual parameters of the model, i.e., the variances and the autoregressive coefficients if they are part of the model. With the standard parameterization, x@transPars = NULL, get.pars(x) returns the output stored in x@pars. When the model is parameterized in terms of an auxiliar set of parameters θ, get.pars return the variance parameters instead of the values of θ that are stored in x@pars. For example, with x@transPars = "square" (where the variances are θ^2), ger.pars returns θ^2 while x@pars contains the vector θ.

Absolute and relative variances.

The model can be defined in terms of relative variances. In this case, the variance that acts as a scaling parameter is stored in the slot cpar. Otherwise, cpar is null and ignored. Typically, the scaling parameter will be chosen to be the variance parameter that is concentrated out of the likelihood function.

If rescale = TRUE, the relative variance parameters are rescaled into absolute variance parameters (i.e., they are multiplied by x@cpar) and then returned by these methods. If rescale = FALSE, relative variance parameters are returned, that is, the variances divided by the scaling parameter cpar. Since the scaling parameter is one of the variances, the relative variance stored in cpar is 1 (the parameter divided by itself).

Transformation of parameters in a model defined in terms of relative variances. When a model is defined so that the parameters are the relative variances (cpar is not null) and a parameterization transPars is also specified, then the transformation of parameters is applied to the relative variances, not to the absolute variances. The relative variances are first transformed and afterwards they are rescaled back to absolute variances if requested by setting rescale = TRUE. The transformation transPars is applied to the parameters defined in pars; cpar is assumed to be chosen following other rationale; usually, it is the value that maximizes the likelihood since one of the variance parameters can be concentrated out of the likelihood function.

Note. When cpar is not null, it is more convenient to store in the slots pars and nopars the values of the relative variances, while the slot cpar stores the value of the scaling parameter rather than the relative variance (which will be 1). If the relative values were stored, then the scaling parameter would need to be recomputed each time the value is requested by get.cpar. Assuming that cpar is the parameter that is concentrated out of the likelihood function, the expression that maximizes the likelihood should be evaluated whenever the value is requested to be printed or to do any other operation. To avoid this, the scaling value is directly stored. This approach makes also sense with the way the method set.cpar works.

Note for users. For those users that are not familiar with the design and internal structure of the **stsm.class** package, it is safer to use the get and set methods rather than retrieving or modifying the contents of the slots through the @ and @<- operators.

Value

`get.cpar`	named numeric of length one.
`get.nopars`	named numeric vector.
`get.pars`	named numeric vector.

See Also

`stsm-class`.

Examples

```
# sample models with arbitrary parameter values

# model in standard parameterization
# internal parameter values are the same as the model parameter
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
m@pars
get.pars(m)

# model parameterized, the variances are the square
# of an auxiliar vector of parameters
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30), transPars = "square")
# auxiliar vector of parameters
m@pars
# parameters of the model, variances
get.pars(m)

# model rescaled, variances are relative to 'var1'
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var2" = 15, "var3" = 30), cpar = c("var1" = 2))
# internal values
m@pars
m@cpar
# relative variances
get.pars(m)
get.cpar(m)
# absolute variances
get.pars(m, rescale = TRUE)
get.cpar(m, rescale = TRUE)

# model defined in terms of relative variances
# and with the parameterization \code{transPars="square};
# the transformation is applied to the relative variances,
# the relative variances are first transformed and afterwards
# they are rescaled back to absolute variances if requested
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var2" = 3, "var3" = 6), cpar = c("var1" = 2),
  transPars = "square")
c(get.cpar(m, rescale = FALSE), get.pars(m, rescale = FALSE))
```

```
c(get.cpar(m, rescale = TRUE), get.pars(m, rescale = TRUE))

# when 'cpar' is defined, 'nopars' is also interpreted as a relative variance
# and therefore it is rescaled if absolute variances are requested
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var2" = 3), cpar = c("var1" = 2), nopars = c("var3" = 6),
  transPars = NULL)
v <- c(get.cpar(m, rescale = FALSE), get.pars(m, rescale = FALSE), get.nopars(m, rescale = FALSE))
v[c("var1", "var2", "var3")]
v <- c(get.cpar(m, rescale = TRUE), get.pars(m, rescale = TRUE), get.nopars(m, rescale = TRUE))
v[c("var1", "var2", "var3")]

# 'nopars' is rescaled as shown in the previous example
# but it is not affected by the parameterization chosen for 'pars'
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var2" = 3), cpar = c("var1" = 2), nopars = c("var3" = 6),
  transPars = "square")
v <- c(get.cpar(m, rescale = FALSE), get.pars(m, rescale = FALSE), get.nopars(m, rescale = FALSE))
v[c("var1", "var2", "var3")]
v <- c(get.cpar(m, rescale = TRUE), get.pars(m, rescale = TRUE), get.nopars(m, rescale = TRUE))
v[c("var1", "var2", "var3")]
```

`stsm-set-methods` *Setter Methods for Class* `stsm`

Description

Setter or modifier methods for objects of class `stsm`.

Usage

```
## S4 method for signature 'stsm'
set.cpar(x, value, check = TRUE, inplace = FALSE)
## S4 method for signature 'stsm'
set.nopars(x, v, check = TRUE, inplace = FALSE)
## S4 method for signature 'stsm'
set.pars(x, v, check = TRUE, inplace = FALSE)
## S4 method for signature 'stsm'
set.sgfc(x, inplace = FALSE)
## S4 method for signature 'stsm'
set.xreg(x, xreg, coefs = NULL)
```

Arguments

`x`	an object of class `stsm`.
`value`	a numeric value.
`v`	a numeric vector.

check	logical. If TRUE, the resulting model is checked for consistency with the definition of the stsm object.
inplace	logical. If TRUE, the input object x is modified in place instead of returning the whole object.
xreg	a matrix or numeric vector of external regressors. The number of rows or length of the vector must be equal to the length of x@y. If column names are specified they are used to name the parameters in the slot pars.
coefs	an optional vector containing the value of the coefficients related to the regressors xreg. If the elements of the vector do not contain names they are assumed to be defined in the same order as the columns in the matrix xreg.

Details

Models parameterized with non-null transPars. If the model is parameterized according to a non-null value of the slot transPars, the argument v must contain the values of the auxiliar set of parameters θ rather than the actual parameters (variances and autoregressive coefficients). For example, with x@transPars = "square" the variances are θ^2. Although this design may seem to disagree with the getter methods stsm-get-methods, the relevant input for the setter methods is actually the auxiliar values θ. Be aware that if transPars is not null the parameters are transformed by get.pars according to the selected parameterization. Therefore, v must be referred to the non-transformed parameters.

The previous comment does not apply to the argument value since cpar is not affected by transPars.

Setter methods are safer. For those users that are not familiar with the design and internal structure of the **stsm.class** package, it is safer to use setter methods rather than modifying the contents of the slots through the @<- operator. See the examples below.

Modifying the input object in-place. Instead of returning the whole object and create a new one or overwrite the original, it is possible to modify just the desired slot in the original object that is passed as input. In the former case the stsm object returned by the method must be assigned to another object using the usual operator <-. In the latter approach, the stsm object that is passed as argument is modified in-place. See the example below. The solution to modify an object in-place is taken from this post. This option is not a customary solution in R, however, it seems suitable in this context. The real benefit of this approach would depend on how R deals with objects that are returned from functions. If assigning the output to a new object involves making copies of all the slots, then modifying the object in-place would most likely be more efficient since the desired slot is directly modified avoiding copying the whole object.

After R version 3.1 this issue may become less critical. One of the new features reported in the release of R 3.1 states: *Avoid duplicating the right hand side values in complex assignments when possible. This reduces copying of replacement values in expressions such as Z$a <- a0.* A related discussion for S4 classes can be found in this post.

Constant terms in the spectral generating function. In pure variance models, some elements of the spectral generating function (s.g.f.) do not depend on the parameters and can be stored as constants. The method set.sgfc computes and stores those elements as a matrix in the slot sgfc. This is useful for example when working with maximum likelihood methods in the frequency domain. In that context, the spectral generating function has to be updated several times for different parameter values. Having the information about the constant terms in the slot sgfc saves several computations whenever the s.g.f. is requested. For details about the s.g.f see stsm.sgf.

Further setter methods. Future versions may include additional setter methods, for example to change the slot `model` or to modify the time series `x@y`. The latter would also require updating the slots `diffy` and `ssd` if requested. Additional methods are not available in the current version because defining a new object by means of `stsm.model` will often be better than modifying one of those slots that do not have a setter method.

Value

If the slot is modified in place, `inplace=TRUE`, nothing is returned, the corresponding slot of the object `m` passed as argument is modified in place.

If `inplace=FALSE`, a new `stsm` object is returned. It contains the same information as the input object `m` except for the slot that has been modified.

See Also

`stsm-class`, `stsm.sgf`.

Examples

```
# sample models with arbitrary parameter values
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
get.pars(m)

# correct modification
m1 <- set.pars(m, c(1, 2, 3))
get.pars(m1)
m1 <- set.pars(m, c(var1 = 11))
get.pars(m1)

# correct but error prone
m1@pars[] <- c(4, 22, 33)
get.pars(m1)
m1@pars <- c(var1 = 1, var2 = 2, var3 = 3)
get.pars(m1)

# inconsistent assignment (error returned)
# 'var4' is not a parameter of model 'llm+seas'
try(m1 <- set.pars(m, c(var4 = 4)))
# inconsistent assignment (no error returned)
# the error is not noticed at this point
# unless 'validObject' is called
m1 <- m
m1@pars["var4"] <- 4
get.pars(m1)
try(validObject(m1))

# modify only one element
m1 <- set.pars(m, v=c(var1=22))
get.pars(m1)
# wrong assignment, the whole vector in the slot is overwritten
# no error returned at the time of doing the assignment
```

```
m1@pars <- c(var1 = 1)
get.pars(m1)
try(validObject(m1))

# consistent assignment but maybe not really intended
# all the elements are set equal to 12
m1 <- m
m1@pars[] <- 12
get.pars(m1)
# warning returned by 'set.pars'
m2 <- set.pars(m, 12)
get.pars(m2)

# wrong value unnoticed (negative variance)
m1 <- m
m1@pars[] <- c(-11, 22, 33)
get.pars(m1)
# negative sign detected by 'set.pars'
try(m1 <- set.pars(m, c(-11, 22, 33)))

# inplace = FALSE
# the whole object 'm' is assigned to a new object,
# which will probably involve making a copy of all the slots
m <- set.pars(m, c(1,2,3), inplace = FALSE)
get.pars(m)

# inplace = TRUE
# the output is not assigned to a new object
# the only operation is the modification of the slot 'pars'
# no apparent additional internal operations such as copying unmodified slots
get.pars(m)
set.pars(m, c(11,22,33), inplace = TRUE)
get.pars(m)

# set a matrix of regressors
xreg <- cbind(xreg1 = seq_len(84), xreg2 = c(rep(0, 40), rep(1, 44)))
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson, xreg = xreg)
m
# set a new matrix of regressors to an existing
xreg3 <- seq(length(m@y))
m2 <- set.xreg(m, xreg3)
m2
# remove the external regressors
m3 <- set.xreg(m, NULL)
m3
m3@xreg
# initialize the coefficients to some values
m <- stsm.class::stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("xreg1" = 10), xreg = xreg)
m
m <- stsm.class::stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("xreg2" = 20, "xreg1" = 10), xreg = xreg)
m
```

stsm-show-methods *Display an Object of Class* stsm

Description

This method displays summary information about an object of class stsm.

Usage

```
## S4 method for signature 'stsm'
show(object)
```

Arguments

object	an object of class stsm.

Details

A succinct summary of the object (name of the model and parameter values) is printed.

Value

Invisible NULL.

See Also

stsm-class.

Examples

```
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
show(m)
# or just
m
```

stsm-transPars-methods *Parameterization of Models Defined in the Class* stsm

Description

This method provides different transformations of the parameters of a structural time series model.

Usage

```
## S4 method for signature 'generic'
transPars(x,
    type = c("square", "StructTS", "exp", "exp10sq"),
    gradient = FALSE, hessian = FALSE,
    rp, sclrho = 1.7, sclomega = 1.7, ftrans = NULL, ...)
## S4 method for signature 'numeric'
transPars(x,
    type = eval(formals(stsm.class::transPars)$type),
    gradient = FALSE, hessian = FALSE,
    rp, sclrho = 1.7, sclomega = 1.7, ftrans = NULL, ...)
## S4 method for signature 'stsm'
transPars(x, type = NULL,
    gradient = FALSE, hessian = FALSE,
    rp, sclrho = 1.7, sclomega = 1.7, ftrans = NULL, ...)
```

Arguments

x	an object of class stsm.
type	a character string indicating the type of transformation. Ignored if x is of class stsm. See details.
gradient	logical. If TRUE, first order derivatives of the transformation function with respect to the parameters in the slot pars are returned.
hessian	logical. If TRUE, second order derivatives of the transformation function with respect to the parameters in the slot pars is returned.
rp	numeric value. Regularization parameter used with type = StrucTS. By default it is the variance of the data x@y divided by 100.
sclrho	numeric value. Currently ignored.
sclomega	numeric value. Currently ignored.
ftrans	a function defining an alternative transformation of the parameters. Ignored if x is of class stsm. See example below.
...	additional arguments to be passed to ftrans.

Details

Rather than using the standard parameterization of the model (in terms of variances and autoregressive coefficients if they are part of the model), it can be parameterized in terms of an auxiliar set of parameters θ. This may be useful for example when the parameters of the model are selected by means of a numerical optimization algorithm. Choosing a suitable parameterization ensures that the solution returned by the algorithm meets some constraints such as positive variances or autoregressive coefficients within the region of stationarity.

The method transPars can be applied both on a named vector of parameters, e.g. x@pars or on a model of class stsm.

When the slot transPars is not null, the model is parameterized in terms of θ. The following transformation of parameters can be considered:

- "square": the variance parameters are the square of θ.
- "StructTS": transformation used in the function StructTS of the **stats** package.
- "exp": the variance parameters are the exponential of θ.
- "exp10sq": the variance parameters are $(exp(-\theta)/10)^2$.

In the model trend+ar2 defined in stsm.model, the autoregressive coefficients, ϕ, are transformed to lie in the region of stationarity: given $z1 = \phi_1/(1 + |\phi_1|)$, $z2 = \phi_2/(1 + |\phi_2|)$, the transformed coefficients are $\phi_1^* = z1 + z2$ and $\phi_2 = -z1 \cdot z2$.

Other transformations can be defined through the argument ftrans, which can also be defined in the slot transPars of a stsm object. ftrans must be a function returning a list containing an element called pars and two other optional elements called gradient and hessian. The parameters to be transformed are identified by their names. The variances follow the naming convention of the regular expression "^var\d{1,2}$", e.g. var1, var2,... The variances of the initial state vector may also be transformed if they are included in the slot pars; their names follow a similar naming convention, P01, P02,... An example of ftrans is given below.

Note: If a transformation is defined by means of ftrans the user may need to update the slots lower and upper if some bounds are still applied to the auxiliar parameters. For example, transPars="StructTS" does not always yield positive variances and hence lower bounds equal to 0 are needed. By default lower and upper bounds are not considered if ftrans is used.

The output of get.pars is given in terms of the actual parameters of the model. For example, if the model is parameterized so that θ^2 are the variances of the model and θ are the auxiliar parameters then, the slot pars contains the values of θ and ger.pars returns θ^2.

The transformation transPars is applied to the parameters included in the slot pars. The transformation does not affect nopars and cpar. The former slot is considered fixed while the latter will in practice be set equal to a particular value, for example the value that maximizes the concentrated likelihood function, for which a specific expression can be obtained.

Value

A list containing a named numeric vector with the values of the transformed parameters. If requested, the gradient and Hessian of the transformation function with respect to the parameters are returned.

See Also

stsm-class, get.pars.

Examples

```
# sample models with arbitrary parameter values

# model in standard parameterization
# ower bounds imposed on the variance parameters
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30), transPars = NULL)
get.pars(m)
m@lower
```

```
# square transformation
# negative values are allowed in 'pars' since
# the square will yield positive variances
# in fact no lower bounds need to be imposed on the auxiliar parameters
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = -2, "var2" = -5, "var3" = 10), transPars = "square")
validObject(m)
m@lower
m@pars
get.pars(m)

# 'ftrans', alternative transformation of parameters;
# the following parameterization is sometimes found:
# variance = exp(-theta) / 10
# the function 'ftrans' following the rules given in the details
# above can be defined as follows:

ftrans <- function(x, gradient = FALSE, hessian = FALSE)
{
  tpars <- x
  p <- length(x)
  nmspars <- names(x)
  idvar <- grep("^var|P0\\d{1,2}$", nmspars, value = FALSE)

  if (gradient) {
    d1 <- rep(NA, p)
    names(d1) <- nmspars
  } else d1 <- NULL
  if (hessian) {
    d2 <- matrix(0, p, p)
    rownames(d2) <- colnames(d2) <- nmspars
  } else d2 <- NULL

  if (length(idvar) > 0) {
    tpars[idvar] <- exp(-x[idvar]) / 10
  } else warning("No changes done by 'transPars'.")

  if (gradient)
  {
    if (length(idvar) > 0)
      d1[idvar] <- -tpars[idvar]
  }
  if (hessian) {
    diag(d2)[idvar] <- tpars[idvar]
  }

  list(pars = tpars, gradient = d1, hessian = d2)
}

# now 'ftrans' can be passed to 'transPars' and be applied
# on a named vector of parameters or on a 'stsm' object
transPars(c("var1" = 2, "var2" = 15, "var3" = 30),
  ftrans = ftrans, gradient = TRUE, hessian = TRUE)
```

```
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30), transPars = ftrans)
get.pars(m)
```

---

stsm-validObject-methods

Check the Validity of an Object of Class stsm

Description

Methods to check the validity of an object of class stsm.

Usage

```
## S4 method for signature 'stsm'
check.bounds(x)
## S4 method for signature 'stsm'
validObject(object)
```

Arguments

x	an object of class stsm.
object	an object of class stsm.

Details

check.bounds checks that the values in the slot pars lie within the lower and upper bounds. These bounds are stored in the slots lower and upper. Default values or specific values can be given when creating the object by means of stsm.model.

check.bounds is called by validObject. In some settings it may be required to check only that the parameters are within the required bounds.

validObject checks additional requirements: e.g. all the parameters taking part in the selected model are either in the slots pars, nopars or cpar;

it is also checked that the parameters are no duplicated in those slots.

This method is called by stsm-set-methods defined for the slots pars, nopars or cpar. That's why it is safer to use the setter methods instead of a direct modification through the operator @<-.

Value

If the input object is valid according to the class definition, the logical TRUE is returned. Otherwise, an error message is returned.

See Also

stsm-class and examples in stsm-set-methods.

Examples

```
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
validObject(m)

# force a wrong value (negative variance)
m@pars[1] <- -1
try(validObject(m))
try(check.bounds(m))

# duplicates not allowed
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
# try to define 'var1', already in 'pars', in the slot 'nopars'
try(m <- set.nopars(m, c(var1=22)))
# force a duplicate
m@nopars <- c(m@nopars, var1 = 22)
try(validObject(m))
```

`stsm.model` *Wrapper for Constructor of Objects of Class* `stsm`

Description

Interface to define an object of class `stsm`. This is a wrapper function to constructor `new`.

Usage

```
stsm.model(model = c("local-level", "local-trend", "BSM",
  "llm+seas", "trend+ar2"),
  y, pars = NULL, nopars = NULL, cpar = NULL, xreg = NULL,
  lower = NULL, upper = NULL, transPars = NULL,
  ssd = FALSE, sgfc = FALSE)
```

Arguments

`model`	a character selecting the structural time series model.
`y`	a univariate time series, `ts`.
`pars`	initial values for the parameters of the model. It must be a named vector.
`nopars`	optional named numeric containing the remaining parameters of the model not included in `pars` and `cpar`.
`cpar`	optional named numeric of length one. See details.
`xreg`	optional matrix or numeric vector of external regressors.
`lower`	optional named vector setting lower bounds to some parameters of the model. The names must follow the same same labelling as `pars`.

upper	optional named vector setting upper bounds to some parameters of the model. The names must follow the same same labelling as pars.
transPars	optional character choosing one of the parameterizations defined in transPars or a function defining an alternative parameterization.
ssd	logical. If TRUE, the sample spectral density (periodogram) of the stationary transformation of the data is computed and stored in the slot ssd. Otherwise, it is ignored.
sgfc	logical. If TRUE, constants terms of the spectral generating function related to the chosen model are computed and stored in the slot sgfc. Otherwise, it is ignored.

Details

Slot pars *and* nopars. In some situations it is convenient to split the vector of parameters in two vectors, the slot pars and the slot nopars. For example, when the parameters are to be estimated by an optimization algorithm, only the parameters in pars are allowed to change while the parameters in nopars are considered fixed.

Scaling parameter cpar. The model can be defined in terms of relative variances. In this case, the variance that acts as a scaling parameter is stored in the slot cpar. Otherwise, cpar is null and ignored. Typically, the scaling parameter will be chosen to be the variance parameter that is concentrated out of the likelihood function.

Naming convention of parameters. The parameters defined in the slots pars, nopars and cpar must be labelled according to the following naming convention. The variance parameters abide by the regular expression "^var\d{1,2}$", e.g. var1, var2,... The variances of the initial state vector, $P0$, follow a similar naming convention, P01, P02,... The elements of the initial state vector, $a0$, are similarly denoted as a01, a02,...

Default values. Default values are assigned to the slots pars, nopars and cpar if they are no defined in their corresponding arguments passed to stsm.model. By default, the variance parameters are defined in the slot pars with value 1. The initial state vector is assigned by default to nopars, it takes on zero values except for the first element that takes the value of the first observation in the data. The variance of the initial state vector is assigned by default to nopars as well. By default it takes on the value 10000 times the variance of the data.

If the argument pars is not NULL, no other parameters are stored in the slot pars. If the argument nopars is not NULL, the parameters in that argument are added to the other default parameters. This is more convenient in practice. See the examples below.

Alternative parameterizations. See transPars for available parameterizations of the model. The definition of a function to be defined in the slot transPars is also explained there.

Stationary transformation of the data. The sample spectral density is computed for the differenced time series y. The differencing filter is chosen so that the data are rendered stationary according to the selected model. The stationary form of each model is given in stsm.sgf.

Value

An object of class stsm.

Available models

The **local level model** consists of a random walk plus a Gaussian disturbance term.

The measurement equation is:

$$y[t] = m[t] + e[t], e[t] \sim N(0, \sigma_1^2)$$

The state equation is:

$$m[t+1] = m[t] + v[t], v[t] \sim N(0, \sigma_2^2)$$

The **local trend model** consists of a trend where the slope evolves as a random walk.

The measurement equation is:

$$y[t] = m[t] + e[t], e[t] \sim N(0, \sigma_1^2)$$

The state equations are:

$$m[t+1] = m[t] + n[t] + v[t], v[t] \sim N(0, \sigma_2^2)$$

$$n[t+1] = n[t] + w[t], w[t] \sim N(0, \sigma_3^2)$$

Setting $var3 = 0$ yields the local level model. The constraint $var2 = 0$ involves a smooth trend.

The **basic structural model** consists of a local trend model plus a seasonal component.

The measurement equation is:

$$y[t] = m[t] + s[t] + e[t], e[t] \sim N(0, \sigma_1^2)$$

The state equations are the same as the local trend model plus a seasonal component:

$$s[t+1] = -s[t] - \ldots - s[t - freq + 2] + w[t], w[t] \sim N(0, \sigma_4^2)$$

The restriction $\sigma_4^2 = 0$ yields a deterministic seasonal pattern.

According to the labelling convention used in the package, the variance parameters σ_1^2, σ_2^2, σ_3^2 and σ_4^2 are respectively denoted `"var1"`, `"var2"`, `"var3"` and `"var4"`.

See Also

stsm-class.

Examples

```
# sample model with arbitrary parameter values
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 6), nopars = c("var3" = 12))
m

# parameter values
v <- c("var1" = 2, "var2" = 6, "var3" = 3, "var4" = 12)

# define the parameter 'cpar'
```

```
# the remaining are defined by default to 'pars' and 'nopars'
m <- stsm.model(model = "BSM", y = JohnsonJohnson,
  pars = NULL, nopars = NULL, cpar = v[1])
m@pars
m@nopars
m@cpar

# define the slot 'pars', only 'v[1]' is stored in 'pars'
# the remaining variances are moved to 'nopars' along
# with the initial state vector and its variances
m <- stsm.model(model = "BSM", y = JohnsonJohnson,
  pars = v[1])
m@pars
m@nopars
m@cpar

# define some of the parameters to be stored in the slot 'npars'
# 'only 'v[1:2]' is added to the remaining elements in 'nopars' by default
# the variances not defined in 'nopars' are assigned to 'pars' with
# default value 1
m <- stsm.model(model = "BSM", y = JohnsonJohnson,
  nopars = v[1:2])
m@pars
m@nopars
m@cpar

# define the slot 'pars' and set a particular value to
# some variances stored in 'nopars', 'v[2:3]'
# 'var4' takes the default value 1 and is stored in 'nopars'
# since the definition 'pars = v[1]' excludes it form 'pars'
m <- stsm.model(model = "BSM", y = JohnsonJohnson,
  pars = v[1], nopars = v[2:3])
m@pars
m@nopars
m@cpar

# define the slots 'pars' and 'cpar'
# the remaining parameters are stored in 'nopars' with the
# values by default
m <- stsm.model(model = "BSM", y = JohnsonJohnson,
  pars = v[2:4], nopars = NULL, cpar = v[1])
m@pars
m@nopars
m@cpar
```

`stsm.sgf` *Spectral Generating Function of Common Structural Time Series Models*

Description

Evaluate the spectral generating function of of common structural models: local level model, local trend model and basic structural model.

Usage

```
stsm.sgf(x, gradient = FALSE, hessian = FALSE, deriv.transPars = FALSE)
```

Arguments

x	object of class stsm.
gradient	logical. If TRUE, the gradient is returned.
hessian	logical. If TRUE, hessian the gradient is returned.
deriv.transPars	logical. If TRUE, the gradient and the Hessian are scaled by the gradient of the function that transforms the parameters. Ignored if x@transPars is null.

Details

The stationary form of the **local level model** is (Δ is the differencing operator):

$$\Delta y[t] = v[t] + \Delta e[t]$$

and its *spectral generating function* at each frequency $\lambda[j] = 2\pi j/T$ for $j = 0, ..., T-1$ is:

$$g(\lambda[j]) = \sigma_2^2 + 2(1 - cos\lambda[j])\sigma_1^2$$

The stationary form of the **local trend model** for a time series of frequency S is:

$$\Delta^2 y[t] = \Delta v[t] + w[t-1] + \Delta^2 e[t]$$

and its *spectral generating function* is:

$$g(\lambda[j]) = 2(1 - cos\lambda[j])\sigma_2^2 + \sigma_3^2 + 4(1 - cos\lambda[j])\sigma_1^2$$

The stationary form of the **basic structural model** for a time series of frequency p is:

$$\Delta\Delta^p y[t] = \Delta^p v[t] + S(L)w[t-1] + \Delta^2 s[t] + \Delta\Delta^p e[t]$$

and its *spectral generating function* is:

$$g(\lambda[j]) = g_v(\lambda[j])\sigma_2^2 + g_w(\lambda[j])\sigma_3^2 + g_s(\lambda[j])\sigma_4^2 + g_e(\lambda[j])\sigma_1^2$$

with

$$g_v(\lambda[j]) = 2(1 - cos(\lambda[j]p))$$
$$g_w(\lambda[j]) = (1 - cos(\lambda[j]p))/(1 - cos(\lambda[j]))$$
$$g_s(\lambda[j]) = 4(1 - cos(\lambda[j]))^2$$
$$g_e(\lambda[j]) = 4(1 - cos(\lambda[j]))(1 - cos(\lambda[j]p))$$

Value

A list containing the following results:

sgf	spectral generating function of the BSM model at each frequency $\lambda[j]$ for $j = 0, \ldots, T-1$.
gradient	first order derivatives of the spectral generating function with respect of the parameters of the model.
hessian	second order derivatives of the spsectral generating function with respect of the parameters of the model.
constants	the terms $g_v(\lambda[j])$, $g_w(\lambda[j])$, $g_s(\lambda[j])$ and $g_e(\lambda[j])$ that do not depend on the variance parameters.

References

Harvey, A. C. (1989). *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press.

See Also

set.sgfc, stsm-class, stsm.model.

Examples

```
# spectral generating function of the local level plus seasonal model
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15), nopars = c("var3" = 30))
res <- stsm.sgf(m)
res$sgf
plot(res$sgf)
res$constants
# the element 'constants' contains the constant variables
# related to each component regardless of whether the
# variances related to them are in the slot 'pars' or 'nopars'
names(get.pars(m))
colnames(res$constants)

# compare analytical and numerical derivatives
# identical values
m <- stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
res <- stsm.sgf(m, gradient = TRUE)

fcn <- function(x, model = m) {
  m <- set.pars(model, x)
  res <- stsm.sgf(m)
  sum(res$sgf)
}

a1 <- numDeriv::grad(func = fcn, x = get.pars(m))
a2 <- colSums(res$grad)
```

```
all.equal(a1, a2, check.attributes = FALSE)

# analytical derivatives are evaluated faster than numerically
system.time(a1 <- numDeriv::grad(func = fcn, x = get.pars(m)))
system.time(a2 <- colSums(stsm.sgf(m, gradient = TRUE)$grad))
```

Index